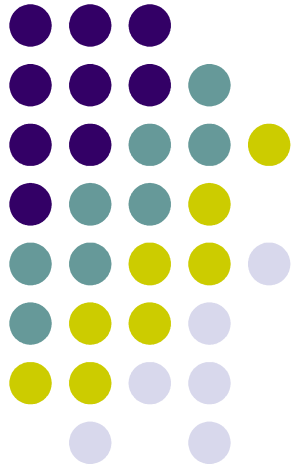
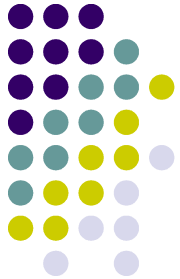
Modificadores



Modificadores – Introducción



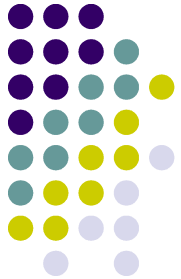
Los *modificadores* son palabras reservadas que proveen información al compilador acerca de la naturaleza del código, datos o clases.

Los modificadores especifican las, por ejemplo, que una *característica* es de tipo *static, final* o *transient.*

Una característica es una clase, un método o una variable.

Un grupo de modificadores, llamados *modificadores de acceso*, indican qué clases tienen permitido utilizar una característica.

Modificadores – Introducción

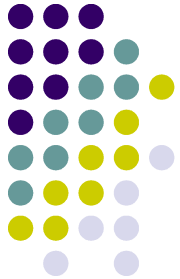


Los modificadores más comunes son los de acceso: public, protected y private.

El resto de los modificadores no se pueden categorizar.

final
abstract
static
native
transient
synchronized
volatile

Modificadores de Acceso



Los modificadores de acceso determinan qué clases pueden utilizar cierta característica.

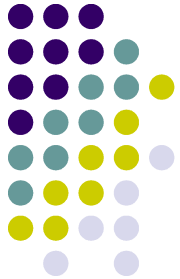
Las características de una clase son:

- La clase misma.
- Sus variables.
- Sus métodos y constructores.

Las únicas variables sobre las que se aplican los modificadores de acceso son las variables de clase.

Las variables de los métodos no llevan modificadores de acceso porque pertenecen únicamente al método.

Modificadores de Acceso



Los modificadores de acceso son:

public
private
protected

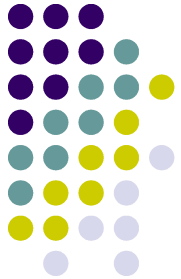
El único modificador de acceso permitido para una clase no interna es public.

Una característica puede tener como máximo un modificador.

Cuando una característica no tiene modificador de acceso se toma un valor por defecto.

El valor por defecto no tiene un nombre estándar (*friendly*, *package* o *default*).

Modificadores de Acceso: ejemplo



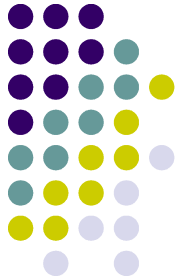
Ejemplo, declaraciones permitidas:

```
class Algo { ... }
public class Objeto { ... }
private int i;
Set conjunto;
protected double metodo() { ... }
```

Declaraciones no permitidas:

```
public protected int x;
default Objeto getObjeto() { ... }
```

Modificadores de Acceso: public

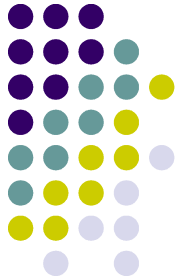


El modificador de acceso más permisivo es public.

Cualquier clase, variable o método publico se puede utilizar en cualquier clase en Java.

Las aplicaciones Java deben declarar el método main().

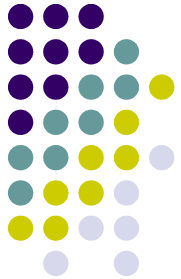
Modificadores de Acceso: private



El modificador private es el más restrictivo.

Una variable o método de tipo privado sólo es accesible para las instancias creadas por la misma clase.

Modificadores de Acceso: default

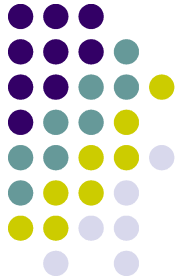


Las clases, variables o métodos que no tienen modificador de acceso toman un valor llamado default.

