RESUMEN
DEL LIBRO
“Cómo Progrmar en Java”
2002

GRUPO # 22

Alumnos:

Aguilar Elba
Barrios Miguel
Camacho Yaquelin
Ponce Rodríguez Jhonny

RESUMEN
DE LOS CAPITULOS 1 AL 6
DEL LIBRO “Como Programar en Java”
Autor: Deitel & Deitel

POR: GRUPO # 22

Alumnos:

Aguilar Elba
Barrios Miguel
Camacho Yaquelin
Ponce Rodríguez Jhonny

JAVA ESTUDIA

- Introducción a las computadores y a las applets de Java.
- Cómo funcionan, y
- Cómo se programan.
- Presenta una breve historia de desarrollo de los lenguajes de programación, desde los lenguajes de máquina hasta los lenguajes de ensamblador y los lenguajes de alto nivel.
- Explica el origen del lenguaje de programación Java (donde detalla la toma de decisiones y las operaciones aritméticas en java. Enseña a escribir programas sencillos).
- El primer cap. desarrolla la POO en Java.
- El primer programa utiliza herencia para crear un applet de java y luego exhibe la ventana de applet.
- Presenta fundamentos de programación **HTML** y explica la relación entre HTML, la World Wide Web y la Internet.
- Se utiliza los rótulos **Label** y campos de texto **TextField** para solicitar entradas y leerlas del teclado.
- Con Java es posible escribir dos tipos de programas:
  Applets diseñadas para transportarse de la www como Netscape Navigator y, aplicaciones diseñadas para ejecutarse en la computadora del usuario.

CAP. I. INTRODUCCION A LAS COMPUTADORAS Y A LAS APPLETS DE JAVA

Objetivos:

- Entender conceptos básicos de las ciencias de la computación.
- Familiarizarse con diferentes tipos de lenguaje de programación.
- Entender el entorno de creación de programas en Java.
- Poder escribir programas sencillos en java
- Poder usar enunciados de E/S sencillos
- Familiarizarse con los tipos de datos fundamentales.
- Poder emplear operadores aritm..
- Entender la procedencia de los operadores aritm.
- Poder escribir enunciados de toma de decisiones sencillos.

1.1. Introducción:

- Java (1995-1996), siendo una herramienta de aprendizaje para principiantes y programadores.
- Presenta los fundamentos de las computadores, su programación y el lenguaje Java.
- El **software** (son instrucciones escritas para ordenar a las computadoras que realicen **acciones** y tomen **decisiones**) **controla las computadoras** (hardware), y **Java** es **un lenguaje** para la creación del software.
- **Java** fue desarrollado por **Sun Microsystems,** y disponible en Internet (http://www.javasoft.com.).
- Se graban en la superficie de pastillas de silicio, bajo costo (ingrediente de la arena común y pequeñas)
- La tecnología de los chips de silicio a hecho a la computación económica.
- Trabaja con objetos (POO) y crea la estructura interna con dichos objetos, utiliza las técnicas de la programación estructurada.

Java ofrece beneficios.

- Aplicaciones provistas de interfaces gráficas con el usuario (**GUI = Graphical User Interfaces**), permite aprovechar las capacidades de multimedia de gráficos, imágenes, animación audio y video. Aplicaciones ejecutable en redes de compiladores con la internet y comunicación con otras aplicaciones. Mejoras de rendimiento y flexibilidad que son posibles con el procesamiento multihilos.

Recapitulación:

1. **Fundamentos de las computadoras y de su programación.**
2. **Escribir programas sencillos en Java.**

1.2. ¿Qué es una computadora?

- Una computadora es un dispositivo capaz de realizar cálculos y tomar decisiones lógicas a velocidades hasta miles de millones de veces más altas que las alcanzables por los seres humanos.
- Procesan **datos** bajo el control de series de instrucciones llamadas **programas de computadora.**
- Los **diversos dispositivos** (teclados, pantalla, discos, memoria y unidades de procesamiento) constituyen un sistema de computo (es el hardware).
- Los **programas que se ejecutan** en la computadora se llaman **software.**

1.3. Organización de las computadoras: Son 6:

1) **unidad de entrada.-** Sección de **recepción** (diversos dispositivos de E.), coloca la información a disposición de las demás unidades para ser procesadas.
2) **unidad de salida.-** Sección de **embarque** (toma la inform. Procesada y lo coloca en diversos dispositivos de S.) exhibe en pantalla, imprime, etc.
3) **unidad de memoria.-** Sección de **bodega**, de rápido acceso (en ella se conserva la información para procesarla)
4) **unidad de artim. Y lógica (ALU).-** Sección de **fabricación** (se encarga de realizar cálculos matem. Baja capac. de la comput.)
5) **Unidad central de procesamiento (CPU).-** Sección **administrativa** es el coordinador de la máquina, supervisa el funcionamiento de las otras secciones)
6) **Unidad de almacenamiento** secundario..- Sección **bodega (**a largo plazo y alta capac. De la comput. discos)

1.4. Evolución de los sistemas operativos:

- Las primeras computadoras, en su procesamiento funcionaban por lotes o tareas (en tarjetas perforadas), luego apareció el Sistema Operativo (Software) con el fín de hacer más cómodo el uso de la computadora.
- Fue aumentando la potencia de las computadoras, apareció la multiprogramación, implica el funcionamiento simultaneo de muchos trabajos.

1.5. Computación personal, distribuida y cliente/servidor

- En 1977, Apple Computer popularizó el fenómeno de la computación personal. De bajo precio.
- En 1981 la IBM, proveedor de computadoras más grande, introdujo el PC. Se distribuyen a través de redes.
- Los servidores de archivos ofrecen un almacen común para programas y datos que pueden ser utilizados por computadoras clientes distribuidos en la red.
- C y C++ son leng. de progr. Para escribir software.
- Sistema Operativo (Software: Unix, win 95, WNT).

1.6. Lenguajes:

- **Leng. de Máquina**, **definido por el diseño del HARDWARE**, consisten en cadena de números 1, 0; ordena a la computadora realizar operaciones una a una.
- **Leng. Ensamblador,** utiliza abreviaturas de palabras en Ingles para representar operaciones.(LOAD SALBASE, ADD pagoEXTRA, STORE SALBRUTO).
- **Leng. de Alto Nivel**, acelera el proceso de programación.
- **Compiladores,** son programas traductores que convierten los programas escritos en leng. de alto nivel a leng. de máquina.

.

1.7. Historia de C++

- Evolucionó a partir de C. En 1967 Martin Richards desarrolló el lenguaje para escribir software de Sist. Oper. Y compiladores.
- Los POO son fáciles de entender, corregir, y modificar.
- C++ es un lenguaje híbrido.

1.8. Historia de JAVA (café) Sun Microsystems

- World Wide Web (1993), la gente de Sun detectó el potencial de **Java para la creación de páginas Web**, con “contenido dinámico”.
- Java fue anunciada en 1995.
- Java fue diseñada con motivos comerciales y generó interés en la comunidad de los negocios por causa de otro avance relacionado con la Internet.
- **Paquetes de Java**: El leng. Java proporciona una serie de paquetes que incluyen ventanas, utilidades, un sistema de E/S, herramientas y comunicaciones.
- En la versión actual del JDK, los paquetes Java que incluyen son:
  - **Java.applet**, este paquete contiene clases diseñadas para usar con applets. Hay una clase Applet y tres interfaces: AppletContext, AppletStub y AudioClip.
  - **Java.awt,** el paquete **Abstract Windowing Toolkit (awt)** contiene clases (Label, TextField) para generar widgets y componentes GUI (interfaz Gráfico de Usuario), incluye las clases Button, Checkbox, Choice, Component, Graphics, Menu, Panel, TextArea y TextField.
  - **Java.io,** el paquete de E/S, contiene las clases de acceso a ficheros: FileInputStream y FileOutpuntStream.
  - **Java.lang,** este paquete incluye las clases de lenguaje Java propiamente dicho (Object, thread, Exception, System, Integer, Flota, Math, String, etc.)

- **Java.net,** este paquete da soporte a las concexiones del protocolo TCP/IP y además incluye las clases Socket, URL y URLConnection.
- **Java.util,** este paquete es una miscelánea de clases útiles para muchas cosas en programación. Se incluyen entre otras, Date (fecha), Dictionary, Random(números aleatorios) y Stack(pila FIFO).

1.9. Biblioteca de clases de java

- Los programas Java consisten en piezas llamadas **clases y métodos.**
- Tiene dos aspectos:
  - Es aprender el leng. java para programar
  - Aprender a usar las clases y métodos de las extensas bibliotecas de clase java.
- Los **creadores de clases son los proveedores de compiladores**.

1.10. Otros lenguajes de alto nivel:

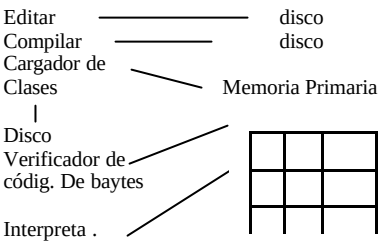
- Fortran (traductor de formulas desarrollado por IBM para ingeniería (1957)
- Cobol (Leng. orientado hacia los negocios comunes, 1959)
- Pascal Creado por Nicklaus Wirth..

1.11. Programación estructurada.

- La creación del software es una actividad compleja.
- En 1971, Nicklaus Wirth en honor a Blaise Pascal, la prog. Estruc. de Pascal, da a conocer un enfoque disciplinado para escribir programas, fáciles de probar, depurar y modificar. En.
- **Ada** (leng. de programación) 1970-1980, en honor de Lady Ada Lovelace (hijo del poeta Lord..) **Ada, Multitares,** su capacidad en actividades son en paralelo..
- **Charles Babbage** lanza la máq. Analítica.
- **JAVA MANEJA EL MULTIHILADO.**

1.12. Fundamentos de un entorno Java típico:

- Los sistemas Java constan de varias partes:
  - Un entorno
  - El lenguaje
  - La Interfaz de Programas de Aplicaciones (API), y
  - Diversas bibliotecas de clases.



- Los **programas Java** normalmente **pasan por cinco fases** antes de ejecutarse:
  - **Editar**, se utiliza un editor de textos y los archivos se guardan con la extensión **.java** (se crea en el editor y se almacena en Disco el programa)
  - **Compilar,** Se utiliza el **comando "javac"** seguido por el **nombre del archivo con extensión java**, luego se crea otro archivo con extensión **.class** el contenido códigos de baytes que serán interpretados durante la fase de ejecución.
    (El compilador crea **código de baytes** y los almacena en disco. El programador emite el comando **JAVAC** para compilar el programa -traduce en códigos de baytes-,
    ej. welcome.java, teclee
    Javac welcome.java
  - **Cargar,** toma el archivo **.class** que contiene códigos de baytes y lo transfiere a la memoria.(El cargador de clases coloca código de baytes en la memoria)
    El archivo .class puede cargarse de un disco de su sistema a través de una red.
    El cargador de clases comienza a cargar archivos .clases en dos situaciones**:**
    Ej. el comando java welcome invoca el INTERPRETE java para el programa welcome y hace que el cargador de clases cargue la información empleada en el programa welcome(que es una Aplicación).
    Las **aplicaciones** son programas que son ejecutados por el intérprete de Java.
    El cargado de clases se ejecuta cuando una applet de java se caraga en un navegador de la WWW como navigator de netscape o Hotjava de Sun.
    Los applets son programs de java, se cargan de la internet mediante un navegdor.
    Los applets pueden ejecutarse desde la línea de comandos utilizando el comando appletviewer –JDK- (welcome.html) incluido en el java Devoloper's Kit .
    Appletviewer requiere un doc. HTML parainvocar una applet.
  - **Verificar,** el verificador de códigos de bytes confirma que todos los códigos sean válidos y que no se violen las restricciones de seguridad de Java (que no causen daños a los archivos y sist. del usuario). (Este confirma que los códigos sean válidos y no se violen las instrucciones de seguridad de java –antes de que el intérprete (o appletviewer) pueda ejecutar los cod. de baytes, estos son verificdos por el verif. de cod. de baytes. Los programas que llegan por la red

podráin causa daños a los archivos para ello java debe hacer cumplir reglas de seguridad-

- **Ejecutar,** el interprete lee códigos de bytes y los traduce a un leng. de la computadora.(Lee códigos de bytes y los traduce a lenguaje de máquina y los almacena los datos conforme ejecuta. – la computadora controla por su CPU interpreta un código de byte a la vez-.

```
//un primer programa en Java -Applet HolaMundo-
import java.awt.Graphics;
import java.applet.Applet;

public class HolaMundo extends Applet{
  public void paint(Graphics g){
    g.drawString("Hola mundo !",25, 25):
  }
}
```

Siguiendo los pasos anteriores, a este código lo editamos y lo debemos de guardar en el disco con el nombre de la clase, seguido por la extensión java, HolaMundo.java, luego lo compilamos ejecutando el sigte comando javac HolaMundo java , el cual no crea un archivo llamado HolaMundo.class, luego como es una applet, editamos un archivo y lo guardamos con nombre HolaMundo html, donde llamamos mediante código html al archivo class, ejemplo <applet code="Addition.class" width=375 height=100. (si no fuera una applet ejecutaríamos el comando java HolaMundo.class) el sistema verifica y se ejecuta la applet Mostrándonos en pantalla.

1.13. Un recorrido por el libro.

- **Cap. 1**, introducción a las computadoras y a las applets de Java.
  El libro utiliza una técnica que llamamos **enfoque de código en vivo.**
  Con Java es posible escribir dos tipos de programas: **applets diseñadas para transportarse por la internet y ejecutarse en navegadores de la www como Netscape Navigator, y aplicaciones autónomas diseñadas para ejecutarse en la computadora del usuario.**
  Las applets de Java y las interfaces gráficas con el usuario.
- **Cap. 2**, Creación de aplicaciones Java.
  Proceso de desarrollo de programas.
  Explica cómo tomar un enunciado de problema..
  Desarrollar un programa Java funcional a partir de él.
- **Cap. 3**, Estructuras de control: secuencia, selección y repetición.
  Práctica de POO en Java (en el interior de los objetos que construimos se utilizan en abundancia las estructuras de control).
- **Cap. 4**, "Métodos" examinando el interior de los objetos. Los objetos contienen datos (variables de ejemplar) y unidades ejecutables(métodos, inclusive el recursivo).
- **Cap. 5** "Arreglos", los arreglos se procesan como objetos en java.
- **Cap. 6**. "Programación basada en objetos"; clase, objeto, construcción y destrucción de éstos, las clases son un mecanismo para empacar software en forma de componentes reutilizables?
- **Cap. 7** "POO", estudia las relaciones entre clases y objetos y la programación con clases relacionadas entre sí.
- **Cap. 8** "Cadenas y caracteres". La diferencia clave entre Java y C (cadenas de Java se tratan como objetos)
- **Cap. 9** "Gráficos"
- **Cap. 10** "Componentes básicos de una interfaz gráfica con el usuario".
- **Cap. 11** Componentes avanzados de una interfaz gráfica con el usuario.
- **Cap. 12** "Manejo de excepciones". En Java, cuándo un componente (esto es un objeto de una clase) se topa con un problema, puede lanzar una excepción. El entorno de ese componente se programa con el fin de atrapar dicha excepción y ocuparse de ella.
- **Cap. 13**. Multhilos, se ocupa de la forma de programar applets y aplicaciones capaces de realizar múltiples actividades en paralelo. Se construyen hoy en día procesadores que trabajan en paralelo, a ellos se llaman multiprocesadores.
- **Cap. 14**, "Multimedia", se ocupa de las capacidades de Java para imágenes, animación, audio.
- **Cap. 15** "Archivos y flujos", Archivos secuenciales, archivos de acceso aleatorio, los buffers.
- **Cap. 16,** "trabajo con redes", se ocupa de las applets y aplicaciones que pueden comunicarse a través de las redes de computadoras.
- **Cap. 17,** "Estructuras de datos", trata la organización de los datos en agregados útiles como listas enlazadas, pilas, colas y árboles.
- **Cap. 18**, "Paquete de utilidades de Java y manipulación de bits", Dos de las más útiles son **Vector y Stack** (una estructura de datos dinámica que permite inserciones y eliminaciones sólo por un extremo llamado tope).

1.14. Notas generales sobre Java y este libro

- Sólo están disponibles intérpretes de Java en el sitio del cliente.
- La applet podría llegar desde cualquier servidor de Web del mundo, por tanto, la applet deberá ser capaz de ejecutarse en cualquier posible plataforma de Java..

1.15. Introducción a la programación en Java

- Desarrollo de programas y el control de programas en Java.

1.16. Un programa sencillo: Impresión de una línea de texto.

- Una **applet** que exhibe una línea de texto.
- **Una applet** es un programa que se ejecuta en un navegador (browser) de la World Wide Web como Navigator de Netscape o Hotjava de Sun.
- Java contiene Componentes predefinidos llamados **clases** que se agrupan mediante directorios de disco en categorías de clases relacionadas entre sí llamados **paquetes** (bibliotecas de clases de java o interfaz de programación de aplicaciones de Java -Java API-).

```
//Un primer programa en Java
```

Los comentarios se insertan para **documentar** los programas y hacerlos más comprensibles, describe el **propósito del programa.**

Paquete *clase*

```
import java.applet.Applet ; //importar la clase Applet
import java.awt.Graphics ;  //importar la clase Graphics

public class Welcome extends Applet{ //inicia una definición Class de applet para la clase
        nombre del archivo              Welcome "bienvenido".
  public void paint( Graphics g )   //introduce un bloque de construcción de programa llamado método.
  {       método pintar
    g.drawString( "¡Bienven..a la program..en Java!", 25, 25) ;//exhibe la frase en pantalla
  }                                                    x, y coordenadas
}
```

- Son enunciados **import**, que cargan las clases necesarias para compilar un programa Java. Le dicen al compilador que debe **cargar la clase Applet del paquete java.applet** y **cargar la clase Graphics del paquete java.awt**.
- Al crear una applet en Java es necesario importar la clase Applet.
- Importamos la clase Graphics para que el programa exhiba información en pantalla.
- **Extendes** (extiende) seguida de un nombre de clase indica la clase de la cual nuestra nueva clase heredará algunas partes.
- Todas las applets de Java deben heredar de la clase Applet (Superclase o clase base).
- Welcome es la subclase o clase derivada.

Clase:

- Las clases sirven para ejemplarizar (o crear) objetos en la memoria.
- Un objeto es una región de la memoria en la que se almacena información para ser utilizada por el programa.
- La clase Welcome sirve para crear un objeto que ejecuta la applet.
- La **palabra Public** (publica), permite al navegador crear y ejecutar la applet. Luego inicia con llaves. Public es necesaria para que el navegador invoque el método paint.
- El **método paint** es invocado durante la ejecución de la applet y sirve para exhibir información en la pantalla. **La lista de parámetros del método paint** indica que **requiere un objeto Graphics g** para realizar su tarea.
- Los **métodos** pueden realizar tareas y devolver información.
- **Void (vacío),** indica que éste método realiza una tarea, no devuelve ninguna información.

Public void paint(Graphics g)

- Esta línea, introduce un bloque de construcción de programa llamado método.
- Todas **las definiciones de métodos comenzarán con** la **palabra clave public.**

g.drawString("¡Bienvenidos a la programación en Java!", 25, 25)

- Esta línea ordena a la computadora a realizar una **acción,** es decir, **usar el método drawString(dibujar cadena)** del objeto **Graphics g**; entonces exhibe el mensaje en la pantalla en las coordenadas indicadas.
- La PC tienen pantallas con una anchura de 640 píxeles y una altura de 480 píxeles, para un total de 307.200 elementos de imagen exhibibles.

- Después de compilar el programa, es necesario **crear un archivo HTML** (Hypertext Markup Language) para **cargar la applet en un navegador** y ejecutarla. **Los navegadores leen archivos de texto**.
- Ejemplo:
- //indican principio y final de las codificaciones html del archivo.
- // Cada clase se compila en un archivo individual que tiene el mismo nombre que la clase y termina con la extensión .class.
- <html>
- indican al navegador que debe cargar una applet específica y definen el tamaño de la applet en el navegador.
- Indica el archivo que contiene la clase de applet compilada que hemos definido.. Arnchura y y altura de la applet
- <applet code=”Welcome.class” widht=275 height=35>
- //indica al navegador que debe cargar una applet específica, define el tamaño de la applete en el navegador.
- </applet>
- </html>
- El **archivo Welcome.html que** carga a la clase de applet definida anteriormente.
- Para demostrar nuestra applet, usamos el comando **appletviewer** del Java Developer’s Kit (JDK) de Sun.
- El comando appletviewer es un programa que prueba applets de java; sólo entiende la etiqueta HTML applet (navegador mínimo). Invocamos appletviewer para la applet Welcome desde la línea de comandos de nuestra computadora:
- Appletviewer Welcome.html// appletviewer necesita un archivo HTML para cargar una applet.
- El mensaje se puede imprimir de varías formas. Dos enunciados drawString en el método pain pueden imprimir múltiples líneas.

1.17. Otro programa en Java

- El usuario digita enteros, calcula la suma de estos valores y exhibe el resultado. El programa lee el entero y lo suma al total.
- Para ello, el programa emplea algunos **componentes de interfaz gráfica con el usuario** (GUI) de la java API.
- Los componentes de GUI facilitan la E/S de datos.
- Todos los programas Java se basan en por lo menos una def. de clase que utiliza mediante herencia.

- Importa el paquete java.awt -usando las clases TextField, Label, (antes Graphics) de java.awt- para que el compilador pueda compilar esta applet.

- El asterisco (*) indica que el compilador debe tener a su disposición todas las clases del paquete java.awt para la compilación.

- Prompt e input son nombres inventados para referirse a **objetos Label y TextField** en el programa. **Un objeto** contiene información que el programa usa.

- number y sum son nombres de variables. **Una variable es similar a un objeto**, la diferencia es que un objeto se define mediante una definición de clase que puede contener información o métodos, en tanto que una variable se define mediante un tipo de datos.

- Cada elemento de datos en un programa Java es un objeto, con excepción de las variables de tipos primitivos.

- La referencia **input** es del tipo **TextField** (campo de texto). Los campos de texto sirven para obtener información que el usuario introduce mediante el teclad, o para exhibir información en pantalla.

- La referencia **prompt** es de tipo **Label** (rótulo). **Un Label** contiene una literal de cadena que se exhibe en pantalla. Cada rótulo está asociado a otro elemento de la interfaz gráfica en pantalla (como el campo de texto en este programa).

- Un **nombre de variable** es cualquier **identificador válido**. Un identificador es una serie de caracteres: letras, dígitos, subrayados y signos de dólar, que no comienza con un dígito. Java permite identificadores de cualquier longitud y es sensible a la caja. Ej. a1 y A1 son dos identificadores distintos.

- La declaración de una variable en un método debe aparecer antes de que se use dicha variable en el método. Una var declarada en un método no puede ser accedida por otros métodos.

- Incluimos el método **init** en un applet para **inicializar** las variables y referencias que se usan en la applet. **El método init se invoca automátic. c**uando la applet comienza su ejecución (1er. Método invocado).

- Las applets siempre comienzas con una serie de tres llamadas a métodos: **init, start y paint.**

- **El método action (acción) procesa interacciones** entre el usuario y los componentes de interfaz gráfica con el usuario (GUI) de la applet.

- .

```
//Programa de suma (propósito del programa)
import java.awt.*;        //importar el paquete java.awt
import java.applet.Applet; //importar la clase Applet (las applets heredan de la clase Applet.

public class Addition enxtends Applet{ //(suma)hereda de Applet.
   Label promt; //(solicitud) pedir al usuario que teclee valor o entrada al usuario
   TextField input; //(introducción) ingresar valores aquí
   Int number; // almacenar valor introducido (var)
   Int sum; // almacenar suma de enteros  (var)

  //preparar componentes de interfaz gráfica de usuario(GUI), e inicializar variables
   //init es un método public, que no devuelve información (void) cuando termina de
   ejecutarse y que no recibe argumentos (init incluido en una applet)

  public void init()
  {
    //*esta línea crea un objeto Label que inicializa con la literal de cadena "teclee...",
    Se usa esta cadena para rotular el campo de texto input en la applet.
    //Pide al usuario que emprenda una acción específica (prompt=solicitud).
    //El operador new crea un objeto durante la ejecución del programa, pide memoria sufic.
    para almacenar un objeto de tipo que se especifica a la der de new. El valor dentro del
    paréntesis se usa para inicializar (asignar un valor inicial a) el objeto creado.
    Asignando a la referencia prompt el resultado de la operación new.

    *prompt = new Label ("Teclee un número y pulse enter:");//prompt obtiene el valor de new label...

    //crea un objeto TextField y asigna el result. a input. 10 es el tamaño en caracteres del campo de texto.
    input = new TextField( 10 );

    //El método add es un método de la clase Applet que heredamos cuando definimos la clase
    Addition. Colocan los componentes Label y TextField  en la applet para que el ususario
    pueda interactura con ellos.

     add( prompt ); // poner prompt (solicitud) en la applet
    add( input ); // poner input(entrada) en la applet
    sum = 0; //hacer sum (suma) igual a cero
  }
   // = es un operador binario, tiene dos operandos: prompt y el valor de la expresión new Label ("Teclee....")
  //procesar la acción del usuario en el campo de texto de entrada

  //El método action procesa interacción entre el usuario y los componentes de GUI de una
  applet.
  Una interacción genera un evento. Los eventos le indican a la applet que el usuario está
  interactuando con uno de los componentes de GUI de la applet.
  Ej. cuando se teclea un número para ingresar en el campo de texto y pulsamos enter, se
  envía un evento a la applet. (que es un objeto de la clase Event del paquete java.awt).
  El método action solo se invoca cuando el usuario pulsa la tecla enter en el campo de
  texto. A ello se llama programación controlada por eventos..
  Action es un método public que devuelve un valor boolean. Recibe argumentos: un Event y
  un object. El argumento  Event sirve para determinar qué evento ocurrió. El argumento
```

Object contiene información para el evento; este contiene el texto que estaba en el campo de texto.

Una característica de Java, **cualquier objeto** de cualquier tipo de clase **se puede convertir** en un **String, empleando el método toString**. Un **String es un objeto** que puede almacenar cadena.

```
public boolean action( Event e, Object o) {  //método public
   //Asigna a number el resultado del método Integer.parseInt que convierte su argumento
   String en un entero. La cadena que se convierte, es el resultado del método
   o.toString(), que convierte el Objetct o en un String.

   number = Interger.parseInt( o.toString());//(método) obtener número

   //Esta línea utiliza el método setText de la clase TextField para asignar la cadena
   vacía al campo de texto input. Esto despeja (borra) el texto que el usuario había
   tecleado en el campo de texto.

   input.setText( ""); // despejar campo de entrada de datos

   calcula la suma de la var sum y number y asigna el resultado a la var sum.
   sum = sum + number; // suma number(número) a sum (suma)

   Luego se exhibe el resultado en la barra de estado (ventana appletviewer)
   Esta línea utiliza el método showStatus de la clase Applet para colocarun String en
   la barra de estado. El String que se exhibe es el resultado del métod
   Integer.ToString (sum) que toma el entero sum y lo convierte en un String..

   showStatus( Integer.toString( sum ) ); //mostrar resultados

   //esta línea le indica a action que devuelva el valor true a la applet después de
   terminar su tarea.  Le dice que ya se procesó la acción del usuario que causó el
   evento.

   return true; // indica que la acción del usuario se procesó
  }
}
```

1.18. Conceptos de memoria.

- **sum y number corresponden a posiciones** en la memoria de la computadora.
- Toda variable tiene un nombre, un tipo, un tamaño y un valor.

1.19. Aritmética

- Casi todos los programas realizan cálculos aritméticos. Sum + value conteien el operador binario + y los dos operandos sum y value.
- Las expresiones aritméticas en Java deben escribirse en forma de línea recta para facilita la introducción de programas en la computadora.
- Usamos paréntesis en las expresiones Java de la misma forma que en las expresiones algebraicas.
- Java aplica los operadores de las expresiones aritméticas en un orden preciso determinado por reglas de precedencia de operadores, que son las mismas que se siguen en álgebra. (), *, /, %, +, -

1.20. Toma de decisiones: Operadores de igualdad y relacionales

- Las condiciones de las estructuras if pueden formarse empleando los operadores de igualdad y operadores relacionales.
- Todos los operadores relacionales tiene le mismo nivel de presencia y se asocian de izq. a der.: ==, ¡=, <=, >=, etc.
- If compara dos números introducidos en campos de texto. Si satisface la condición if, se ejecuta el enunciado drawString asociado a ese if. Las comparaciones se efectúan en el método paint.

If (event.target == input2) {

Esta línea determina si el usuario pulsó return en el campo de texto input2 }.

Cuando el usuario presiona la tecla enter en un campo de texto se genera un evento que produce una llamada al **método action.** El **objeto Event** que se pasa al **método action** contiene muchos elementos de información.

El elemento más importante es el **target (objetivo)** del evento. El **target es el componente de la interfaz gráfica** con la que el usuario interactuó para **generar el evento**.

La condición if –event.target == input2- pregunta: ¿es el objetivo del evento el campo de texto input2?

Los enunciados: num1=Interger.parseInt(input1.getText());
Num2=Interger.parseInt(input2.getText());.

Leen el texto introducido por el usuario en los campos de texto input1 e input2, convierten en texto en enteros y almacenan los valores en las variables num1 y num2.

El **método de getText** de la clase TextField **sirve para leer el texto de cada campo de texto**. Cuando se ejecuta **input1.getText()** se **obtiene el texto** del campo de texto input1.

Repaint() invoca al método paint a nuestro nombre. Este método es heredado cuando definimos nuestra **clase comparison**(comparación) para extender la clase Applet.

Al llamar a repaint, se invoca otro método heredado llamado update (actualizar) . Este método borra cualquier información que se haya exhibido antes en la applet con paint y luego invoca otra vez a paint.

