

Construcción dinámica de menús en Java[®].

La clase *MiMenu*.

Ernesto Ponsot Balaguer^π

Mérida, Diciembre de 2004

Resumen. -

Se presenta la discusión, el diseño e implantación de un objeto genérico Java, bautizado *MiMenu*, cuyo fin es proporcionar un menú de opciones a partir de una matriz de cadenas de caracteres dada. Dicha matriz debe contener los pares *clave – opción deseada* y se espera que dicha *clave*, además de identificar la opción, ~~le~~ ubique en la secuencia y jerarquización del conjunto de opciones. La clase *MiMenu* es de utilidad a los programadores de sistemas, al encapsular las complejidades de objetos como *JMenuBar*, *JMenu*, *JMenuItem*, entre otros. Otra ventaja que recibe el programador, es un incremento en la legibilidad y mantenibilidad de sus programas, ya que habrá de preocuparse sólo de establecer las opciones de sus menús de acuerdo con el diseño que haya escogido y programar las respectivas respuestas, manteniendo el árbol de las alternativas como una estructura de datos definida, en una única sección de su programa.

[™] Sun Microsystems. Inc.

^π Profesor en la Cátedra de Computación, Departamento de Estadística – FACES, Universidad de los Andes. Mérida, Venezuela. E-Mail: ernesto@ula.ve

Contenido

Introducción.....	2
El Problema	2
La Clase <i>MiMenu</i>	4
Comentarios sobre el uso de la clase <i>MiMenu</i>	10
Referencias	12

Introducción

Java es definitivamente genial. Sus precursores y propulsores, la empresa *Sun Microsystems*, apostaron a una idea brillante: construir un compilador que hiciera su trabajo independientemente de la plataforma computacional en la que se encontrara. Llegaron aún más allá: bgraron un compilador para un lenguaje de programación, que funciona en cualquier ambiente, perfectamente orientado a objetos (OO) y tan general que puede o no estar conducido por eventos. Una de sus más apreciables bondades es su capacidad de crecer gracias al aporte de propios y extraños.

Ahora bien, para sacar verdadero provecho de Java es imprescindible no solo entender, sino también practicar la programación OO. Sus virtudes comienzan a verse rápidamente si se adopta el paradigma OO, pero nunca antes del segundo programa. Sólo cuando el programador nota que la forma como ha utilizado los objetos Java puede, a su vez, ser encapsulada en nuevos objetos de creación propia, se vislumbra todo su poderío.

Estudiando Java, en la tarea de utilizarle como herramienta para la programación de sistemas de información, emerge la idea de construir un objeto genérico, que automatizara la elaboración de menús, sin que preocuparan detalles técnicos o gráficos, sino más bien, conceptos y consideraciones de diseño del árbol de opciones. Propósito sencillo que, sin embargo, es de utilidad práctica y didáctica. Las líneas que siguen, cuentan la historia y su resultado, y tienen la pretensión de ser interesantes a cualquiera que, como el autor, se inicie en la fascinante actividad de la programación Java.

El Problema

La idea de un menú es proporcionar al usuario una interfaz gráfica, valiéndose del ratón, teclas de abreviación o flechas de dirección. Navegando en el menú, el usuario puede ubicar la opción que desea y ordenarle a la aplicación que la ejecute.

La base para la creación de menús de opciones en Java son las clases *JMenuBar*, *JMenu* y *JMenuItem*. La primera establece la estructura principal que contendrá las opciones. La segunda crea una opción cuya propósito será desplegar nuevas opciones y la tercera, crea una opción que ejecutará acciones concretas del programa. Una vez instanciados los objetos, el árbol se define anexando submenús y opciones finales a la barra de menús, submenús y opciones finales a los submenús que se desprenden de la barra de menús, y así sucesivamente. Esto se logra con el método `add` del objeto receptor.