El modificador de acceso default permite que las características de una clase sean accesibles para cualquier clase en el mismo paquete.

Una clase que hereda de otra no tiene acceso a las características default del padre si no se encuentra en el mismo paquete.

Modificadores de Acceso: protected



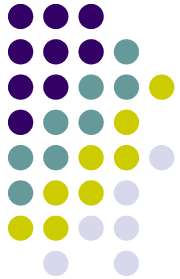
Las características que tienen el modificador de acceso protected son más accesibles que las default.

Únicamente los métodos y las variables se pueden declarar como protected.

Las características de tipo protected de una clase son accesibles desde las clases que heredan.

Al igual que default, protected permite el acceso a las clases del mismo paquete.

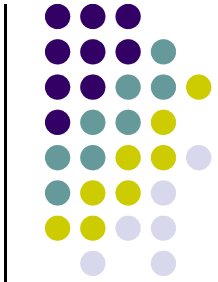
Modificadores de Acceso: Resumen



Los modificadores de acceso en Java son:

- public – una característica public es accesible para cualquier clase.
- protected – una característica protected puede ser accedida desde una clase que hereda o desde cualquier clase que pertenece al mismo paquete.
- default – las características default son accesibles para las clases del mismo paquete.
- private – una característica private sólo es visible para la clase a la cual pertenece.

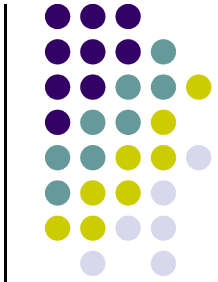
Otros modificadores



Los modificadores que no son de acceso son los siguientes:

- final
- abstract
- static
- native
- transient
- synchronized
- volatile

Modificadores: final



El modificador final se puede aplicar a clases, métodos y variables.

Dependiendo del contexto, el significado del modificador varía.

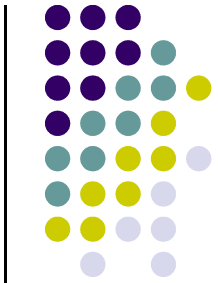
La idea central del modificador es simple: las características de tipo final no pueden ser modificadas.

Una clase final no puede tener herencia.

Una variable de tipo final no puede ser modificada una vez que se le haya asignado un valor.

Un método final no puede ser redefinido por una clase que herede.

Modificadores: abstract



El modificador abstract se aplica a clases y métodos.

Las clases abstractas no pueden ser instanciadas.

Una clase con uno o más métodos abstractos debe ser abstracta.

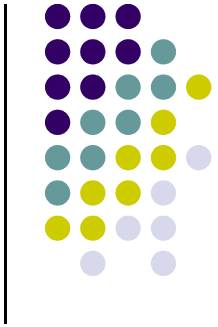
Una clase que herede de un padre abstracto y que no implante todos sus métodos abstractos debe ser abstracta.

abstract es lo opuesto a final porque las clases abstractas deben tener herencia para ser útiles.

Para los métodos abstractos se define únicamente la firma:

```
abstract void metodo(int x);
```

Modificadores: static



El modificador static se aplica a métodos, variables y bloques de código que no pertenecen a un método.

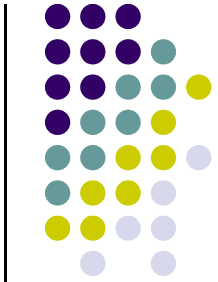
Las características static pertenecen a la clase, no a las instancias de ésta.

```
class Estatica {
   static int x;
   public Estatica()
   {
      x++;
   }
}
```

La variable x es estática, esto implica que sólo existe una copia de x para todas las instancias de la clase.

Puede haber una, N o ninguna instancia de la clase Estatica, sin embargo, siempre hay un valor y una referencia para x.

Modificadores: static



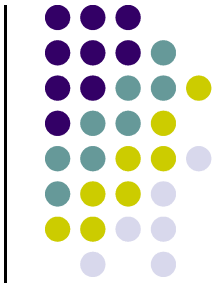
Los métodos también pueden ser declarados static.