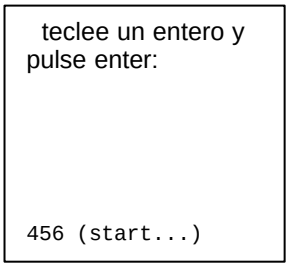
La estructura if en paint:

```
if (num1 == num2 )
g.drawString (num1 + “==” + num2, 100, 90 );
```

compara valores de las var num1 y num2 para determinar si son iguales. Si lo son, el método drawString exhibe el resultado de la expresión num1 + “== + num2en la pantalla. Suma un entero a una literal de cadena y luego suma otro entero al resultado de la primera suma.
En Java el **signo +** permite juntar una cadena y un valor de otro tipo de datos (u otra cadena), el resultado es una cadena nueva.

```
//Empleo del enunciado if, operadores relacionales y operadores de igualdad
import java.awt.*;
import java.applet.Applet ;

public class comparison extends Applet{

   Label prompt1; //pedir al usuario que teclee primer valor
   TextField input1; // introducir primer valor aquí
   Label prompt2; //pedir al usuario que teclee 2do valor
   TextField input2; // guardar valores introducidos aquí
   Int num1, num2; // almacenar valor introducido

   //establecer componentes de interfaz gráfica con el usuario e inicializar variables.
   Public void init()
   {
         prompt1 = new Label ( “Teclee un número” );
         input1 = new TextField (10);
         prompt2 = new Label ( “Teclee un número entero y pueso enter” );
         input2 = new TextField (10);
         add( prompt1 ); //colocar prompt1 en la applet
         add( input1 ); //colocar input1 en la applet
         add( prompt2 ); //colocar prompt2 en la applet
         add( input2 ); //colocar input2 en la applet }

   }
   //exhibir los resultados

   public void paint (Graphics g)
   {
       g.drawString (« Los resultados de la comparación son:”, 70, 75);

       if (num1 == num2 )
          g.drawString (num1 + “==” + num2, 100, 90 );
       if (num1 != num2 )
          g.drawString (num1 + “!=” + num2, 100, 105 );
       if (num1 < num2 )
          g.drawString (num1 + “<” + num2, 100, 120 );
       if (num1 > num2 )
          g.drawString (num1 + “>” + num2, 100, 135 );
       if (num1 >= num2 )
          g.drawString (num1 + “>=” + num2, 100, 150 );
```

```
          if (num1 <= num2 )
            g.drawString (num1 + "<=" + num2, 100, 165 );.
      }
      // procesar la acción del usuario en el campo de texto input2
      public bolean action ( Event event, Object o )
      {
          if ( event.target == input2 ) {
          num1 = Integer.parseInt( input1.getText() );
          num2 = Integer.parseInt( input2.getText() );
          repaint();
         }
         return true; //indica que se procesó la acción del usuario
      }
   }
import java.awt.*;// importar el paquete java.awt
import java.applet.*;

public class Suma extends Applet {

      Label L1;//solicitar entrada al usuario
      TextField E1;// introducir valores aquí
      int number; //para almacenar valor introducido
      int sum;// para almacenar suma de enteros

      //preparar componentes de interfaz gráfica de usuario e inicializar variables
      public void init() {
            L1 = new Label("teclee un entero y pulse enter:");
            E1 = new TextField( 10);

            add(L1);//poner L1(solicitud) en la applet
            add(E1);// poner E1(entrada) en la applet
            sum = 0;// hacer sum igual a cero
      }
//procesar la acción del usuario en el campo de texto de entrada
      public boolean action (Event e, Object o) {
            number = Integer.parseInt(o.toString()); // obtener número
            E1.setText("");// despejar campo de entrada de datos
            sum = sum + number; //suma número a suma
            showStatus(Integer.toString(sum)); //mostrar resultados
            return true; // indica que la acción del usuario se proceso
      }
}
```



Son declaraciones:	**Detalle**	//
Label prompt(L1)	- Prompt (solicitud) **L1** y Input(introducción) **E1** son referencias o nombres que usamos para referirnos a nuestros **objetos Label(rotulo) y TextField (campo de texto)** en el programa. - Un objeto contiene información. Un objeto se define mediante una def. de clases que puede contener tanto información como métodos.	Solicitar entrada al usuario
TextField input(E1)		Introducir valores aquí
Int number	- Number y sum son variables. Una var. se define mediante	Almacenar valor introducido

Int sum	un tipo de datos primitivo. - Un nombre de variable o de referencia es cualquier **identificador válido.**	Almacenar suma de enteros
public void init()	- El **método init** en una applet inicializa las variables y referencias - **Se invoca automáticamente cuando la applet comienza su ejecución** (1er. Mét. invocado) - Prepara componentes de interfaz gráfica de usuario e **inicializar variables y referencias que se usan en la applet.** - **Es un método públic que no devuelve información (void)** cuando termina de ejecutarse y que **No recibe argumentos**	
L1 = new Label("teclee un entero y pulse enter:");	- **New** crea un **objeto Label L1** que inicializa con una **cadena.** - Se usa esta cadena para rotular el campo de texto **E1** en la applet. Pide al usuario que emprenda una acción específica. - New pide memoria para almacenar un objeto del tipo que especifica a la derecha de new. - Asigna un valor inicial a la referencia L1 el resultado de la operación new (“ ...”) - Se asigna memoria dinámicamente a los objetos de tipo clase. - Java asigna memoria automáticam. A las variables de tipos primitivos.	
E1 = new TextField(10);	- **Crea un objeto TextField** empleando el operador new y asigna el resultado a **E1** - 10 es el tamaño en caracteres del campo de texto en la pantalla.	
add(L1); add(E1);	- El **método add** es de la clase Applet que heredamos cuando definimos la clase Addition. **Colocan los componentes Label y TexField en la applet** para interactuar con ellos.	poner L1(solicitud) en la applet poner E1(entrada) en la applet
sum = 0;	Asigna el valor 0 a sum.	hacer sum igual a cero
	- **action (acción**) es un **método**	

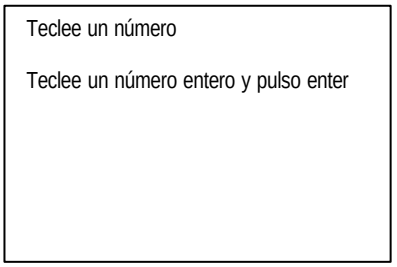
public **boolean action** (Event e, Object o) {	**public** que devuelve un **valor boolean** (true o false) al terminar su tarea. - El **método action sólo se invoca** cuando el usuario **pulsa la tecla enter** en el campo de texto. - Recibe dos argumentos: un **event (evento) y un object (objeto).** - **Event** determina **qué evento ocurrió**. - **Object** contiene información para el evento (contiene el texto que estaba en el campo de texto.) - **Action procesa interacción entre usuario y componentes de GUI** de una applet. - **Genera un EVENTO**. Los **EVENTOS** le indican a la applet que el usuario está interactuando con GUI de la applet. - Al pulsar **ENTER, se ENVÍA UN EVENTO** (que es un objeto de la clase Event del paquete java.awt).	

number = Integer.parseInt(o.toString());
 	
//toString: convierte en cadena **o** (obj)
//Integer.parseInt: convierte su argumento string en un entero.
	- En java cualquier **objeto** de cualquier tipo de clase **se puede convertir** en un **String(cadena**) empleando el método **toString.**
	- Un String es un objeto que puede almacenar cadena.
	- Asigna a number el resultado del método **Integer.parseInt** que **convierte su argumento string en un entero**.
	- La cadena (string u object) que **se convierte es el resultado del método o.toString()**
	- La cadena que se convierte, es el resultado del método o.toString(), que convierte el Objetct o en un String.
		obtener número

E1.setText("");	- Se Utiliza el **método setText** de la clase TextField para **asignar la cadena vacía**, al **campo de texto E1(input)**. Esto despeja (borra) el texto que el usuario había tecleado en el campo de texto.	Despeja(borra) campo de entrada de datos
sum = sum + number;	- Calcula la suma de las variables sum y number y asigna el resultado a la var	suma número a suma

	sum	
showStatus(Integer.toString(sum)); //	- Se utiliza el **método showStatus** de la clase Applet para **colocar un String en la barra de estado**(inferior de la ventana del navegador o applet started), **d**espués de realizar el cálculo. - El String que se exhibe es el resultado del método **Integer.toString(sum)** que toma el entero sum y lo convierte en un String.	mostrar resultados
return	- Le indica a action que devuelva el valor true a la applet cuando ha terminado su tarea.	Indica que la acción del usuario se procesó

```
//Empleo del enunciado if, operadores relacionales y operadores de igualdad
import java.awt.*;
import java.applet.*;

public class Comparison extends Applet
{//declarando variables
      Label prompt1; //pedir al usuario que teclee primer valor
      TextField input1; // introducir primer valor aquí
      Label prompt2; //pedir al usuario que teclee 2do valor
      TextField input2; // guardar valores introducidos aquí
      int num1, num2; // almacenar valor introducido

//establecer componentes de interfaz gráfica con el usuario (GUI) e inicializar
variables.
      public void init()
      {
          ⎧ prompt1 = new Label ( "Teclee un número" );
asigno    ⎪ input1 = new TextField (10);
memoria   ⎨ prompt2 = new Label ( "Teclee un número entero y pulso enter" );
          ⎩ input2 = new TextField (10);

          ⎧ add( prompt1 ); //colocar prompt1 en la applet
coloco    ⎪ add( input1 ); //colocar input1 en la applet
en la     ⎨ add( prompt2 ); //colocar prompt2 en la applet
applet    ⎩ add( input2 ); //colocar input2 en la applet }

      }
//exhibir los resultados

      public void paint (Graphics g)
      {
            g.drawString (" Los resultados de la comparación son:", 70, 75);

                  if (num1 == num2 )
            g.drawString (num1 + "==" + num2, 100, 90 );
                  if (num1 != num2 )
            g.drawString (num1 + "!=" + num2, 100, 105 );
                  if (num1 < num2 )
            g.drawString (num1 + "<" + num2, 100, 120 );
                  if (num1 > num2 )
```



```
            g.drawString (num1 + ">" + num2, 100, 135 );
                  if (num1 >= num2 )
            g.drawString (num1 + ">=" + num2, 100, 150 );
                  if (num1 <= num2 )
            g.drawString (num1 + "<=" + num2, 100, 165 );
      }
// procesar la acción del usuario en el campo de texto input2
      public boolean action ( Event event, Object o )
      {
            if ( event.target == input2 ) {
                  num1 = Integer.parseInt( input1.getText() );
                  num2 = Integer.parseInt( input2.getText() );
                  repaint();
      }
   return true; //indica que se procesó la acción del usuario
      }
}
```

if (num1 == num2) g.drawString(num1 +"=="+ num2, 100, 90);	- La estructura if en paint: - **Compara valores** de las **var num1 y num2** para determinar si son iguales. Si lo son, el **método drawString exhibe** el resultado de la expresión num1 + "==" + num2 en la pantalla. - Suma un entero a una literal de cadena y luego suma otro entero al resultado de la primera suma. - En Java el **signo +** permite juntar una cadena y un valor de otro tipo de datos (u otra cadena), el resultado es una cadena nueva.	
Number = Integer.parseInt(o.toString());	- Sum y number son posiciones en la memoria. - Toda var. tiene un nombre, un tipo, un tamaño y un valor. - Si se introduce 45, la computadora coloca 45 en la posición number.	
Sum = sum + number.	- Una vez obtenido el valor, lo suma a la suma y coloca el resultado en la var sum. - La suma implica la destrucción de un valor.	

If (event.target == input2)
{
 	
input1 = E1
input2 = E2
	- Esta línea determina si el usuario pulsó **enter** en el campo de texto input2 para que se genere un evento al llamar al método action.
	- El objeto **Event** que se pasa al **método action,** contiene elementos de información que permiten al programador procesar el evento.
	- El elemento **target(objetivo**) del evento.
	- El **target es el componente de la interfaz gráfica** para generar el evento.
	- La **condición if** –event.target ==input2- pregunta: **¿es el objetivo del evento el campo de texto**	

Num1= Integer.parseInt(input1.getText()); Num2= Integer.parseInt(input2.getText());
 	
//Integer.parseInt=convierte en entero
// getText()=lee el texto de input1 //obtiene el texto
	input2?; si se cumple la condición, se ejecuta el cuerpo de la estructura if.
	- Estas líneas **leen el texto introducido en input1 e input2, convierten el texto en enteros y almacena los valores en las var. num1 y num2**.
	- El **método getText()** de la clase TextField **lee el texto** de cada campo de texto.
	- cuando se **ejecuta input1.getText(),** se **obtiene el texto** del campo de texto input1.	

Repaint()	- Este método invoca al método paint. - El **método repaint** se ha heredado cuando se ha definido nuestra clase **comparison(comparación**) para extender la clase Applet. - Invoca otro método heredado llamado update(actualizar). Update borra cualquier información que se haya exhibido antes en la applet con paint y luego invoca otra vez a paint.	

Ejercicios de autoevaluacion

1.1 Llene los espacios en blanco en cada una de las siguientes afirmaciones:

a) La compañía que popularizó la computación personal fue **Apple.**

b) La computadora que hizo respetable la computación personal en los negocios y la industria fue **Computadora personal IBM.**

c) Las computadoras procesan datos controladas por series de instrucciones llamadas **programas.**

d) Las seis unidades lógicas clave de la computadora son:
unidad de entrada,
unidad de salida,
unidad de memoria,
unidad de aritmética y lógica,
unidad central de procesamiento, y
unidad de almacenamiento secundario.

e) Las tres clases de lenguajes que se mencionaron en el capítulo son:
Lenguajes de máquina
Lenguajes de ensamblados
Lenguajes de alto nivel.

f) Los programas que traducen programas escritos en lenguajes de alto nivel a lenguaje de máquina se llaman **compiladores.**

g) El lenguaje **Pascal** fue desarrollado por Wirth para enseñar programación estructurada en las universidades.

h) El Departamento de la Defensa de Estados Unidos desarrolló el lenguaje Ada con una capacidad llamada **multitareas** que permite a los programadores especificar que muchas actividades pueden realizarse en paralelo..

1.2 Llene los espacios en blanco en cada una de las siguientes afirmaciones acerca del entorno Java.

El comando **appletviewer** del Java Developer's Kit ejecuta una applet de Java .

El comando **java** del Java Developer's Kit ejecuta una aplicación de Java.

El comando **javac** del Java Developer's Kit compila un programa Java.

Se requiere un archivo **HTML** para invocar una applet de Java.

Todo archivo de programa Java debe terminar con la extensión de archivo **.java**

Cuando se compila un programa Java, el archivo producido por el compilador termina con la extensión de archivo **.class**

El archivo producido por el compilador de Java contiene **códigos de bytes** que son interpretadas para ejecutar un applet o aplicación de Java.

1.3 Llene los espacios en blanco en cada una de las siguientes afirmaciones.

La **llave izquierda** comienza el cuerpo de todo método y la **llave derecha** termina el cuerpo de todo método.

Todo enunciado termina con un **signo de punto y coma (;)**

La estructura **if** sirve para tomar decisiones.

1.4 Indique si son verdaderas o falsas las siguientes afirmaciones. Si alguna es falsa, explique por qué.

a) Los comentarios hacen que la computadora imprima en la pantalla el texto que sigue a // cuando se ejecuta el programa.
Falso. Los comentarios no dan pie a ninguna acción cuando se ejecuta el programa. Sirven para documentar los programas y hacerlos más comprensibles.

b) Todas las variables que se usan en un método deben declararse antes de usarse.
Verdadero

c) A todas las variables se les debe dar un tipo al declararlas.
Verdadero.

d) Java considera idénticas las variables number y NuMbEr.
Falso. Java es sensible a la caja, de modo que estas variables son distintas.

e) La declaraciones pueden aparecer en casi cualquier lugar dentro del cuerpo de un método Java.
Verdadero.

f) El operador de residuo (%) solo puede usarse con operandos enteros.
Verdadero.

g) Los operadores aritméticos *./.%, + y – tienen todos el mismo nivel de precedencia.
Falso. Los operadores *, / y % están en el mismo nivel de precedencia, y los operadores + y – están en un nivel de precedencia más bajo.

1.5 Escriba enunciados en Java que lleven a cabo lo siguiente:

a) Declarar las variables c.estaEsUnaVariable, q76354 y number de tipo int.
int c, estaEs Una Variable, q76354, number;.

b) Solicitar al usuario que teclee un entero. Sugerencia: declare una referencia Label, cree un objeto Label y especifique cómo se debe colocar el Label en la applet.

```
Label prompt;
Prompt = new Label ("Teclee un entero");
Add( prompt );
```

c) Leer un entero del teclado y almacenar el valor introducido en una variable entera edad. Suponga que el TextField llamado input existe y que el argumento Object o del método action contiene el texto que el usuario introdujo.
Edad = Integer.parseInt(o.toString());

d) Si la variable number no es igual a 7, exhibir "La variable number no es igual a 7" en la coordenada x 10 y la coordenada y 10 del método paint. Suponga que el objeto Graphics g es el argumento del método paínt.

```
If ( number ¡= 7 )
  g.drawString("La variable number no es igual a 7..", 10, 10);
```

e) Imprima el mensaje "Este es un programa Java" en una línea del método paint. Suponga que el objeto Graphics g es el argumento del método paint y escoja sus propias coordenadas.
g.drawString("Este es un programa java", 10, 10);

f) Imprima el mensaje "Este es un programa java" en dos líneas, la primera de las cuales termina con programa. Suponga que el objeto Graphics g es el argumento del método paint y escoja sus propias coordenadas.
g.drawString("Este es un programa", 10, 10);
g.drawString("Java", 10, 25);.

1.6 Identifique y corrija los errores en cada uno de los siguientes enunciados:

```
If ( c < 7 );
       g.drawString( "c es menor que 7", 25, 25 );
If ( c => 7 )
       , 25, 25 );.
```

Error: Punto y coma después de paréntesis derecho de la condición del enunciado if. Corrección: elimine el punto y coma después del paréntesis derecho. Nota: El resultado de este error es que se ejecutará el enunciado de salida sea que se cumpla o no la condición del enunciado if. El punto y coma después del paréntesis derecho se considera como un enunciado vacío; un enunciado que no hace nada.

Error: El operador relacional => Corrección cambiar >=

1.7 Clasifique cada una de las siguientes cosas como hardware o software:

CPU = Hardware
Compilador de Java = Software
ALU = software
Intérprete de Java = software
Unidad de entrada = hardware
Un programa editor = software

1.8 ¿Por qué querría alguien escribir un programa en un lenguaje independiente de la máquina en lugar de en uno dependiente de la máquina? Por qué podría ser más apropiado usar un lenguaje dependiente de la máquina para escribir ciertos tipos de programas?

Porque los lenguajes de máquina son difíciles de usar para las personas.

1.9 Llene los espacios en blanco en cada uno de los siguientes enunciados:

¿Cuál unidad lógica de la computadora recibe información de fuera de la computadora para usarse dentro de ella?
La unidad de entrada

El proceso de instruir a la computadora para que resuelva problemas específicos se denomina **software.**

¿Qué tipo de lenguaje de computadora usa abreviaturas de palabra en inglés para las instrucciones de lenguaje de máquina?
Los lenguajes de ensamblador.

¿Cuál unidad lógica de la computadora envía información que ya ha sido procesada por la computadora a diversos dispositivos para que dicha información se pueda utilizar fuera de la computadora?
La Unidad central de procesamiento.

¿Cuál unida lógica de la computadora retiene información?
La unidad de memoria y la unidad de almacenamiento secundario.

¿Cuál unidad lógica de la computadora realiza cálculos?
La unidad de aritmética lógica.

¿Cuál unidad lógica de la computadora toma decisiones lógicas?
La unidad central de procesamiento.

El nivel de lenguaje de computadora más cómodo para el programador, para escribir programas rápida y fácilmente, es:
El lenguaje de alto nivel que reejecutan en programas intérpretes.

El único lenguaje que una computadora puede entender directamente es el **lenguaje de máquina** de esa computadora.

¿Cuál unidad lógica de la computadora coordina las actividades de todas las demás unidades lógicas?
Unidad central de procesamiento.

1.10 Llene los espacios en blanco en las siguientes afirmaciones:
Los **comentarios** sirven para documentar los programas y hacerlos más comprensibles.
Un objeto de la clase **TextField** puede recibir entradas que el usuario teclea o **exhibir** información en una applet.
Un enunciado de Java que toma una decisión es **if.**
Los cálculos normalmente se realizan mediante enunciados de **operación aritmética.**
Un objeto de la clase **Graphics** sirve para exhibir texto que normalmente está asociado a otro componente de GUI en una applet.

1.11 Escriba enunciados en Java que realicen las siguientes tareas:
a) Exhibir el mensaje **"Introduzca dos números "** empleando un **Label.**

```
Label L1
L1 = new Label ("Introduzca dos números");
```

b) Asignar el producto de las variables **b** y **c** a la variable **a.**

```
a = b * c;.
```

c) Indicar al programa que realice un cálculo de nómina sencillo (use texto que ayude a documentar el programa).

```
Public int sum(int a, int b)
{
   int s = a + b;
   return s;.
}
```

1.12 Indique cuáles de las siguientes afirmaciones son verdaderas y cuales falsas. Explique sus respuestas:
Los operadores de Java se evalúan de izquierda a derecha **True**
Todos los siguientes son nombres de variables válidos: under bar, m928134, t5, j7, her sales, his account total, a, b, c, z, z2

Una expresión aritmética válida en Java que no lleva paréntesis se evalúa de izquierda a derecha.

Los siguientes son nombres de variables no válidos: 3g, 87, 67h2, h22, 2h.

1.3. Llene los espacios en blanco en los siguientes enunciados:
a) ¿Cuáles operaciones aritméticas están en el mismo nivel de precedencia que la multiplicación?
/ y %
b) Cuando los paréntesis están anidados, cuál juego de paréntesis se evalúa primero en una expresión aritmética?
Desde el centro hacia afuera.
c) Una posición en la memoria de la computadora que puede contener diferentes valores en diferentes momentos durante la ejecución de un programa es una **variable.**

1.4 ¿Qué, si acaso, se imprime cuando se ejecuta cada uno de los siguientes enunciados en Java? Si no se emprime nada, responda "nada", Suponga que x = 2 y Y = 3.

```
g.drawString ( Integer.toString( x ), 25, 25 );

g.drawString ( Integer.toString( x + x ), 25, 25 );.

g.drawString ( "x=", 25, 25 );

g.drawString ( "x=" + x, 25, 25 );

g.drawString ( x + y ) + "=" + ( y + x ), 25, 25 );

z = x + y;
```

1.15 ¿Cuáles de los siguientes enunciados en Java contienen variables cuyos valores se destruyen?

```
p = i + j + k + 7;

g.drawString ("valores de variables destruidos", 25, 25 );

g.drawString( "a = 5", 25, 25 );
```

Dado $y = ax^3 + 7$ ¿cuáles de los siguientes son enunciados correctos para esta ecuación?

```
y = a * x * x * x + 7;
```

```
2  y = a * x * x * (x + 7);
3  y = (a * x) * x * (x + 7);
4  y = (a * x) * x * x + 7;
5  y = a * (x * x * x )+ 7;
6  y = a * x * (x * x + 7);
```

Indique el orden de evaluación de los operadores en cada uno de los siguientes enunciados en Java y diga qué valor tiene x después de ejecutarse cada enunciado.

```
x = 7 + 3 * 6 / 2 – 1; = 1
x = 2 % 2 + 2 * 2 – 2 / 2; = 2
x = (3*9*(3+(9*3/(3)))); = 3
```

1.18 Escriba un applet de Java que pida al usuario introducir dos números, obtenga los dos números de usuario e imprima la suma, el producto, la diferencia y el cociente de los dos números. Utilice las técnicas de interfaz gráfica con el usuario que se ilustran en la fig. 1.13

```
import java.awt.*;   // Librerías para graficos(Abstrac Window Toolkit)
import java.applet.Applet;

public class TNumeros extends Applet{

      Label      L1, L2;
      TextField  E1, E2;
      Button     B1;

      private int X, Y;

      public void Asignar (int N1, int N2){

            X = N1;
            Y = N2;
      }

//-------------------------------------------------------------------------------------------------


      public int Sumar(){

            return (X + Y);
      }

//-------------------------------------------------------------------------------------------------


      public int Restar(){

            return (X - Y);
      }

//-------------------------------------------------------------------------------------------------


      public int Multiplicar(){

            return (X * Y);
      }

//-------------------------------------------------------------------------------------------------


      public float Dividir(){
```

```
		if (Y == 0)
			return (0);
		else
			return (X/Y);
	}

//-----------------------------------------------------------------------------------------------

//-----------------------------------------------------------------------------------------------


	public void init(){

		X = Y = 0;

		L1 = new Label ("Introduzca el primer número: ");
		L2 = new Label ("Introduzca el segundo número: ");
		E1 = new TextField (5);
		E2 = new TextField (5);
		B1 = new Button ("Calcular");

		add (L1);
		add (E1);
		add (L2);
		add (E2);
		add (B1);
	}

//-----------------------------------------------------------------------------------------------

	public boolean action (Event E, Object O){

		if (E.target == B1){
			Asignar (Integer.parseInt(E1.getText()), Integer.parseInt(E2.getText()) );

			repaint ();
		}
		return true;
	}

//-----------------------------------------------------------------------------------------------

	public void paint (Graphics G){

		G.drawString ("La suma es: " + Sumar() , 70, 75);
		G.drawString ("La resta es: " + Restar(), 70, 95);
		G.drawString ("La multiplicación es: " + Multiplicar(), 70, 115);
		G.drawString ("La división es: " + Dividir(), 70, 135);

	}
}
```

==========

1.19 Escriba un applet de Java que pida al usuario introducir dos enteros, obtenga los números del usuario y exhiba el número más grande seguido por las palabras “es mayor” en la barra de estado de la applet. Si los números son iguales, se debe imprimir el mensaje “Los dos números son iguales”. Utilice las técnicas de interfaz gráfica con el usuario ilustradas en la figura 1.13.

```
import java.awt.*; // Libreria para graficos
import java.applet.Applet;

public class Comparacion extends Applet{

	Label	L1, L2;
	TextField	E1, E2;
```

```
	Button	B1;

	private int A, B;

	public void Asignar(int N1, int N2){

		A = N1;
		B = N2;
	}

//-------------------------------------------------------------------------------------

	public int Comparar (){

		if (A > B)
			return (1);
		if (B > A)
			return (2);
		else
			return (0);
	}

//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------

	public void init (){

		A =  B = 0;

		L1 = new Label ("Introduzca el número 1) ");
		L2 = new Label ("Introduzca el número 2) ");
		E1 = new TextField (5);
		E2 = new TextField (5);
		B1= new Button ("Aceptar");

		add (L1);
		add (E1);
		add (L2);
		add (E2);
		add (B1);
	}

//-------------------------------------------------------------------------------------

	public boolean action (Event E, Object O){

		if (E.target == B1){
			Asignar (Integer.parseInt (E1.getText()), Integer.parseInt(E2.getText()) );
			repaint();
		}

		return  true;
	}

//-------------------------------------------------------------------------------------
	public void paint (Graphics G){

		switch  (Comparar()){
			case  0: G.drawString("Los números son iguales", 200, 200); break;
			case  1: G.drawString("El número A es mayor que B", 200, 200); break;
			case  2: G.drawString("El número B es mayor que A", 200, 200); break;
			default: G.drawString("No existe ningún número!!!", 200, 200);
		}
	}
}
```

==========

1.20 Escriba una applet de Java que reciba tres enteros desde el teclado e imprima la suma, la media, el producto, el mínimo y el máximo de dichos números. Utilice las técnicas de GUI ilustradas en la fig. 1.13

```
import java.awt.*;
import java.applet.Applet;

public class TNumeros extends Applet{

	Label	L1, L2, L3;
	TextField E1, E2, E3;
	Button	B1;

	private int A, B, C;

	public void Asignar (int X, int Y, int Z){

		A = X;
		B = Y;
		C = Z;
	}

//-----------------------------------------------------------------------------------------------
---

	public int Sumar (){

		return (A + B+ C);
	}

//-----------------------------------------------------------------------------------------------
---

	public float Media(){

		return (Sumar() / 3);
	}

//-----------------------------------------------------------------------------------------------
---

	public int Producto(){

		return (A * B * C);
	}

//-----------------------------------------------------------------------------------------------
---

	public int Minimo(){

		if (A < B &&  A <C)
			return (A);
		else
			if (B < C)
				return (B);
			else
				return (C);
	}

//-----------------------------------------------------------------------------------------------
---

	public int Maximo(){

		if (A > B &&  A >C)
			return (A);
		else
```

```
                if (B > C)
                        return (B);
                else
                        return (C);
        }

//-----------------------------------------------------------------------------------------------
---
//-----------------------------------------------------------------------------------------------
---

        public void init(){

                A = B = C = 0;

                L1 = new Label("Introduzca el primer número: ");
                L2 = new Label("Introduzca el segundo número: ");
                L3 = new Label("Introduzca el tercer número: ");
                E1 = new TextField (5);
                E2 = new TextField (5);
                E3 = new TextField (5);
                B1 = new Button ("Aceptar");

                add (L1);
                add (E1);
                add (L2);
                add (E2);
                add (L3);
                add (E3);
                add (B1);
        }

//-----------------------------------------------------------------------------------------------
---
        public boolean action (Event E, Object O){

                if (E.target == B1){
                        Asignar (Integer.parseInt(E1.getText()), Integer.parseInt(E2.getText()),
Integer.parseInt(E3.getText()) );
                        repaint();
                }

                return true;
        }

//-----------------------------------------------------------------------------------------------
---
        public void paint (Graphics G){

                G.drawString ("La suma de los números es: " + String.valueOf(Sumar() ), 200, 200);

                G.drawString ("El producto es:  " + String.valueOf(Producto()), 200, 220 );
                G.drawString ("La media es: " + String.valueOf(Media()), 200, 240);
                showStatus ("El mínimo es: " +  Integer.toString(Minimo()) + " ;  El máximo es:" +
Integer.toString(Maximo()) );
        }

}
```

==========

1.21 Escriba una applet de Java que reciba el radio de un círculo e imprima el diámetro, la circunferencia y el área de ese círculo. Utilice el valor constante 3.14159 para pi. Emplee las técnicas de GUI ilustradas en la fig. 1.6

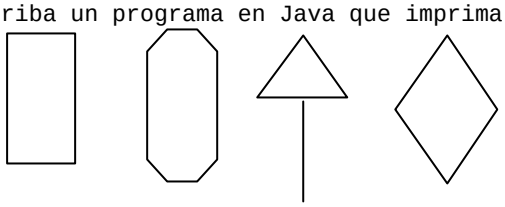
```
import java.awt.*;
import java.applet.Applet;

public class TRadio extends Applet{

        Label     L1;
```

```
	TextField E1;


	private byte Radio;


	public void Asignar (byte R){

		Radio = R;
	}

//-----------------------------------------------------------------------------------------------

	static int Diametro (byte R){

		return (2 * R);
	}

//-----------------------------------------------------------------------------------------------

	static double Area (byte R){

		return (3.14159 * (R*R));
	}

//-----------------------------------------------------------------------------------------------

	static void Dibujar (byte R, Graphics G){

		G.fillOval (200, 200, R*R, R*R);
	}

//-----------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------

	public void init (){

		L1 = new Label("Introduzca el radio del circulo: ");
		E1 = new TextField (5);
		add (L1); add (E1);
	}

//-----------------------------------------------------------------------------------------------

	public boolean action (Event E, Object O){

		showStatus ("Made By 1) ACP   2) LRL");

		Asignar (Byte.parseByte (E1.getText()));
		E1.setText ("");
		repaint ();

		return (true);
	}

//-----------------------------------------------------------------------------------------------

	public void paint (Graphics G){

		G.drawString ("El diametro es: " + Integer.toString(Diametro(Radio)), 100, 100);
		G.drawString ("El area es: " + Double.toString(Area(Radio)), 100, 120);
		Dibujar (Radio, G);
	}
}
************************************************
public class TGrito extends ArithmeticException{


	public TGrito(){
```

```
            super ("Valores no validos!!!");
      }
}
```

==========

1.22 Escriba un programa en Java que imprima un rectángulo, un óvalo, una flecha y un rombo como sigue:



```
import java.awt.*;
import java.applet.Applet;

public class TGraficos extends Applet{

      Label       L1;
      TextField E1;
      Button     B1;

      private String C;
      private Color Colorcito;

      public void Asignar(String Aux){
            C = Aux;
      }

//-------------------------------------------------------------------------------------------------
---

      public void Rectangulo (Graphics G){

            for (byte I = 1; I <= 70; I+=4){
                  G.drawString (C, I+50, 150);
                  G.drawString (C, I+50, 250);
            }

            for (byte I = 1; I <= 100; I+=4){
                  G.drawString (C, 50, I + 150);
                  G.drawString (C, 120, I + 150);
            }
      }

//-------------------------------------------------------------------------------------------------
---

      public void Elipse (Graphics G){

            for (byte I = 1; I <= 70; I+=4){
                  G.drawString (C, 350, I + 160);
                  G.drawString (C, 420, I + 160);
            }

            for (byte I = 1; I <= 40; I+=8){
                  G.drawString (C, I + 370, 140);
                  G.drawString (C, I + 370, 250);
            }

            G.drawString (C, 360, 150);
            G.drawString (C, 410 ,150);
            G.drawString (C, 360, 240);
            G.drawString (C, 410, 240);
      }

//-------------------------------------------------------------------------------------------------
---

      public void Flecha (Graphics G){
```

```
            G.drawString(C, 180, 152);

            for (byte J = 1; J <= 5; J++)                        // Columnas
                  for (byte I = 0; I<=J; I++){          //  Filas
                        G.drawString (C, 180 - (I * 8), 150 + (J * 8));
                        G.drawString (C, 180 + (I * 8), 150 + (J * 8));
                  }

            for (int I = 1; I <= 8; I++)
                  G.drawString (C, 180, 200 + (I * 6) );
      }

//-------------------------------------------------------------------------------------------------
---

      public void Rombo (Graphics G){

            for (byte I = 0; I <= 7; I++){
                  G.drawString (C, 270 + (I * 5), 150 + (I * 8));  // Arriba
                  G.drawString (C, 270 - (I * 5), 150 + (I * 8));

                  if (I < 7){
                        G.drawString (C, 270 + (I * 5), 260 - (I * 8));  // Abajo
                        G.drawString (C, 270 - (I * 5), 260 - (I * 8));
                  }
            }
      }


//-------------------------------------------------------------------------------------------------
---
//-------------------------------------------------------------------------------------------------
---

      public void init (){

            L1 = new Label ("Introduzca un carácter: ");
            B1 = new Button ("Aceptar");
            E1 = new TextField (5);

            add (L1);
            add (E1);
            add (B1);
      }

//-------------------------------------------------------------------------------------------------
---

      public boolean action (Event E, Object O){

            if (E.target == B1){
                  String C = E1.getText();
                  Asignar (C);
                  repaint();
            }

            return (true);
      }

//-------------------------------------------------------------------------------------------------
---

      public void paint (Graphics G){

            G.setColor  (Color.blue);

            Rectangulo(G);
            Flecha (G);
```

```
            Rombo (G);
            Elipse (G);
            showStatus ("Made By EAA");
      }
}
```

==========

1.23 He aquí una mirada hacia adelante. La clase Graphics de Java provee muchas capacidades de dibujo que veremos con detalle en el cap. 9 “Gráficos”. Escriba una applet de la java que utilice el siguiente enunciado para dibujar un rectángulo desde el método paint de la applet.

```
            g.drawRect ( 10, 10, 50, 50 );
```

El método drawRect (dibujar rectángulo) de la clase Graphics requiere cuatro argumentos. Los primeros dos son las coordenadas de la esquina superior izquierda del rectángulo. El tercer argumento es la anchura de rectángulo en pixeles. El cuarto argumento es la altura del rectángulo en pixeles.

```
import java.awt.*;
import java.applet.Applet;

public class TRectangulo extends Applet{

      Label       L1, L2, L3, L4;
      TextField E1, E2, E3, E4;
      Button     B1;

      private  int X, Y, Grosor, Altura;
//-------------------------------------------------------------------------------------------------
---

      public void Asignar (int A, int B, int C, int D){

            X = A;
            Y = B;
            Grosor = C;
            Altura = D;
      }

//-------------------------------------------------------------------------------------------------
---//-------------------------------------------------------------------------------------------------
------

      public void init (){

            X = Y = Grosor = Altura = 0;

            L1 = new Label ("Introduzca X1: ");
            L2 = new Label ("Introduzca Y1: ");
            L3 = new Label ("Introduzca X2: ");
            L4 = new Label ("Introduzca Y2:  ");

            E1 = new TextField (5);
            E2 = new TextField (5);
            E3 = new TextField (5);
            E4 = new TextField (5);

            B1 = new Button ("Aceptar");

            add (L1);  add (E1);
            add (L2);  add (E2);
            add (L3);  add (E3);
            add (L4);  add (E4);
            add (B1);
      }

//-------------------------------------------------------------------------------------------------
---

      public boolean action (Event E, Object O){

            if (E.target == B1){
```

```
                    Asignar (Integer.parseInt(E1.getText()), Integer.parseInt(E2.getText()),
Integer.parseInt(E3.getText()), Integer.parseInt(E4.getText()) );
                    repaint ();
             }

             return (true);
      }

//-------------------------------------------------------------------------------------------------
---

      public void paint (Graphics G){

             G.setColor (Color.blue);
             G.drawRect (X, Y, Grosor, Altura);
      }
}
=============================================================
```

