N REINAS
(BACKTRACK)
2002

GRUPO # 22

Alumnos:

Aguilar Elba
Barrios Miguel
Camacho Yaquelin
Ponce Rodríguez Jhonny

Práctica 1: Programación orientada a objetos (el lenguaje java)

1. Objetivos

En esta práctica el alumno se familiarizará con el uso del lenguaje de programación java y el entorno de desarrollo JDK 1.3. Para ello en la práctica se desarrollará:

- Una aplicación de consola, cuyo objetivo es aprender cómo java implementa los conceptos más importantes de la orientación a objetos.
- Una aplicación con interfaz gráfica de usuario cuyo objetivo es aprender los fundamentos básicos del desarrollo de aplicaciones java con las clases del paquete Swing.

2. Entorno de trabajo

La práctica se realizará en `merlin`, donde se creará un directorio de nombre `/insii/practica1`.

Para el manejo del entorno JDK 1.3, sun® dispone de una completa documentación [1] que puede ser consultada a través de la red.

Para crear los ficheros fuente de las aplicaciones java (`nomClase.java`) puede utilizarse cualquiera de los editores instalados en `merlin`: vi, emacs, nedit, etc.

Las siguientes secciones describen las herramientas, variables, etc. a tener en cuenta para desarrollar en JDK 1.3.

2.1. La API de java

Java proporciona un extenso conjunto de clases e interfaces conocido como Application Program Interface (API). Estas clases e interfaces resultan imprescindibles a la hora de programar en java. Será por tanto necesario tener abierto un navegador con la página web [2] que documenta la API de java.

A modo de ejemplo buscar en dicha página el paquete `javax.swing` y en él la clase `JDialog` que será util para crear diálogos tanto modales como no modales. Dedicar algún minuto a familiarizaros con la documentación que proporciona sobre sus atributos (`Field Detail`), constructores (`Constructor Detail`) y métodos (`Method Detail`).

2.2. Las herramientas de JDK

JDK tiene herramientas y utilidades que te resultarán necesarias para compilar y depurar tus fuentes java, ejecutar las aplicaciones, crear documentación, etc.

En esta primera práctica básicamente usarás:

- javac, compila los fuentes java (`.java`) en bytecodes (`.class`).
  En la página web [3] tienes una documentación sencilla y precisa de esta herramienta.
- java, el interprete que ejecuta aplicaciones java.
  En la página web [4] tienes una documentación sencilla y precisa de esta herramienta.

2.3. Variables del entorno de java

- Las herramientas de JDK (compilador, depurador, etc.) están en el directorio `dir_bin_java`, si es necesario actualiza la variable `PATH` de tu fichero de inicio (`.profile` si usas kshell, si usas otro shell entonces el fichero correspondiente):

  ```
  PATH=$PATH:dir_bin_java
  ```

  donde `dir_bin_java` puedes obtenerlo con la orden `which java`
- La variable de entorno `CLASSPATH` es utilizada por las herramientas de JDK para identificar la ubicación de las clases de java compilado. Por defecto tanto las clases de la API de java como las del directorio de trabajo actual son accesibles a las herramientas sin necesidad de actualizar dicha variable. Cuando sea necesario añádela en tu fichero de inicio (ejemplo con kshell):

  ```
  export CLASSPATH=lista_directorios
  ```

  donde `lista_directorios` son los directorios en los que decidas ubicar tus clases java compiladas, por ejemplo, `$HOME/insii/practica1`.

2.4. Demos

Puede ser interesante que antes de escribir tu primera aplicación java intentes ejecutar y comprender el código de alguna de las aplicaciones de demostración que puedes encontrar en el directorio de `merlin`, `/opt/java1.3/demo`.

2.5. Resumen del lenguaje java

En reprografía puedes encontrar con el nombre “Resumen del lenguaje java” un documento que resume la sintaxis del mismo, extraído de [5]. Además, la página web [1] contiene diversos tutoriales.