Los métodos estáticos no tienen acceso a las características no estáticas de la clase.

Los métodos estáticos no realizan operaciones sobre objetos, por lo que pueden ser llamados sin crear una sola instancia de la clase.

El método main es un ejemplo.

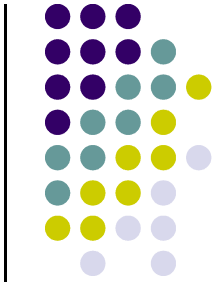
Modificadores: static



El tercer uso de static es para inicializar bloques de código.

```
public class Estatica {
    static int x = 1;
    static {
       System.out.println(“Hola … 1“ + x++);
    }

    public static void main(String args[])
    {
       System.out.println(“Hola … 2“ + x++);
    }
}
```

Modificadores: native



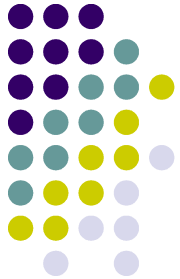
El modificador native se emplea únicamente con métodos.

native indica que la implantación del método no se encuentra en la misma clase, sino fuera de Java.

native sirve para llamar a funciones escritas en otros lenguajes de programación, usualmente C.

La implantación de dichas funciones depende de la plataforma en la cual se desarrolla.

Modificadores: transient, synchronized y volatile.



transient se aplica a variables.

Una variable transient no se almacena como parte del objeto cuando éste se persiste.

El modificador synchronized se emplea para controlar el acceso a código crítico en una aplicación multi-hilos.

El modificador volatile indica que una variable puede ser modificada de forma asíncrona. Esta condición se puede presentar en sistemas con procesadores múltiples.

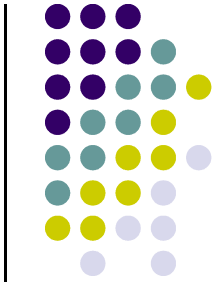
Modificadores

Modificador	Clase	Variable	Método	Constructor	Bloque de código
public	sí	sí	sí	sí	no
protected	no	sí	sí	sí	no
(default)*	sí	sí	sí	sí	no
private	no	sí	sí	sí	no
final	sí	sí	sí	no	no
abstract	sí	no	sí	no	no
static	no	sí	sí	no	sí
native	no	no	sí	no	no
transient	no	sí	no	no	no
volatile	no	sí	no	no	no
synchronized	no	no	sí	no	sí

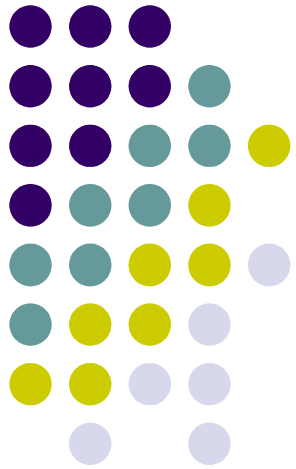
* default no se refiere a una palabra reservada en Java.

Modificadores

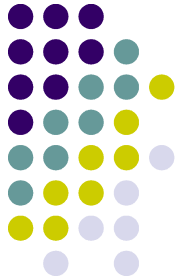
Fin ...



Conversión y Casting



Conversión de primitivos

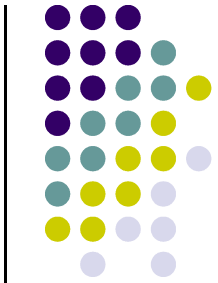


Los tipos de datos primitivos de Java pueden ser convertidos en otros tipos.

La conversión de un primitivo se puede dar en los siguientes casos.

- Asignación.
- Llamada a método.
- Promoción aritmética.