1.24 Modifique la solución del ejercicio 1.23 para exhibir diversos rectángulos de diferente tamaño.

```
import java.awt.*;
import java.applet.Applet;

public class TRectangulos extends Applet{

      Label      L1, L2, L3, L4;
      TextField E1, E2, E3, E4;
      Button    B1;

      private int X1, Y1, X2, Y2;

//-------------------------------------------------------------------------------------------------
-

      public void Asignar (int A, int B, int C, int D){

             X1 = A;
             Y1 = B;
             X2 = C;
             Y2 = D;
      }

//-------------------------------------------------------------------------------------------------
---

      public void Dibujar (Graphics G){

                    int N = 1 + (int) (Math.random () * 6);

                    switch (N){
                           case 1: G.setColor(Color.red); break;
                           case 2: G.setColor(Color.pink); break;
                           case 3: G.setColor(Color.green); break;
                           case 4: G.setColor(Color.blue); break;
                           case 5: G.setColor(Color.yellow); break;
                           case 6: G.setColor(Color.black); break;
                    }

             for (byte I = 0; I <= 4; I++){
                    int Aux = 1 + (int) (Math.random() * 7);

                    G.drawOval (X1 + (I * 50), Y1, X2 * Aux, Y2 * Aux);
      }

//-------------------------------------------------------------------------------------------------
---
//-------------------------------------------------------------------------------------------------
---
```

```
	public void init (){

		L1 = new Label ("Introduzca el valor X1: ");
		L2 = new Label ("Introduzca el valor Y1: ");
		L3 = new Label ("Introduzca el valor X2: ");
		L4 = new Label ("Introduzca el valor Y2: ");

		E1 = new TextField (5);
		E2 = new TextField (5);
		E3 = new TextField (5);
		E4 = new TextField (5);

		B1 = new Button("Aceptar");

		add (L1); add (E1);
		add (L2); add (E2);
		add (L3); add (E3);
		add (L4); add (E4);
		add (B1);

		X1 = X2 = Y1 = Y2 = 50;
	}

//-------------------------------------------------------------------------------------------------
---

	public boolean action (Event E, Object O){

		if (E.target == B1){
			Asignar (Integer.parseInt(E1.getText()), Integer.parseInt(E2.getText()),
Integer.parseInt(E3.getText()), Integer.parseInt(E4.getText()) );
			repaint();
		}

		return (true);
	}

//-------------------------------------------------------------------------------------------------
---
	public void paint (Graphics G) {

		Dibujar (G);
	}
}
================================================================
====================================
```

1.25 La **clase Graphics** contiene un **método drawOval** (dibujar óvalo que toma exactamente los mismos argumentos que el **método drawRect**, solo que en este caso los argumentos especifican el rectángulo circunscrito o “delimitador” del óvalo. Los lados del rectángulo delimitador son las cotas del óvalo. Escriba una applet de java que dibuje un óvalo y un rectángulo con los mismos cuatro argumentos. Verá que el óvalo toca el rectángulo en le centro de cada lado.

```
import java.awt.*;
import java.applet.Applet;

public class TFiguras extends Applet{

	Label		L1, L2, L3, L4;
	TextField	E1, E2, E3, E4;
	Button		B1;

	private int A, B, C, D;

//-------------------------------------------------------------------------------------------------
---

	public void Asignar (int X, int Y, int Z, int W){
```

```
		A = X;
		B = Y;
		C = Z;
		D = W;
	}

//-------------------------------------------------------------------------------------------------
---
	public void Dibujar (Graphics G){

		int N = 1 +  (int) (Math.random() * 6);

		switch (N){
			case 1: G.setColor (Color.red); break;
			case 2: G.setColor (Color.blue); break;
			case 3: G.setColor (Color.pink); break;
			case 4: G.setColor (Color.orange); break;
			case 5: G.setColor (Color.green); break;
			case 6: G.setColor (Color.black); break;
		}

		G.drawRect (A, B, C, D);
		G.drawOval (A, B, C, C);
	}

//-------------------------------------------------------------------------------------------------
---
//-------------------------------------------------------------------------------------------------
---

	public void init (){
		L1 = new Label ("Introduzca el valor X1: ");
		L2 = new Label ("Introduzca el valor Y1: ");
		L3 = new Label ("Introduzca el valor X2: ");
		L4 = new Label ("Introduzca el valor Y2: ");

		E1 = new TextField (5);
		E2 = new TextField (5);
		E3 = new TextField (5);
		E4 = new TextField (5);

		B1 = new Button("Aceptar");

		add (L1); add (E1);
		add (L2); add (E2);
		add (L3); add (E3);
		add (L4); add (E4);
		add (B1);

		A = B = C = D = 50;
	}

//-------------------------------------------------------------------------------------------------
---

	public void paint (Graphics G){

		Dibujar (G);
	}

//-------------------------------------------------------------------------------------------------
---

	public boolean action (Event E, Object O){

		if (E.target == B1){
			Asignar (Integer.parseInt(E1.getText()), Integer.parseInt(E2.getText()),
Integer.parseInt(E3.getText()), Integer.parseInt(E4.getText()) );
			repaint();
```

```
            }

            return (true);
      }
}
```

==========

1.26 Modifique la solución del ejercicio 1.25 para exhibir diversos óvalos con diferentes formas y tamaños.

```
import java.awt.*;
import java.applet.Applet;

public class TFiguras extends Applet{

      Label        L1, L2, L3, L4;
      TextField  E1, E2, E3, E4;
      Button      B1;

      private int A, B, C, D;

//-------------------------------------------------------------------------------------------------
---
      public void Asignar (int X, int Y, int Z, int W){

            A = X;
            B = Y;
            C = Z;
            D = W;
      }

//-------------------------------------------------------------------------------------------------
---

      public void Dibujar (Graphics G){

            int N = 1 +  (int) (Math.random() * 6);

            switch (N){
                  case 1: G.setColor (Color.red); break;
                  case 2: G.setColor (Color.blue); break;
                  case 3: G.setColor (Color.pink); break;
                  case 4: G.setColor (Color.orange); break;
                  case 5: G.setColor (Color.green); break;
                  case 6: G.setColor (Color.black); break;
            }

            for (byte I = 0; I <= 4; I++){
                  int Aux = 1 + (int) (Math.random() * 7);

                  G.drawOval (A + (I * 50), B, C * Aux, D * Aux);
            }
      }

//-------------------------------------------------------------------------------------------------
---
//-------------------------------------------------------------------------------------------------
---

      public void init (){
            L1 = new Label ("Introduzca el valor X1: ");
            L2 = new Label ("Introduzca el valor Y1: ");
            L3 = new Label ("Introduzca el valor X2: ");
            L4 = new Label ("Introduzca el valor Y2: ");

            E1 = new TextField (5);
            E2 = new TextField (5);
            E3 = new TextField (5);
            E4 = new TextField (5);
```

```
            B1 = new Button("Aceptar");

            add (L1); add (E1);
            add (L2); add (E2);
            add (L3); add (E3);
            add (L4); add (E4);
            add (B1);

            A = B = C = D = 50;
      }

//-------------------------------------------------------------------------------------------------
---

      public void paint (Graphics G){

            Dibujar (G);
      }

//-------------------------------------------------------------------------------------------------
---

      public boolean action (Event E, Object O){

            if (E.target == B1){
                  Asignar (Integer.parseInt(E1.getText()), Integer.parseInt(E2.getText()),
Integer.parseInt(E3.getText()), Integer.parseInt(E4.getText()) );
                  repaint();
            }

            return (true);
      }
}
=================================================================================================
=
1.27 ¿ Qué imprime el siguiente código?
      g.drawString( “*”, 25, 25 );
      g.drawString( “*****”, 25, 55 );
      g.drawString( “******”, 25, 85 );
      g.drawString( “****”, 25, 70 );
      g.drawString( “**”, 25, 40 );

import java.awt.*;
import java.applet.Applet;

public class TAsterisco extends Applet{

      public void paint (Graphics G){

            G.drawString ("*", 25, 25);
            G.drawString ("***", 25, 55);
            G.drawString ("******", 25, 85);
            G.drawString ("*****", 25, 70);
            G.drawString ("**", 25, 40);
      }
}
=================================================================================================
```

1.28 Escriba una applet Java que lea enteros y determine e imprima el entero más grande y el más pequeño del grupo. Utilice sólo las técnicas de programación que aprendió en este capítulo.

```
public class TN_Numeros{

      final int Max = 100;

      private int V[] = new int[Max];
      private byte Dim;
```

```
	TN_Numeros(){
		Dim = 0;
	}

//-------------------------------------------------------------------------------------------------
--
	public void Cargar (int E){

		Dim++;
		V[Dim] = E;
	}

//-------------------------------------------------------------------------------------------------
---
	public int Mayor(){

		if (Dim == 1)
			return (V[1]);
		else{
			int M = V[1],  I = 2;

			while (I <= Dim){
				if (V[I] > M)
					M = V[I];
				I++;
			}

			return (M);
		}
	}

//-------------------------------------------------------------------------------------------------
---
	public int Menor(){

		if (Dim == 1)
			return (V[1]);
		else{
			int M = V[1], I = 2;

			while (I <= Dim){
				if (V[I] < M)
					M = V[I];
				I++;
			}

			return (M);
		}
	}

//-------------------------------------------------------------------------------------------------
---//-------------------------------------------------------------------------------------------------
------
	public static void main (String args[]){

		TN_Numeros MiNumero = new TN_Numeros();

		for (byte I = 0; I < args.length; I ++)
			MiNumero.Cargar (Integer.parseInt(args[I]));

		if (MiNumero.Dim == 0)
			System.out.println ("No existen elementos!!!");
		else
```

```
                    System.out.println ("El numero mayor es: " + MiNumero.Mayor() + ",  el numero
menor es: " + MiNumero.Menor());
      }
}
```

==========

1.29 Escriba una applet java que lea un entero y determine e indique si es impar o part. (Sugerencia: Utilice el operador de residuo. Un número par es múltiplo de 2 Cualquier múltiplo de 2 deja un residuo de cero cuando se divide entre 2)

```
import java.awt.*;
import java.applet.Applet;

public class TPar_Impar extends Applet{

      Label        L1;
      TextField  E1;

      private int A;

      public void Asignar (int N){

            A = N;
      }

//-----------------------------------------------------------------------------------------------
---

      public boolean Par (){

            return (A % 2 == 0);
      }

//-----------------------------------------------------------------------------------------------
---
//-----------------------------------------------------------------------------------------------
---

      public void init (){

            L1 = new Label ("Introduzca un número: ");
            E1 = new TextField (5);

            add (L1); add (E1);

            A = 0;
      }

//-----------------------------------------------------------------------------------------------
---

      public boolean handleEvent (Event Evento){

            switch (Evento.id){
                  case Evento.ACTION_EVENT:{
                              Asignar (Integer.parseInt(E1.getText()));
                              repaint ();
                        }

                        return (true);

                  default:      {
                        showStatus ("No existen ningún valor!!!");
                        return (false);
                  }
            }
      }

//-----------------------------------------------------------------------------------------------
---
```

```
	public void paint (Graphics G){

		if (Par())
			G.drawString ("El valor es par.", 100, 100);
		else
			G.drawString ("El valor es impar.", 100, 100);
	}

}
```

==========

1.30 Escriba unma applet Java que lea dos enteros y determine e indique si el primero es o no múltiplo del segundo (Sugerencia: Utilice el operador de residuo)

```
import java.awt.*;
import java.applet.Applet;

public class TMultiplo extends Applet{

	Label	L1, L2;
	TextField  E1, E2;
	Button	B1;

	private int A, B;

	public void Asignar (int N, int M){

		A = N;
		B = M;
	}
//-----------------------------------------------------------------------------------------------
---

	public boolean Multiplo (){

		return (A % B == 0);
	}

//-----------------------------------------------------------------------------------------------
---
//-----------------------------------------------------------------------------------------------
---

	public void init (){

		L1 = new Label ("Introduzca el primer número: ");
		L2 = new Label ("Introduzca el segundo número: ");
		E1 = new TextField (5);
		E2 = new TextField (5);
		B1 = new Button ("Aceptar");

		add (L1); add (E1);
		add (L2); add (E2);
		add (B1);

		A = B = 0;
	}

//-----------------------------------------------------------------------------------------------
---

	public boolean handleEvent (Event Evento){

		switch (Evento.id){
			case Evento.ACTION_EVENT:{
```

```
                                Asignar (Integer.parseInt(E1.getText()),
Integer.parseInt(E2.getText()));
                                repaint ();
                        }

                        return (true);

                default:      {
                        showStatus ("No existen ningún valor!!!");
                        return (false);
                }
        }
    }

//-------------------------------------------------------------------------------------------------
---

    public void paint (Graphics G){

        if (Multiplo())
                G.drawString ("El número " + Integer.toString(A) + " es multiplo de " +
Integer.toString(B), 100, 100);
        else
                G.drawString ("El número " + Integer.toString(A) + " no es multiplo de " +
Integer.toString(B), 100, 100);
    }

}
=================================================================================================
==
```

1.31 Escriba una applet Java que exhiba un patrón de tablero de ajedrez como sigue:

```
import java.awt.*;
import java.applet.Applet;

public class TTablero extends Applet{

    Label        L1;
    TextField  E1;
    Button      B1;

    private String S;

//-------------------------------------------------------------------------------------------------
---

    public void Asignar (String Cad){

        S = Cad;
    }

//-------------------------------------------------------------------------------------------------
---

    public void Dibujar (Graphics G){

        for (byte I = 0; I <= 7; I++)
                for (byte J = 0; J <= 7; J++)
                        G.drawString (S, 100 + (I*10), 100 + (J*10));
    }

//-------------------------------------------------------------------------------------------------
---
//-------------------------------------------------------------------------------------------------
---

    public boolean action (Event E, Object O){

        if (E.target == B1){
```

```
                Asignar (E1.getText());
                repaint();
            }

            return (true);
        }

//-------------------------------------------------------------------------------------------------
---

        public void init (){

            L1 = new Label ("El tablero de ajedrez es: ");
            E1 = new TextField (5);
            B1 = new Button ("Aceptar");

            add (L1); add (E1);
            add (B1);
        }

//-------------------------------------------------------------------------------------------------
---

        public void paint (Graphics G){

            G.setColor (Color.blue);
            Dibujar (G);
        }
}
```

==========

1.32 Explique la diferencia entre los términos “error fatal” y “error no fatal” ¿Por qué podría ser preferible experimentar un error fatal que uno no fatal?

Error fatal hacen que los programas terminen de inmediato sin llegar a feliz término. Los errores no fatales permiten a los programas terminar, pero a menudo producen resultados incorrectos.

==========

1.33 He aquí otra mirada al futuro. En este capítulo estudiamos los enteros y el tipo **int.** Java también puede representar letras mayúsculas, letras minúsculas y una amplia variedad de símbolos especiales. Cada carácter tiene una representación entera correspondiente. El juego de caracteres que usa una computadora y la representación entera correspondiente a dichos caracteres constituyen el conjunto de caracteres de esa computadora. Podemos indicar un carácter encerrándolo en apóstrofos, como en ‘A’.

Podemos determinar el equivalente entero de un carácter anteponiendo (**int)** a ese carácter; esto se denomina mutación (hablaremos más de las mutaciones en el cap. 2);

```
(int) ‘A’
```

El siguiente enunciado imprime un carácter y su equivalente entero desde el método páint de una applet;

```
g.drawString( “El carácter” + ‘A’ + “tiene el valor “ + (int) ‘A’, 25, 25 );
```

Cuando se ejecuta el enunciado anterior, exhibe el carácter A y el valor 65 (en los sistemas que utilizan el conjunto de caracteres llamado Unicode) como parte de la cadena. Escriba una applet en Java que imprima los equivalentes enteros de algunas letras mayúsculas, letras minúsculas, dígitos y símbolos especiales. Como mínimo, determine los equivalentes enteros de los siguientes: A B C a b c 0 1 2 $ * + / y el carácter de espacio.

```
import java.awt.*;
import java.applet.Applet;

public class TEquivalente extends Applet{

    Label       L1;
    TextField  E1;
    Button      B1;

    private char A;

    public void Asignar (char X){
```

```
            A = X;
      }

//-------------------------------------------------------------------------------------------

      public int Equivalente(){

            return ((int) A);
      }

//-------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------

      public void init (){

            L1 = new Label ("Introduzca el caráter: ");
            E1 = new TextField (5);
            B1 = new Button ("Aceptar");

            add (L1); add (E1);
            add (B1);

            A = 'a';
      }

//-------------------------------------------------------------------------------------------

      public boolean handleEvent (Event Evento){

            switch (Evento.id){
                  case Evento.ACTION_EVENT:{
                              String S = E1.getText();
                              Asignar (S.charAt(0));
                              repaint ();
                        }

                        return (true);

                  default:      {
                        showStatus ("No existen ningún valor!!!");
                        return (false);
                  }
            }
      }

//-------------------------------------------------------------------------------------------
      public void paint (Graphics G){

            G.setColor (Color.blue);
            G.drawString("El equivalente de " + A + " es:   " +  Integer.toString(Equivalente
() ), 100, 100 );
      }

}
```

==========

1.34 Escriba una applet en Java que reciba un número de cinco dígitos, separe el número en sus dígitos individuales e imprima los dígitos separados con tres espacios entre cada uno. Por ejemplo, si el usuario teclea 42339 el programa deberá imprimir: `4   2   3   3   9`

```
import java.awt.*;
import java.applet.Applet;

public class TDigitos extends Applet{

      Label        L1, L2, L3, L4, L5;
      TextField  E1, E2, E3, E4, E5;
      Button      B1;
```

```
//-------------------------------------------------------------------------------------------
	public void init (){

		L1 = new Label ("Introduzca el primer número: ");
		L2 = new Label ("Introduzca el segundo número: ");
		L3 = new Label ("Introduzca el tercer número: ");
		L4 = new Label ("Introduzca el cuarto número: ");
		L5 = new Label ("Introduzca el quinto número: ");

		E1 = new TextField (5);
		E2 = new TextField (5);
		E3 = new TextField (5);
		E4 = new TextField (5);
		E5 = new TextField (5);
		B1 = new Button ("Aceptar");

		add (L1); add (E1);
		add (L2); add (E2);
		add (L3); add (E3);
		add (L4); add (E4);
		add (L5); add (E5);
		add (B1);
	}

//-------------------------------------------------------------------------------------------

	public boolean action (Event E, Object O){

		if (E.target == B1)
			repaint ();

		return (true);
	}

//-------------------------------------------------------------------------------------------

	public void paint (Graphics G){

		G.drawString (E1.getText() + "     ," + E2.getText() + "     ," + E3.getText() +
"     ,"  + E4.getText() + "     ," + E5.getText(),100,100);

	}
}
```

==========

CAP. 2

OBJETIVOS

- Estudiar las técnicas básicas de resolución de problemas.
- Poder crear algoritmos siguiendo el proceso de reficiación descendente por pasos.
- Poder utilizar las estructuras de selección if e if/else para escoger interacciones alternativos.
- Poder utilizar las estructuras de repetición while para ejecutar repetidamente enunciados de un programa.
- Entender la repetición controlada por el contador y la repetición controlada por la centilenal.
- Poder usar operadores de incremento y decremento.

2.1 Introducción

- Al escribir programas, es necesario entender los tipos de **BLOQUES DE CONTRUCCIÓN** que están disponibles y utilizar principios de construcción de programas probados y comprobados.
- Teoría y principios de la programación estructurada son útiles en la contrucción y manipulación de objetos.
- Conceptos en el contexto de aplicaciones Java. **Una aplicación** es un programa autónomo que **NO** requiere el Appletviewer ni el navegador de la World Wide Web para ejecutarse.

2.2. Algoritmos

Procedimiento para resolver un problema:
1. las acciones por ejecutar y
2. el orden en que dichas acciones debe ejecutarse es un algoritmo.

Es decir, la especificación del orden en que se ejecutan los enunciados (acciones) de un programa para computadora se llama **CONTROL DE PROGRAMA.**

2.3. Pseudocódigo

El pseudocódigo es un lenguaje informal que ayuda a desarrollar algoritmos, o ayudan al programador a razonar.

2.4. Estructuras de control

- Los enunciados de un programa se ejecutan uno tras otro (ejecución secuencial)
- **Bohn y Jacopini** demostraron que los programas podían escribirse en **sólo tres estructuras de control**: **SECUENCIAL,** de **SELECCIÓN Y DE REPETICIÓN**.

2.5. La estructura de selección if

La ESTRUCTURA DE SELECCIÓN <u>única</u> if(si) <u>realiza</u> (selecciona) **<u>una acción</u>** si se cumple una condición o bien pasa por alto una acción si la condición no se cumple.

- Ej. calificación para aprobar un examen es 60 (de 100)

  Pseudocódigo:

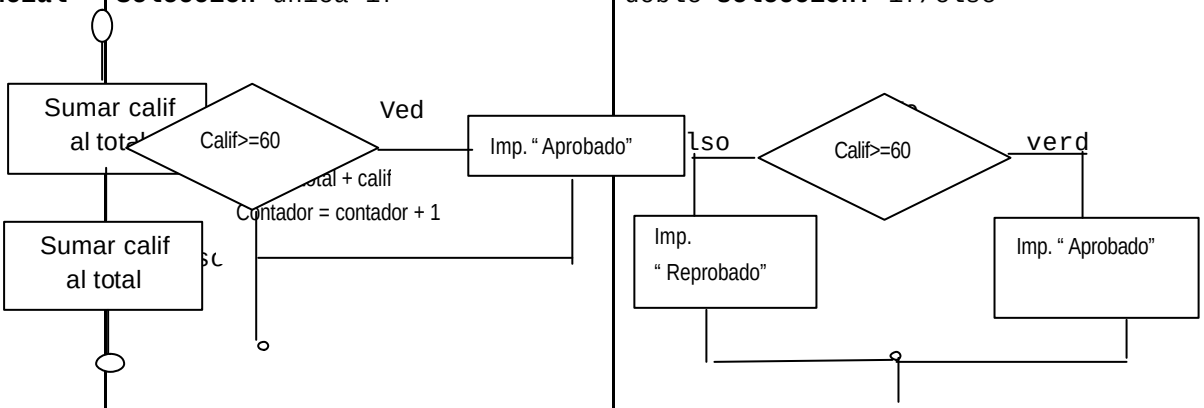
  If calificación del estudiante mayor que o igual a 60 | if (calif >= 60)
  Imprimir "aprobado"
  System.out.println("Aprobado");

  Si la condición se cumple, imprime, de lo contrario salta a la siguiente instrucción.

2.6. La estructura de selección if/else

La estructura de selección <u>doble</u> if/else (si/si no) <u>realiza (selecciona entre dos acciones diferentes)</u> una acción si se cumple una condición y realiza una acción diferente si la condición no se cumple.

La estructura de selección <u>(múltiple)</u> switch(conmutar) realiza una de las muchas acciones distintas dependiendo del valor de una expresión. <u>(selecciona entre muchas acciones distintas)</u>

secuencial	**Selección** única if	doble **selección:** if/else
		

	if (calif >= 60)
	System.out.println("Aprobado");
		If (calif >= 60)
		System.out.println("aprobado")
		Else

		System.out.printl(“Reprobado”) System.out.println(calif>=60 ?”apr”:”Rep”)

El operador condicional (?☺ se relacina con if/else (operador ternario en Java)

- Las estructuras if/else anidadas prueban múltiples casos.

- Palabras claves de Java: if, else, switch, while, do y for, etc.

abstract	extends	int	return	true
byte	false	interface	short	try
case	final	long	static	void boolean
catch	finally	native	super	volatile break
char	float	new	switch	while
class	for	null	synchronized	
continue	if	package	this	
default	implements	private	throw	
do	import	protected	throws	
double	instanceof	public	transient	
else				

- Un error de lógica **FATAL**(cuando se omiten ambas llaves de un enunciado compuesto) hace que el programa falle y termine prematuramente.
- Un error de lógica **NO FATAL** permite al programa continuar su ejecución pero produciendo resultados incorrectos.
- Un enunciado compuesto se denomina bloque.
- Forma de conectar las estructura de control: Método de anidamiento de estructuras de control.

2.7. La estructura de repetición while.

- Java ofrece tres tipos de **ESTRUCTURAS DE REPETICIÓN**: While, do/while y for (do/while y for).
- En total son **siete(7) estructuras de control**: secuencia, tres tipos de selección y tres tipos de repetición.
- La estructura de repetición permite especificar que una acción se repita en tanto se cumpla una condición.

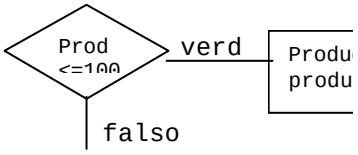
Ej. pseudocódigo:

```
While haya más artículos en mi lista de compras
      Comprar el siguiente artículo y tacharlo de la lista
```

Esta acción se ejecutará repetidamente mientras la condición siga siendo verdadera.

Ej. Encontrar la primera potencia de 2 mayor que 1000. Variable entera product se inicializa en 2, luego al ejecutarse while, product contiento el siguiente resultado:

```
Int product = 2;

While ( product <= 1000)
      Product = 2* product;
```



valores: 4, 8, 16, 32, 64, 128, 256, 512 y 1024 luego se sale.

2.8. Formulación de algoritmos: Estudio de caso 1 (repetición controlada por contador)

Un contador es una variable que sirve para contar, y se inician en 0 o 1.
Java asegura que las variables counter(contador) y total se inicializan antes de utilizarse en un cálculo.

Aplicaciones en Java:

Una **aplicación comienza su ejecución en el método main**. Las aplicaciones se ejecutan empleando el **intérprete de Java** en lugar del Applet-viewer o un navegador de la World Wide Web.

Ej. Promedio de un grupo con repitición controlada por un contador

```
Import java.io.*; //importa el paquete de E/S de Java, permite leer datos del
     teclado y exhibir datos por pantalla

Public class Average{ // el método main (principal) se define dentro de la clase
     Average(promedio) en esta aplic.
 public static void main ( String arg[] ) throws IOException
                                          //  throws  IOException  constituye  la
                                        cláusula throws del método main
 {
     int counter, grade, total, average;
     // inicialización
     total = 0;
     counter = 1;

     //procesando
     while (counter <= 10){
       System.out.print("Teclee calificación de letra: ");
       System.out.flush();  //mostrar, resplandor
       Grade = System.in.read();

       If ( grade == 'A' )
             total = total + 4;

       Else if (grade == 'B' )
             total = total + 3;
       Else if (grade == 'C' )
             total = total + 2;
       Else if (grade == 'D' )
             total = total + 1;
       Else if (grade == 'F' )
             total = total + 0;
       System.in.skip( 1); // saltar el caracter de nueva línea
         Counter = counter +1;

     }
     //terminación
     average = total / 10;
     System.out.println("El promedio del grupo es " + average );
 }
}
```

2.9. Formulación de algoritmos con refinación descendente por pasos: Estudio de caso 2 (repetición controlada por centinela)
2.10 Formulación de algoritmos con refinación descendente pro pasos: Estudio de caso 3 (estructuras de control anidadas)
2.11. Operadores de asignación
2.12. Operadores de incremento y decremento
2.13. Tipos de datos primitivos
2.14. Secuencias comunes de escape

Ejercicios de autoevaluacion

2.1. Conteste las siguientes preguntas

2 Todos los programas pueden escribirse en términos de tres tipos de estructuras de control: ***secuencia, selección y repetición.***

b) La estructura de selección ***if / else*** sirve para ejecutar una acción cuando se cumple una condición y otra cuando no se cumple dicha condición.

c) La repetición de una serie de instrucciones un número especifico de veces se denomina repetición *controlada por contador definida.*
d) Cuando se sabe anticipadamente cuantas veces se va a repetir un conjunto de enunciados, se puede usar un valor *centinela, señal, bandera o ficticio* para determinar la repetición.

2.2. Escriba cuatro enuncia dos diferentes en Java que sumen 1 a la variable entera x.

```
X = X  + 1;
X += 1 ;
++x;
x++;
```

2.3. Escriba cuatro enunciados diferentes en Java que realicen las siguientes acciones:

3. Asignar la suma de x y y a z e incrementar el valor de x en 1 despues del calculo.
   `Z = x++  +y;`
4. Provar si el valor de la variable cuenta es mayor que 10. Si es asi, imprimir, “Cuenta es mayor que 10”.
   ```
   If ( cuenta >10 )
      System.out.println(“Cuenta es  mayor que 10” );
   ```
5. Decrementar la variable x en 1 y luego restarla a la variable total.
   `Total- = --x;`
6. Calcular el residuo después de dividir q entre divisor y asignar el resultado a q. Escriba este enunciado en dos formas distintas.
   `Q %=divisor;`

2.3. Escriba un enunciados en Java que realice cada una de las siguientes tareas:

3. Declarar las variables suma y x de tipo int.
   `Int sum, x;`
4. Asignar 1 a la variable x.
   `X = 1;`
5. Asignar 0 a la variable suma.
   `Sum =0;`
6. Sumar la variable x a la variable suma y asignar el resultado a la varible suma.
   `Sum += x; o suma = suma + x;`
7. Imprimir”La suma es: ”seguido del valor de la variable suma.
   `System.out.println(”La suma es: ”+ suma );`

2.4. Combine los resultados que escribio en el ejercicio 2.4 en una aplicación que calcule e imprima la suma de dos enteros del 1 al 10. Utilice la estructura while para ejecutar cíclicamente los enunciados de calculo e incremento. El ciclo debe terminar cuando el valor de x sea 11.

```
public class calcular {
   public static void main(String args [ ] ){
          int suma, x ;
          x = 1;
           suma =0;
           while ( x<=10 ){
   suma += x ;
   ++x;
           }
           System.out.println( “La suma es: ”+ suma );
   }
}
```

2.5. Determine los valores de cada variable después de realizarse el calculo. Suponga que cuando se inicia la ejecución todas las variables tienen el valor entero 5.

3. `producto *= x++;` R. producto = 25, x = 6;
4. `cociente /= ++x;` R. cociente = 0, x = 6;

2.7 Identifique y corrija los errores en los siguientes enunciados:

2 `while ( c < = 5 ){`

```
            producto *= c;
            ++c;
b) if ( sexo == 1 )
     System.out.println( “Mujer “);
     else;
      System.out.println( “Hombre “);
```

Corrección.-

a)Error: falta la llave derecha que cierra el cuerpo del while.
Corrección: Agregue la llave derecha después del enunciado ++c;

b) Error: El punto y coma después de else causa un error de lógica. El segundo enunciado de salida siempre se ejecutará.
Corrección: Quite el punto y coma después de else.

2.8 ¿Qué esta mal en la siguiente estructura de repetición while?

```
    While ( z>=0 )
      Sum += z;
```

El valor de la variable z nunca se modifica dentro de la estructura while; por lo tanto, si la condición para continuar el ciclo (z>=0) se cumple, se creará un ciclo infinito. Para evitar el ciclo infinito es preciso decrementar z para que en algún momento sea menor que 0.

2.9 Identifique y corrija los errores en los siguientes fragmentos de código.(Nota: Puede haber más de un error en cada fragmento):

```
3  if ( age >= 65 );
      System.out.println (“Edad mayor o igual que 65”) ;
      else
       System.out.println (“ Edad menor que 65) ” ;
```

Error: En la intruccion System.out.printl (“Edad mayor o igual que 65)”;la comilla debe ir dentro del paréntesis.

Corrección: System.out.println (“Edad mayor o igual que 65”);

```
4  int x =1, total;
   while (x >=10){
      total +=x;
      ++x;
   }
```

No hay error.

```
c)While ( x <=100 )
   total +=x;
      ++x;
```

Error: Le faltan las llaves de inicio y terminación del ciclo de while. Otro error while esta escrito con mayúscula y Java produce un error.

```
d) while (y >0){
      System.out.println ( y ) ;
      ++y;
   }
```

Error: La variable “y” no esta convertida en string para que se muestre su valor.

4.9¿Qué imprime el siguiente programa?

```
public class Mystery{
public static void main( String args [] ){
            int y, x = 1, total = 0;
            while( x <=10 ){
            y = x *x ;
               System.out.println ( y ) ;
            Total += y;
            ++x;
            }
      System.out.println ( “total es “+total ) ;
      }
}
```

Imprime de salida:

```
1
4
9
16
25
36
49
64
81
100
el total 385
```

2.19 ¿Qué imprime el siguiente programa?

```
 public class Mystery2{
public static void main( String args [] ){
            int count = 1;
            while(count <=10 ){
                        System.out.println  (count    %  2  ==  1  ?  “****”  :
                        “++++++++”) ;
            ++ count;
            }
      }
}
```

Imprime de salida:

```
****
 ++++++++
 ****
++++++++
****
++++++++
****
++++++++
****
 ++++++++
```

2.20 ¿Qué imprime el siguiente programa?

```
public class Mystery3{
public static void main( String args [] ){
            int row = 10, column;
            while(row >=1 ){
                  column = 1;
                  while ( column<=10 ){
                         System.out.println (row  % 2 == 1 ? “<” : “>”) ;
                    ++ count;
            }
            --row;
            System.out.println();
            }
      }
}
```

Imprime de salida:

2.10 Qué imprime el siguiente programa?

```
public class Mystery{

	public static void main (String Args[]){

		int Total = 0, Y;

		for (int Cont = 1; Cont <= 10; ++Cont){
			Y = Cont * Cont;
			System.out.println (Y);
			Total += Y;
		}

		System.out.println ("El total es: " + Total);
	}
}
```

Para los ejercicios 2.11 al 2.14 realice los siguientes pasos :

a) Lea el enunciado del problema
b) Formule el algoritmo empleando pseudocódigo y refinación descendente por pasos.
c) Escriba una programa en java.
d) Pruebe, depure y ejecute el programa en Java..