Cada opción final de un menú, se trata en Java como un botón que al presionarse ejecuta una acción preestablecida, por lo cual, es necesario proporcionar al programa una forma de inspección que vigile constantemente si el usuario ha o no seleccionado algo del menú y actúe en consecuencia. La combinación de la cláusula `implements ActionListener`, en la declaración de la clase, y su método requerido `actionPerformed`, son los responsables de controlar este comportamiento. Cada opción que conduzca a una acción concreta, habrá de tener un identificador para el sistema de manera tal que cuando el “oyente” note que ha sido seleccionada, pase la novedad al método “actuante” donde, seguramente a base de instrucciones de decisión anidadas, estará la llamada a la rutina que implementa la acción.

El problema consiste entonces en cómo encapsular esta complejidad, creando un objeto de menús al cual sólo deban pasársele como parámetros, las opciones y el método que resuelve las acciones, sin tener que preocuparse de nada más.

Supóngase que se desea crear un menú con la estructura mostrada en la Figura 1.

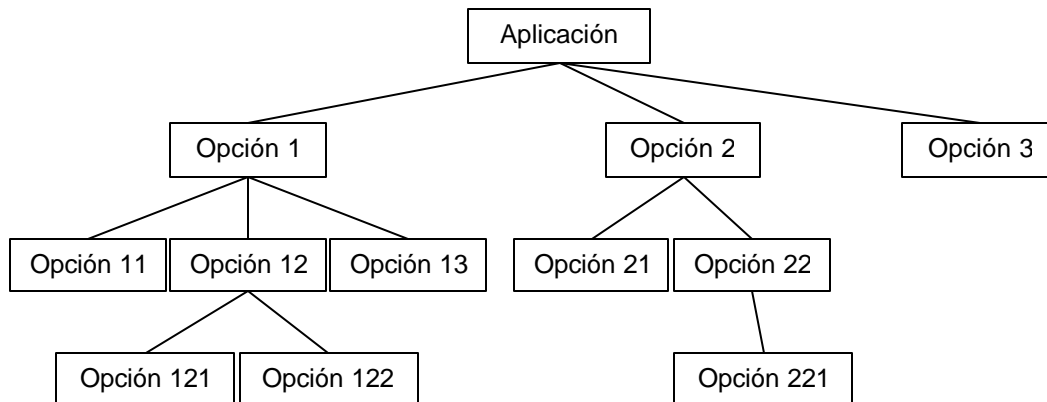


Figura 1. Ejemplo de un menú de opciones típico desplegado como un árbol

La Figura 1 muestra que la estructura deseada contiene dos submenús (Opción 1 y 2) y una opción de acción (Opción 3) que forman la barra de menús. Al interior del submenú Opción 1, hay un submenú (Opción 12) y dos opciones de acciones (Opción 11 y 13). Al interior del submenú Opción 12, hay dos opciones de acciones (Opción 121 y 122). Al interior del submenú Opción 2 hay un submenú (Opción 22) y una opción de acción (Opción 21). Por último, al interior del submenú Opción 22, hay una opción de acciones (Opción 221). La estructura puede visualizarse como un árbol con tres niveles, que contiene una barra de menús, cuatro submenús y siete opciones de acción. Todo ello debería verse en la práctica como muestra la Figura 2.