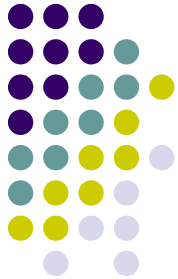
Conversión de primitivos: Asignación



La conversión de asignación se da cuando se asigna un valor a una variable de un tipo distinto al del valor original.

```
// Legal
int i;
double d;
i =10;
d = 10; // asigna un tipo int a una variable double

// Ilegal
double d;
short s;
double d = 1.2345;
s = d; // asigna un tipo double a una variable short
```

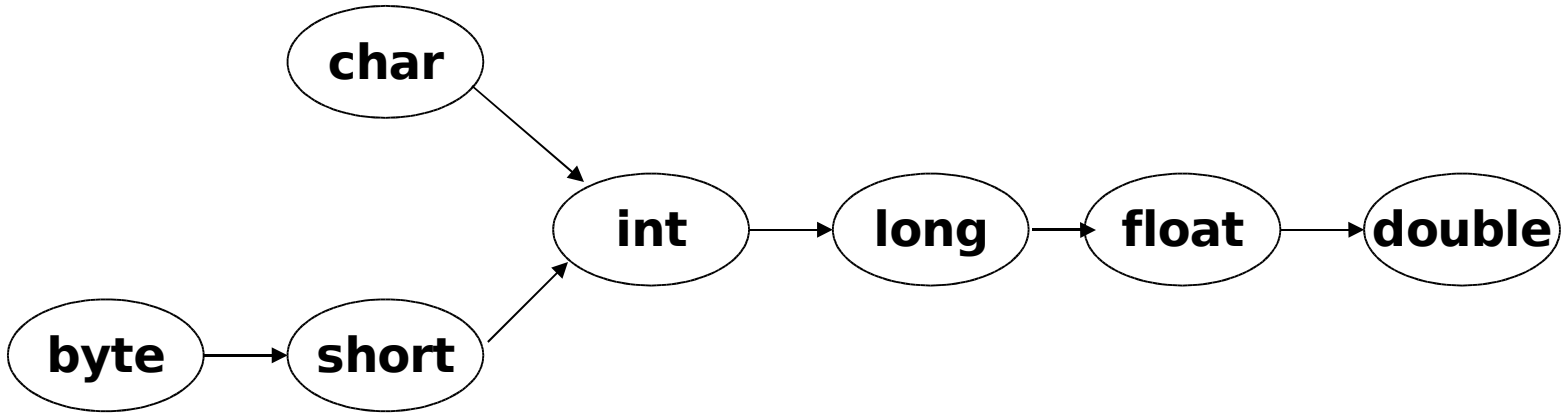
Conversión de primitivos: Asignación



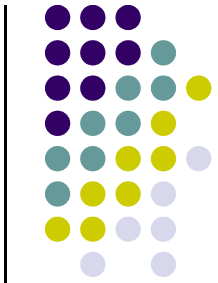
Las reglas generales para la conversión de tipos en asignaciones son las siguientes:

- Un tipo boolean no puede ser convertido a ningún otro tipo.
- Un tipo no booleano puede ser convertido a otro tipo no booleano siempre y cuando el segundo tipo tenga mayor tamaño (widening conversion).
- Un tipo no booleano no puede ser convertido a otro tipo no booleano cuando el segundo tipo tenga un tamaño menor (narrowing conversion).

Conversiones widening válidas



Conversión de primitivos en métodos

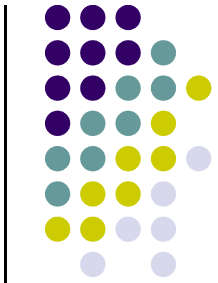


Una conversión de primitivos en un método se da cuando se pasa un valor como argumento con un tipo distinto al esperado.

Por ejemplo, la clase Math define el método estático cos(),el cual recibe como argumento un double.

```
float rads;
double d;
rads = 1.2345f;
d = Math.cod(rads);
```

Conversión de primitivos: promoción aritmética



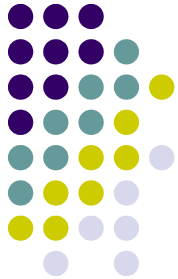
El último tipo de conversión es la promoción aritmética.

La promoción aritmética se da en sentencias con operadores aritméticos.