2.11 En vista del alto precio de la gasolina, a los conductores les preocupa el kilometraje que pueden lograr sus automóviles. Un conductor ha llevado un registro durante un periodo en el que ha llenado el tanque de combustible de su automóvil varias veces, anotando los kilómetros recorridos y los litros consumidos entre dos llenados del tanque sucesivos. Desarrolle una applet de Java que introduzca los kilómetros recorridos y los litros consumidos (ambos como enteros) para cada llenado del tanque. El programa deberá calcular y exhibir los kilómetros por litro obtenidos con cada tanque e imprimir los kilómetros por litro combinados para todos los tanques hasta el presente. Todos los cálculos de promedios deben producir resultados de punto flotante. Utilice dos campos de texto para introducir los datos. Vea Fig. 1.13 para obtener ideas.

```
import java.awt.*;
import java.applet.Applet;
//import java.TRegistro;

public class TKilometraje extends Applet{

	static final byte Max = 50;


	Label		L1, L2;
	TextField	E1, E2;
	Button		B1;

	static private int V[][] = new int [Max][2];
	static private int Dim;
-------------------------------------------------------------------------
	static void Introducir_Datos (int L, int K){

		Dim++;
		V[Dim][1] = L;
		V[Dim][2] = K;
	}
-------------------------------------------------------------------------

	public void Calcular (Graphics G){


		G.drawString (Integer.toString (V[1][ 1]), 100, 100);
		G.drawString (Integer.toString (V[1][ 2]), 100, 110);

		//G.drawString ();
/**		if (Dim > 1){
			for (byte I = 1; I<=Dim; I++)
				G.drawString (Integer.toString(V[I][1] * V[I][2]), 100, 100);
		}		*/
	}
```

```
//-------------------------------------------------------------------------------------

	public void init (){

		Dim = 0;

		L1 = new Label ("Introduzca los kilometros recorridos: ");
		L2 = new Label ("Introduzca los litros de gasolina: ");
		E1 = new TextField (5);
		E2 = new TextField (5);
		B1 = new Button ("Ejecutar");

		add (L1); add (E1);
		add (L2); add (E2);
		add (B1);
	}

//-------------------------------------------------------------------------

	public boolean action (Event E, Object O){

		showStatus ("Made By 1) ACP   2) LRL");

		if (E.target == B1){
		  Introducir_Datos(Integer.parseInt(E1.getText()),  Integer.parseInt(E2.getText()));

			repaint ();
		}

		return (true);
	}

//-------------------------------------------------------------------------------------

	public void paint (Graphics G){

		Calcular (G);
	}
}
-------------------------
class TRegistro {

	int X, Y;

	TRegistro (){
		X = Y = 0;
	}
}
---------------------------
```

==========

2.15 El proceso de encontrar el valor más grande (esto es, el máximo de un grupo de valores) se utiliza con frecuencia en las aplicaciones de cómputo. Por ej., un programa que determina el ganador de un concurso de ventas introduciría el número de unidades vendidas por cada vendedor. El vendedor que vende más unidades gana el concurso. Escriba un programa en pseudocógico y luego un programa en Java que introduzca una serie de 10 números de un solo dígito y determine e imprima el más grande de ellos. Sugerencia : su programa deberá utilizar tres variables como sigue.

```
public class TBig{

	public static void main (String Args[]){

		int May = 0;

		for (byte I = 0; I <= 9; I++){
			if (May < Integer.parseInt(Args[I]))
				May = Integer.parseInt(Args[I]);
		}

		System.out.println ("El valor mayor es: " + May);
	}
}
```

==========

2.16 Escriba una aplicación en Java que utilice ciclos para imprimir la siguiente tabla de valores.

N	10*N	100*N	1000*N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000

```
public class TTabla{

	static void Dibujar (){

		System.out.println ("N" + "        " + "10*N" + "       " + "100*N" + "       " +
"1000*N");
		System.out.println ();

		for (byte I=1; I<= 5; I++)
			System.out.println (I + "        " + I*10 + "          " + I*100 + "
" + I*1000);
	}

//---------------------------------------------------------------------------

	public static void main (String Args[]){

		System.out.println ();
		Dibujar ();
	}
}
```

```
====================================
2.17 Utilizando un enfoque similar al ejercicio 2.15, encuentre los dos valores más grandes de
los 10 dígitos introducidos. Nota: Sólo se puede introducir cada número una vez.
import java.io.*;
public class TBig2 {

//-------------------------------------------------------------------------------------------
       public static void main (String Args[])throws IOException{

              System.out.println ();

              int Mayor = 0, Num, I = 1;

              while (I <= 2){
                     System.out.flush ();
                     Num = System.in.read ();
                     Num = Num - 48;

                     System.in.skip (1);
                     System.out.println ("El numero es: " + Num);
              }


/**           while (I <= 3){
                     System.out.print ("Introduzca el " + I  + " digito");
                     System.out.flush();
                     Num = System.in.read ();
                     Num = Num - 48;

                     if (Mayor < Num)
                            Mayor = Num;

                     I++;
                     System.in.skip (1);

                     System.out.println ();
              }*/
       }
}
====================================
2.18 Modifique el programa de la fig. 2.11 de modo que verifique la validez de sus entradas. En
     cualquier entrad si el valor introducido es distinto de 1 ó 2, repita el ciclo hasta que el
     usuario introduzca un valor válido.
import java.io.*;
public class TAnalisis{

       public static void main (String Arg[]) throws IOException{

              int  I = 1, R, Aprobados = 0, Reprobados = 0;

              while (I <= 10 ){
                     System.out.println ("Introduzca un resultado (1=A, 2=R): ");
                     System.out.flush();
                     R = System.in.read ();

                     if (R == '1')
                            Aprobados++;
                     else
                            Reprobados++;

                     I++;
              }

              System.out.println ("Los aprobados son: " + Aprobados);
              System.out.println ("Los reprobados son: " + Reprobados);
       }
}
====================================
```

2.19 ¿Qué imprime el siguiente programa?

```
public class Mystery{
  public static void main (String Args[]){
    for (int Cont = 1; Cont <= 10; ++Cont)
       System.out.println (Cont % 2==1? "****":"++++++++");
       }
}
```

```
//Public class Mystery2{
//public static void main (String arg[]){
   //int count =1;
   //while count <= 10}{
   //   System.out.prinln(count%2==1? “*****”
     //                  : “++++++++”);
   //++count;
    }
   }
```

==========

} Obs. del libro

2.20 ¿Qué imprime el siguiente programa?

```
public class Mystery3{
  public static void main (String Args[]){
       for (byte I = 10; I > 0; I--){
         for (byte J = 1; J <= 10; J++)
           System.out.println (I % 2 == 1 ? "<" : ">");
       System.out.println();
             }
       }
             system.out.println();
}
```

```
int rouw=10,colum;
while (row>=1){
  colum=1
   While colum <=10){
System.out.print(row%2==1?”<”:”>”);
++colum;
}
   --row:
}
```

==========

2.21 (problema del else colgante) Determine la salida de los sigtes fragmentos de código cuando x es 9 y Y es 11 y cuando x es 11 y Y es 9. Recuerde que el compilador no tiene en cuenta las sangrías de los programas en Java. Además, el compilador de java siempre asocia un else con el if previo a menos que se indique otra cosa insertando llaves {}. Dado que, a primera vista, el programador tal vez no esté seguro de cuál if corresponde un else, éste se conoce como el problema del “else colgante”. Hemos eliminado las sangrías de los siguientes fragmentos de código para hacer el problema más difícil. (Sugerencia: Aplique las convenciones de sangrado que ha aprendido)

```
public class Prueba{
       public static void main (String Args[]){

              int X = 5, Y = 8;

              if (Y == 8)
                     if (X == 5)
                            System.out.println ("@@@@@");
                     else
                            System.out.println ("######");

              System.out.println ("$$$$$$");
              System.out.println ("&&&&&&");
       }
}
```

```
a)  a) a)     a)     if (x<10)
     if (y>10)
    S.o.p. (“*****”);
    Else
    S.o.p.(“ ######”);
    S.o.p.(“ $$$$$$”);
b)     b)     b)     b)     if
       (x<10){
     if (y>10)
    S.o.p. (“*****”);
    }Else{
    S.o.p.(“ ######”);
```

==========

obs: del libro

2.22(otro problema de else colgante) Modifique el siguiente código para producir la salida que se muestra. Utilice técnicas de sangrado apropiadas. No puede hacer ninguna modificación como no sea insertar llaves. El compilador hace caso omiso de las sangrías en los programas en Java. Hemos eliminado las sangrías del siguiente fragmento de código para hacer el problema más difícil. Nota: Podría no ser necesaria ninguna modificación.

```
public class Prueba{

       public static void main (String Args[]){

              int X = 5, Y = 8;

              if (Y == 8)
                     if (X == 5)
                            System.out.println ("@@@@@");
                     else{
                            System.out.println ("######");
```

```
                        System.out.println ("$$$$$$");
                        System.out.println ("&&&&&&");
                  }
      }
}
```

```
a)  a) a)     a)     suponiendo
   que x=5 y y=8, se produce la
   sigte salida:
   @@@@@
   $$$$$
   &&&&&
b)  b) b)     b)     suponiendo
   que x=5 y y=8, se produce la
   sigte salida:
   @@@@@
c)  c) c)     c)     suponiendo
   que x=5 y y=8, se produce la
   sigte salida:
   @@@@@
   &&&&&
d)  d) d)     d)     suponiendo
   que x=5 y y=7, se produce la
   sigte salida:
   #####
   $$$$$$
   &&&&&
```

==========

2.23 Escriba una applet que lea el tamaño del lado de un cuadro y luego imprima un cuadrado hueco de ese tamaño con asteriscos empleando el método drawString dentro del método **paint** de su applet. El programa deberá funcionar para cuadrados de cualquier tamaño entre 1 y 20. Por ej., si su programa lee un tamaño de 5 deberá imprimir:

```
Tcuadrado. Java                                                    ******
                                                                   *
import java.awt.*;                                                 *
import java.applet.Applet;                                         *
                                                                   *
public class TCuadrado extends Applet{                             *
                                                                   *
      Label     L1, L2;                                          *
      TextField E1, E2;
      Button    B1;

      private byte Ancho, Alto;

//---------------------------------------------------------------------------

      public void Asignar (byte An, byte Al){

            Ancho = An;
            Alto  = Al;
      }

//---------------------------------------------------------------------------
      public void Dibujar (Graphics G){

                  for (int I = 1; I <= Ancho; I++){
                        G.drawString ("*", (I * 8) + 100, 100);
                        G.drawString ("*", (I * 8) + 100, 100 + (Alto * 8));
                  }

                  for (int I = 1; I <= Alto; I++){
                        G.drawString ("*", 108 , 100 + (I * 8));
                        G.drawString ("*", (Ancho * 8) + 100, 100 + (I * 8));
                  }
      }

//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------
	public void init (){

		L1 = new Label("Introduzca el ancho: ");
		L2 = new Label("Introduzca el alto : ");

		E1 = new TextField (5);
		E2 = new TextField (5);

		B1 = new Button ("Aceptar");

		add (L1); add (E1);
		add (L2); add (E2);

		add (B1);
	}

//---------------------------------------------------------------------------

	public boolean action (Event E, Object O){

		if (E.target == B1){
			Asignar (Byte.parseByte(E1.getText()), Byte.parseByte(E2.getText()));
			repaint();
		}

		return (true);
	}

//---------------------------------------------------------------------------

	public void paint (Graphics G){

		Dibujar(G);
	}
}
```

==========

2.24 Un palindromo es un número o una frase que se lee igual de izquierda a derecha que de derecha a izquierda. Por ej. los sigtes números de cinco dígitos son palindromos: 12321, 55555, 45554 y 11611. Escriba una applet que lea un entero de cinco dígitos y determine si es o no un palíndromo.

==========

2.25 Escriba una applet que introduzca un enteo que contenga sólo ceros y unos (esto es, un entero “binario”) e imprima su equivalente decimal. (Sugerencia: utilice los operadores de residuo y división para extraer los dígitos del número “binario” uno por uno de der. a izq. . Así como en el sistema de numeración decimal donde el dígito de la extrema der. tiene un valor de posición de 1 y el sigte. Dígito a su izq. Tiene un valor de posición de 10, luego 100, luego 1000, etc., en el sistema de numeración binario el dígito de la extrema derecha tiene un valor de posición de 1, el sigte. Dígito a la izq. Tiene un valor de posición de 2, luego 4, luego 8, etc. Así el número decimal 234 se puede interpretar como 4*1+3*10+2*100. El equivalente decimal del 1101 binario es 1*1+0*2+1*4+1*8º1+0+4+8=13)

```
import java.awt.*;
import java.applet.Applet;

public class TBin extends Applet{
	Label     L1;
	TextField E1;
	int       numero;

//-----------------------------------------------------------------------
	public void Asignar(int n){
		numero = n;
	}
//-----------------------------------------------------------------------
	public void init(){
		L1 = new Label("Digite el numero ");
		E1 = new TextField(8);
		add(L1); add(E1);
	}
//-----------------------------------------------------------------------
	public int binario(){
		int k =1,aux=0;
		while (numero>0){
			aux = (numero %10)*k + aux;
			k = k*2;
			numero=numero/10;
		}
		return(aux);
	}
//-----------------------------------------------------------------------
	public boolean action(Event E, Object O){
		if( E.target == E1){

			Asignar(Integer.parseInt(E1.getText()));
			//binario();
			repaint();
		}
		return(true);
	}
//-----------------------------------------------------------------------
	public void paint(Graphics G){
		G.drawString("El numero en decimal es: "+Integer.toString(binario()),100,90);
	}

}
==========
```

2.26 Escriba una aplicación que imprima la siguiente tabla de valores. El programa sólo puede usar tres enunciados de salida, uno de la forma :

```
System.out.print(“*”);
```

uno de la forma:

```
System.out.print (“ ”);
```

y uno de la forma:

```
System.out.println();
```

Observe que el enunciado anterior indica que el programa debe enviar a la salid un solo carácter de nueva línea para pasar a la siguiente línea de la salida.

```
public class TGrafico{
       public static void main(String Args[]){
                for(int i = 1;i <= 8; i++){
                     System.out.println();
                     //System.out.print("");
                       for(int y = 1; y <= 8;y++){
                            System.out.print( "*");
                 System.out.print("");
                      }

                }

       }

}
```

```
********
 ********
********
 ********
********
 ********
********
 ********
```

==========

2.27 Escriba una applet que continuamente exhiba en la barra de estado los múltiplos del entero 2, a saber d, 4, 8, 16, 32, 64, etc. El ciclo no deberá terminar (es decir, hay que crear un ciclo infinito). ¿Qué sucede cuando ejecuta este programa?

```
import java.awt.*;
import java.applet.Applet;

public class TMultiplo2 extends Applet{
       int Val=1;
       public void init(){
            while(Val>0){
              showStatus(Integer.toString(Val*2));
              Val++;
            }

       }
}
```

==========

2.28 ¿Qué error tiene el sigte. Enunciado? Escriba el enunciado correcto para lograr lo que el programador probablemente estaba tratando de hacer.

```
                          System.out.println(++(x+y));

public class TPrueva{
       public static void main(String Args[]){
              int x=0,y=1, c = x + y;

              System.out.println(Integer.toString(++(c)));

       }
}
```

==========

2.29 Escriba una applet que lea tres valores distintos de cero introducidos por el usuario en campos de texto, determine si podrían representar los lados de un triángulo e imprima un mensaje alusivo.

==========

2.30 Escriba una applet que lea tres valores distintos de cero, determine si podrían repsentar los lados de un triángulo rectángulo, e imprima un mensaje alusivo.

```
import java.awt.*;
import java.applet.Applet;

public class TDigitos extends Applet{

	Label	L1;
	TextField	E1;

	private int N;

//-------------------------------------------------------------------------------------

	public void Asignar (int X){

		N = X;
	}

//-------------------------------------------------------------------------------------

	public void Intercambiar (byte Pos1, byte Pos2){

		byte Mem1, Mem2, Cont = 1;
		int Aux = Valor;

		while (Aux > 0){
			if (Cont == Pos1)
				Mem1 = Aux % 10;
			if (Cont == Pos2)
				Mem2 = Aux % 10;

			Aux /= 10;
			Cont++;
		}

		Cont = 1;
		int Expo = 1;

		while (Valor > 0){
			if (Cont == Pos1)
				Aux = Aux + Mem2 * Expo;
			if (Cont == Pos2)
				Aux = Aux + Mem1 * Expo;

			Valor /= 10;
			Expo *= 10;
			Cont++;
		}

		Valor = Aux;
	}

//-------------------------------------------------------------------------------------

	public void Convertir(){

		int Aux = 0, Expo = 1;

		while (Valor > 0){
			Aux = Aux + (((Valor % 10) + 7) % 10) * Expo;
			Valor /= 10;
		}

		Valor = Aux;
	}

//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
```

```
	public void init(){

		N = 0;

		L1 = new Label("Ingrese el primer número: ");
		E1 = new TextField (5);

		add (L1); add (E1);
	}

//-------------------------------------------------------------------------------

	public boolean action(Event E, Object O){

		Asignar (Integer.parseInt(E1.getText()));
		repaint();

		return (true);
	}

//-------------------------------------------------------------------------------

	public void paint (Graphics G){

		Convertir();
		Intercambiar (1, 3);
		Intercambiar (2, 4);
		G.drawString ("El número es: " , 100, 100);
	}
}
```

==========

2.31 Una Cía. desea transmitir datos por teléfono, pero están preocupados por la posibilidad de que sus teléfonos estén intervenidos. Todos sus datos se transmiten como enteros de cuatro dígitos. Se le ha pedido a usted escribir un programa que cifre los datos para poderlos transmitir con mayor seguridad. Su applet deberá leer un entero de cuatro dígitos introducido por el usuario en un campo de texto y cifrarlo como sigue: sustituya cada dígito por (ese dígito + 7) % 10. Luego, intercambie el primer y el tercer dígito, y luego el segundo y cuarto dígitos. Luego imprima el entero cifrado. Escriba otra applet que reciba un entero de cuatro dígitos cifrado y lo descifre para forma el número original.

```
import java.awt.*;
import java.applet.Applet;

public class TDigitos extends Applet{

	Label	L1;
	TextField	E1;
	Button	B1;

	private int Valor;

//-------------------------------------------------------------------------------

	public void Asignar (int X){

		Valor = X;
	}

//-------------------------------------------------------------------------------

	public void Intercambiar (int Pos1, int Pos2){

		int Mem1 = 0, Mem2 = 0, Cont = 1;
		int Aux = Valor;

		while (Aux > 0){
			if (Cont == Pos1)
				Mem1 = Aux % 10;
```

```
                if (Cont == Pos2)
                        Mem2 = Aux % 10;

                Aux /= 10;
                Cont++;
        }

        Cont = 1;
        int Expo = 1, Dig;
        Aux = 0;

        while (Valor > 0){
                Dig = Valor % 10;

                if (Cont == Pos1)
                        Dig = Mem2;
                if (Cont == Pos2)
                        Dig = Mem1;

                Aux = Aux + Dig * Expo;
                Valor = Valor/10;
                Expo =Expo* 10;
                Cont++;
        }

        Valor = Aux;
    }

//-------------------------------------------------------------------------------------

    public void Convertir(){

        int Aux = 0, Expo = 1;

        while (Valor > 0){
                Aux = Aux + (((Valor % 10) + 7) % 10) * Expo;
                Valor =Valor/ 10;
                Expo =Expo*10;
        }

        Valor = Aux;
    }

//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------

    public void init(){

        Valor = 0;

        L1 = new Label("Ingrese el primer número: ");
        E1 = new TextField (5);

        add (L1); add (E1);
    }

//-------------------------------------------------------------------------------------

    public boolean action(Event E, Object O){

        if (E.target == E1){
                Asignar (Integer.parseInt(E1.getText()));
                repaint();
        }

        return (true);
    }

//-------------------------------------------------------------------------------------
```

```
	public void paint (Graphics G){

		Convertir();
		Intercambiar (1, 3);
		Intercambiar (2, 4);
		G.drawString ("El número es: " + Integer.toString(Valor), 100, 100);
	}
}
=====================================
```

2.32 El factorial de un entero no negativo n se escribe n! (se lee “n factorial”) y se define como sigue:
n! = n(n-1)(n-2)....1 para los valores de n mayores o iguales a 1
y n! = 1 para n = 0
por eje. 5! = 5.4.3.2.1 = 120

a) Escriba una applet que lea un entero no negativo de un campo de texto y calcule e imprima su factorial.

b) Escriba una applet que estime el valor de la constante matemática e utilizando la fórmula:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

c) Escriba una applet que calcule el valor de ex utilizando la fórmula

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Ejercicios de autoevaluación

3.1 Indique si las siguientes afirmaciones son verdaderas o falsas. Si la respuesta es falsa, explique porqué.

2 El caso **default** es obligatorio en la estructura de selección **switch**.
Falso. El caso **default** es opcional. Si no se requiere una acción por omisión, no hay necesidad de incluir un caso **default.**

3 El enunciado **break** es obligatorio en el caso por omisión de una estructura de selección **switch.**
Falso.El enunciado **break** sirve par salir de una estructura **switch.** El enunciado **break** no es necesario cuando el caso **default** es él ultimo caso.

4 La expresión **(x > y && a <b)**es verdadera si **x>y** es verdadera o bien **a<b** es verdadera.
Falso. Cuando se usa &&, ambas expresiones relacionales deben ser verdaderas para que toda la expresión sea verdadera.

5 Una expresión que contiene el operador **||** es verdadera si cualquiera de sus operandos es verdadero, o ambos. **Verdadero.**

3.2 Escriba un enunciado en Java o una serie de enunciados en Java para realizar cada una de las siguientes tareas:

6 Sumar los entero impares entre 1 y 99 empleando una estructura **for**.Suponga que ya se declararon las variables enteras **suma** y **resta.**

```
Suma=0;
for(cuenta=1;cuenta<=99;cuenta +=2)
    Suma+=cuenta;
```

7 Calcular el valor de 2.5 elevado a la potencia 3 usando el método **pow.**

```
Math.pow(2.5,3)
```

8 Imprimir los enteros del 1 al 20 usando el ciclo **while** y la variable de conteo **x.** Suponga que ya se declaro la variable **x**, pero no se ha inicializado. Imprima sólo cinco enteros en cada línea. Sugerencia: Utilice el cálculo **x % 5**.Cuando el valor de esto sea 0, imprima un carácter de nueva línea; en los demás casos imprima un carácter de tabulacion. Suponga que esta es una aplicación: utilice el método **System.out.println()**para enviar a la salida el carácter de nueva línea y el método **System.out.print**(‘/t’)para enviar a la salida el carácter de tabulación.

```
X=1;
while(x<=20){
   System.out.print(x);
   if(x%5==0)
     System.out.println(‘/t’);
   else
     System.out.println(‘/t’);
   X++;
}
```

9 Repita el ejercicio 3.2(c)usando una estructura **for**.

```
for(x=1;x<=20;x++){
  System.out.print(x);
   if (x%5==0)
      System.out.println();
  else
       System.out.print(‘/t’);
}   o
for(x=1;x<=20;x++){

   if (x%5==0)
      System.out.println(x);
  else
       System.out.print(x+“/t”);
```

3.3 Encuentre el error en cada uno de los siguientes segmentos de código y explique cómo corregirlo.

```
10  x = 1 ;
    while(x <= 10);
      x++;
    }
```

Error: El signo de punto coma después de la cabecera while causa un ciclo infinito. Sustituir el signo de punto y coma por { o quitar él ; coma la } seria correcto.

```
11  for (y = .1; y != 1.0; y+= .1)
```

```
      System.out.println( y );
```

Error: Usar un numero de punto flotante para controlar una estructura de repetición for:correcto seria usar un Entero

```
      for(y=1;y!=10; y++)
         System.out.println((float)y/10);
```

12

```
switch( n ){
  case1:
  System.out.println("El número es 1");
  Case2:
  System.out.println("El número es 2");
  break;
  default:
  System.out.println("El número no es 1 ni es 2");
  break;
}
```

Error: Falta el enunciado **break** en los enunciados del primer case.Correcto seria colocar un break al final del primer case

13 El siguiente código debería imprimir los valores 1 a 10.

```
N = 1;
while (N < 10)
   System.out.println(n++);
```

Error: En el empleo del operador relacional incorrecto en la condición para continuar la repetición while. Seria correcto utilizar <= en lugar de < o cambie 10 a 11.

3.6 Escriba una aplicación que encuentre la más pequeña de varias letras. Suponga que el primer valor leído es un carácter de dígito que especifica el número de letras que quedan.

```
import java.io.*;

public class TCaracteres{

     final byte Max = 100;

     private char V[] = new char[Max];
     private byte Dim;

     TCaracteres(){

          Dim = 0;
     }


     public void Asignar (char C){

          V[++Dim] = C;
     }


     public char Pequenia (){

          char A = V[1];

          for (byte I = 2; I <= Dim; I++)
               if (V[I] < A)
                    A = V[I];
```

```
            return (A);
      }


      public static void main (String Args[]) throws IOException{

            TCaracteres MiCaracter = new TCaracteres();

            System.out.println();
            System.out.print ("Introduzca el numero de caracteres que desea: ");
            System.out.flush ();
            int D = System.in.read ();

            D = D - 48;
            System.out.println();

            for (int I = 1; I <= D; I++){
                  System.out.print ("Introduzca el " + Integer.toString(I) + "
caracter: ");
                  System.out.flush ();
                  System.in.skip(2);
                  MiCaracter.Asignar ((char) System.in.read());
                  System.out.println();
            }

            System.out.print("El caracter mas pequenio es: " +
MiCaracter.Pequenia () );
            System.out.println();
      }
}
---------------------------------
import java.io.*;

public class TDibujo{

      private char C;

      TDibujo(){

      }


      public void Asignar (char c){

            C = c;
      }


      public void Dibujar (){

            for (byte I = 1; I <= 10; I++){
                  for (byte J = 1; J <= I; J++)
                        System.out.print (C);

                  System.out.println ();
```

```
            }
      }


      public static void main (String Args[]) throws IOException{

            TDibujo MiDibujo = new TDibujo();

            char car;
            System.out.println();
            System.out.print ("Introduzca un caracter");
            System.out.flush();
            car = (char) System.in.read();

            MiDibujo.Asignar (car);

            MiDibujo.Dibujar ();
      }
}
-------------------------------------------------------------

import java.awt.*;
import java.applet.Applet;

public class TAsterisco extends Applet{
      Label L1;
      TextField E1;
      int Val;
      public void Asignar(int V){
            Val  = V;
      }
      public void init(){
            L1 = new Label("Digite un numero");
            E1 = new TextField (5);

            add(L1); add(E1);

      }
      public void paint(Graphics G){
      for(int I=1; I<=Val; I++){
            G.setColor(Color.blue);
            G.drawString("*", I*10, 100);
            }
      }
      public boolean action(Event E, Object O){
            if(E.target==E1){
                    int Vl=Integer.parseInt(E1.getText());
                  E1.setText("");
                  Asignar(Vl);
                  repaint();
            }
            return(true);

      }
}
```

```
------------------------------------------------------------
import java.awt.*;
import java.applet.Applet;

public class TVentas extends Applet{
     Label L1, L2;
     TextField E1, E2;
     int Np=0, Cv=0;
     double Uno=1;

     public void init(){
          L1 = new Label("Digite el numero de producto");
          E1 = new TextField (10);
          L2 = new Label("Digite la cantidad vendida en un dia");
          E2 = new TextField(10);

          add(L1);
          add(E1);
          add(L2);
          add(E2);
     }

     public void paint(Graphics G){
          G.drawString( "El valor total de los porductos vendidos en una
semana"+(Uno*7),75, 100);
     }

     public boolean action(Event E, Object O){

          if (E.target==E2){
               Np =Integer.parseInt(E1.getText());
               Cv =Integer.parseInt(E2.getText());
               switch ( Np ){
                    case 1:
                       Uno=(double) Cv*2.98;
                       break;
                    case 2:
                       Uno=(double) Cv*4.50;
                       break;
                    case 3:
                       Uno=(double) Cv*9.98;
                       break;
                    case 4:
                      Uno=(double) Cv*4.49;
                      break;
                    case 5:
                      Uno=(double) Cv*6.87;
                      break;
                    default:
                      showStatus("Estamos cantinfleando");

               }
          repaint();

          }
```

```
            return(true);
      }


}

-------------------------------------------------------------

import java.awt.*;
import java.applet.Applet;

public class TPrueba extends Applet{
      Label L1;
      TextField E1;

      int Con=0, aCon=0;//, bCon=0, cCon=0, dCon=0, Con=0;
      double fCon=0;
      public void init(){
            L1=new Label("Digite una calificacion");
            E1=new TextField(2);
            add(L1);
            add(E1);
      }


      public void paint(Graphics G){

            G.drawString("El poromedio total de todas las calificaciones es :
",25,40);
            G.drawString(Double.toString(fCon),25,60);
      }


      public boolean action(Event E,Object O){
            String Val = O.toString();
            char Nota =Val.charAt(0);

            showStatus("");
            E1.setText("");
            switch(Nota){
                  case 'A':case 'a':{
                              aCon = aCon + 4;
                              Con++;}; break;

                  case 'B':case 'b':{
                              aCon = aCon + 3;
                              Con++;}; break;

                  case 'C':case 'c':{
                              aCon = aCon + 2;
                              Con++;}; break;

                  case 'D':case 'd':{
                              aCon += aCon;
                              Con++;}; break;
```

```
                case 'F':case 'f':{
                          aCon += aCon;
                          Con++;}; break;
                default:
                     showStatus("Digite una Letra calificacion.");
           }

           fCon=(double)(aCon)/Con;
           repaint();
           return (true);
      }

}

import java.io.*;

public class TCantidades{

      final byte    Max   = 100;
      final double  Dolar = 5.5;

      private int V[] = new int[Max];
      private byte Dim;

      TCantidades(){

           Dim = 0;
      }


      public void Asignar (int C){

           V[++Dim] = C;
      }


      public double Pequenia (){

           int A = V[1];

           for (byte I = 2; I <= Dim; I++)
                if (V[I] < A)
                     A = V[I];

           return ((double) (A / Dolar));
      }


      public static void main (String Args[]) throws IOException{

            try{
                TCantidades MiCantidad = new TCantidades();

                System.out.println();
                System.out.print ("Introduzca el numero de cantidades: ");
                System.out.flush ();
```

```
                int D = System.in.read ();

                D = D - 48;

                for (int I = 1; I <= D; I++){
                     System.out.print ("Introduzca la " + Integer.toString(I) +
" cantidad: ");
                     System.out.flush ();
                     System.in.skip(2);
                     MiCantidad.Asignar ((int) System.in.read());
                     System.out.println();
                }

                System.out.print("La cantidad mas pequenia es: " +
MiCantidad.Pequenia () );
                System.out.println();
            }
            catch (Exception e){
                System.err.println ("Error: Sus datos no son validos!!!");
            }
      }
}
```

```
import java.io.*;

public class TSystem{

      public static void main (String Args[]){

            try{
                System.out.println (5 == 4);
            }
            catch(Exception E){
                System.err.println ("Error: Sus datos no son validos!!!");
            }
      }
}
```

```
import java.io.*;

public class TNumero{


      public static String Binario (byte N) throws Exception{

            try{
                String Cad = "";

                while (N > 0){
                     if (N % 2 == 0)
                          Cad = "0" + Cad;
                     else
```

```
                    Cad = "1" + Cad;

                N /= 2;
            }

            return (Cad);
        }
        catch (Exception E){
            throw E;
        }
        finally{
            N = 0;
        }
    }
    public static String Octal (byte N) throws Exception{

        try{
            String Aux = "";

            while (N > 0){
                Aux = (N % 8) + Aux;
                N /= 8;
            }

            return (Aux);
        }
        catch (Exception E){
            throw E;
        }
        finally{
            N = 0;
        }
    }


    public static String Hexadecimal (byte N) throws Exception{

        try{
            String Cad = "";

            while (N > 0){
                if (N % 16 > 9)
                    Cad = (char) ((N % 16) + 55) + Cad;
                else
                    Cad = (N % 16) + Cad;

                N /= 16;
            }

            return (Cad);
        }
        catch (Exception E){
            throw E;
        }
        finally{
            System.out.println();
```

```
          }
     }


     public static void main (String Args[]) throws IOException{

          try{
               System.out.println ();
               System.out.println (" Binario" + "             Octal"  +
"              Hexadecimal");

               for (byte I = 1; I <= 15; I++)
                    System.out.println ("  " + Binario(I) +  "
" + Octal(I)  + "                         " + Hexadecimal(I));
          }
          catch (Exception E){

          }
          finally{

          }
     }
}
```

```
import java.io.*;

public class TSerie{

     public static void main (String Args[]) throws IOException{

          System.out.println ();
          System.out.print ("Introduzca la cantidad de terminos que desea ver
(0-9): ");
          System.out.flush();
          int D = System.in.read (), Signo = 1;
          System.out.println ();
          float Acu = 0;
          byte J = 1;

          for (byte I = 1; I <= D - 48; I++){
               System.out.print (Signo + "4/" + Integer.toString(J) + " ");
               Acu = Acu + Signo * (4/J);
               Signo = Signo * (-1);
               J += 2;
          }

          System.out.println ();
          System.out.println (Acu);
     }
}
```

```
import java.io.*;
```

```
public class TPitagoras2 {

	static void Calcular (int N){

		int h = 5, a, b, Tiempo = 0;

		if (N < 5)
			System.out.println ("No exiten valores!!!");
		else{
			while (h <= N){
				a = 3;

				while (a <= N){
					b = 3;

					while (b <= N){
						if (h * h == a * a + b * b){
							System.out.println ("La hipotenusa es: " +
h);
							System.out.println ("El cateto opuesto es:
" + a);
							System.out.println ("El cateto adyacente
es: " + b);
							System.out.println();
						}

						b++;
						Tiempo++;
					}

					a++;
				}

				h++;
			}


			System.out.println ("El tiempo de iteraciones es: " + Tiempo);
		}
	}
	public static void main (String Args[]) throws IOException{

		String S = "";

		System.out.println ();
		System.out.print ("Teclee un numero: ");

		int c;

		do{
			System.out.flush ();
			c = System.in.read ();

			if (c != 32 && c!= 13 && c!= 0)
				S = S + (c - 48);
```

```
            }
            while (c != 32 && c!= 13 && c!= 0);

            System.out.println ();

            Calcular (Integer.parseInt(S));
      }
}
```

```
import java.io.*;

public class TMorgan{

      public static void main (String Args[]) throws IOException{

             try{
                  TCantidades MiCantidad = new TCantidades();

                  System.out.println();
                  System.out.print ("Introduzca el primer numero: ");
                  System.out.flush ();
                  int X = System.in.read ();
                  System.out.skip (2);

                  System.out.println();
                  System.out.print ("Introduzca el segundo numero: ");
                  System.out.flush ();
                  int Y = System.in.read ();
                  System.out.skip (2);

                  System.out.println();
             }
             catch (Exception e){
                  System.err.println ("Error: Sus datos no son validos!!!");
             }
      }
}
```

```
import java.awt.*;
import java.applet.Applet;

public class TGraficos extends Applet{

      Label       L1;
      TextField E1;
      Button     B1;

      private String C;
      private Color Colorcito;

      public void Asignar(String Aux){
            C = Aux;
```

```
}


public void Rectangulo (Graphics G){

     for (byte I = 1; I <= 70; I+=4){
          G.drawString (C, I+50, 150);
          G.drawString (C, I+50, 250);
     }

     for (byte I = 1; I <= 100; I+=4){
          G.drawString (C, 50, I + 150);
          G.drawString (C, 120, I + 150);
     }
}


public void Elipse (Graphics G){

     for (byte I = 1; I <= 70; I+=4){
          G.drawString (C, 350, I + 160);
          G.drawString (C, 420, I + 160);
     }

     for (byte I = 1; I <= 40; I+=8){
          G.drawString (C, I + 370, 140);
          G.drawString (C, I + 370, 250);
     }

     G.drawString (C, 360, 150);
     G.drawString (C, 410 ,150);
     G.drawString (C, 360, 240);
     G.drawString (C, 410, 240);
}


public void Flecha (Graphics G){

     G.drawString(C, 180, 152);

     for (byte J = 1; J <= 5; J++)                          // Columnas
          for (byte I = 0; I<=J; I++){         //  Filas
               G.drawString (C, 180 - (I * 8), 150 + (J * 8));
               G.drawString (C, 180 + (I * 8), 150 + (J * 8));
          }

     for (int I = 1; I <= 8; I++)
          G.drawString (C, 180, 200 + (I * 6) );
}


public void Rombo (Graphics G){

     for (byte I = 0; I <= 7; I++){
          G.drawString (C, 270 + (I * 5), 150 + (I * 8));  // Arriba
          G.drawString (C, 270 - (I * 5), 150 + (I * 8));
```

```
                if (I < 7){
                     G.drawString (C, 270 + (I * 5), 260 - (I * 8));  // Abajo
                     G.drawString (C, 270 - (I * 5), 260 - (I * 8));
                }
           }
     }


     public void init (){

           L1 = new Label ("Introduzca un carácter: ");
           B1 = new Button ("Aceptar");
           E1 = new TextField (5);

           add (L1);
           add (E1);
           add (B1);
     }


     public boolean action (Event E, Object O){

           if (E.target == B1){
                String C = E1.getText();
                Asignar (C);
                repaint();
           }

           return (true);
     }



     public void paint (Graphics G){

           G.setColor  (Color.blue);

     //    Rectangulo(G);
     //    Flecha (G);
           Rombo (G);
     //    Elipse (G);
           showStatus ("Made By ACP");
     }
}

-------------------------------------------------------------


-------------------------------------------------------------

import java.awt.*;
import java.applet.Applet;

public class TProducto extends Applet{
```

```
      public void paint(Graphics G){

            int Aux = 1;

            for (byte I = 1; I <= 15; I+= 2)
                        Aux *= I;

            Font Fondo = new Font ("Courier", Font.ITALIC + Font.BOLD, 24);

            G.setFont (Fondo);
            G.setColor (Color.blue);
            G.drawString ("El resultado es: " + Integer.toString(Aux), 100, 100);

      }
}

-------------------------------------------------------------

import java.awt.*;
import java.applet.Applet;

public class TFactorial extends Applet{

      Label L1;
      TextField E1;

      private byte Valor;

      public void Asignar(byte N){

            Valor = N;
      }


      public long Factorial(byte N){

            long M = 1;

            for (int I = 1; I<=N; I++)
                  M*= I;

            return(M);
      }


      public void Dibujar (Graphics G){

            for (byte I = 1; I <= this.Valor; I++)
                  G.drawString (Byte.toString(I) + "          " +
Long.toString(Factorial(I)), 50, (I * 15) + 80);
      }


      public void init(){
            L1 = new Label("Introduzca un numero ");
            E1 = new TextField (5);
```

```
            add(L1); add(E1);
      }

      public boolean handleEvent (Event Evento){

            switch (Evento.id){
                  case Event.ACTION_EVENT:{
                              Asignar (Byte.parseByte(E1.getText()));
                              repaint ();
                        }

                        return (true);

                  default:{

                        return (false);
                  }
            }
      }


      public void paint(Graphics G ){

            Font Fondo = new Font("Helvetica", Font.ITALIC + Font.BOLD, 15);
            G.setFont (Fondo);
            G.setColor (Color.blue);

            Dibujar (G);
      }
}

-------------------------------------------------------------

import java.awt.*;
import java.applet.Applet;

public class TInteres extends Applet{

      Choice    CB1;
      Label     L1;

      private double Valor;

      public void Asignar (double N){

            Valor = N;
      }


      public void init (){

            CB1 = new Choice ();
            CB1.addItem (" 5% ");
            CB1.addItem (" 6% ");
            CB1.addItem (" 7% ");
```

```
        CB1.addItem (" 8% ");
        CB1.addItem (" 9% ");
        CB1.addItem (" 10% ");

        L1 =  new Label ("Elija la tasa de interes: ");
        add (L1); add (CB1);
    }


    public boolean action (Event E, Object O){

        if (E.target instanceof Choice){

            if (CB1.getSelectedItem() == " 5% ")
                this.Asignar (0.05);

            if (CB1.getSelectedItem() == " 6% ")
                this.Asignar (0.06);

            if (CB1.getSelectedItem() == " 7% ")
                this.Asignar (0.07);

            if (CB1.getSelectedItem() == " 8% ")
                this.Asignar (0.08);

            if (CB1.getSelectedItem() == " 9% ")
                this.Asignar (0.09);

            if (CB1.getSelectedItem() == " 10% ")
                this.Asignar (0.10);

            repaint();
        }

        return (true);
    }


    public void paint (Graphics G){
        int Pos=40;

        for(int A=1;A<=10;A++){
            double R=1000.0*Math.pow(1.0+Valor, A);
            G.drawString(Integer.toString(A), 25, Pos);
            G.drawString(Double.toString(R),100, Pos);
            Pos+=15;
        }
    }
}
```