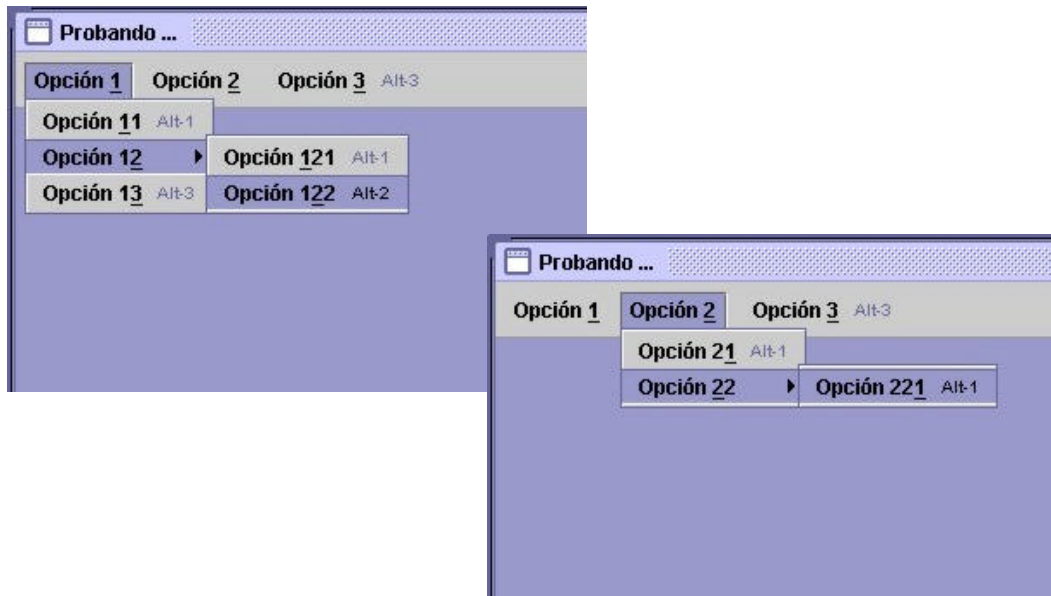


Figura 2. Ejemplo de un menú de opciones típico desplegado en la interfaz gráfica

En la Figura 2 se observa cómo luce el diseño de menús dado como ejemplo. Se han adicionado elementos como las teclas de acceso abreviado (señaladas con un subrayado) e indicaciones de cómo acceder directamente a cada opción de acciones (del tipo Alt-Tecla), ubicando los elementos dentro de una ventana que se constituye en la aplicación.

La Clase *MiMenu*

El Listado 1, muestra el código fuente Java de la clase *MiMenú* que implementa la solución buscada. Los comentarios al programa siguen la cadena “//” o se encierran entre las combinación “/*” y “*/”.

Listado 1. La Clase *MiMenu*

```
// Objetos para la elaboración de menús
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
// Objetos para las teclas de abreviación
import javax.swing.KeyStroke;
// Objetos de la interfaz
import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
// Objetos de lista ordenada por clave
import java.util.Hashtable;
```

```

// Declaración de la clase MiMenu
public class MiMenu extends JMenuBar implements ActionListener {
    private String vtOpc[][];           // Matriz de opciones
    private String MetodoAccion = "";   // Nombre del método de acciones
    private int LonOpc;                 // Número de filas de la matriz de opciones

    // Constructor de la clase MiMenu
    public MiMenu(String vtOpcPar[][], String MetAccion) {

        // Asignación de parámetros y variables de instancia
        vtOpc = vtOpcPar;
        MetodoAccion = MetAccion;
        LonOpc = vtOpc.length;

        /* Definición de la orientación y aspecto del menú. Necesaria
           para que opciones de acción directamente colocadas en la
           barra de menús restrinjan su tamaño al mínimo */
        setLayout(new FlowLayout(FlowLayout.LEFT));
        // Definición de la lista que contendrá los objetos del menú
        Hashtable Lista = new Hashtable();
        // Encontrando los tipos de objetos y llenando la lista con ellos
        JMenu tmpMenu;
        JMenuItem tmpItem;
        for (int i=0; i<LonOpc; i++) {
            if (TieneHijos(vtOpc[i][0])) {
                // Si tiene hijos, debe ser un submenú
                tmpMenu = new JMenu(vtOpc[i][0]);
                // El nombre del objeto es la clave de la opción
                tmpMenu.setName(vtOpc[i][0]);
                /* El texto que se mostrará va sin el señalador del
                   mnemónico */
                tmpMenu.setText(QuitaCar(vtOpc[i][1], '_'));
                /* El mnemónico se asigna como el caracter después del
                   señalador */
                tmpMenu.setMnemonic(
                    EncuentraMnemonic(vtOpc[i][1], '_'));
                // Se agrega el objeto a la lista, ordenado por su clave
                Lista.put(vtOpc[i][0], tmpMenu);
            } else {
                // Si no, debe ser una opción de acción
                tmpItem = new JMenuItem(vtOpc[i][0]);
                // El nombre del objeto es la clave de la opción
                tmpItem.setName(vtOpc[i][0]);
                /* El texto que se mostrará va sin el señalador del
                   mnemónico */
                tmpItem.setText(QuitaCar(vtOpc[i][1], '_'));
                /* El mnemónico se asigna como el caracter después del
                   señalador */
                char Nemonico = EncuentraMnemonic(vtOpc[i][1], '_');
                tmpItem.setMnemonic(Nemonico);
                // La tecla de aceleración es el mismo mnemónico
                tmpItem.setAccelerator(KeyStroke.getKeyStroke(
                    Nemonico, ActionEvent.ALT_MASK));
                /* El comando de acción del objeto es también la clave
                   de la opción */
                tmpItem.setActionCommand(vtOpc[i][0]);
            }
        }
    }
}