Ejemplo:

```
short s = 9;
int i = 10;
float f = 11.1f;
double d = 12.2;
if( -s * i >= f / d)
   System.out.println(“>>>>>>>>>>>>>”);
else
   System.out.println(“<<<<<<<<<<<<<”);
```

Promoción aritmética



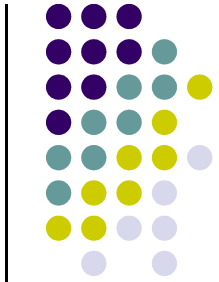
Las reglas que se aplican a la promoción aritmética dependen del tipo de operador: unario o binario.

Los operadores unarios se aplican a un solo valor.

Los operadores binarios requieren dos valores.

Operadores unarios	+	-	++	--	~			
Operadores binarios	+	-	*	/	%	>>	>>	<<
		&	^	|			>	

Promoción aritmética

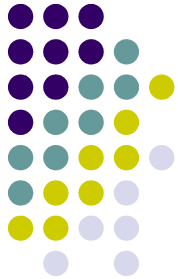


Para los operadores unarios se aplican dos reglas:

Si el operando es de tipo byte, short o char, se convierte a un int (con excepción de ++ y --).

De lo contrario, no hay conversión.

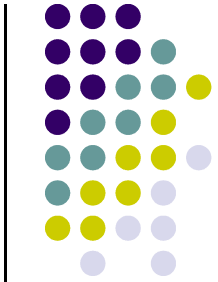
Promoción aritmética



Para los operadores binarios se aplican las siguientes reglas:

- Si uno de los operandos es de tipo double, el otro operando es convertido a double.
- Si uno de los operandos es de tipo float, el otro operando es convertido a float.
- Si uno de los operandos es de tipo long, el otro operando es convertido a long.
- De lo contrario, ambos operandos son convertidos a int.

Primitivos y casting



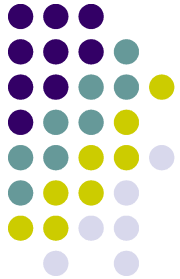
En los casos vistos previamente, Java hace automáticamente la conversión de datos.

casting significa indicarle al compilador explícitamente que se requiere una conversión.

Para hacer una operación de casting se usa el tipo de dato entre paréntesis.

```
int i = 5;
double d = (double)i;

// igual
int i = 5;
double d = i;
```

Casting

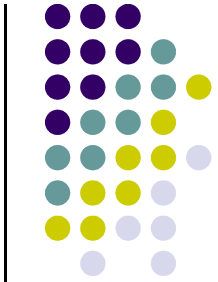


Las operaciones de casting son necesarias cuando el tipo de dato al cual se va a convertir es más chico que el tipo original.

Este tipo de conversión no se hace implícitamente.

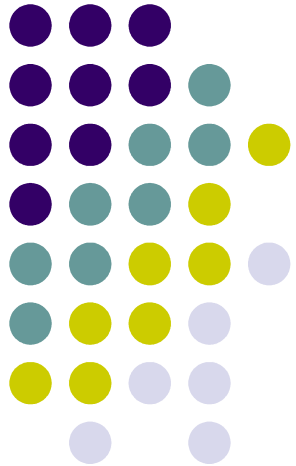
```
// Ilegal
short s = 259;
byte b = s;                          // error
System.out.println(“s = ” + s + “, b = “ + b);

// Legal
short s = 259;       // 100000011
byte b = (byte)s;              // casting explícito
System.out.println(“s = ” + s + “, b = “ + b);
```

Conversión y casting

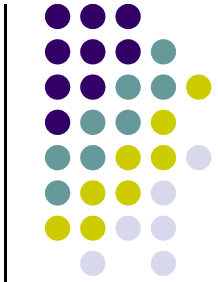


Fin

Control de flujo y excepciones



Ciclos

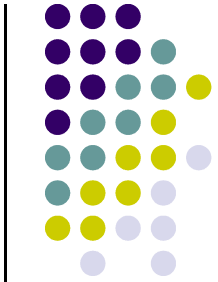


Existen 5 construcciones que operan en ciclos:

- while
- do
- for
- continue
- break

while

```
while(condición boolean)
    bloque o sentencia

int 1 = 0;
while(i <= 5)
    System.out.println(“cinco veces”);
System.out.println(“una vez”);

int 1 = 0;
while(i <= 5)
{
    System.out.println(“cinco veces”);
    System.out.println(“una vez”);
}
```



do – while

```
do
    ciclo
while(condición booleana)

do
{
    ciclo
}
while(condición booleana)
```

El ciclo do – while se ejecuta por lo menos una vez.

for

```
for( sentencia ; condición ; expresión)
    ciclo

for( sentencia ; condición ; expresión)
{
    ciclo
}
```

La sentencia se ejecuta una vez antes del ciclo.