4.1. Introducción

Para crear y mantener un programa grande se construirá a partir de piezas pequeñas o módulos. Para ello, se utilizará la técnica divide y vencerás.

Este cap. describe las características clave del lenguaje java que facilitan:

- El diseño
- Implementación
- Operación y
- Mantenimiento de programas grandes

4.2. Módulos del programa en Java

- Los módulos en Java se llaman métodos y clases.
- La java API ofrece colección de métodos y clases para realizar:
  - cálculos matemáticos
  - manipular cadenas
  - manipular caracteres
  - realizar la entrada y salida de datos
  - verificar errores, etc.
- Los métodos de Java API forman parte del Java Developer's Kit (JDK).
- Un método se invoca con una llamada de método, este proporciona información

Ej. **un jefe(**método que invoca) **pide a un trabajador**(método invocado) **que realice una tarea y devuelva el resultado**

- El método MAIN se comunica con varios métodos trabajadores de forma jerárquica.

4.3. **Métodos de la clase MATH** //La clase Math forma parte del paquete java.lang.

- Estos métodos permiten realizar cálculos matemáticos.

Ej. Calcular e imprimir la raíz cuadrada de 900.0:

System.out.println(Math.sqrt(900.0) // imprimirá 30. recibe(argum) y devuelve un resultado de tipo double

Normalmente invocamos métodos a través de un objeto o directamente, como en las sigtes líneas de código de una applet.7

g.drawString("Bienvenidos...", 25, 25); // podría aparecer en el método **paint** de la applet.
showStratus(" Hola"); // invoca y exhibe una cadena en la barra del Apletviewer

La 1ra. línea invoca el **método drawString** a **través del objeto g** de la **clase Graphics** .(*enviar el mensaje drawString al objeto g*).

La 2da. Línea podría aparecer en cualquier método de la applet.

Los argumentos de los métodos son: constantes, variables, expresiones, etc..

4.4. Métodos

Los métodos permiten modularizar programas.

Las variables declaradas en métodos son locales y son la lista de parámetros, permiten comunicar entre métodos.

4.5. Definiciones de Métodos

Cada programa constituye una definición de clase, que contiene al menos una definición de método que invoca métodos de la Java API para llevar a cabo sus tareas.

En Java todos los métodos deben definirse dentro de una definición de clase. Ejemplo:

```
Public class SquareInt extends Applet
{
        métodos
}
```

Formato de una definición de método:

```
                                        Es cualquier identificador válido  el método recibe cuando se le invoca
        Tipo-de-valor-devuelto   nombre-de-método(      lista de parámetros      )
//Tipo de datos del resultado que el método devuelve al invocador .      // no recibe valores si está vacía
        {
                declaraciones y enunciados //incluye el cuerpo del método llamado también bloque
        }
```

definir un parámetro de método otra vez como variable local en el método es error de sintaxis.

El tipo-de-valor-devuelto VOID indica que el método no devuelve ningún valor

Se debe listar un tipo para cada parámetro

Los bloques pueden anidarse.

No definir un método dentro de otro método.

La cabecera del método y las llamadas a ese método deben coincidir en el número, tipo y orden de los argumentos y parámetros y en el tipo del valor devuelto.

Método definido por el programador.

Una applet usa un **método square se invoca en paint //** calcula cuadrados => square(x)

Ej. método definido por el programador

```
Import java.awt.Graphics;
Import java.applet.Applet;

Public class SquareInt extends Applet {
//exhibir los cuadrados de los valores 1 a 10
        public void paint ( Graphics g) //paint se invoca autom. para que la applet pueda exhibir
        {//invoca repetidamente el método square para realizar el cálculo
        int xPosition = 25;
                for(int x = 1; x <= 10; x++ ){ //proceso que se repite 10 veces
                    g.drawString( String.valueOf ( square ( x )), xPosition, 25);// llama al método y se exhbe
                xPosition += 20;
                }
        }
        //definición del método square, este recibe una copia del valor de x en el parámetro y

        public int square( int y )
        {
        return y * y;   // calcula y el resultado pasa a paint, allí donde invoca square y exhibe
        }
}
```

En este programa contiene dos definiciones de métodos: paint y square.
Existe formas de devolver el control al punto en el que se invocó un método.

- el método no devuelve resultado alguno, entonces el control se devuelve cuando llega a la llave der. o al ejecutarse return (regresar)
- el método devuelve el valor de la expresión al invocador en return expresión.

Ej.2: Método definido (maximum)por el programador: devuelve el más grande de tres números.

```
Import java.awt.*;
Import java.applet.Applet ;

Public class Maximun extends Applet{{

    Label L1, L2, L3, resultLabel;
    TextFieldnumber1, number2, number3, result;
    Int num1, num2, num3, max;

    //El usuario introduce 3 #s en campos de texto.
    //Se oprime enter en uno de los campos de texto, se invoca el método action, se leen las cadenas de los campos de texto y se convierten en enteros, y los enteros se pasan al método maximum, entonces se determina cuál de ellos es el más grande.
    Este valor se devuelve al método action mediante return del método maxium. El valor devuelto se asigna a la variable max y luego se exhibe en el campo de texto result (resultado).

    //estableciendo rótulos y campos de texto.

    public void init()
    {
        L1 = new Label (“teclee el primer entero:”);
        number1 = new TextField( “0”, 10)
        L2 = new Label (“teclee el segundo entero:”);
        number2 = new TextField( “0”, 10)
        L3 = new Label (“teclee el tercer entero:”);
        number3 = new TextField( “0”, 10)
        resultLabel1 = new Label(“El valor máximos es:”);
        result  = new TextField( “0”, 10)
        result.setEditable( false );

        add( L1 );
        add( number1 );
        add( L2 );
        add( number2 );
```

```
        add( L3 );
        add( number3 );
        add( resultLabel );
        add( result );
    }
    //implementación o definición del método máximun

        public int maximun( int x, int y, int z )
```

El método devuelve el más grande de los 3 enteros, recibe tres parámetros enteros para realizar su tarea. El cuerpo del método contiene:

```
        {
```

segundo método primer método.

```
            return Math.max( x, Math.max( y, z) );

            // llama dos veces al método Math.max.
```

//el **primer método Math.max** se invoca con los valores de las vars. "y, z" para determinar el mayor. A continuación se pasan al método **Math. Max** el valor de la variable x y el resultado de la primera llamada a **Math.max**.

//Por último, el resultado de la 2da. llamada a **Math.max** se devuelve al punto en el que se invocó a **maximun** (esto es, al **método action** de este programa).

```
        }

    //obtener los enteros e invocar el método maximum
      public bolean action( Event e, Object o )
      {
        num1 = integer.parseInt( number1.getText() );
        num2 = integer.parseInt( number2.getText() );
        num3 = integer.parseInt( number3.getText() );
        result.setText( Integer.toString( max ) );
        returrn true;
      }
}El enunciado: System.out.println( Math.sqrt( 4 ));  evalúa Math.sqrt( 4 ) e imprime el valor 2.
```

4.6. Paquetes de la Java API

Java contiene muchas piezas predefinidas llamadas clases que se agrupan mediante sudirectorios del disco en categorías de clases relacionadas entre sí llamadas paquetes. Juntos, estos paquetes se conocen como la interfaz de programación de aplicaciones de Java API (applications programming interface)

- **Import** carga las clases necesarias para compilar un programa en Java.

Ej. **import java.applet.Applet**;

Ventajas de java: gran número de clases contenidas en los paquetes de la Java API que se puede reutilizar .

Paquetes de Java API: (Descripción)

java.applet	**El paquete de applets** de Java (Java Applet Package) Este paq. Contiene la clase Applet y interfaces que permiten la creación de applets, la interacción de las applets con el navegador y la reproducción de clips de audio.
java.awt .	**Paquete de herramientas** abstractas para trabajar con ventanas de Java (Abstract Windowing Toolkit Package). Este paq. Contiene todas las clases e interfaces necesarias para crear y manipular interfaces gráficas con el usuario.
Java.awt.image	**El paq. de imágenes** de las herramientas abstractas para trabajar con ventanas de Java (Java Abstract Windowing Toolkit Image Package). Este paq. Contiene clases e interfaces que permiten almacenar y manipular imágenes en un programa.
Java.awt.peer	**El paq. de pares** de las herramientas abstractas para trabajar con ventanas de Java.

Este paq. contiene interfaces que permiten a los componentes de interfaz gráfica con el usuario de java interactuar con sus versiones en plataformas específicas.

Java.io **El paq. de E/S de** Java (Java Input/Output Package)
Este paq. contiene clases de permiten a los programas introducir y producir datos.

Java.lang **El paq. del lenguaje** Java (Java Lenguaje Package)
El compilador importa automátic. Este paq. en todos los programas Contiene las claes e interfaces básicas requeridas por muchos programs en java. Se ven en todo el texto estas clases.

Java.net **El paq. de trabajo con redes** de Java (Java Networking Package)
Este paq. contiene clases que permiten a los programs comunicarse a través de la Internet o de intrarredes corporativas.

Java.util **El paq. de utilidades** de Java (Java Utilities Package)
Este paq. contiene clases e interfaces de utilidad como manipulaciones de fechas y horas (date), diversas capacidades de procesamiento de números aleatorios (Random), almacenamiento y procesamiento de cantidades de datos, dividiendo las cadenas en fragmentos más pequeños llamados unidades lexicográficas o tokens (StringTokenize).

4.7. Generación de números aleatorios

Programas de simulación y juegos:

- La posibilidad de que la suerte convierta un puñado de dinero en una montaña de riqueza.
- El elemento del azar se puede introducir mediante el método **random de la clase Math.**
  Ej. **double randomValue = Math.random();**

Como demostración de **random**, crearemos un programa para simular 20 lanzamientos de un dado de seis caras e imprimiremos el valor de cada tirada. Utilizaremos el operador de multiplicación con random:

(int) (Math.random) () * 6)

para producir enteros en el intervalo de 0 a 5. Esto se determina escalado, y el número 6 es el factor de escala.

```
//Enteros aleatorios desplazados escalados. La anchura de este intervalo es de 6 y
que el número inicial del intervalo es 1
import java.awt.Graphics;
import java.applet.Applet ;

public class RandomInt extends Applet {
   public void paint ( Graphics g)
   {
      int xPosition = 25 ;// posición x
      int yPosition = 25 ; // posición y
      int value; // valor

      for ( int i = 1; i <= 20;  i++ ) {
         value  = 1 + (int) ( Math.random() * 6 ); //produce enteros en el intervalo
      de 0 a 5
         g.drawString( Integer.toString( value ), xPosition, yPosition );
         if ( I % 5 != 0 )
             xPosition += 40;
         else{
             xPosition = 25;
             yPositon += 15;
         }
      }
   }

}
```

- A continuación desplazamos el intervalo de números producido sumando 1 a nuestro resultado anterior.

- Los resultados están en el intervalo de 1 a 6.
- Se **utiliza el operador de mutación a entero** para truncar la parte de punto flotante de cada valor producido por la expresión anterior.
- Con objeto de demostrar que estos números ocurren con aproximadamente la misma probabilidad, simularemos 6000 tiradas de un dado. Cada entero de 1 a 6 deberá aparecer más o menos 1000 veces.
- Con la ayuda del escalado y el desplazamiento hemos utilizado el método **random** para simular de forma realista el lanzamiento de un dado de seis caras. No se incluye un caso **default** en la estructura **switch.** Los valores producidos por **random** siempre están dentro del intervalo:

0.0 <= Math.random() < 1.0

Un solo enunciado para simular el lanzamiento de un dado de seis caras:

Int face = 1 + (int) (Math.random() * 6);

Que siempre asigna un entero (al azar) a la variable face (cara) en el intervalo 1<=face<=6.

La anchura del intervalo está determinada por el número empleado para **escalar random** con el operador de multiplicación (es decir, 6) y número inicial del intervalo es igual al número (1 en este caso) que se suma a (int) (Math.random() * 6).

```
//tirar un dado de seis caras 6000 veces
import java.awt.Graphics;
import java.applet. Applet;

public class RollDie extends Applet {

  int frequency1 = 0, int frequency2 = 0, int frequency3 = 0,
   int frequency4 = 0, int frequency5 = 0, int frequency6  = 0, // frecuencias de c/u
de los seis resultados.

  //resumir los resultados
  public void start()
  {
     for I int  roll = 1; roll <= 6000; roll++ ){
       int face = 1 + (int) ( Math.random() * 6 ) ;

       switch ( face ) {
       case 1 :
            ++ frequency1 ;
          break ;
       case 2 :
          ++ frequency2;
          break ;
       case 3 :
          ++ frequency3 ;
          break ;
       case 4  :
          ++ frequency4 ;
          break ;
       case 5 :
          ++ frequency5 ;
          break ;
       case 6 :
       ++ frequency6 ;
          break ;
       }
     }
  }
```

estos números ocurren con aproximadamente la misma probabilidad, se simula 6000 tiradas de un dado en el sigte. Programa, c/entero de 1 a 6 deberá aparecer más o menos 1000 veces.

```
   //exhibir los resultados
   public void paint( Graphics g )
   {
      g.drawString( "Cara", 25, 25 );
      g.drawString( ""frecuencia", 100, 25);
      g.drawString( ""1", 25, 40 );
      g.drawString( Integer.toString( frequency1 ), 100, 40 );
      g.drawString( "2", 25, 25);
      g.drawString( Integer.toString( frequency2 ), 100, 55 );
      g.drawString( "3", 25, 70 );
      g.drawString( Integer.toString( frequency3 ), 100,70 );
      g.drawString( "4", 25, 85 );
      g.drawString( Integer.toString( frequency4 ), 100,85 );
      g.drawString( "5", 25, 100 );
      g.drawString( Integer.toString( frequency5 ), 100, 100 );
      g.drawString( "6", 25, 115 );
      g.drawString( Integer.toString( frequency6), 100, 115 );
   }
}
```

```
cara  frecuencia
   1     1020
   2     1008
   3     981
   4     990
   5     995
   6     1006
```

Generalizando el resultado:

```
                n = a + (int ( Math.random() * b );
```

donde **a** es el valor de desplazamiento (y es igual al primer número del intervalo de enteros consecutivos deseados) y **b** es el factor de escala (que es igual a la anchura del intervalo de enteros consecutivos deseados).
Es posible escoger enteros al azar de conjuntos de valores que no sean intervalos de enteros consecutivos.

4.8. Ejemplo: Un juego de azar

Juegos al azar: el juego de dados

Reglas:

- al lanzar dos dados, si se suma 7 u 11 en el primer lanzamiento, el jugador gana.
- Si la suma es 2, 3 ó 12 en el primer lanzamiento (craps), el jugador pierde (la casa gana).
- Si la suma es 4, 5, 6, 8, 9 ó 10 en el primer lanzamiento, esta suma se convierte en el "punto" del jugador. Para ganar, el jugador debe seguir tirando los dados hasta "lograr su punto" (esto es, obtener otra vez esa suma en una tirada).
- El jugador perderá si tira un 7 antes de lograr su punto. El programa simula un juego de craps:

```
//Programa para simular el juego de craps
import java.awt.*;
import java.applet.Applet;

public class Craps extends Applet{
//crea vars. que definen los tres estados de un juego de craps: juego ganado, perdido o continuar tirando los dados.
//la palabra clave final al principio de la declaración indica que éstas son variables constantes. Estas vars. deben inicializarse cuando se declaran y no pueden modificarse..
   final int WON = 0, LOST = 1, CONTINUE = 2  // ganar/perder/continuar  (var. constantes para el juego).
   boolean  firstRoll = true;     // true si es la primera tirada
   Int dieSum = 0;                // suma de datos
```

```
Int myPoint = 0;                // punto si no gana/pierde en primera tirada

Int gameStatus = CONTINUE ;  // GANAR, PERDER, CONTINUAR
Var gameStatus (situación del juego) sirve para seguir la pista a esto
//componentes de la interfaz gráfica con el usuario
Label  die1Label, die2Label, sumLabel, pointLabel;
TextField firstDie, secondDie, Sum, point;
Button roll;

// preparar componentes de la interfaz gráfica con el usuario
public void init()
{
      die1Label = new Label ( “Dado 1”);
      firstDie = new TextField ( 10 ); // campo de texto del primer dado
      firstDie.setEditable( false );
      die2Label = new Label ( “Dado 2 “ );
      secondDie = new TextField( 10 );
      secondDie.setEditable( false );.
      sumLabel = new Label (“La suma es” );
      sum = new TextField (10 );// campo de texto de la suma
      sum.setEditable (False);
      roll = new Button (“Tirar dados”);// hacer clic en este boton para tirar los dados. Esto invoca el método
      action de la applet, que a su vez invoca el  método play(jugar).
      pointLabel = new Label (“El punto es” );
      point = new TextField( 10 );
      point.setEditable( false);

      add( die1Label);
      add( firstDie);
      add( die2Label);
      add( secondDie);
      add( sumLabel);
      add( sum);
      add( pointLabel);
      add( point);
      add( roll);
}
// procesar un lanzamiento de los dados
public void play()// El método play verifica la var firsRool (1ra. Tirada) de tipo boolean para determinar si es true o
false; si es true se trata de la primera tirada del juego.
{
      if ( firstRoll ) { // primer lanzamiento de los dados
         dieSum = rollDice(); //1 vez invocado

         switch( dieSum ) {
            case 7: case 11:          // gana en la primera tirada
               gameStatus = WON;
               point.setText( “ ” ); //borrar campo de texto de punto
               firstRoll = true; //permitir inicio de nuevo juego
               break;
            case 2: case 3:  case 12..// pierde en la primera tirada
               gameStatus = LOST;
               point.setText( “ ” );//borrar campo de texto de punto
               firstRoll = true; // permitir inicio de nuevo juego
               break;
```

```
                default:            // recordar punto
                    gameStatus = CONTINUE;
                    myPoint = dieSum;
                    point.setText( Integer.toString( myPoint ) );
                    firstRoll = false;
                    break;
            }
        }
     }else{
        dieSum = rollDice(); //2 veces invocado
        if ( dieSum == myPoint )          // ganar logrando el punto
                gameStatus = WON;
        else
                if ( dieSum == 7 )        // perder por tirar 7
                        gameStatus = LOST;
        }
        if ( gameStatus == CONTINUE )
                showStatus( "Tire otra vez");
        else {
            if ( gameStatus == WON )
                showStatus( "el jugador gana." + "haga clic en tirar dados par jugar otra vez.");
            else
                showStatus ("El jugadro pierde." + "Haga clic en tirar dados para jugar otra vez");
            firstRoll = true;
        }
    }

    // llamar al método play cuando se hace clic en el botón
    public bolean action ( Event e, Object o )
    {
        play();
        return true;
    }

    // método creado para tirar los dados y calcular y exhibir su suma.
    //Este método no recibe argumentos, lista de parámetros vacía.
    //Este método si devuelve la suma de los dados, de tipo entero.
     int rollDice()// definido una vez, invocado dos veces
     {
        int die1, die2, workSum;

        die1 = 1 + (int) ( Math.random() * 6 );
        die2 = 1 + (int) ( Math.random() * 6 );
        workSum = die1 + die2;

        firstDie.setText( Integer.toString( die1 ));
        secondDie.setText( Integer.toString( die2 ));
        sum.setText( Integer.toString( workSum ) );

        return workSum;
     }
}
```

dado1 = 2 **dado2** = 3
la suma es: 5 **el punto es** 5

tire otras vez

el jugador debe lanzar dos dados en la primera tirada, y también todas las tiradas subsecuentes. .

4.9.- Variables automáticas

Java tiene identificadores de duración estática.

- Las **vars. y métodos de duración estática existen** desde el punto en el que la clase en la que se define se carga en la memoria para ejecutarse hasta que el programa termine.
- Las vars. con duración estática, se asigna memoria y se efectúa la inicialización una vez en el momento en que su clase se carga en la memoria.
- Los **métodos con duración estática,** el nombre del método existe cuando su clase se carga en la memoria.
- Las variables y nombres de métodos son duración estática ,existen al cargarse su clase en la memoria.

Todos los identificadores de un programa tienen otros atributos que incluyen la **duración y el alcance.**

4.10. Reglas de alcance

Un nombre de una variable es cualquier **identificador** válido.

Un **identificador** es una serie de caracteres que consiste en letras, dígitos, subrayados (_) y signos de dólar ($) que no comienza con un dígito. **Java permite identificadores de cualquier longitud**. (Ej. las letras mayúsculas y minúsculas son identificadores diferentes a2 y A2).

Sus atributos de estas variables son: nombre, tipo, tamaño y valor. De la misma manera utilizamos **identificadores para métodos definidos**. Estos **identificadores tienen otros atributos** que incluyen la **duración y el alcance** :

- **La duración** (tiempo de vida) **de un identificador** determina el periodo durante el cual dicho identificador existe en la memoria.
- **Alcance de un identificador,** es la **porción del programa** en la que se puede hacer referencia a dicho identificador.

  Los **alcances de los identificadores** son: **alcance de clase y alcance de bloque.**

  - **Alcance de clase.-** El alcance de clase comienza en la llave izquierda (que inicia la definición de clase) y termina en la llave derecha (que cierra dicha definición).

    El alcance de clase permite a los métodos de una clase invocar directamente todos los métodos definidos o heredados en la clase. Una variable de ejemplar o global declarada fuera de cualquier método tiene alcance de clase.

  - **Alcance de bloque.-** El alcance de bloque comienza en la declaración del identificador y termina en la llave derecha que cierra el bloque.

    Las variables locales declaradas al principio de un método tienen alcance de bloque, lo mismo que los parámetros de un método (que son var. loc. del método).

    El compilador genera un mensaje de error de sintaxis, cuando un **identificador de un bloque exterior tiene el mismo nombre que un identificador de un bloque interior** (cuando se anidan los bloques).

    ```
    Si una var local de un método tiene el mismo nombre que una var de ejemplar, la
    variable de ejemplar queda oculta. Cualquier bloque puede contener declaraciones de
    variables.
    ```

Ej. de determinación de alcance, con **var de ejemplar** y **var locales** automáticas.

```
Import java.awt.Graphics;
Import java.applet.Applet;

public class Scoping extends Applet{
  int x = 1;       // var. de ejemplar, ésta var queda oculta en cualquier bloque o método en el que se declara una va x

      public void paint( Graphics g)
      {
            g.drawString ( "vea la salida en la línea de comandos", 25, 25 );
            int x = 5          // var. local de paint. var que se imprime, la var ejemplar x está oculta en paint
```

```
            System.out println(( “la x local en paint es” + x ) ;
            //El programa define otros dos métodos que no reciben argumentos y no devuelven nada.
            a() ;      // a tiene una x local automática
            b() :      //  b usa la var de ejemplar x
            a();       // a reinicializa la x local automática
            b();       // la var de ejemplar x conserva su valor

            System.out.println( “\n la x local en paint es” + x);
      }

      void a()          //El método a define la  var automática x y la inicializa en 25.
      {
          int x = 25;           // se inicializa cada vez que se invoca a
          System.out.println (“\ nla x local en a es “ + x + “después de ingresar en a”);
          ++x;
          System.out.println ( “la x local en a es “ + x + “antes de salir de a”);
      }
      void b()          //El método b no declara ninguna var , entonces usa la var ejemplar  x.
      {
            System.out.println( “\nla variable de ejemplar x es “ + x + “al ingresar en b”);
            x *=10;
            System.out.println(“la var de ejemplar x es “ + x + “al salir de b”);
      }
}
```

Nota: El programa imprime la var local x de paint para demostra que ninguna de las llamadas del método modificó el valor de x porque todos los métodos hicieron referencia a variables de otros alcances.

4.11 Recursión.

Para resolver ciertos problemas resulta útil hacer que los métodos se invoquen a sí mismos. Es así que el método recursivo es un método que se invoca a sí mismo de forma ya sea directa o indirecta.
Si un método recursivo se invoca con un caso base, simplemente devuelve un resultado. Si el método se invoca con un problema más complejo, divide el problema en dos o más partes conceptuales: una parte que el método sabe cómo resolver y otra parte que el método no sabe cómo resolver. Dado que este nuevo problema se asemeja al problema original, el método emite una llamada recursiva para trabajar con el problema reducido.

Cuando el método reconoce el caso base, devuelve el resultado a la llamada de método previa, y se inicia una secuencia de devoluciones que termina cuando la llamada de método original devuelve el resultado final. La secuencia de problemas cada vez más pequeños debe convergir hacia el caso base. En este punto, el método reconoce el caso base, devuelve un resultado a la copia anterior del método y se inicia una secuencia de devoluciones en retroceso hasta que la llamada original del método devuelve el resultado final al invocador.

Ej. el factorial de un entero no negativo n, es decir “ n!” ó 5!

n! = n(n-1)(n-2)...1
5! = 5*4*3*2*1 = 120
5! = 5 (4!)
Este mismo factorial se puede calcular iterativamente:

```
factorial = 1
for ( int counter = number; counter >= 1; counter--)
    factorial *=counter;
```

```
5!                          = valor final = 120
 5 * 4!   |        5!= 5 * 24 = 120 valor devuelto
  4 * 3!  |        4!= 4 * 6= 24 valor devuelto
   3 * 2!         3!= 3 * 2= 6 valor devuelto
    2 * 1!        2!= 2 * 1= 2 valor devuelto
       1!              1 valor devuelto
Llamadas recursivas     Valores devueltos de
                              c/llamada
```

Por tanto, la iteración como la recursión se basan en una estructura de control; la iteración usa una estructura de repetición; la recursión usa una estructura de

selección también repetitiva, ya que logra la repetición mediante llamadas de método repetidas.

```
// Método recursivo factorial
Import java.awt.Graphics;
Import java.aaplet.Applet ;

public class FactorialTest extends Applet {
      public void paint ( Graphics g)
      {
            int yPosition = 25;
            for ( long  i = 0;  i <= 10;   i++ ) {
            g.drawString ( i + "!  = " +  factorial ( i  ), 25
            yPosition += 15;
            }
      }

      //  def. Recursiva del método factorial
      public long factorial ( long number )
      {
      if ( number <= 1 )      // caso base si number es 0 ó 1
            return 1;
      else
            return  number * factorial(  number  – 1 );  //llamada recursiva factorial
que evalúa
      }
}
```

Applet viewer:factorialTest

0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 10320
9! = 362880
10! = 3628800

Tanto la iteración como la recursión incluyen una prueba de terminación; la iteración termina cuando deja de cumplirse la condición para continuar el ciclo; la recursión termina cuando se reconoce el caso base.

Ocurre un ciclo infinito con la iteración si la condición para continuar el ciclo nunca deja de cumplirse; ocurre una recursión infinita si el paso de recursión no reduce el problema de modo tal que converja hacia el caso base.

La recursión invoca repetidamente el mecanismo de las llamadas de métodos, y por tanto incurre en su gasto extra. Esto puede ser costoso tanto en tiempo de procesador como en espacio de memoria.