```

```

        /* Este es de acciones por lo que debe ser escuchado
           por el sistema */
        tmpItem.addActionListener(this);
        // Se agrega el objeto a la lista, ordenado por su clave
        Lista.put(vtOpc[i][0], tmpItem);
    }
}
String IdPapa;
// Variable temporal utilizada sólo para la comparación de clases
JMenu tmpMenuPrb = new JMenu();
/* Conectando los objetos de la lista de acuerdo con las
   jerarquías establecidas */
for (int i=0; i<LonOpc; i++) {
    if (EsPrincipal(vtOpc[i][0])) {
        /* Si es una opción principal, no tiene padre y se agrega a
           la barra de menús. Dependiendo del tipo de objeto se
           recupera de la lista por su clave */
        if (Lista.get(vtOpc[i][0]).getClass() ==
            tmpMenuPrb.getClass()) {
            add((JMenu) Lista.get(vtOpc[i][0]));
        } else {
            add((JMenuItem) Lista.get(vtOpc[i][0]));
        }
    } else {
        /* Si no, tiene un padre. Dependiendo del tipo de objeto
           se recupera de la lista por su clave y se anexa al padre
           encontrado */
        IdPapa = vtOpc[i][0].substring(0,vtOpc[i][0].length()-1);
        tmpMenu = (JMenu) Lista.get(IdPapa);
        if (Lista.get(vtOpc[i][0]).getClass() ==
            tmpMenuPrb.getClass()) {
            tmpMenu.add((JMenu) Lista.get(vtOpc[i][0]));
        } else {
            tmpMenu.add((JMenuItem) Lista.get(vtOpc[i][0]));
        }
    }
}
} // Fin del constructor MiMenu

/* Método que determina, dada la clave de una opción,
   si ésta tiene subopciones */
private boolean TieneHijos(String Item) {

    /* Cuenta el número de veces que aparece la clave dada iniciando
       otras claves. Si ésta aparece más de una vez, la opción tiene
       subopciones */
    int NVeces = 0;
    int LonItem = Item.length();
    for (int i=0; i<LonOpc; i++) {
        if (vtOpc[i][0].length() >= LonItem) {
            if (vtOpc[i][0].substring(0,LonItem).equals(Item)) {
                NVeces++;
                if (NVeces > 1) {
                    return true;
                }
            }
        }
    }
}