3. Aplicación de consola

El conocido problema de las ocho reinas servirá de ejemplo sencillo para realizar la primera aplicación en java que permita tomar contacto con los elementos básicos del lenguaje, el estilo de la programación orientada a objetos y el entorno de desarrollo.

El problema consiste en situar ocho reinas en un tablero de ajedrez de modo que no se ataquen entre sí, la Figura 1 muestra una posible solución.

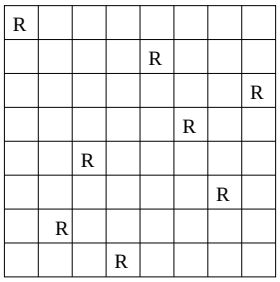

Figura 1: Una solución.

La solución orientada a objetos [6] consistirá en crear las reinas y dotarlas del comportamiento para que ellas mismas descubran la solución. Cada reina puede ser asignada a una columna, quedando el problema reducido a que cada reina encuentre su fila apropiada. Para ello una reina sólo necesita enviar mensajes a su vecina de un lado (p.e. el izquierdo) consultando si en la posición en la que se encuentra puede ser atacada (función `puedeAtacar(f,c)`).

Una *solución aceptable* para una columna `n` será una configuración de las columnas `1` a `n` en la que ninguna reina puede atacar a otra reina en tales columnas. La solución al problema en su conjunto consiste en pedir a la reina del extremo derecho que encuentre una solución aceptable (función primera()).

Cuando se pide a una reina en la columna `n` que produzca una solución aceptable, deberá pedir a su vecina que produzca una solución aceptable para la columna `n-1`. La primera columna es una caso especial, mientras que cualquier otra reina debe consultar si puede ser atacada (función pruebaOAvanza()) en la posición en la que se encuentra, si es así debe avanzar a una nueva posición y consultar de nuevo si puede ser atacada (función siguiente()). Si la reina ha llegado a la última fila y no se encuentra en una posición segura pedirá a su vecina que explore la posición siguiente.

Por útimo se imprime la solución (procedimiento `imprimir`).

El perfil de las funciones necesarias para resolver el problema aparece a continuación, así como el seudocódigo de `pruebaOAvanza()` e `imprimir()`.

```
función puedeAtacar(f,c): booleana;
función primera():booleana;
función siguiente():booleana;

función pruebaOAvanza(): booleana;
   si vecina.puedeAtacar(fila,columna) devuelve misma.siguiente()
   sino devuelve verdadero
fin

procedimiento imprimir
   vecina.imprimir;
   escribe fila, columna
fin
```

Utiliza el siguiente código como base para implementar el programa `ReinasApp.java`, que constará de las dos clases que aparecen, la primera de ellas ya está implementada.

```
//declarar paquetes necesarios
import ...
//clase para el programa principal
class ReinasApp {
    public static void main (String args[]) throws IOException{
        //Crear las reinas
        Reina ultimaReina = null;//la primera reina no tiene vecina
        for(int i=1; i<=8; ++i)
            ultimaReina = new Reina(i, ultimaReina);
        //Generar sol.
        if( ultimaReina.primera() ) ultimaReina.imprimir();
    }
}
//clase que modela una reina
class Reina {
    //atributos
    //Constructor
    public Reina(int c, Reina vec) {
        ...
    }
    //resto de metodos
}
```

Modifica el programa para que:

1. Calcule todas las soluciones posibles (no sólo la primera).
2. El usuario decida para cada columna si la reina que la ocupa se debe comportar como una reina o como una torre.

4. Aplicación GUI

Se desarrollará un sencillo prototipo de aplicación con interfaz gráfica y la implementación de dos cuadros de diálogo. Todo ello basado en los componentes Swing de java. Los objetivos son aprender a: crear aplicaciones basadas en menús, manejar los eventos que se generan, gestionar diálogos, manejar componentes gráficos más avanzados (tabla y árbol).