4.12 Ejemplo de uso de recursión: La serie de Finobacci

0, 1, 1, 2, 3, 5, 8, 13, 21, .../// suma de los dos números de Fibonacci previos.
Calcula el i-ésimo número de Fibonacci recursivamene empleando el método de fibonacci.
Cada par de líneas de salida corresponde a una ejecución individual del programa.

```
// Método fibonacci recursivo
import java.awt.*;
import java.applet.Applet ;

public  class  FibonacciTest  extends
Applet {
  Label numLabel, resultLabel;
  TextField num, result;

  Public void init()
  {
     numLabel = new Label ("teclee un
entero y pulse enter");
    resultLabel = new Label ("El valor
de Fibonacci es");
    result = new TextField ( 15 );
    result.setEditable ( false );

    add( numLabel );
    add( num );
    add( resultLabel );
    add( result );
  }
```

```
Public boolean action ( Event e, Object o)
{
  long number, fobonacciVal;
  number = Long.parseLong( num.getText());

  showStatus( "Calculando….");
  fibonacciVal = fibonacci( number );
  shoStatus( "Listo.");

result.setText( Long.toString( fibonacciVal ) );
  return true;
}
     // Def. recursiva del método fibonacci
long fibonacci ( long n )
 {
     if ( n== 0  ó n == 1 ) // caso base
              return n;
     else
             return fibonacci( n – 1) + fibonacci
( n – 2 );
  }
}
```

- Un método que no devuelve un valor se declara con un tipo devuelto void. Si se intenta devolver un valor del método o utilizar el resultado de la invocación del método en el método invocador, el compilador informará un error.
- El calificador **final** crea variables constantes. Una variable constante debe inicializarse cuando se declara la variable, y no puede modificarse posteriormente. Las variables constantes también se conocen como constantes con nombre o variables solo de lectura.

4.14. Sobrecarga de métodos

Java permite definir varios métodos con el mismo nombre, pero estos métodos deben tener diferentes juegos de parámetros(los tipos y el orden de los parámetros).

Cuando se invoca un método sobrecargado, el compilador selecciona el método correcto examinando los argumentos de la llamada.

Los métodos sobrecargados pueden tener diferentes valores devueltos, y deben tener listas de parámetros distintas. Si dos métodos difieren únicamente en el tipo devuelto, el compilador informará un error.

```
Ej. //MethodOverload.Java (empleo de métodos sobrecargados)
Import java.awt.Graphics;
Import java.applet.Applet;
Public class MethoOverload extends Applet{
  Public void pain ( Graphics g )
  {
     g.drawString( “El cuadrado del entero 7 es” + square( 7 ), 25, 25 );
     g.drawString( “El cuadrado del double 7.5 es”  + square( 7.5 ), 25, 40 );
  }

  int square( int x )
  {
    return   x * x
  }
  double square( double y )
  {
    return  y * y;
  }
}
```

El compilador podría usar el nombre lógico “square de int y square de double para el método square y que especifica distinto parámetro.

Void foo(int a, flota b).

El compilador podría usar el nombre lógico foo de flot e int, sin embargo, el compilador considera distintos los dos métodos foo anteriores

4.15 Métodos de la clase Applet

Los métodos clave de la clase Applet se invocan automáticamente durante la ejecución de una applet. Estos métodos de Applet están definidos por la Java API

Ej. MethodOverload.Java (métodos sobrecargados con firmas idénticas y diferentes tipos devueltos.

```
Import java.awt.Graphics;
Import java.applet.Applet;

Public class MetrhodOverload extends Applet{
```

```
 int square( double x )
 {
        return   x * x
 }

 double square( double y )
 {
        return  y * y;
 }
}
```

El compilador lanza un mensaje de error generados por los métodos sobrecargados con listas de parámetros idénticas y tipos devueltos diferentes.

Los métodos resize(redimensionar) y repaint(repintar) también resulta de interés. Cuando se invoca una applet a través de una página HTML, la anchura y altura de la applet en pixeles están especifcads en el código HTML.

Ej. <html>

<applet code="Velcome.class" width=275 height=35>

</applet>

</html>

el tamaño de la applet se puede modificar durante la ejecución de la appplet invocando el método resize.

Ej. resize(200, 400);

Hace que la anchura de la applet sea de 200 pixeles, y su altura de 400 pixeles. Este enunciado se coloca en el método **init** de la applet.

El método **init** de una applet es invocado una vez por el Appletviewer o el navegador cuando se carga la applet para ejecutarse. Este método se encarga de inicializar la applet.

El método start (comenzar) de la applet se llama después de que el método init termina de ejecutarse y cada vez que el usuario del navegador regresa la página HTML en la que la applet reside (después de navegar en otra página HTML).

El método **paint** de la applet se invoca después de que termina la ejecución del método **init,** y una vez que ha comenzado la ejecución del método **start**, paradibujar en la applet. Este método también se invoca automáticamente cada vez que es necesario redibujar la applet.

El método **stop** (parar) de la applet se invoca cuando es preciso suspender la ejecución de la applet; esto sucede normalmente cuando el usuario del navegador abandona la página HTML en la que la applet reside.

El método **destroy** (destruir) de la applet se invoca cuando la applet se va a borrar de la memoria; esto normalmente sucede cuando el usuario del navegador sale de la sesión de navegación.

Ejercicios de autoevaluación

4.1 Llene los espacios en blanco.

2 Los módulos de programa en Java se llaman **métodos y clases.**

3 Un método se invoca con una **llamada de método.**

4 Una variable que se conoce solo dentro del método en el que se define es una **variable local.**

5 El enunciado **return** en un método invocado puede servir para devolver al método invocador el valor de la expresión.

6 La palabra clave **void** se incluye en una cabecera de método para indicar que el método no devuelve un valor.

7 El **alcance** de un identificador es la porción del programa en la que puede usarse el identificador.

8 Las tres formas de devolver el control de un método invocado son **return; o return expresión; o llegar a la llave derecha que cierra el método.**

9 El método **init** se invoca una vez cuando una applet inicia su ejecución.

10 El método **Math.random** sirve para producir números aleatorios.

11 El método **start** se invoca cada vez que el usuario de un navegador vuelve a visitar la página HTML en la que la applet reside.

12 El método **paint** se invoca para dibujar en una applet.

13 Se entiende que las variables declaradas en un bloque o en la lista de parámetros de un método tienen duración **automática.**

14 El método **resize** se invoca para modificar la anchura o la altura de una applet durante la ejecución de la applet.

15 El método **repaint** invoca al método update de la applet, el que a su vez invoca al método páint de la applet.

16 El método **stop** se invoca para una applet cada vez que el usuario de un navegador abandona la página HTML en la que la applet reside.

17 Un método que se invoca a sí mismo de forma ya sea directa o indirecta es un método **recursivo.**

18 Un método recursivo por lo regular tiene dos componentes: uno que provee una forma de terminar la recursión probando el caso **base** y otro que expresa el problema como una llamada recursiva para revolver un problema un poco más sencillo que el resuelve la llamada original.

19 En Java es posible tener varios métodos con el mismo nombre, cada uno de los cuales opera con diferentes tipos y/o números de argumentos. Esto se denomina **sobrecarga** de métodos.

20 Se emplea el calificador **final** para declarar variables sólo de lectura.

4.2 Para el siguiente programa, indique el alcance (ya sea de clase o de bloque) de cada uno de los siguientes elementos.

21 La variable x. //**alcance de clase**

22 La variable y. //**alcance de bloque**

23 El método cube. //**alcance de clase**

24 El método paint.//**alcance de clase**

25 La variable yPos.//**alcance de bloque**

```
Public class CubeTest extends Applet{
      Int x;
      Public void paint ( Graphics g )
      {
            int yPos = 25;

            for ( x = 1;  x <= 100;  x++ ) {
            g.drawString( cube( x), 25, yPos );
            yPos += 15;
            }
      }
      public int cube ( int Y )
      {
            return y * y * y ;
      }
}.
```

4.3 Escriba una aplicación que pruebe si los ejemplos de llamadas a **métodos de la biblioteca de matemáticas** que se muestran en la fig. 4.2 realmente producen los resultados indicados.

La siguiente solución demuestra los **métodos de la clase Math** de la fig. 4.2

```
// Ejercicio 4.3: MathTest.java
//Prueba de los métodos de la clase Math
```

```
public class MathTest{

 public static void main (String args[])
{
 System.out.println( "Math.abs( 23.7) = " + Math.abs( 23.7 ) );// = 23.7
 //(valor absoluto de x; si x > 0 => abs(x) es x;    //float, int y long
 System.out.println( "Math.abs( 0.0) = " + Math.abs( 0.0 ) ); //= 0
                          si x = 0 => abs(x) es 0;
 System.out.println( "Math.abs( -23.7) = " + Math.abs( -23.7 ) );// = 23.7
                          si x < 0 => abs(x) es –x)
 System.out.println( "Math.ceil( 9.2) = " + Math.ceil( 9.2 ) ); //= 10
 //redondear x al entero más pequeño no menor que x
 System.out.println( "Math.ceil( -9.8) = " + Math.ceil( -9.8 ) );// = -9
 System.out.println( "Math.cos( 0.0 ) = " + Math.cos( 0.0 ) );//= 1
 // coseno trigonométrico de x (x en radianes)
 System.out.println( "Math.exp( 1.0 ) = " + Math.exp( 1.0 ) );// = 2.71828
 System.out.println( "Math.exp( 2.0 ) = " + Math.exp( 2.0 ) );// = 7.38906
 //método exponencial e^x
 System.out.println( "Math.floor( 9.2 ) = " + Math.floor( 9.2) );// = 9
 System.out.println( "Math.floor( -9.8) = " + Math.floor( -9.8));// = -10
 //redondea x al entero más grande no mayor que x
 System.out.println("Math.log(2.718282)="+ Math.log(2.718282));// = 1
 System.out.println("Math.log(7.389056)="+ Math.log(7.389056));// = 2
 //logaritmo natural de x (base e)
 System.out.println( "Math.max(2.3, 12.7)="+Math.max(2.3, 12.7));// = 12.7
 System.out.println("Math.max(-2.3, -12.7)="+Math.max(-2.3,-2.7));// = -2.3
 //Valor mayor de x y y
 System.out.println("Math.min(2.3, 12.7)="+Math.min(2.3,  2.7));// = 2.3
 System.out.println(" Math.min(-2.3, -12.7)= " + Math.min(-2.3, -12.7));// = -12.7 //Valor menor de x y y
 System.out.println("Math.pow(2,  7)=" + Math.pow(2, 7));// = 128
 System.out.println("Math.pow(9, .5)=" + Math.pow(9, .5));// = 3
 // x elevado a la potencia y  x^y
 System.out.println("Math.sin(0.0)="+ Math.sin(0.0 ));// = 0
 //seno trigonométrico de x (x en radianes)
 System.out.println("Math.sqrt(25.0)="+ Math.sqrt(25.0));// = 5
 //raiz cuadrada de x
 System.out.println("Math.tan(0.0)=" + Math.tan(0.0));// = 0
 //tangente trigonométrica de x (xen radianes)
 }
}
```

Nota: abs(x), ceil(x), cos(x), exp(x), floor(x), log(x), max(x, y), min(x, y), pow(x, y), sin(x), sqrt(x), tan(x)

4.4 Escriba la cabecera de método de cada uno de los siguientes métodos.

2 Método hipotenusa que recibe dos argumentos de punto flotante de doble precisión, cateto1 y cateto2 y devuelve un resultado de punto flotante de doble precisión.
double hypotenuse(double side1, double side2)

3 Método menor que tome tres enteros x, y, z, y devuelva un entero.
Int menor(intx, int y, int z).

4 Método instructions que no reciba argumentos y no devuelva ningún valor.(Nota. Es común usar este tipo de métodos para mostrar instrucciones al usuario.)
Void instrucciones()

5 Método intToFloat que recibe un argumento entero, number, y devuelve un resultado de punto flotante.
float IntToFloat(int number)

4.5 Encuentre el error en cada uno de los siguientes segmentos de programa y explique cómo puede corregirse el error.

2
```
int g(){
   System.out.println( "Método interior g" );
   Int h(){
   System.out.println( "Método interior h" );
   }
}
```
Error: El método h se define dentro del método g.
Corrección: Saque la definición de h de la definición de g.

3
```
int sum( int x, int y ) {
   int result;
```

```
      result = x + y;
   }
```
Error: se supone que el método devuelve un entero, pero no lo hace.
Corrección: elimine la variable resultado y coloque el siguiente enunciado en el método:
```
return x + y;
```
```
   int sum( int x, int y ) {
         return x + y;
4  int sum( int n ) {
   if ( n == 0 )
      return 0;
   else
      n + sum(n-1);
   }
```
Error: El resultado de n + suma(n-1) no se devuelve; suma devuelve un resultado inapropiada.
Corrección: Rescriba el enunciado de la cláusula else así: return n + sum(n – 1);
```
   int sum( int n ) {
     if ( n == 0 )
          return 0;
     else
         return n + sum(n-1);
   }
5  void f( float a ); {
      float a;

      System.out.println( a );
   }
```
Error: Signo de punto y coma después del paréntesis derecho que encierra la lista de parámetros, y redefinir el parámetro **a** en la definición del método.
Corrección: Quite el punto y coma después del paréntesis derecho de la lista de parámetros y elimine la declaración float a;
```
   void f( float a ) {

      System.out.println( a );
   }
6  void product() {
      int a = 6, b=5, c=4, result;
      result=a*b*c;
      Sustem.out.println( “El resultado es”  + result);
      return result;
   }
```
Error: El método devuelve un valor y no se supone que lo haga.
Corrección: Elimine el enunciado **return.**
```
   void product() {
      int a = 6, b=5, c=4, result;
      result=a*b*c;
      System.out.println( “El resultado es”  + result);
}
```

4.6 Escriba un programa completo en Java que solicite al usuario el **radio double de una esfera** y llame al **método volumenEsfera** para calcular y exhibir el **volumen de esa esfera** empleando la asignación:
```
      volume = ( 4/3)* Math.PI * Math.pow( radio, 3)
```
El usuario deberá introducir el radio a través de un campo de texto. Podemos convertir una cadena en un valor double como sigue (suponga que o es el String que se pasa al método action cuando el usuario oprime la tecla Enter en el campo de texto):
```
// crear un objeto Double usando el valor del campo de texto.
Double val = new Double( o.toString() );
//Extraer el valor de tipo de datos primitivo double del objeto  Double.
      double radio = val.doubleValue();
// El resultado se puede convertir en un String con el método Double.toString.
```
La sigte. solución **calcula el volumen de una esfera** usando el radio introducido por el usuario.
```
//Ejercicio 4.6: PruebaEsfera.java

import java.applet.Applet;
import java.awt.* ;

public class PruebaEsfera extends Applet{
```

```
            Label solicitud;
            TextField entrada;

            Public void init()
            {
                  prompt = new Label( “Teclee radio de esfera:”);
                  input = new TextField( 10 );
                  add( solicitud );
                  add( entrada );
            }

            public boolean action (Event e, Object o)
            {
                  Double val = new Double(o.toString() );
                  Double radius = val.doubleValue();
                  ShowStatus( “El volumen es ”
                  + Double.toString(sphereVolume(radio))) ;
                  Return true ;
            }


            public double volumenEsfera( double radio )
            {
                  double volumen;
                  volume = (4/3)*Math.PI * Math.pow( radio, 3 );
                  return volume;
            }
      }
```

4.7 Indique el valor de x después de ejecutarse cada uno de los siguientes enunciados:

```
2   x = Math.abs(7.5 );// x =7.5
3   x = Math.floor(7.5); // x = 7
4   x = Math.abs(0.0); // x = 0
5   x = Math.ceil(0.0); // x = 0
6   x = Math.abs(-6.4); // x = 6.4
7   x = Math.ceil(-6.4); // x = -6
8   x = Math.ceil(-Math.abs(-8 + Math.floor( -5.5))); // x = - 14
```

ARREGLOS

- Presenta una estructura de datos.
- Se emplea para almacenar, ordenar y examinar listas y tablas de valores.
- Sus elementos son del mismo tipo, sus posiciones de memoria son contiguas
- Son entidades estáticas en cuanto a su tamaño.
- El 1er. elemento del arreglo se lee así:

c[0], c[1], c[i-1]

nombre del arreglo: similar a una variable.
Nro. de posición del elemento: subíndice.

- Un nombre de arreglo con subíndice es un **Ivalue**
- La longitud se determina con la sigte. expresión:

c.length

nombre del arreglo — longitud

Todo arreglo en Java conoce su propia longitud.

En Java un arreglo:
- **se declara**
- **se inicializa**

Para declarar:	**Para asignar espacio de almacenamiento al Nro. de elem requeridos por cada arreglo:**
- Se especifica **tipo de elementos**	- Se usa el operador **new**

int c[ ] = new int[12]; ó

int c[ ]; //declara el arreglo de max elem = 12
c[ ] = new int[12]; //asigna memoria al arreglo

- Los elementos se **inicializan automáticamente a cero** en caso de variables de tipos primitivos numéricos, a false en el caso de boolean o a null en el caso de referencias.

- se pasa arreglos a métodos.

- Se Usa técnicas de ordenamiento básicas.
- Se manipulea con múltiples subíndices.

CAP. 5 ARREGLOS

Objetivos:

- Presentar la estructura de datos de arreglo
- Entender el empleo de arreglos para almacenar, ordenar y examinar listas y tablas de valores.
- Entender la forma de declarar un arreglo, inicializar un arreglo a elementos individuales de un arreglo.
- Poder pasar arreglos a métodos.
- Entender las técnicas de ordenamiento básicas.
- Poder declarar y manipular con múltiples subíndices.

5.1. Introducción:

- Los arreglos son estructuras de datos que consisten en elementos de información del mismo tipo relacionados entre sí.
- Los arreglos son entidades estáticas en cuanto a su tamaño.

5.2. Arreglos

- Un arreglo es un grupo de posiciones de memoria contiguas, ya tienen el mismo nombre y el mismo tipo.
- Para referirnos a una posición(elemento) , especificamos el nombre del arreglo y el número de posición del elemento.
- Nos referimos así: primer elemento del arreglo: c[0], segundo elem c[1] ,... y en general al i-esimo elem del arreglo c[i-1]**.**
- El nombre de un arreglo es similar a una variable.
- El número de posición recibe el nombre de subíndice.
- La expresión se evaluará para determinar el subíndice.
- Un nombre de arreglo con subíndice es un **Ivalue,** se puede escribir en lado izquierdo de un asignación para colocar un nuevo valor en un elem de un arreglo.
- La longitud del arreglo se determina con la siguiente expresión: **c.length**// nombre del arreglo y la longitud.
- Todo arreglo en Java conoce su propia longitud.

5.3. Declaración y Asignación de almacenamiento a arreglos

- Los arreglos ocupan espacio en la memoria.
- Se **especifica el tipo de los elementos** y **se usa** el operador **new** para asignar espacio de almacenamiento al número de elementos requeridos por cada arreglo. Los **elementos se inicializan** automáticamente a **cero** en caso de var. de tipos primitivos numéricos, a **false** en el caso de boolean o a **null** en el caso de referencias. Usamos la declaración:

int c[ ] = new int[12]; ó int c[ ]; //declara el arreglo de max elem=12

c[ ] = new int[12]; //asigna memoria al arreglo

5.4. Ejemplos del uso de arreglos

El operador **new** es para asignar dinámicamente espacio de almacenamiento a un arreglo de 10 elementos.

```
// Ej. 1   Inicialización de los elementos de un arreglo en ceros
import java.awt.Gaphics;
import java.applet.Applet;

public class Initarray extends Applet {
    int n[ ]; // declarando un arreglo de enteros – tamaño del arreglo

    //  inicializar variables de ejemplar
    public void init( )
    {
        n = new int[ 10 ]; // asignar memoria dinámicamente al arreglo
    }
    // pintar la applet
    public void paint( Graphics g )
    {
        int yPosition = 25 ; // posición y inicial de la applet

        g.drawString( "Elemento", 25, yPosition );//enunciado exhibe la cabecera
        g.drawString( "Valor", 100, yPosition );   //     "                "

        for ( int i = 0;  i< n.length;  i++ )
        {
              yPosition += 15;
              g.drawString( String.valueOf (i  ), 25, yPosition );
              g.drawString(String.valueOf, ( n[ i ] ), 100,  yPosition );.
        }
    }
}
```

```
Elemento valor
   10    0
   11    0
   12    0
   13    0
   14    0
   15    0
   16    0
   17    0
   18    0
   19    0
```

La var **yPosition** sirve para determinar la **posición vertical** en la que el método drawString dibujará en la applet.
El **método String.valueOf convierte cada entero en una cadena que se pueda exhibir en la applet.**
n.length determina la longitud del arreglo.

```
//Ej. 2  Inicialización de arreglo enteros con 10 valores e imprime el arreglo en forma tabular.(con una declarac.)
import java.awt.Gaphics;
import java.applet.Applet;

public class Initarray extends Applet {
    int n[ ] = {32, 27, 64, 18, 95 14, 90, 70, 60, 37};

    // pintar la applet
    public void paint( Graphics g )
    {
        int yPosition = 25 ; // posición y inicial en la applet

        g.drawString( "Elemento", 25, yPosition );//enunciado exhibe la cabecera
        g.drawString( "Valor", 100, yPosition );   //     "                "

        for ( int i = 0;  i< n.length;  i++ )
        {
              yPosition += 15;
              g.drawString( String.valueOf (i  ), 25, yPosition );
              g.drawString(String.valueOf, ( n[ i ] ), 100,  yPosition );.
```

```
Elemento valor
   10    32
   11    27
   12    64
   13    18
   14    95
   15    14
   16    90
   17    70
   18    60
   19    37
```

```
        }
    }
}

//Ej. 3 inicializar arreglo s con enteros pares de  2 a 20
    import java.awt.Gaphics;
    import java.applet.Applet;

    public class Initarray extends Applet {
        final int  arraySize = 10;          // inicializa los elem. de un arreglo  s de 10 elem con los enteros 2, 4, 6 … 20;

        //utiliza el calificador final para declarar una var constante arraySize (tamaño de arreglo) cuyo valor es 10.
        Las var. const. se deben inicializar con una expresión constante cuando se declaran y no se pueden modificar.
        Si intenta modificar una var. final después de haberla declarado, el compilador producirá el mensaje de error
        (cant't assign a value to a final variable)..
                int s[ ];

        //  inicializar variables de ejemplar
        public void init( )
        {
            s = new int[ arraySize ];
        //estos números se generan multiplicando cada valor sucesivo del contador del ciclo por 2  y sumándolo 2
        //  establecer los valores del arreglo

            for ( int i = 0 ; i <= n.length;  i++ ).
                s[ i ] = 2 +2 * i ;
        }
        // pintar la applet
        public void paint( Graphics g )
        {
            int yPosition = 25 ; // posición y inicial de la applet

            g.drawString( "Elemento", 25, yPosition );//enunciado exhibe la cabecera
            g.drawString( "Valor", 100, yPosition );   //     "                "

            for ( int i = 0;  i< n.length;  i++ )
            {
                yPosition += 15;
                g.drawString( String.valueOf (i  ), 25, yPosition );
                g.drawString(String.valueOf, ( s[ i ] ), 100,  yPosition );.
            }
        }
    }
```

```
Elemento valor
   10 2
   11 4
   12 6
   13 8
   14 10
   15 12
   16 14
   17 16
   18 18
   19 20
```

```
Ej. 4  Empleo de una var constante debidamente inicializada
Import java.awt.Graphics;
Import java.applet.Applet ;

Public class FinalTest extends Applet
{
        final int x = 7;  //inicializar la var constante

        public void paint( Graphics g )
        {
        g.drawString( "El valor de x es: " + x, 25,  25 );
        }
```

```
}
```

Ej.5 Calculo de la suma de los elementos de un arreglo

```
Import java.awt.Graphics;
Import java.applet.Applet ;

Public class SumArray extends Applet
{
        int a[ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int total;

        //inicializar variables de ejemplar
        public void init( ).
        {
        total = 0;
        for ( int i = 0;   i < a.length;  i++ )
                        total += a [ i ];
        }
        //pinta la applet
        public void paint( Graphics g )
        {                                                          55
        g.drawString( "Total de los elementos del arreglo:"  + total,  25,  25 );
        }
}
```

Se pidió a 40 estudiantes calificar la calidad de la comida que se sirve en la cafetería en una escala de 1 a 10 (1 denota horrible y 10 excelente) . Coloque las 40 respuestas en un arreglo de enteros y resuma los resultados del sondeo.

Ej.6 Programa de sondeo de 40 estudiantes//numero de respuestas(arreglo **responses)** de cada tipo(de 1 a 10)

```
Import java.awt.Graphics;
Import java.applet.Applet ;
```

Calificación	frec
11	2
12	2
13	2
14	2
15	5
16	11
17	5
18	7
19	1
20	3

```
Public class StudentPoll extends Applet
{               arreglo que contiene las repuestas de 40 elem (estud.)
        int responses [ ] = {  1, 2, 6, 4, 8, 5, 9, 7, 8, 10
                               1, 6, 3, 8, 6, 10, 3, 8, 2, 7
                               6, 5, 7,  6, 8, 6, 7, 5, 6, 6,    };
                               5, 6, 7, 5, 6, 4, 8, 6, 8, 10
        int frec [ ];

      //inicializar variables de ejemplar
        public void init( ).
        //frec. arreglo de 11 elem. para contar el número de ocurrencias de c/respuesta.

            frec = new  int [ 11 ] ;//los elem. del arreglo frec se inicializan automát. en cero, se asigna memoria con new

                                                  long. 40
            for ( int answer = 0;   answer <  response.length; answer++ )//for toma las respuest. una a una del arreglo
response

                ++frec[ responses [ answer ] ];
                          0, 1, 2, 3, ….40
                      1, 2, 6, 4…10// increm 6 del arreglo
                  1, 2, 3, 4.....11

            // incrementa el contador dependiendo del valor de response[answer]  es frec [ 1 ] y  es ++frec[1]
        }
```

answer	response[answer]	++frec[response[answer]]
0	1	++frec[1]
1	2	++frec[2]
2	6	++frec[6]
3	4	++frec[4]

Calificación	frecuencia]
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

```
        //pintar la applet
        public void paint( Graphics g )
        {
                int yPosition = 25; //posición inicial en la applet

        g.drawString( "Calificación!", 25, yPosi tion);
                g.drawString( "frec",100, yPosi tion);

        for (int rating = 1;  rating < frec.length;  rating ++ )
        {
                yPosition += 15;
                g.drawString( String.valuof( rating ), 25, yPosition );
                        g.drawString( String.valuof( frec[ rating]  ), 100, yPosition );
        }
        }
}
```

Java cuenta con mecanismos para evitar el acceso a elementos fuera de los límites del arreglo, al compilar, verifica las referencias a elementos de arreglos para asegurarse de que sean válidas.
Si se hace una referencia no valida a un elem de un arreglo en el momento de la ejecución java genera una excepción. (ArrayIndexOutOfBoundsException -índice de arreglo fuera de límite).

Ej. 7. lee números de un arreglo y grafica la información en forma de graficas de barras o histograma, se imprime cada número y junto a el se imprime una barra formada por esa misma cantidad de asteriscos. El ciclo for anidado se encarga de dibujar las barras.

```
Import java.awt.Graphics;
Import java.applet.Applet ;

Public class Histogram extends Applet
{
        int n[ ] = {19, 3, 15, 7, 11, 9, 13, 15, 17, 1 };

        //pintar la applet
        public void paint( Graphics g )
        {
        int xPosition; //posición de * en el histograma
        int yPosition = 25; //posición vertical en la applet

                g.drawString( "Elemento ", 25, yPosition );
                g.drawString( "Valor", 100, yPosition );
                g.drawString( "Histograma", 175, yPosition );
```

Elemento	Valor	Histograma
0	19	**********
1	3	***
2	15	**********
3	7	*******
4	11	**********
5	9	*********
6	13	**********
7	5	*****
8	17	********** *
9	1	*

```
            for ( int  i  = 0;  i  <  n.length;  i++)
            {
                yPosition += 15;
                g.drawString( String.valueOf( i ), 25, yPosition );
                g.drawString(String.valueOf( n [ i ] ),  100, yPosition );.

                XPosition = 175;

                El ciclo  for anidado  se encarga de dibujar las barras.
                for ( int  j  = 0;  j  <  n[ i ];  j++ ) // imprimir una barra
                {
                      g.drawString ( " * ", xPosition, yPosition);
                      xPosition  += 7;
                }
            }
        }
}
```

5.5. Referencias y parámetros de referencia

- Dos **formas de pasar argumentos a los métodos**, en C y C++ son llamar **por valor** y llamar **por referencia**. Cuando se pasa un **argumento por valor** , **se crea una copia de valor del argumento y se pasa al método invocado**. Por **referencia, el invocador confiere al método invocado la capacidad de acceder directamente a los datos del invocador y modificar dichos datos** si el método invocado lo desea.

- En Java no permite escoger cómo quiere pasar los argumentos, si por valor o por referencia. **Las vars de tipos de datos primitivos siempre se pasan por valor y los objetos siempre se pasan por referencia.**.
- Para **pasar un objeto por referencia** especificar **en la llamada de método la referencia al objeto por su nombre**.
- **Java trata a los arreglos como objetos**, los **arreglos se pasan a los métodos por referencia:** un método invocado puede acceder a los valores de los elementos en los arreglos originales de los innovadores.
- El **nombre de un arreglo es una referencia a un objeto** que contiene los elementos del arreglo y la **variable de ejemplar length** que indica el número de elementos en el arreglo.
- Al devolver información de un método mediante un enunciado **return** las variables de tipos de datos primitivos siempre se devuelve por valor y los objetos siempre se devuelven por referencia.

`Ej. Tirar un dado de seis caras 6000 veces para probar si el generador de números aleatorios producía números en verdad aleatorios. Empleando arreglos en vez de switch`

```
//RollDie.java – Tirar un dado de seis caras 600 veces

Import java.awt.Graphics;
Import java.applet.Applet;
Import java.util.Random;

Public class RollDie extends Applet
{
        int face;  //cara
        int frec [ ];
        Random r; // crear el generador de números aleatorios


        //inicializar variables de ejemplar
        public void init( ).
        {
```

```
        frec  = new int [ 7 ];.
                r = new Random  ();

        for ( int roll =1;  roll < = 6000;   roll++ ){
                        face = 1 + Math.abs( r.nextInt() % 6 );
                        ++ frec[ face ];
                }
        }
        //pintar la applet
        public void paint( Graphics g )
        {
        int yPosition = 25;
                g.drawString( "cara"   25, yPosition );
                g.drawString( "Frecuencia", 100, yPosition );

                for ( face = 1; face < frec.length; face++ ){
                yPosition += 15;
                        g.drawString( String.valueOf( face ), 25, yPosition );
                        g.drawString(String.valueOf( frec [ face] ),  100, yPosition );.
                }
        }
}
```

1	1030
2	1049
3	977
4	973
5	988
6	983

5.6. Cómo pasar arreglos a los métodos

Si quiere pasar un argumento de arreglo a un método, especifique el nombre del arreglo sin los corchetes.
Ej. si se declaró el arreglo como:

Int **temperaturasHorarias[ ] = new int [24];**

La llamada del método:

ModificarArreglo(**temperaturasHorarias**); .// método y arreglo

En **JAVA** todo objeto de arreglo conoce su propio tamaño (por la var ejemplar length). Cuando pasamos un objeto de arreglo a un método, no necesitamos pasar el tamaño del arreglo como argumento.

Los arreglos enteros se pasan por referencia; es decir, los elementos de arreglo individuales que no son de tipos primitivos) a estos elementos se denominan escalares o cantidades escalares.

Los elementos de arreglo individuales de tipos de datos primitivos se pasan por valor..

Al pasar un elemento de arreglo a un método usamos el nombre con subíndice del elemento de arreglo como argumento en la llamada de método.

Para que un método reciba un arreglo a través de una llamada de método, la lista de parámetros del método debe indicar que se recibirá un arreglo. Ej.

Void modificarArreglo(int b[ ])

Modificar arreglo espera recibir un arreglo entero en el parámetro b. **Dado que**, los arreglos se pasan por referencia, cuando el método invocado utiliza el nombre de **arreglo b.,** se refiere al arreglo real en el invocador (el arreglo temperaturaHorarias en la llamada anterior)

El ejemplo demuestra la diferencia entre **pasar un arreglo completo y pasar un elemento de arreglo.**
Primero imprime los cinco elementos del arreglo **enteros a.** Luego se pasa a al **método modifyArray**(modificar arreglo) donde cada uno de los elementos de **a** se multiplica por 2. Después, **a** se vuelve a imprimir en **main**. ModiffyArray modifica los elementos de **a.** Luego imprime el valor de a[3] y lo pasa al **método modifyElemen**t(modificar elemento). Este método multiplica su argumento por 2, cuando a[3] se vuelve a imprimir en **main**, no

ha sufrido modificación porque los elementos del arreglo individuales de tipos de datos primitivos se pasan por **valor.**

```
//PassArray.java
//paso de arreglos y elementos individuales de arreglos a métodos
Import java.awt.Graphics;
Import java.applet.Applet;

public class PassArray extends Applet
{
        int a [ ] = { 0, 1, 2, 3, 4};

        //pintar la applet
        public void paint( Graphics g )
        {
        int yPosition = 25, yPosition = 25;
                g.drawString( “efectos de pasar  todo el arreglo por referencia” , xPosition,  yPosition );
                yPosition += 15;
                g.drawString( “Los valores del arreglo original son”, xPosition,  yPosition );
                xPosition += 15;
                yPosition += 15;

                for ( int i = 0;  i < a.length; i++ ){
                g.drawString( String.valueOf( a[ i ] ), xPosition, yPosition );
                        xPosition += 15;
                }
                xPosition = 25;
                yPosition += 30;

                modifyArray( a );          // el arreglo a se pasa por referencia
                g.drawString( “Los valores del arreglo modificados son”, xPosition,  yPosition );.
                xPosition += 15;
                yPosition += 15;

                for ( int i = 0;  i < a.length; i++ ){
                g.drawString( String.valueOf( a[ i ] ), xPosition, yPosition );
                        xPosition += 15;
                }
                xPosition = 25;
                yPosition += 30;

                g.drawString( “efectos de pasar  un elemento de un arreglo por valor” , xPosition,  yPosition );
                yPosition += 15;
                g.drawString( “a[3] antes de modifyElement:” + a [ 3 ], xPosition,  yPosition );
                xPosition = 25;
                modifyElement(  a[3 ] ) ;
                g.drawString( “a[3] después de modifyElement:” + a [ 3 ], xPosition,  yPosition );

        }

        public void modifyArray( int b[ ] )
        {
        for ( int j = 0 ; j < b.length; j++)
                        b[ j ] *= 2;
        }
```

```
Efectos de pasar todo el arreglo  pro referencia:
Los valores del arreglo original son:
0 1 2 3 4
Los valores del arreglo modificados son:
0 2 4 6 8
Efectos de pasar un elemento de una arreglo por valor :
A[3] abantes de modifyElemnto: 6
A[3] después de modifyElement: 6
```

```
        public void modifyElement( int e)
        {
        e *= 2;
        }
}
```

5.7. Ordenamiento de arreglos

`Ordenar es colocar los datos en algún orden específico(ascendente o descendente).`

Ej. Ordenar los valores del arreglo de 10 elementos **a** en **orden ascendente**.