```

```

    }
    return (NVeces > 1);
} // Fin de TieneHijos

/* Método que determina, dada la clave de una opción,
   si ésta pertenece a la barra de menús */
private boolean EsPrincipal(String Item) {

    // En la barra de menús se esperan claves de un solo dígito
    return (Item.length() == 1 );

} // Fin de EsPrincipal

/* Método de propósito general que quita un caracter específico
   de una cadena */
private String QuitaCar(String Cad, char c) {

    int Pos = Cad.indexOf(c);
    int Lon = Cad.length();
    if (Pos != -1) {
        // Si está el caracter
        if (Pos == 0) {
            return Cad.substring(1, Lon);
        } else {
            if (Pos == Lon-1) {
                return Cad.substring(0, Lon-1);
            } else {
                return Cad.substring(0, Pos) +
                    Cad.substring(Pos+1, Lon);
            }
        }
    }
    return Cad;
} // Fin de QuitaCar

/* Método que retorna el caracter siguiente al señalado,
   entendiendo que dicho caracter es el mnemónico de una
   opción de menú */
private char EncuentraMnemonico(String Cad, char c) {

    int Pos = Cad.indexOf(c);
    if (Pos >= Cad.length() - 1) {
        /* El señalador de mnemónico no debe ser el último caracter
           de la cadena */
        return 0;
    }
    return Cad.charAt(Pos+1);
} // Fin de EncuentraMnemonico

```

```

/* Método para la resolución de las acciones en respuesta a
los eventos de MiMenu que estén siendo escuchados */
public void actionPerformed(ActionEvent e) {

    try {
        /* Si el contexto en que se utiliza el objeto MiMenu es
        una ventana, dentro de la cual hay una barra de menús,
        extendiendo la barra MiMenu, el objeto que instancia
        la clase está en la tercera generación. Si este no es el
        caso, habrá que alterar la instrucción, referenciando
        el objeto padre en la generación correcta */
        Object objPapa = getParent().getParent().getParent();
        Class MiPapa =
            getParent().getParent().getParent().getClass();
        /* Estableciendo que los parámetros del método de acciones
        en la clase que llama a MiMenu son de tipo String y
        pasando como argumento a dicho método la clave de la
        opción seleccionada */
        Class[] TiposParametros = new Class[] {String.class};
        Object[] argumentos = new Object[] {e.getActionCommand()};
        /* Definiendo el método de acciones de la clase que llama
        a MiMenu y sus parámetros para luego invocarlo
        ocasionando su ejecución */
        Method target = objPapa.getClass().getMethod(
            MetodoAccion, TiposParametros);
        Object param[] = { e.getActionCommand() };
        target.invoke(objPapa,argumentos);

    } catch ( Exception exp ) {
        exp.printStackTrace();
    }
} // Fin de actionPerformed

} // Fin de la clase MiMenu

```

La clase *MiMenu* mostrada en el Listado 1, extiende la clase *JMenuBar* e implementa el “oyente” *ActionListener*.

Se definen tres variables de instancia, *vtOpc*, *MetodoAccion* y *LonOpc* que reciben de quien invoca la clase, el menú de opciones en forma de matriz de cadenas, el nombre del método que resuelve las acciones en forma de cadena y el número de filas de la matriz de opciones, respectivamente.

La matriz de opciones esperada por la clase tiene dimensiones *LonOpc* x 2. Las filas representan las distintas opciones del menú, la primera columna representa las claves de cada opción y la segunda columna, el texto de cada opción. Para el ejemplo señalado, la matriz de opciones necesaria se muestra en la Tabla 1.

Tabla 1. Ejemplo de un menú de opciones típico. La matriz de opciones

Fila / Columna	1	2
1	"1"	"Opción _1"
2	"11"	"Opción 1_1"
3	"12"	"Opción 1_2"
4	"121"	"Opción 12_1"
5	"122"	"Opción 12_2"
6	"13"	"Opción 1_3"
7	"2"	"Opción _2"
8	"21"	"Opción 2_1"
9	"22"	"Opción 2_2"
10	"221"	"Opción 22_1"
11	"3"	"Opción _3"