La condición debe ser una expresión booleana, similar a la condición de while. El ciclo se ejecuta repetidamente mientras la condición tenga el valor true.

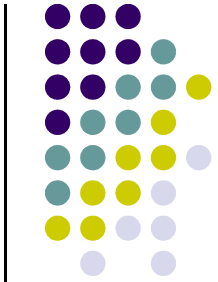
La expresión se ejecuta inmediatamente después del bloque del ciclo.

continue

continue se emplea para salir de ciclos anidados o para interrumpir un bloque dentro de un ciclo.

```
salida: for(i = 0 ; i < 5 ; i++) {
                    for(j = 0 ; j < 10 ; j++) {
                                if( (i*2) == j ) {
                                            continue salida;
                                }
                    }
          }

while( i < 15 ) {
    if( (i % 2) == 0 )
          continue;
    System.out.println(“i : “ + i );
    i++;
}
```

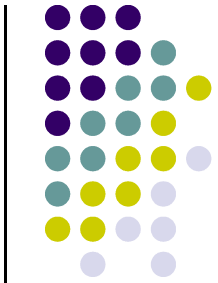
break



break se utiliza para interrumpir ciclos.

```
j = 0;
while(j < 10)
{
      if(j == 7)
      {
            break;
      }
      System.out.println(“j : “ + j );
      j++;
}
```

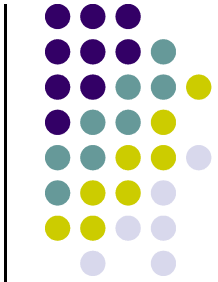
Sentencias de selección



En Java existe dos tipos de sentecnias de selección:

- if / else
- switch

if /else

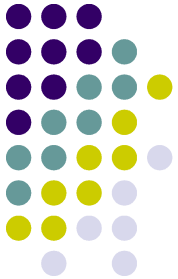


La estructura condicional if / else toma una sentencia booleana como condición.
Se ejecuta únicamente un bloque.
Tres tipos de combinaciones para estructuras if.

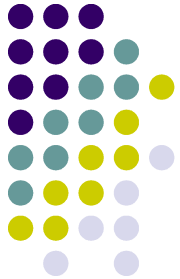
```
if(condición)
{
        bloque
}

if(condición)
{
        bloque1
}
else
{
        bloque2
}
```

if

```
if(condición)
{
        bloque1
}
else if(condición2)
{
        bloque2
} // ...
else if(condiciónN)
{
        bloqueN
}
else
{
        bloque 3
}
```



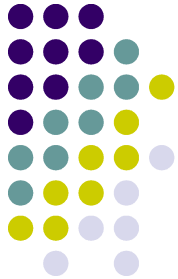
switch



La estructura de selección switch tiene la siguiente estructura:

```
swith(valor int)
{
	case 1:
		// 1
	case 2:
		// 2
	break;
	case 3:
		// 3
	break;
	default:
		// bloque defecto (opcional)
	brea;
}
```

Excepciones



Las excepciones indican anormalidades en la ejecución de una aplicación.

Interrumpen el flujo de la aplicación.

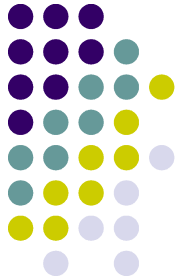
Se pueden *cachar* con un bloque try {} catch {}.

Hay dos tipos de excepciones a considerar java.lang.Exception y java.lang.RuntimeException.

Las primeras deben ser cachadas, las segundas no.

Con el manejo de excepciones se separa el ódigo con la lógica del código de manejo de

Excepciones



```
try
{
    // bloque con excepción
}
catch(Exception e)
{
    // bloque manejo error
}
finally
{
    // siempre se ejecuta
}
```