Técnica a usar: ordenamiento de burbuja u ordenamiento descendente. Los valores más pequeños suben hacia el principio del arreglo, como las burbujas del aire ascienden en el agua, mientras que los valores más grandes se hunden o descienden hasta el final del arreglo. Para ello se realizará varias pasadas por el arreglo. **En cada pasada se comparan pares de elementos sucesivos.** Si un par está en orden creciente (o los valores son idénticos), dejamos los valores tal como están. Si un par está en orden decreciente, intercambiamos sus valores en el arreglo.

```
import java.awt.Graphics;
import java.applet.Applet ;
public class BubbeSort extends Applet
{
        int a[ ] = {2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

        int hold; // area de retención temporal para el intercambio

        public void paint ( Graphics g)
        {
                print( g, "Datos en el orden original", a, 25, 25 );
                sort();
                print( g, "Datos en el orden ascendente", a, 25, 55);
        }
        public void sort( )
        {//for anidado
        for ( int pass = 1; pasa < a.length; pass++ ) // pasadas
                for ( int i = 0;  i < a.length – 1;  i++) // una pasada
                                if ( a[ i ] > a[ i  + 1 ]) //compara a[0] con a[1], luego a[1] con a[2]…hasta completar la
                        pasada,
                                solo realiza 9 comparaciones.
                                El valor grande baja varias posiciones en una sola pasada, valor pequeño sube solo una
                                posición.
                                1ª pasada el valor grande se hunde hasta el último elemento, a[9], en la 2ª hundirá hasta
                                a[8], en la 9ª pasada, el noveno valor más grande se hunde a a[1], Esto deja al valor
                                más pequeño con a[ 0 ], de modo que sólo se necesitan nueve pasadas para ordenar un
                                arreglo de 10 elementos.
                                {// intercambiar dos valores, si es posible
                                hold = a[ i ];
                        //var adicional  hold(retener) almacena temporalmente uno de los dos valores que se
                        intercambian.
                                a[ i ] = a[ i  + 1];
                                        a[ i + 1 ] = hold;
                                }
        }
        public void print ( Graphics g,  String head, int b [ ], int x, int y )
        {
```

```
Datos en el orden original
2 6 4 8 10 12 89 68 45 37
Datos en el orden ascendente
2  4 6  8 10 12 37 45 68 89
```

```
        g.drawString( head, x, y );
                x += 15;
                y += 15;
                for ( int i = 0;  i < b.length;  i++)
                {
                g.drawString( String.valueof( b[ i  ] ), x,  y );
                x += 20;
                }
        }
}
```

Ordenamiento burbuja es fácil de programar, pero se ejecuta con lentitud.

5.8. Búsqueda en arreglos: Búsqueda lineal y Búsqueda binaria.

- Determinar si un arreglo contiene un valor que coincide con cierto **valor clave.**
- El proceso de encontrar un elemento de un arreglo se llama **búsqueda.**
- **El método de búsqueda lineal funciona con arreglos pequeños y no ordenados.**
- **Búsqueda lineal:** compara cada elemento del arreglo con la clave de búsqueda. Al comparar la clave de búsqueda con la mitad de los elementos del arreglo, en promedio.
- **Para el método de búsqueda binaria el arreglo debe estar ordenado, ya que es de alta velocidad .**
- El **algoritmo de búsqueda binaria elimina de la búsqueda la mitad de los elementos del arreglo** después de cada comparación. Si son iguales, se habrá encontrado la clave y se devolverá el subíndice del elemento en cuestión; si no, el problema se reduce a buscar en la mitad del arreglo. La búsqueda continua hasta que la clave
- En el peor de los casos, una búsqueda en un arreglo de **1024 elementos requiere sólo de 10 comparaciones** empleando búsqueda binaria.
- La división repetitiva 1024/2 después de cada comparación, podemos eliminar la mitad del arreglo, produce los valores 512, 256, 128, 64, 32, 16, 8, 4, 2 y 1. El número 1024 (2^{10}) se divide entre 2 sólo 10 veces para obtener el valor 1.
- Un arreglo de 1.048..576 (2^{20}) elementos requiere un máximo de 20 comparaciones para encontrar la clave

```
// búsqueda binaria de un arreglo = BinarySearch.java

import java.awt.* ;
improt java.applet.Applet ;
public class BinarySearch extends Applet
{
        int a[ ];
        int element;
        String searchKey; //clave de búsqueda
        Int xPosition; // posición horizontal pra dibujar en la applet
        Int yPosition; // posición vertical para dibujar en la applet
        Label enterLabel;
        TextField enter;
        Label resultLabel;
        TextField result; // campo de texto para el resultado
        Bolean timeToSearch = false; // hora de buscar

        Public void init ( )
        {
        a = new int [ 15   ];
                for ( int i = 0; i < a.length;  i++ ) // crear los datos
```

```
                a [ i ] = 2 * i;

        enterLabel = new Label ("Tecle la clave" );
        enter = new TextField ( 5 );
        resultLabel = new Label ( "Resultado " );
        result = new TextField ( 22 );
        result.setEditable( false );
        add( enterLabel );
        add( enter );
        add( resultLabel );
        add( result );
}

public void paint ( Graphics g )
{
if ( timeToSearch ) // evita buscar en la 1ª invocación
        {
        element. BinarySearch( Integer.parseInt ( searchKey ), g );
                if ( element != - 1 )
                        result.setText( "valor hallado en elemento " + element );
                else
                        result.setText( "No se halló el valor ");

        }
}

public bolean action ( Event e, Object o )
{
if ( e.target == enter)
        {
        timeToSerarch = true;
                xPosition = 25;
                yPosition = 75;
                searchKey = e.arg.toString( );
                repaint (); // llamar a paint para iniciar búsqueda y salida.
        }
        return true;.
}

//Búsqueda binaria
public int  binarySearch ( int key, Graphics gg)
{
gg.drawString ( "proporciones del arreglo en que se buscó", xPositicon, yPosition );
yPosition += 15;

int low = 0; // subíndice bajo
int high = a.length – 1; // subíndice alto
int middle;       // subíndice medio

while ( low <= high )
        {
        middle = ( low + high ) / 2;

        printRow( low, middle, high, gg );
        if ( key == a[ middle ] ) // se encontró
```

```
                return middle;
        else if ( key < a[ middle ])
                        high = middle –1 // buscar parte baja del arreglo
                else
                        low = middle + 1; // buscar parte alta del arreglo
        }
        return – 1; // no se encontró searchKey
    }
    // imprimir una fila de salida mostrando la parte del arreglo que se está procesando ahora

    void printRow ( int low, int mid, int high, Graphics gg)
    {
    xPosition = 25;

        for ( int i = 0; i < a.length;  i++ ) //
        {
            if ( i < low       i > high )
            gg.drawString( “  “,  xPositicon, yPosition );
            else if ( i == mid ) // marcar  valor medio
            gg.drawString( String.valueOf ( a [ i  ] + “ * ”, xPosition, yPosition );
            else
            gg.drawString( String.valueOf ( a [ i  ] ,  xPosition, yPosition );

        xPosition += 20;
        }
        yPosition += 15;
    }
}
```

5.9. Arreglos con múltiples subíndices.

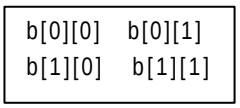
- Las tablas o arreglos requieren dos subíndices (filas, columnas), para identificar un elemento (el 1ro. identifica la fila del elem, y el 2do. la columna del elem).
- Java no maneja arreglos de múltiples subíndices.
- Un arreglo con doble subíndice b[2] [2] se podría declarar e inicializar con:

```
int b[ ] [ ] = { ( 1, 2 ), ( 3, 4 )};
```

- Los valores se agrupan por fila encerrados en llaves.

Así, 1 y 2 inicializan b[0] [0] y b[0] [1] y

3 y 4 inicializan b[1] [0] y b[1] [1]



.

```
//InitArray.java
//Inicialización de arreglos multidimensionales
import java.awt.Graphics;
import java.applet.Applet ;

public class InitArray extends Applet
```
{//la declaración array1 proporciona **seis inicializadores en dos sublistas** . La 1ra. Sublista inicializa la primera fila del arreglo con los valores 1, 2 y 3; la 2da. Sublista inicializa la 2da. fila del arreglo con los valores 4, 5 y 6.

```
    int array1[ ] [ ] = { ( 1, 2, 3 ), (4, 5, 6 ) };
```

//la declaración array2 proporciona **tres inicializadores en dos sublistas** . La 1ra. Sublista inicializa la primera fila del arreglo con dos elementos, con los valores 1, 2 ; la 2da. Sublista de la 2da. fila inicializa con un elemento con el valor 4.

```
    int array2[ ] [ ] = { ( 1, 2 ), (4 ) };

    //pintar la applet
    public void paint ( Graphics g)
    {
        g.drawString( « Los valores de array1 por fila son », 25, 25 ) ;
        printArray(array1, g, 40 );  //imprimir arreglo, exhibe los elems

        g.drawString( « Los valores de array2 por fila son », 25, 70 ) ;
        printArray(array2, g, 85 );
    }
    public void printArray(  int a[ ] [ ], Graphics g, int y ) //método printArray
```

```
Lo a[ i ].length valores de array1 por fila
son
1 2 3
4 5 6
los valores de array2 por fila son
1 2
```

//La def. del método especifica el parámetro de arreglo como int a[ ] [ ] para indicar que se recibirá un arreglo con doble subíndice como argumento. Se pasa una referencia Graphics como argumento para que el método (printArray) pueda exhibir el contenido de cada arreglo en la applet.

```
    {
        int x = 25 ;

        for ( int i = 0 ; i < a.length ; i++ ) //for anidada para exhibir las filas de cada arreglo con doble subíndice
        {//a.length determina el número de filas
                for ( int j = 0 ; j < a[ i ].length ;  j++ )
                {// a[ i ].length determina el número de columnas de cada fila del arreglo
                        g.drawString(  String.valueof( a[ i ] [ j ], x,  y  );
                        x += 15;
                }
                x = 25;
                y += 15;.
        }
    }
}
```

Ej. For asigna cero a todos los elems. de la 3ra fila del arreglo

```
        for ( int col = 0 ; col < a[ 2 ].length ; col++ )
                a[ 2 ] [ col ] = 0;
```

```
a [ 2 ][ 0 ]= 0;
a [ 2 ][ 1 ]= 0;
a [ 2 ][ 2 ]= 0;
a [ 2 ][ 3 ]= 0;
```

Ej. for anidada calcula la sumatoria de todos los elementos del arreglo **a**. for suma los elementos fila por fila

```
        total = 0;
        for ( int row = 0 ; row < a.length ; row++ )
                for ( int col = 0;  col < a[row ].length ; col++ )

                        total += a[ row] [ col ];
```

Ej. de arreglo con doble subíndice: Se utiliza 4: **métodos: minimum y maximun**, determinan la calificación más baja y más alta de cualquier estudiante durante el semestre, **el método average** determina **el promedio** de un estudiante en particular. El **método printArray** exhibe el arreglo de double subíndice en un formato tabular.

Cada fila representa un estudiante, cada columna representa una calificación.

```
import java.awt.Graphics;
import java.applet.Applet ;

public class DoubleArray extends  Applet.
{
      int grades[ ][ ] = {( 77, 68, 86, 73 ), (96, 87, 89, 81 ), (70, 90, 86, 81 ) };
      int studensts, exams;
      int xPosition, yPosition;

      // inicializar var de ejemplar
      public void init( )
      {
          students = grades.length;
          exams = grades [0].length;
      {

      //pintar las applets
      public void paint( Graphics g)
      {
          xPosition = 25;
          yPosition = 25;

          g.drawString( “El arreglo es: “, xPosition, yPosition );
          yPosition  += 15;
          printArray ( g );
          xPosition = 25;
          yPosition += 30;
          g.drawString( “calificación más bajas; “, xPosition, yPosition );
          int min = minimun( ) ;
          g.drawString ( String.valuof( min ), xPositon + 85, yPositon ) ;
          yPositon += 15;
          g.drawString( “calificación más alta:” , xPosition, yPosition );
          int max = maximun( ) ;
          g.drawString ( String.valuof( max ), xPositon + 85, yPositon ) ;
          yPositon += 15;

          for( int i = 0;  i < studensts, i++)
          {
                    g.drawString( “el promedio del estudiante ”  + i + es”, 25,  yPosition );
                    double ave = average ( grades [ i ] ;
                    g.drawString ( String.valuof( ave ), 165, yPositon ) ;
                    yPositon += 15;
          }
      }
      //encontrar la calificación más baja.
      Public int minimun()
      {
          int lowGrade = 100 ;
          for( int i = 0 ; i < studentes ;  i++ )
                    for( int j = 0 ; j < exams ;  j++ )
                              if ( grades[ I ]  <  lowGrade)
                                        lowGrade = grades[ I][ j ];
```

```
    return lowGrade;.
  }
  //encontrar la calificación más alta
  Public int maximun()
  {
    int highGrade = 0 ; //calificación alta
    for( int i = 0 ; i < studentes ;  i++ )
        for( int j = 0 ; j < exams ;  j++ )
            if ( grades[ I ]  <  highGrade)
                highGrade = grades[ I][ j ];

    return highGrade;.
  }
  //determinar la calificación para un estudiante en particular ( o un grupo de calificaciones)
  Public double average ( int setOfGrades[ ])
  {
    int total = 0;
    for( int i = 0 ; i < setOfGrades.length ;  i++ )
        total += setOfGrades[ I ];
    return (double) total/ setOfGrades.length;
  }
  //imprimir el arreglo
  Public void printArray (Graphics g )
  {
    xPosition = 80 ;

    for( int i = 0 ; i < exams ;  i++ )
    {
        g.drawString( " [ " + I + "]", xPosition, yPosition );
        xPosition += 30;
    }
    for( int i = 0 ; i < estudents ;  i++ )
    {
        xPosition = 25;
        yPosition += 15;
        g.drawString( " [ " grades[" + I + "]", xPosition, yPosition ];
        xPosition = 80;.

        for( int j = 0 ; j < exams ;  j++ )
        {
        g.drawString( String.valuOf( grades [ I] [ j ], xPosition, yPosition );
        xPosition += 30;

        }
     }
  }
}
```

```
El arreglo es:
            [0]  [1 ]  [ 2 ]  [ 3 ]

grades[0]   77    68     86    73
grades[ 1 ]  96   87     89     81
grades[ 2 ]  70   90     86     81
calificación más baja . 68
calificación más alta 96
el promedio del estudiante 0 es 76
el promedio del estudiante 1 es 88.25
el promedio del estudiante 2 es 81.87
```

ARREGLOS

Objetivos:

- Presentar la estructura de datos de arreglo
- Entender el empleo de arreglos para almacenar, ordenar y examinar listas y tablas de valores.
- Entender la forma de declarar un arreglo, inicializar un arreglo a elementos individuales de un arreglo.
- Poder pasar arreglos a métodos.
- Entender las técnicas de ordenamiento básicas.
- Poder declarar y manipular con múltiples subíndices.

5.10. Introducción:

- Los arreglos son estructuras de datos que consisten en elementos de información del mismo tipo relacionados entre sí.
- Los arreglos son entidades estáticas en cuanto a su tamaño.

5.11. Arreglos

- Un arreglo es un grupo de posiciones de memoria contiguas, ya tienen el mismo nombre y el mismo tipo.
- Para referirnos a una posición(elemento) , especificamos el nombre del arreglo y el número de posición del elemento.
- Nos referimos así: primer elemento del arreglo: c[0], segundo elem c[1] ,... y en general al í-esimo elem del arreglo c[i-1]**.**
- El nombre de un arreglo es similar a una variable.
- El número de posición recibe el nombre de subíndice.
- La expresión se evaluará para determinar el subíndice.
- Un nombre de arreglo con subíndice es un **Ivalue,** se puede escribir en lado izquierdo de un asignación para colocar un nuevo valor en un elem de un arreglo.
- La longitud del arreglo se determina con la siguiente expresión: **c.length**// nombre del arreglo y la longitud.
- Todo arreglo en Java conoce su propia longitud.

5.12. Declaración y Asignación de almacenamiento a arreglos

- Los arreglos ocupan espacio en la memoria.
- Se **especifica el tipo de los elementos** y **se usa** el operador **new** para asignar espacio de almacenamiento al número de elementos requeridos por cada arreglo. Los **elementos se inicializan** automáticamente a **cero** en caso de var. de tipos primitivos numéricos, a **false** en el caso de boolean o a **null** en el caso de referencias. Usamos la declaración:

int c[ ] = new int[12]; ó int c[ ]; //declara el arreglo de max elem=12

c[ ] = new int[12]; //asigna memoria al arreglo

5.13. Ejemplos del uso de arreglos

El operador **new** es para asignar dinámicamente espacio de almacenamiento a un arreglo de 10 elementos.

```
// Ej. 1  Inicialización de los elementos de un arreglo en ceros
import java.awt.Gaphics;
import java.applet.Applet;

public class Initarray extends Applet {
    int n[ ]; // declarando un arreglo de enteros – tamaño del arreglo
```

```
    //  inicializar variables de ejemplar
    public void init( )
    {
        n = new int[ 10 ]; // asignar memoria dinámicamente al arreglo
    }
    // pintar la applet
    public void paint( Graphics g )
    {
        int yPosition = 25 ; // posición y inicial de la applet

        g.drawString( "Elemento", 25, yPosition );//enunciado exhibe la cabecera
        g.drawString( "Valor", 100, yPosition );   //     "                "

        for ( int i = 0;  i< n.length;  i++ )
        {
              yPosition += 15;
              g.drawString( String.valueOf (i  ), 25, yPosition );
              g.drawString(String.valueOf, ( n[ i ] ), 100,  yPosition );.
        }
    }
}
```

```
Elemento valor
   0     0
   1     0
   2     0
   3     0
   4     0
   5     0
   6     0
   7     0
   8     0
   9     0
```

La var **yPosition** sirve para determinar la **posición vertical** en la que el método drawString dibujará en la applet.
El **método String.valueOf convierte cada entero en una cadena que se pueda exhibir en la applet.**
n.length determina la longitud del arreglo.

```
//Ej. 2  Inicialización de arreglo enteros con 10 valores e imprime el arreglo en forma tabular.(con una declarac.)
import java.awt.Gaphics;
import java.applet.Applet;

public class Initarray extends Applet {
    int n[ ] = {32, 27, 64, 18, 95 14, 90, 70, 60, 37};

    // pintar la applet
    public void paint( Graphics g )
    {
        int yPosition = 25 ; // posición y inicial en la applet

        g.drawString( "Elemento", 25, yPosition );//enunciado exhibe la cabecera
        g.drawString( "Valor", 100, yPosition );   //     "                "

        for ( int i = 0;  i< n.length;  i++ )
        {
              yPosition += 15;
              g.drawString( String.valueOf (i  ), 25, yPosition );
              g.drawString(String.valueOf, ( n[ i ] ), 100,  yPosition );.
        }
    }
}
```

```
Elemento valor
   0     32
   1     27
   2     64
   3     18
   4     95
   5     14
   6     90
   7     70
   8     60
   9     37
```

```
//Ej. 3 inicializar arreglo s con enteros pares de  2 a 20
    import java.awt.Gaphics;
    import java.applet.Applet;

    public class Initarray extends Applet {
        final int  arraySize = 10;          // inicializa los elem. de un arreglo  s de 10 elem con los enteros 2, 4, 6 … 20;
```

```
        //utiliza el calificador final para declarar una var constante arraySize (tamaño de arreglo) cuyo valor es 10.
        Las var. const. se deben inicializar con una expresión constante cuando se declaran y no se pueden modificar.
        Si intenta modificar una var. final después de haberla declarado, el compilador producirá el mensaje de error
        (cant't assign a value to a final variable)..
                int s[ ];

        //  inicializar variables de ejemplar
        public void init( )
        {
            s = new int[ arraySize ];
        //estos números se generan multiplicando cada valor sucesivo del contador del ciclo por 2  y sumándolo 2
        //  establecer los valores del arreglo

            for ( int i = 0 ; i <= n.length;  i++ ).
                  s[ i ] = 2 +2 * i ;
        }
        // pintar la applet
        public void paint( Graphics g )
        {
            int yPosition = 25 ; // posición y inicial de la applet

            g.drawString( "Elemento", 25, yPosition );//enunciado exhibe la cabecera
            g.drawString( "Valor", 100, yPosition );   //     "                "

            for ( int i = 0;  i< n.length;  i++ )
            {
                  yPosition += 15;
                  g.drawString( String.valueOf (i  ), 25, yPosition );
                  g.drawString(String.valueOf, ( s[ i ] ), 100,  yPosition );.
            }
        }
    }
```

```
Elemento valor
   0  2
   1  4
   2  6
   3  8
   4  10
   5  12
   6  14
   7  16
   8  18
   9  20
```

Ej. 4 Empleo de una var constante debidamente inicializada

```
Import java.awt.Graphics;
Import java.applet.Applet ;

Public class FinalTest extends Applet
{
        final int x = 7;  //inicializar la var constante

        public void paint( Graphics g )
        {
        g.drawString( "El valor de x es: " + x, 25,  25 );
        }
}
```

Ej.5 Calculo de la suma de los elementos de un arreglo

```
Import java.awt.Graphics;
Import java.applet.Applet ;

Public class SumArray extends Applet
{
        int a[ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int total;
```

```
        //inicializar variables de ejemplar
        public void init( ).
        {
        total = 0;
        for ( int i = 0;   i < a.length;  i++ )
                          total += a [ i ];
        }
        //pinta la applet
        public void paint( Graphics g )
        {                                                          55
        g.drawString( "Total de los elementos del arreglo:"  + total,  25,  25 );
        }
}
```

Se pidió a 40 estudiantes calificar la calidad de la comida que se sirve en la cafetería en una escala de 1 a 10 (1 denota horrible y 10 excelente) . Coloque las 40 respuestas en un arreglo de enteros y resuma los resultados del sondeo.

Ej.6 Programa de sondeo de 40 estudiantes//numero de respuestas(arreglo **responses)** de cada tipo(de 1 a 10)

```
Import java.awt.Graphics;
Import java.applet.Applet ;

Public class StudentPoll extends Applet
{               arreglo que contiene las repuestas de 40 elem (estud.)
        int responses [ ] = {  1, 2, 6, 4, 8, 5, 9, 7, 8, 10
                               1, 6, 3, 8, 6, 10, 3, 8, 2, 7
                               6, 5, 7,  6, 8, 6, 7, 5, 6, 6,    };
                               5, 6, 7, 5, 6, 4, 8, 6, 8, 10
        int frec [ ];

      //inicializar variables de ejemplar
        public void init( ).
        //frec. arreglo de 11 elem. para contar el número de ocurrencias de c/respuesta.

           frec = new  int [ 11 ] ;//los elem. del arreglo frec se inicializan automát. en cero, se asigna memoria con new

                                                  long. 40
           for ( int answer = 0;   answer <  response.length; answer++ )//for toma las respuest. una a una del arreglo
response

                ++frec[ responses [  answer ] ];
                                0, 1, 2, 3, ….40
                              1, 2, 6, 4…10// increm 6 del arreglo
                    1, 2, 3, 4.....11

                // incrementa el contador dependiendo del valor de response[answer]  es frec [ 1 ] y  es ++frec[1]
        }
        //pintar la applet
        public void paint( Graphics g )
        {
                int yPosition = 25; //posición inicial en la applet

        g.drawString( "Calificación!", 25, yPosi tion);
                g.drawString( "frec",100, yPosi tion);

        for (int rating = 1;  rating < frec.length;  rating ++ )
        {
```

Calificación	frec
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

answer	response[answer]	++frec[response[answer]]
0	1	++frec[1]
1	2	++frec[2]
2	6	++frec[6]
3	4	++frec[4]

```
            yPosition += 15;
            g.drawString( String.valuof( rating ), 25, yPosition );
                g.drawString( String.valuof( frec[ rating]  ), 100, yPosition );
        }
        }
}
```

Calificación	frecuencia]
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

Java cuenta con mecanismos para evitar el acceso a elementos fuera de los límites del arreglo, al compilar, verifica las referencias a elementos de arreglos para asegurarse de que sean válidas.
Si se hace una referencia no valida a un elem de un arreglo en el momento de la ejecución java genera una excepción. (ArrayIndexOutOfBoundsException -índice de arreglo fuera de límite).

Ej. 7. lee números de un arreglo y grafica la información en forma de graficas de barras o histograma, se imprime cada número y junto a el se imprime una barra formada por esa misma cantidad de asteriscos. El ciclo for anidado se encarga de dibujar las barras.

```
Import java.awt.Graphics;
Import java.applet.Applet ;

Public class Histogram extends Applet
{
        int n[ ] = {19, 3, 15, 7, 11, 9, 13, 15, 17, 1 };

        //pintar la applet
        public void paint( Graphics g )
        {
        int xPosition; //posición de * en el histograma
        int yPosition = 25; //posición vertical en la applet

                g.drawString( “Elemento “, 25, yPosition );
                g.drawString( “Valor”, 100, yPosition );
                g.drawString( “Histograma”, 175, yPosition );

                for ( int  i  = 0;  i  <  n.length;  i++)
                {
                    yPosition += 15;
                    g.drawString( String.valueOf( i ), 25, yPosition );
                    g.drawString(String.valueOf( n [ i ] ),  100, yPosition );.

                    XPosition = 175;

                    El ciclo for anidado se encarga de dibujar las barras.
                    for ( int  j  = 0;  j  <  n[ i ];  j++ ) // imprimir una barra
```

Elemento	Valor	Histograma
0	19	**********
1	3	***
2	15	***************
3	7	*******
4	11	***********
5	9	*********
6	13	*************
7	5	*****
8	17	***************** *
9	1	*

```
                {
                    g.drawString ( " * ", xPosition, yPosition);
                    xPosition  += 7;
                }
            }
        }
}
```

5.14. Referencias y parámetros de referencia

- Dos **formas de pasar argumentos a los métodos**, en C y C++ son llamar **por valor** y llamar **por referencia**. Cuando se pasa un **argumento por valor** , **se crea una copia de valor del argumento y se pasa al método invocado**. Por **referencia, el invocador confiere al método invocado la capacidad de acceder directamente a los datos del invocador y modificar dichos datos** si el método invocado lo desea.

- En Java no permite escoger cómo quiere pasar los argumentos, si por valor o por referencia. **Las vars de tipos de datos primitivos siempre se pasan por valor y los objetos siempre se pasan por referencia.**.
- Para **pasar un objeto por referencia** especificar **en la llamada de método la referencia al objeto por su nombre**.
- **Java trata a los arreglos como objetos**, los **arreglos se pasan a los métodos por referencia:** un método invocado puede acceder a los valores de los elementos en los arreglos originales de los innovadores.
- El **nombre de un arreglo es una referencia a un objeto** que contiene los elementos del arreglo y la **variable de ejemplar length** que indica el número de elementos en el arreglo.
- Al devolver información de un método mediante un enunciado **return** las variables de tipos de datos primitivos siempre se devuelve por valor y los objetos siempre se devuelven por referencia.

Ej. Tirar un dado de seis caras 6000 veces para probar si el generador de números aleatorios producía números en verdad aleatorios. Empleando arreglos en vez de switch

```
//RollDie.java – Tirar un dado de seis caras 600 veces

Import java.awt.Graphics;
Import java.applet.Applet;
Import java.util.Random;

Public class RollDie extends Applet
{
        int face;  //cara
        int frec [ ];
        Random r; // crear el generador de números aleatorios


        //inicializar variables de ejemplar
        public void init( ).
        {
        frec  = new int [ 7 ];.
                r = new Random  ();

        for ( int roll =1;  roll < = 6000;   roll++ ){
                        face = 1 + Math.abs( r.nextInt() % 6 );
                        ++ frec[ face ];
                }
        }
        //pintar la applet
        public void paint( Graphics g )
        {
```

1	1030
2	1049
3	977
4	973
5	988
6	983

```
        int yPosition = 25;
                g.drawString( "cara"   25, yPosition );
                g.drawString( "Frecuencia", 100, yPosition );

                for ( face = 1; face < frec.length; face++ ){
                yPosition += 15;
                        g.drawString( String.valueOf( face ), 25, yPosition );
                        g.drawString(String.valueOf( frec [ face] ),  100, yPosition );.
                }
        }
}
```

5.15. Cómo pasar arreglos a los métodos

Si quiere pasar un argumento de arreglo a un método, especifique el nombre del arreglo sin los corchetes.
Ej. si se declaró el arreglo como:

Int **temperaturasHorarias[ ] = new int [24];**

La llamada del método:

ModificarArreglo(**temperaturasHorarias**); .// método y arreglo

En **JAVA** todo objeto de arreglo conoce su propio tamaño (por la var ejemplar length). Cuando pasamos un objeto de arreglo a un método, no necesitamos pasar el tamaño del arreglo como argumento.

Los arreglos enteros se pasan por referencia; es decir, los elementos de arreglo individuales que no son de tipos primitivos) a estos elementos se denominan escalares o cantidades escalares.

Los elementos de arreglo individuales de tipos de datos primitivos se pasan por valor..

Al pasar un elemento de arreglo a un método usamos el nombre con subíndice del elemento de arreglo como argumento en la llamada de método.

Para que un método reciba un arreglo a través de una llamada de método, la lista de parámetros del método debe indicar que se recibirá un arreglo. Ej.

Void modificarArreglo(int b[ ])

Modificar arreglo espera recibir un arreglo entero en el parámetro b. **Dado que**, los arreglos se pasan por referencia, cuando el método invocado utiliza el nombre de **arreglo b.,** se refiere al arreglo real en el invocador (el arreglo temperaturaHorarias en la llamada anterior)

El ejemplo demuestra la diferencia entre **pasar un arreglo completo y pasar un elemento de arreglo.**
Primero imprime los cinco elementos del arreglo **enteros a.** Luego se pasa a al **método modifyArray**(modificar arreglo) donde cada uno de los elementos de **a** se multiplica por 2. Después, **a** se vuelve a imprimir en **main**. ModiffyArray modifica los elementos de **a.** Luego imprime el valor de a[3] y lo pasa al **método modifyElemen**t(modificar elemento). Este método multiplica su argumento por 2, cuando a[3] se vuelve a imprimir en **main**, no ha sufrido modificación porque los elementos del arreglo individuales de tipos de datos primitivos se pasan por **valor.**

```
//PassArray.java
//paso de arreglos y elementos individuales de arreglos a métodos
Import java.awt.Graphics;
Import java.applet.Applet;

public class PassArray extends Applet
{
        int a [ ] = { 0, 1, 2, 3, 4};

        //pintar la applet
```

```
public void paint( Graphics g )
{
int yPosition = 25, yPosition = 25;
        g.drawString( “efectos de pasar  todo el arreglo por referencia” , xPosition,  yPosition );
        yPosition += 15;
        g.drawString( “Los valores del arreglo original son”, xPosition,  yPosition );
        xPosition += 15;
        yPosition += 15;

        for ( int i = 0;  i < a.length; i++ ){
        g.drawString( String.valueOf( a[ i ] ), xPosition, yPosition );
                xPosition += 15;
        }
        xPosition = 25;
        yPosition += 30;

        modifyArray( a );          // el arreglo a se pasa por referencia
        g.drawString( “Los valores del arreglo modificados son”, xPosition,  yPosition );.
        xPosition += 15;
        yPosition += 15;

        for ( int i = 0;  i < a.length; i++ ){
        g.drawString( String.valueOf( a[ i ] ), xPosition, yPosition );
                xPosition += 15;
        }
        xPosition = 25;
        yPosition += 30;

        g.drawString( “efectos de pasar  un elemento de un arreglo por valor” , xPosition,  yPosition );
        yPosition += 15;
        g.drawString( “a[3] antes de modifyElement:” + a [ 3 ], xPosition,  yPosition );
        xPosition = 25;
        modifyElement(  a[3 ] ) ;
        g.drawString( “a[3] después de modifyElement:” + a [ 3 ], xPosition,  yPosition );

}

public void modifyArray( int b[ ] )
{
for ( int j = 0 ; j < b.length; j++)
                b[ j ] *= 2;
}


public void modifyElement( int e)
{
e *= 2;
}
}
```

```
Efectos de pasar todo el arreglo pro referencia:
Los valores del arreglo original son:
0 1 2 3 4
Los valores del arreglo modificados son:
0 2 4 6 8
Efectos de pasar un elemento de una arreglo por valor :
A[3] abantes de modifyElemnto: 6
A[3] después de modifyElement: 6
```

5.16. Ordenamiento de arreglos

`Ordenar es colocar los datos en algún orden específico(ascendente o descendente).`

Ej. Ordenar los valores del arreglo de 10 elementos **a** en **orden ascendente**.