Nótese en la Tabla 1, en primer lugar, la incorporación de las marcas que determinan el carácter mnemónico en cada opción, señalizadas a voluntad del programador, a continuación del carácter de subrayado (“_”). Recuérdese que entendemos por mnemónico de una opción de menú, una tecla que al ser presionada o combinada con otras, conduce directamente a la ejecución de la opción. Los mnemónicos son alternativas de teclado al uso del ratón. Estas marcas no aparecen en el menú construido pero se consideran indicativos de la tecla de abreviación que deberá ser establecida por la clase *MiMenu*. Otro aspecto importante a notar es la jerarquía del árbol de opciones, auto-representada por las claves en forma numérica. La disposición de las filas se espera esté en concordancia con dicha jerarquía. Por ejemplo, de la opción cuya clave es “2” podemos decir lo siguiente:

- ◆ Es una opción principal puesto que se trata de un sólo dígito. Ello implica que habrá de disponerse directamente en la barra de menús.
- ◆ Tiene subopciones (y por tanto no es una opción de acción) ya que el dígito “2” aparece más de una vez en la matriz al inicio de otras claves.

Algunos comentarios sobre los métodos:

- ◆ *MiMenu*: Constructor de la clase. Dada la matriz de opciones, en primer lugar dilucida el tipo de objeto que le corresponde a cada opción (submenú u opción de acción), almacenándolo en una lista de objetos Java organizada por clave. En el camino, quita del texto de cada opción el carácter indicativo del mnemónico, encuentra dicho mnemónico y lo establece, define la combinación de las teclas de aceleración, establece para las opciones de acción el “oyente” de eventos y el comando por medio del cual serán conocidas. Por último, recorre la lista y recupera cada objeto, asignándole su disposición en la jerarquía del menú, de acuerdo con la auto-representación entendida de las claves numéricas.
- ◆ *TieneHijos*: Método privado. Determina si una clave dada tiene o no subclaves asociadas, mediante el examen de la matriz de opciones. Retorna un valor lógico.

- ◆ **EsPrincipal:** Método privado. Determina si una clave dada debe ser ubicada directamente sobre la barra de menús o si corresponde a una opción interior. Retorna un valor lógico.
- ◆ **QuitaCar:** Método privado. Rutina de carácter general (es decir que puede ser empleada en otro contexto), cuya misión es retirar de una cadena de caracteres, un carácter específico. Retorna la cadena dada suprimiendo dicho carácter.
- ◆ **EncuentraMnemonic:** Método privado. Asumiendo que un carácter cualquiera que pasa como parámetro, señala otro carácter a continuación que representa el mnemónico de una opción de menú (representada por una cadena de caracteres también dada), devuelve dicho carácter en caso que sea encontrado.
- ◆ **actionPerformed:** Método público obligatorio en Java cuando se implementa en una clase *ActionListener*. Es responsable de “escuchar” permanentemente los eventos que ocurren asociados al menú y decidir las acciones correspondientes. Tratándose de una clase de propósito general, que puede ser utilizada por cualquier otra clase, no se conoce de antemano cuáles serán las acciones que se dispararán en cada evento de menú. La aplicación particular que hace uso de *MiMenu* establece estas acciones según su conveniencia. En consecuencia, los eventos son escuchados y se determina la opción de menú seleccionada por el usuario, pero no se realizan acciones particulares de la aplicación. Una vez establecida la opción seleccionada, simplemente se pasa como parámetro su clave a la aplicación, en la que se espera exista un método llamado como indica la variable *MetodoAccion*, cual es en realidad el responsable de conducir la respuesta al evento. Esta rutina utiliza la clase *Method* de Java para invocar al método de la aplicación encargado de tales acciones. Tres llamadas al método *getParent()* son necesarias para instanciar un objeto que apunte a la aplicación original, lo que presupone que dicha aplicación es la primera de tres generaciones de objetos (de no ser así, será necesario incorporar o quitar llamadas a *getParent()*). Esta última característica es tal vez la única que resta generalidad a la clase *MiMenu*, pero la circunstancia supuesta es la más frecuente en la práctica.