Comenzar creando un "front-end" como el que aparece en la Figura 2. Este se compone de los siguientes items de menú: Fichero (Nuevo, Abrir,Salvar,Salir), MisDialogos (Tabla,Arbol). Además aparecerá un campo de texto donde se escribirá el nombre de la opción de menú que se seleccione cada vez. Utilizar como guía el esqueleto que aparece a continuación. Para obtener una ayuda detallada consulta en la API la clase `JFrame`, allí encontrarás la sección "How to make frames".

```
//declarar paquetes necesarios
import ...
//clase que crea el menu
public class Aplicacion extends JFrame{
    Container frameContainer;
    //Crear la barra de menu (JMenuBar).
    //Crear las opciones del menu principal (JMenu).
    //Crear los items de menu(JMenuItem) y el separador(JSeparator).
    //Crear el campo de texto (JTextField)
    public Aplicacion(){
        super(TITLE);
        crearGUI();
        crearManejadoresEvts();
        //fijar tamaño de la ventana, modo de operacion y mostrarla.
    }
    void crearGUI(){
        crearMenu();
        situarComponentes();
    }
    void crearMenu(){
        //Añadir los items de menu a su opcion principal.
        //Añadir las opciones principales a la barra de menu.
        //Configurar la barra de menu (setJMenuBar(..)).
    }
    void situarComponentes(){
        frameContainer = getContentPane();
        frameContainer.setLayout(null);
        //Dar tamaño al campo de texto y unirlo al frameContainer.
    }
    void crearManejadoresEvts(){
        addWindowListener(new ManejadorVentana());//gestiona evts. vent. ppal.
        //AñadirManejadores a los items de menu.
    }
    public static void main(String[] args) {
        Aplicacion app = new Aplicacion();
    }
    public class ManejadorVentana extends WindowAdapter{
        //implementar el metodo correspondiente
        //para que la aplicacion termine correctamente.
    }
    public class ManejadorItemMenu implements ActionListener{
        //implementar el metodo correspondiente para que Fichero/Salir
        //termine la Aplicacion y el resto de opciones escriban
        //escriban en el campo de texto su nombre.
    }
}
```

Una vez realizado el menú se debe crear un diálogo modal que aparecerá cuando se seleccione la opción de menú MisDialogos/Tabla. Este diálogo contendrá una

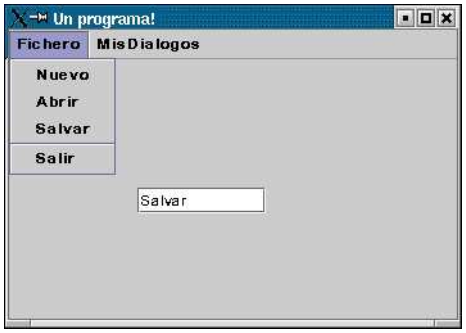

Figura 2: Menú de la aplicación.

tabla con scroll. Las columnas mostrarán información de Clientes: Nombre, Apellido, Teléfono, Edad. Cada fila de la tabla se corresponderá con la información de un cliente, que será tomada de un fichero de texto en el que cada línea es un cliente con los diferentes campos separados por espacios en blanco. La tabla podrá ser ordenada por cualquier campo. Se permitirá añadir y borrar clientes de la tabla así como actualizar el fichero de texto con la información de la tabla. Para obtener detalles de cómo manipular tablas leer en la clase `JTable`, "How to create a table". Para manipular el fichero utilizar las clases `FileReader, BufferedReader, StringTokenizer`.

Por último, la opción MisDialogos/Arbol mostrará un diálogo modal que contendrá un árbol (objeto `JTree`) con la estructura jerárquica de los ficheros y directorios en tu cuenta de `merlin`. Se permitirá cambiar nombre y permisos de los ficheros y directorios así como crear nuevos y borrar los existentes. Para obtener detalles de cómo manipular árboles leer en la clase `JTree`, "How to use trees". Por simplicidad, este diálogo no debe tener ninguna relación con las opciones del menú principal (Fichero Nuevo, Fichero Abrir, etc.). No utilizar la clase `JFileChooser`.