Técnica a usar: ordenamiento de burbuja u ordenamiento descendente. Los valores más pequeños suben hacia el principio del arreglo, como las burbujas del aire ascienden en el agua, mientras que los valores más grandes se hunden o descienden hasta el final del arreglo. Para ello se realizará varias pasadas por el arreglo. **En cada pasada se comparan pares de elementos sucesivos.** Si un par está en orden creciente (o los valores son idénticos), dejamos los valores tal como están. Si un par está en orden decreciente, intercambiamos sus valores en el arreglo.

```
import java.awt.Graphics;
import java.applet.Applet ;
public class BubbeSort extends Applet
{
        int a[ ] = {2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

        int hold; // area de retención temporal para el intercambio

        public void paint ( Graphics g)
        {
                print( g, "Datos en el orden original", a, 25, 25 );
                sort();
                print( g, "Datos en el orden ascendente", a, 25, 55);
        }
        public void sort( )
        {//for anidado
        for ( int pass = 1; pasa < a.length; pass++ ) // pasadas
                for ( int i = 0;  i < a.length – 1;  i++) // una pasada
                                if ( a[ i ] > a[ i  + 1 ]) //compara a[0] con a[1], luego a[1] con a[2]…hasta completar la
                        pasada,
                                solo realiza 9 comparaciones.
                                El valor grande baja varias posiciones en una sola pasada, valor pequeño sube solo una
                                posición.
                                1ª pasada el valor grande se hunde hasta el último elemento, a[9], en la 2ª hundirá hasta
                                a[8], en la 9ª pasada, el noveno valor más grande se hunde a a[1], Esto deja al valor
                                más pequeño con a[ 0 ], de modo que sólo se necesitan nueve pasadas para ordenar un
                                arreglo de 10 elementos.
                                {// intercambiar dos valores, si es posible
                                hold = a[ i ];
                        //var adicional  hold(retener) almacena temporalmente uno de los dos valores que se
                        intercambian.
                                a[ i ] = a[ i  + 1];
                                        a[ i + 1 ] = hold;
                                }
        }
        public void print ( Graphics g,  String head, int b [ ], int x, int y )
        {
        g.drawString( head, x, y );
                x += 15;
                y += 15;
                for ( int i = 0;  i < b.length;  i++)
                {
                g.drawString( String.valueof( b[ i  ] ), x,  y );
                x += 20;
                }
        }
```

```
Datos en el orden original
2 6 4 8 10 12 89 68 45 37
Datos en el orden ascendente
2  4 6  8 10 12 37 45 68 89
```

}
Ordenamiento burbuja es fácil de programar, pero se ejecuta con lentitud.

5.17. Búsqueda en arreglos: Búsqueda lineal y Búsqueda binaria.

- `Determinar si un arreglo contiene un valor que coincide con cierto` **`valor clave.`**
- El proceso de encontrar un elemento de un arreglo se llama **búsqueda.**
- **El método de búsqueda lineal funciona con arreglos pequeños y no ordenados.**
- **`Búsqueda lineal:`** `compara cada elemento del arreglo con la clave de búsqueda. Al comparar la clave de búsqueda con la mitad de los elementos del arreglo, en promedio.`
- **Para el método de búsqueda binaria el arreglo debe estar ordenado, ya que es de alta velocidad .**
- El **algoritmo de búsqueda binaria elimina de la búsqueda la mitad de los elementos del arreglo** después de cada comparación. Si son iguales, se habrá encontrado la clave y se devolverá el subíndice del elemento en cuestión; si no, el problema se reduce a buscar en la mitad del arreglo. La búsqueda continua hasta que la clave
- En el peor de los casos, una búsqueda en un arreglo de **1024 elementos requiere sólo de 10 comparaciones** empleando búsqueda binaria.
- La división repetitiva 1024/2 después de cada comparación, podemos eliminar la mitad del arreglo, produce los valores 512, 256, 128, 64, 32, 16, 8, 4, 2 y 1. El número 1024 (2^{10}) se divide entre 2 sólo 10 veces para obtener el valor 1.
- Un arreglo de 1.048..576 (2^{20}) elementos requiere un máximo de 20 comparaciones para encontrar la clave

```
// búsqueda binaria de un arreglo = BinarySearch.java

import java.awt.* ;
improt java.applet.Applet ;
public class BinarySearch extends Applet
{
        int a[ ];
        int element;
        String searchKey; //clave de búsqueda
        Int xPosition; // posición horizontal pra dibujar en la applet
        Int yPosition; // posición vertical para dibujar en la applet
        Label enterLabel;
        TextField enter;
        Label resultLabel;
        TextField result; // campo de texto para el resultado
        Bolean timeToSearch = false; // hora de buscar

        Public void init ( )
        {
        a = new int [ 15   ];
                for ( int i = 0; i < a.length;  i++ ) // crear los datos
                        a [ i ] = 2 * i;

                enterLabel = new Label ("Tecle la clave" );
                enter = new TextField ( 5 );
                resultLabel = new Label ( "Resultado " );
                result = new TextField ( 22 );
                result.setEditable( false );
                add( enterLabel );
                add( enter );
```

```
        add( resultLabel );
        add( result );
}

public void paint ( Graphics g )
{
if ( timeToSearch ) // evita buscar en la 1ª invocación
        {
        element. BinarySearch( Integer.parseInt ( searchKey ), g );
                if ( element != - 1 )
                        result.setText( "valor hallado en elemento " + element );
                else
                        result.setText( "No se halló el valor ");

        }
}

public bolean action ( Event e, Object o )
{
if ( e.target == enter)
        {
        timeToSerarch = true;
                xPosition = 25;
                yPosition = 75;
                searchKey = e.arg.toString( );
                repaint (); // llamar a paint para iniciar búsqueda y salida.
        }
        return true;.
}

//Búsqueda binaria
public int  binarySearch ( int key, Graphics gg)
{
gg.drawString ( "proporciones del arreglo en que se buscó", xPositicon, yPosition );
yPosition += 15;

int low = 0; // subíndice bajo
int high = a.length – 1; // subíndice alto
int middle;       // subíndice medio

while ( low <= high )
        {
        middle = ( low + high ) / 2;

        printRow( low, middle, high, gg );
        if ( key == a[ middle ] ) // se encontró
                return middle;
        else if ( key < a[ middle ])
                        high = middle –1 // buscar parte baja del arreglo
                else
                        low = middle + 1; // buscar parte alta del arreglo
        }
        return – 1; // no se encontró searchKey
}
// imprimir una fila de salida mostrando la parte del arreglo que se está procesando ahora
```

```
        void printRow ( int low, int mid, int high, Graphics gg)
        {
        xPosition = 25;

                for ( int i = 0; i < a.length;  i++ ) //
                {
                        if ( i < low       i > high )
                        gg.drawString( "  ",  xPositicon, yPosition );
                        else if ( i == mid ) // marcar  valor medio
                        gg.drawString( String.valueOf ( a [ i  ] + " * ", xPosition, yPosition );
                        else
                        gg.drawString( String.valueOf ( a [ i  ] ,  xPosition, yPosition );

                xPosition += 20;
                }
                yPosition += 15;
        }
}
```

5.18. Arreglos con múltiples subíndices.

- Las tablas o arreglos requieren dos subíndices (filas, columnas), para identificar un elemento (el 1ro. identifica la fila del elem, y el 2do. la columna del elem).
- Java no maneja arreglos de múltiples subíndices.
- Un arreglo con doble subíndice b[2] [2] se podría declarar e inicializar con:

  int b[ ] [ ] = { (1, 2), (3, 4)};

  - Los valores se agrupan por fila encerrados en llaves.

    Así, 1 y 2 inicializan b[0] [0] y b[0] [1] y

    3 y 4 inicializan b[1] [0] y b[1] [1]

```
b[0][0]   b[0][1]
b[1][0]   b[1][1]
```

.

```
//InitArray.java
//Inicialización de arreglos multidimensionales
import java.awt.Graphics;
import java.applet.Applet;

public class InitArray extends Applet
{//la declaración array1 proporciona seis inicializadores en dos sublistas. La 1ra. Sublista inicializa la primera fila del arreglo con los valores 1, 2 y 3; la 2da. Sublista inicializa la 2da. fila del arreglo con los valores 4, 5 y 6.
    int array1[ ] [ ] = { ( 1, 2, 3 ), (4, 5, 6 ) };

//la declaración array2 proporciona tres inicializadores en dos sublistas. La 1ra. Sublista inicializa la primera fila del arreglo con dos elementos, con los valores  1, 2 ; la 2da. Sublista  de la 2da. fila inicializa con un elemento con el valor 4.
    int array2[ ] [ ] = { ( 1, 2 ), (4 ) };

    //pintar la applet
    public void paint ( Graphics g)
```

```
Lo a[ i ].length valores de array1 por fila son
1 2 3
4 5 6
los valores de array2 por fila son
1 2
4
```

```
    {
        g.drawString( « Los valores de array1 por fila son », 25, 25 ) ;
        printArray(array1, g, 40 );  //imprimir arreglo, exhibe los elems

        g.drawString( « Los valores de array2 por fila son », 25, 70 ) ;
        printArray(array2, g, 85 );
    }
    public void printArray(  int a[ ] [ ], Graphics g, int y ) //método printArray
```

//La def. del método especifica el parámetro de arreglo como int a[ ] [ ] para indicar que se recibirá un arreglo con doble subíndice como argumento. Se pasa una referencia Graphics como argumento para que el método (printArray) pueda exhibir el contenido de cada arreglo en la applet.

```
    {
        int x = 25 ;

        for ( int i = 0 ; i < a.length ; i++ ) //for anidada para exhibir las filas de cada arreglo con doble subíndice
        {//a.length determina el número de filas
                for ( int j = 0 ; j < a[ i ].length ;  j++ )
                {// a[ i ].length determina el número de columnas de cada fila del arreglo
                        g.drawString(  String.valueof( a[ i ] [ j ], x,  y  );
                        x += 15;
                }
                x = 25;
                y += 15;.
        }
    }
}
```

Ej. For asigna cero a todos los elems. de la 3ra fila del arreglo

```
        for ( int col = 0 ; col < a[ 2 ].length ; col++ )
                a[ 2 ] [ col ] = 0;
```

```
a [ 2 ][ 0 ]= 0;
a [ 2 ][ 1 ]= 0;
a [ 2 ][ 2 ]= 0;
a [ 2 ][ 3 ]= 0;
```

Ej. for anidada calcula la sumatoria de todos los elementos del arreglo **a**. for suma los elementos fila por fila

```
        total = 0;
        for ( int row = 0 ; row < a.length ; row++ )
                for ( int col = 0;  col < a[row ].length ; col++ )

                        total += a[ row] [ col ];
```

Ej. de arreglo con doble subíndice: Se utiliza 4: **métodos: minimum y maximun**, determinan la calificación más baja y más alta de cualquier estudiante durante el semestre, **el método average** determina **el promedio** de un estudiante en particular. El **método printArray** exhibe el arreglo de double subíndice en un formato tabular.

Cada fila representa un estudiante, cada columna representa una calificación.

```
import java.awt.Graphics;
import java.applet.Applet ;

public class DoubleArray extends Applet.
{
    int grades[ ][ ] = {( 77, 68, 86, 73 ), (96, 87, 89, 81 ), (70, 90, 86, 81 ) };
    int studensts, exams;
    int xPosition, yPosition;

    // inicializar var de ejemplar
    public void init( )
    {
        students = grades.length;
```

```
    exams = grades [0].length;
{

//pintar las applets
public void paint( Graphics g)
{
    xPosition = 25;
    yPosition = 25;

    g.drawString( “El arreglo es: “, xPosition, yPosition );
    yPosition  += 15;
    printArray ( g );
    xPosition = 25;
    yPosition += 30;
    g.drawString( “calificación más bajas; “, xPosition, yPosition );
    int min = minimun( ) ;
    g.drawString ( String.valuof( min ), xPositon + 85, yPositon ) ;
    yPositon += 15;
    g.drawString( “calificación más alta:” , xPosition, yPosition );
    int max = maximun( ) ;
    g.drawString ( String.valuof( max ), xPositon + 85, yPositon ) ;
    yPositon += 15;

    for( int i = 0;  i < studensts, i++)
    {
            g.drawString( “el promedio del estudiante ”  + i + es”, 25,  yPosition );
            double ave = average ( grades [ i ] ;
            g.drawString ( String.valuof( ave ), 165, yPositon ) ;
            yPositon += 15;
    }
}
//encontrar la calificación más baja.
Public int minimun()
{
    int lowGrade = 100 ;
    for( int i = 0 ; i < studentes ;  i++ )
            for( int j = 0 ; j < exams  ;  j++ )
                    if ( grades[ I ]  <  lowGrade)
                            lowGrade = grades[ I][ j ];

    return lowGrade;.
}
//encontrar la calificación más alta
Public int maximun()
{
    int highGrade = 0 ; //calificación alta
    for( int i = 0 ; i < studentes ;  i++ )
            for( int j = 0 ; j < exams  ;  j++ )
                    if ( grades[ I ]  <  highGrade)
                            highGrade = grades[ I][ j ];

    return highGrade;.
}
//determinar la calificación para un estudiante en particular ( o un grupo de calificaciones)
Public double average ( int setOfGrades[ ])
{
```

```
        int total = 0;
        for( int i = 0 ; i < setOfGrades.length ;  i++ )
                total += setOfGrades[ I ];
        return (double) total/ setOfGrades.length;
    }
    //imprimir el arreglo
    Public void printArray (Graphics g )
    {
        xPosition = 80 ;

        for( int i = 0 ; i < exams ;  i++ )
        {
                g.drawString( " [ " + I + "]", xPosition, yPosition );
                xPosition += 30;
        }
        for( int i = 0 ; i < estudents ;  i++ )
        {
                xPosition = 25;
                yPosition += 15;
                g.drawString( " [ " grades[" + I + "]", xPosition, yPosition ];
                xPosition = 80;.

                for( int j = 0 ; j < exams ;  j++ )
                {
                g.drawString( String.valuOf( grades [ I] [ j ], xPosit ion, yPosition );
                xPosition += 30;

                }
         }
    }
}
```

```
El arreglo es:
            [0]  [1 ]  [ 2 ]  [ 3 ]

grades[0]   77    68     86    73
grades[ 1 ]  96   87     89     81
grades[ 2 ]  70   90     86     81
calificación más baja . 68
calificación más alta 96
el promedio del estudiante 0 es 76
el promedio del estudiante 1 es 88.25
el promedio del estudiante 2 es 81.87
```

Ejercicios de autoevaluación

6.1 Llene los espacios en blanco.

a) El acceso a miembros de una clases se logra con el operador **punto (.)** junto con un objeto de la clase.

b) Los miembros de una clase especificados como **private** sólo están accesibles para métodos de la clase.

c) Un **constructor** es un método especial que se usa para inicializar las variables de ejemplar de una clase.

d) Se usa un **metodo set** para asignar valores a variables de ejemplar **privaten** de una clase.

e) Los métodos de una clase normalmente se hacen **public** y las variables de ejemplar de la clase normalmente se hacen **private.**

f) Se emplea un **metodo get** para obtener los valores de los datos **private** de una clase.

g) La palabra clave **class** introduce una definición de clase.

h) Los miembros de una clase especificada como **public** están accesibles en cualquier lugar en el que un objeto de la clase esté dentro del alcance.

i) El operador **new** asigna dinámicamente memoria a un objeto de un tipo especificado y devuelve una **referencia** a ese objeto.

j) Los objetos constantes deben **inicializarse**; no pueden modificarse después de haberse creado.

k) Una variable de ejemplar **static** representa información que abarca toda la clase.

l) La referencia **this** se pasa como primer argumento imlícito a todos los métodos de una clase.

m) La palabra clave **final** espñecifica que un objeto o variable no es modificable después de que haberse inicializado.

n) Un método declarado **static** no puede acceder a miembros de clase **no static.**

6.2 Cree una clase llamada C**omplex (complejo)** para realizar aritmética de números complejos. Escriba un programa para obtener su clase.

Los números complejos tienen la forma:
RealPart + imaginar/Part * i donde i es:
Utilice variables de punto flotante para representar los datos privados de la clase. Proporcione un método constructor que permita inicializar un objeto de esta clase cuando se declara. Incluya un constructor sin argumentos con valores por omisión en caso de que no se proporcionen inicializadores. Incluya métodos public para lo siguiente:

a) suma de dos números Complex las partes reales y las imaginarias se suman aparte.
b) Resta de dos números **Complex:** la parte real del operando derecho se resta a la parte real del operando izquierdo y la parte imaginaria del operando derecho se resta a la parte imaginaria del operando izquierdo.
c) Impresión de números Complex en la forma (a, b), donde a es la parte real y b es la parte imaginaria.

6.3 Cree una clase llamada Rational (racional) para realizar aritmética de fracciones. Escriba un programa para probar su clase.

Utilice variables enteras para representar las variables de ejemplar private de la clase: el numerator (numerador) y el denominator (denominador). Proporcione un método constructor que permita inicializar un objeto de esta clase cuando se declara. El constructor deberá almacenar la fracción en forma reducida (es decir, la fracción 2/4 se almacenaría en el objeto como 1 en numerator y 2 en denominator . Incluya un constructor sin argumentos convalores por omisión en caso de que no se proporcionen argumentos. Incluya métodos public para cada una de los siguientes operaciones.

a) Suma de dos números Rational. El resultado de la suma se debe almacenar en forma reducida.

b) Resta de dos números Rational. El resultado de la resta se debe almacenar en forma reducida.
c) Multiplicación de dos números Rationl. El resultado de la multiplicación se debe almacenar en forma reducida.
d) División de dos números Rational. El resultado de la división se debe almacenar en forma reducida.
e) Impresión de números Rational en la forma a/b , donde a es el numerador y be es el denominador.
f) Impresión de números Rational en formato de punto flotante. (Considere ofrecer capacidades de formateo que permitan al usuario de la clase especificar el número de dígitos de precisión a la derecha del punto decimal)

```
// Figura 6.5 Time.java
// Definición de la clase Time

public class Time{
     private int hour;    // 0 - 23
     private int minute;  // 0 - 59
     private int second;  // 0 - 59

     // El constructor de Time inicializa cada variable de
     // ejemplar en cero. Esto asegura que el objeto Time inicia
     // en un estado consistente.
     public Time() { setTime( 0, 0, 0 ); }

     // Constructor de Time: se especifíca hour, minute y
     // second es 0 por omisión.
     public Time( int h ) { setTime( h, 0, 0 ); }

     //  Constructor de Time: se especifíca hour y minute;
     // second es 0 por omisión.
     public Time( int h, int m ) { setTime( h, m, 0 ); }

     //  Constructor de Time: se especifíca hour, minute y second.
     public Time( int h, int m, int s ) { setTime( h, m, s ); }

     // Métodos set
     // Fijar un nuevo valor de Time empleando hora militar
     // Verificar la validez de los datos. Fijar en cero los
     // datos no válidos.
     public void setTime( int h, int m, int s )
     {
          setHour( h );
          setMinute( m );
          setSecond( s );
     }

     // poner la hora
     public void setHour( int h )
     { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }

     // poner el minuto
     public void setMinute( int m )
     { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }
}
```

```
      // poner el segundo
      public void setSecond( int s )
      { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }

      // Métodos get
      // obtener la hora
      public int getHour() { return hour; }

      // obtener el minuto
      public int getMinute() { return minute; }

      // obtener el segundo
      public int getSecond() { return second; }

      // convertir Time en String en formato de hora militar
      public String toMilitaryString()
      {
            return ( hour < 10 ? "0" : "" ) + hour +
                   ( minute < 10 ? "0" : "" ) + minute;
      }

      // convertir Time en String en formato de hora estándar
      public String toString()
      {
            return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
                   ":" + ( minute < 10 ? "0" : "" ) + minute +
                   ":" + ( second < 10 ? "0" : "" ) + second +
                  ( hour < 12 ? "AM" : "PM" );
      }

}
```

```
import java.awt.*;
import java.applet.Applet;

public class TimeTest extends Applet {
      private Time t;
      private Label hrLabel, minLabel, secLabel;
      private TextField hrField, minField, secField, display;
      private Button tickButton;

      public void init()
      {
            t = new Time();

            hrLabel = new Label ( "Poner Hora" );
            hrField = new TextField ( 10 );
            minLabel = new Label ( "Poner minuto" );
            minField = new TextField ( 10 );
            secLabel = new Label ( "Poner segundo" );
            secField = new TextField ( 10 );
            display = new TextField ( 30 );
            display.setEditable ( false );
            tickButton = new Button ( "Agregar 1 segundo" );

            add( hrLabel );
            add( hrField );
            add( minLabel );
            add( minField );
            add( secLabel );
            add( secField );
            add( display );
            add( tickButton );
            updateDisplay();
      }

      public boolean action( Event e, Object o )
      {
            if ( e.target == tickButton)
                  tick();
            else if ( e.target == hrField ){
                  t.setHour( Integer.parseInt( e.arg.toString() ) );
                  hrField.setText( "" );
            }
            else if ( e.target == minField ){
                  t.setMinute( Integer.parseInt( e.arg.toString() ) );
                  minField.setText( "" );
            }
            else if ( e.target == secField ){
                  t.setSecond( Integer.parseInt( e.arg.toString() ) );
                  secField.setText( "" );
            }

            updateDisplay();

            return true;
      }
```

```
      public void updateDisplay()
      {
            display.setText( "Hora: " + t.getHour() +
               "; Minuto: " + t.getMinute() +
               "; Segundo: " + t.getSecond() );
            showStatus( "La hora estandar es: " + t.toString() +
               "; La hora Militar es: " + t.toMilitaryString() );
      }

      public void tick()
      {
            t.setSecond( ( t.getSecond() + 1 ) % 60 );

            if ( t.getSecond() == 0 ) {
                  t.setMinute( ( t.getMinute() + 1 ) % 60 );
                  if ( t.getMinute() == 0 )
                        t.setHour( ( t.getHour() + 1 ) % 24 );
            }
      }
}
```

6.4 Modifique la clase Time de la figura 6.5 a modo de incluir el método tick (“tic”) que incrementa el tiempo almacenado en un objeto Time en un segundo. Proporcione también el método incrementMinute para incrementar el minuto y el método increméntHour para incrementar la hora. El objeto Time siempre deberá quedar en un estado consistente. Escriba un programa que pruebe los métodos tick, incrementMinute e incrementHour a fin de comprobar que funcionan correctamente. Asegúrese de probar los siguientes casos:

a) Incrementar pasando al siguiente minuto.
b) Incrementar pasando a la siguiente hora.
c) Incrementar pasando al siguiente día (ej. 11:59:59 PM a .
:00:00 AM)

6.7

```
      public void SetTime(int h, int m, int s)
      {
      SetHour( h);
      SetMinute(m);
      SetSecond(s);
      }

      public void SetHour(int h)
      {
      if (h>=0 && h<=24)
            hour = h;
      else
            Sistem.out.println("El dato no esta entre el rango de la hora");
      }

      public void SetMinute(int m)
      {
      if(m>=0 && m<=60)
            minute = m;
      else
            Sistem.out.println("El dato no esta en el rango de los minutos");
```

```
      };

      public void SetSecond(int s)
      {
      if(s>=0 && s<=60)
            second = s;
      else
            Sistem.out.println("El dato no esta en rango de los segundos");

      }
```

6.8

```
import java.applet.Applet;
import java.awt.*;

public class Rectangulo2 Applet
{

      private Double x1,y1,x2,y2;

      public void init()
      {
            x1=0;
            y1=0;
            x2=0;
            y2=0;
      }


      public void SetAlto(Double x, Double y)
      {
            if((x>0)&&(x<=20)&&(y>0)&&(y<=20))
            {
                  x1=x;
                  y1=y;
            }
            else
                  System.Out.println("Error... las coordenadas deben ser mayores
que 0 y menores o igual que 20");
      }

      public void SetAncho( Double x, Double y)
      {
            if((x>0)&&(x<=20)&&(y>0)&&(y<=20))
            {
                  x2=x;
                  y2=y;
            }
            else
                  System.out.println("Error... las coordenadas deben ser mayores
que 0 y menores o igual que 20");
      }

      public void SetRectangulo( Double x, Double y, Double a, Double b)
      {
```

```
            SetAlto(x,y);
            SetAncho(a,b);
      }

      public Double Longitud()
      {
            return (x2-x1);
      }

      public Double Anchura()
      {
            return (y2-y1);
      }

      public Double Perimetro()
      {
            return ((Longitud()+Anchura())*2);
      }

      public Double Area()
      {
            return (Longitud()*Anchura());
      }

      Boolean Double EsRectangulo()
      {
            return (Longitud() != Anchura());
      }
}
```

6.9

```
import java.applet.Applet;
import java.awt.*;

public class Rectangulo2 Applet
{

      private Double x1,y1,x2,y2;

      public void init()
      {
            x1=0;
            y1=0;
            x2=0;
            y2=0;
      }


      public void SetAlto(Double x, Double y)
      {
            if((x>0)&&(x<=20)&&(y>0)&&(y<=20))
            {
                  x1=x;
                  y1=y;
            }
            else
```

```
                System.Out.println("Error... las coordenadas deben ser mayores
que 0 y menores o igual que 20");
      }

      public void SetAncho( Double x, Double y)
      {
            if((x>0)&&(x<=20)&&(y>0)&&(y<=20))
            {
                  x2=x;
                  y2=y;
            }
            else
                  System.out.println("Error... las coordenadas deben ser mayores
que 0 y menores o igual que 20");
      }

      public void SetRectangulo( Double x, Double y, Double a, Double b)
      {
            SetAlto(x,y);
            SetAncho(a,b);
      }

      public Double Longitud()
      {
            return (x2-x1);
      }

      public Double Anchura()
      {
            return (y2-y1);
      }

      public Double Perimetro()
      {
            return ((Longitud()+Anchura())*2);
      }

      public Double Area()
      {
            return (Longitud()*Anchura());
      }

      Boolean Double EsRectangulo()
      {
            return (Longitud() != Anchura());
      }
}
6.10
import java.applet.Applet;
import java.awt.*;

public class Rectangulo3 Applet
{

      private Double x1,y1,x2,y2;
      Button Boton;
```

```
      TextField Edit1,Edit2,Edit3,Edit4;

      public void init()
      {
            Boton = new Button("Aceptar");
            Edit1 = new TextField(12);
            Edit2 = new TextField(12);
            Edit3 = new TextField(12);
            Edit4 = new TextField(12);
            x1=0;
            y1=0;
            x2=0;
            y2=0;
      }


      public void SetAlto(Double x, Double y)
      {
            if((x>0)&&(x<=20)&&(y>0)&&(y<=20))
            {
                  x1=x;
                  y1=y;
            }
            else
                  System.Out.println("Error... las coordenadas deben ser mayores
que 0 y menores o igual que 20");
      }

      public void SetAncho( Double x, Double y)
      {
            if((x>0)&&(x<=20)&&(y>0)&&(y<=20))
            {
                  x2=x;
                  y2=y;
            }
            else
                  System.out.println("Error... las coordenadas deben ser mayores
que 0 y menores o igual que 20");
      }



      public Double SetRectangulo( Double x, Double y, Double a, Double b)
      {
            SetAlto(x,y);
            SetAncho(a,b);
      }

      public Double Longitud()
      {
            return (x2-x1);
      }

      public Double Anchura()
      {
            return (y2-y1);
```

```
	}

	public Double Perimetro()
	{
		return ((Longitud()+Anchura())*2);
	}

	public Double Area()
	{
		return (Longitud()*Anchura());
	}

	Boolean EsRectangulo()
	{
		return (Longitud() != Anchura());
	}


	public boolean action(Event e, Object o)
	{

		if(e.target == Boton)
		{
			x1=Double.parseDouble(Edit1.getText());
			y1=Double.parseDouble(Edit2.getText());
			x2=Double.parseDouble(Edit3.getText());
			y2=Double.parseDouble(Edit4.getText());
			repaint();
		}
		return true;
	}

	public void paint( Graphics g)
	{
		g.drawRect(x1,y1,x2,y2);
	}
}
```

6.15

```
public class Fecha2
{
	private int dia;   // 1-31 depende del mes
	private int mes;   // 1-12
	private int año;   // cualquiera

	private int analizarDia(int d)
	{
		int DiasPorMes[] = {0,31,28,31,30,
				    31,30,31,31,30,
				    31,30,31};       // el 0 es por el vector empieza
desde 0.

		if(d > 0 && d <= DiasPorMes[mes])
```

```
                return d;

          if(mes == 2 && d == 29 && (año % 400 == 0 || (año % 4 == 0 && año %
100 != 0)))
                return d;  // si es febrero hay que ver si es bisiesto

          return 1;

     }

     public Fecha2()
     {
          ponerFecha(1,1,1);
     }


     public Fecha2(int d)
     {
          ponerFecha(d,1,1);
     }


     public Fecha2(int d, int m)
     {
          ponerFecha(d,m,1);
     }


     public Fecha2(int d, int m, int a)
     {
          ponerFecha(d, m, a);
     }



     public void ponerFecha( int d, int m, int a)
     {
          if(m >= 1 && m <= 12)
               mes = m;

          dia = analizarDia(d);

          if(a > 0)
              año = a;

     }


     public String Mostrar1()
     {
          return mes + "/" + dia + "/" + año;
     }

     public String Mostrar2()
     {
          String m = "";
```

```
            switch (mes)
            {
                  case 1: m = "Enero";
                        break;
                  case 2: m = "Febrero";
                              break;
                  case 3: m = "Marzo" ;
                        break;
                  case 4: m = "Abril" ;
                        break;
                  case 5: m = "Mayo" ;
                        break;
                  case 6: m = "Junio" ;
                        break;
                  case 7: m = "Julio" ;
                        break;
                  case 8: m = "Agosto" ;
                        break;
                  case 9: m = "Septiembre" ;
                        break;
                  case 10: m = "Octubre" ;
                        break;
                  case 11: m = "Noviembre" ;
                        break;
                  case 12: m = "Diciembre" ;
                        break;

            }

            return m + "  " + dia + "," + "  " + año;

      }

      public String Mostrar3()
      {
            return dia + "  " + año;
      }

}


import java.applet.Applet;
import java.awt.*;

public class P615b extends Applet
{
      private Fecha2 f1,f2,f3,f4;


      public void init()
      {
            f1 = new Fecha2();
            f2 = new Fecha2(3);
            f3 = new Fecha2(16, 5);
            f4 = new Fecha2(16, 5, 2001);
```

```
    }


    public void paint(Graphics g)
    {
         g.drawString(f1.Mostrar1(), 25, 25);
         g.drawString(f1.Mostrar2(), 25, 50);
         g.drawString(f1.Mostrar3(), 25, 75);

         g.drawString(f2.Mostrar1(), 25, 100);
         g.drawString(f2.Mostrar2(), 25, 125);
         g.drawString(f2.Mostrar3(), 25, 150);

         g.drawString(f3.Mostrar1(), 25, 175);
         g.drawString(f3.Mostrar2(), 25, 200);
         g.drawString(f3.Mostrar3(), 25, 225);

         g.drawString(f4.Mostrar1(), 25, 250);
         g.drawString(f4.Mostrar2(), 25, 275);
         g.drawString(f4.Mostrar3(), 25, 300);

    }


}
```

Resumen:

- El acceso a los miembros de una clase se logra empleando el operador de acceso a miembros: el operador de punto (.)
- Las clases permiten al programador modelar objetos con atributos y comportamientos. Los tipos de clase se definen en Java empleando la palabra clave **class**.
- Las definiciones de clase comienzan con la palabra clave **class**. El cuerpo de la definición de la clase se encierra en llaves {y}.
- Cualquier variable de ejemplar o método declarado **public** en una clase está visible para cualquier método con acceso a un objeto de la clase.
- Cualquier variable de ejemplar o método declarado como **private** sólo está visible para los demás miembros de la clase.
- Los modificadores de acceso a miembros pueden aparecer varias veces y en cualquier orden en una definición de clase.
- Un constructor es un método especial con el mismo nombre que la clase y que sirve para inicializar los miembros de un objeto de esa clase. Los constructores se invocan cuando se ejemplarizan objetos de sus clases.
- El conjunto de métodos **public** de una clases se conoce como la interfaz publica de la clases.

- La invocación de métodos es más concisa que la invocación de funciones en los lenguajes de programación prodecimientales porque la mayor parte de los datos que el método usa ya están en el objeto.
- Dentro del alcance de una clase, podemos hacer referencia a los miembros de la clases simplemente con sus nombres. Fuera del alcance de unas clase, hacemos referencia los miembros de la clase mediante una referencia a un objeto.
- El operador de selección de miembros sirve para acceder a los miembros de una clase.
- Los miembros de clase públicos presentan una visión de los servicios que la clase ofrece a sus clientes.
- Para que los clientes puedan leer datos **private**, la clase puede proveer métodos get(obtener). Para que los clientes puedan modificar datos **private**, la clase puede proveer métodos set(fijar).
- Las variables de ejemplar de una clases normalmente se hacen **private** y los métodos de la clase normalmente se hacen **public**. Algunos pueden ser **private** y actuar como métodos de utilidad para los demás métodos de la clase.
- Los constructores pueden sobrecargarse.
- Una vez debidamente inicializada un objeto de una clase, todos los métodos que manipulen el objeto deberán asegurar que el objeto permanezca en un estado consistente.
- Cuando se declara un objeto de una clase, se pueden incluir inicializadores. Estos inicializadores se pasan al constructor de la clase.
- Los constructores no pueden especificar tipos devueltos, ni pueden intentar devolver valores.
- Un finalizador realiza aseo al terminar antes de que la función de recolección de basura del sistema recupere la memoria ocupada por el objeto.
- La palabra clave **final** especifica que un objeto no es modificable.
- Un objeto **final** debe inicializarse en su declaración.
- Las clases pueden estar compuestas por objetos de otras clases.
- La referencia **this** se utiliza implícitamente para hacer referencia tanto a métodos como a variables de ejemplar desde el interior de un objeto.
- El operador **new** crea automáticamente un objeto del tamaño apropiado y devuelve una referencia de tipo correcto.
- Una variable de clase estática representa información que abarca toda la clase. La declaración de un miembro estático comienza ocn la palabra clave **static**.
- Las variables de clase estáticas tienen alcance de clase.
- Un método declarado **static** no puede acceder a miembros de clase no estáticos. Un método **static** no tiene referencia **this** porque las variables de clase static y los métodos **static** existen independientemente de que existan o no objetos de la clase.