Comentarios sobre el uso de la clase *MiMenu*

El Listado 2 muestra un programa Java construido para probar la utilización de la clase *MiMenu*, empleando el ejemplo descrito en las figuras 1 y 2.

Listado 2. Probando la Clase *MiMenu*

```
// Objetos para la construcción de ventanas
import javax.swing.JFrame;
import java.awt.*;

public class ParaProbarMiMenu extends JFrame {

    // Constructor de la clase ParaProbarMiMenu
    public ParaProbarMiMenu() {
```

```

super("Probando ... ");
Dimension screenSize =
    Toolkit.getDefaultToolkit().getScreenSize();
setBounds(50, 50,
    screenSize.width - 100,
    screenSize.height - 100);

/* Definición e inicialización de la matriz n x 2 contentiva
de las opciones que se desea dar a MiMenu. Se espera que
esta matriz contenga sólo tipos String, y que cada elemento,
en sentido vertical, esté compuesto por el par Clave, Opción,
donde Clave es un número que mantiene la jerarquía del
árbol de opciones, por ejemplo, 112 es subopción de 11, la
cual es subopción de 1, y así en todos los demás casos. */
String vtOpciones[][] = {
    {"1", "Opción _1"},
    {"11", "Opción 1_1"},
    {"12", "Opción 1_2"},
    {"121", "Opción 12_1"},
    {"122", "Opción 12_2"},
    {"13", "Opción 1_3"},
    {"2", "Opción _2"},
    {"21", "Opción 2_1"},
    {"22", "Opción 2_2"},
    {"221", "Opción 22_1"},
    {"3", "Opción _3"}
};

/* Llamada de MiMenu que envía la matriz de opciones y el
Método de la clase que lo invoca que resuelve las acciones
del menú dado */
MiMenu mnPrin = new MiMenu(vtOpciones, "AccionesMenu");
// Establecimiento de MiMenu como el menú de la aplicación
setJMenuBar(mnPrin);

} // Fin del constructor de ParaProbarMiMenu

/* Método que resuelve las acciones a tomar cuando se ha seleccionado
una opción de MiMenu, la cual pasa como parámetro String en Opc y
representa una clave de la matriz de opciones definida. */
public void AccionesMenu(String Opc) {

    /* En este ejemplo, estas son las claves de opciones terminales
    (esto es, aquellas que provocan acciones) definidas. Por
    supuesto si cambia el menú de opciones, será necesario
    alterar el contenido de este método, en consecuencia. */
    if (Opc.equals("11")) {
        System.out.print("11");
    } else if (Opc.equals("121")) {
        System.out.print("121");
    } else if (Opc.equals("122")) {
        System.out.print("122");
    } else if (Opc.equals("13")) {
        System.out.print("13");
    } else if (Opc.equals("21")) {
        System.out.print("21");
    } else if (Opc.equals("221")) {

```

```

        System.out.print("221");
    } else if (Opc.equals("3")) {
        System.out.print("3");
    }
} // Fin de AccionesMenu

// Principal de ParaProbarMiMenu
public static void main(String[] args) {

    JFrame.setDefaultLookAndFeelDecorated(true);
    ParaProbarMiMenu frame = new ParaProbarMiMenu();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);

} // Fin de main

} // Fin de la clase ParaProbarMiMenu

```

La clase mostrada en el Listado 2 se utiliza para probar *MiMenu*. En ella se crea la ventana de la aplicación – del tipo `JFrame` – a la que se asigna como barra de menús una instancia de *MiMenu*. Previamente se han establecido las opciones deseadas en la forma de un arreglo bidimensional de cadenas de caracteres, siguiendo las reglas esperadas por *MiMenu*. Tratándose de un ejemplo, el método `AccionesMenu` se construye de manera tal que responde a cada opción terminal seleccionada, simplemente escribiendo en la consola estándar de Java su clave. El método `main` se encarga de ejecutar el programa de prueba.

Nótese que el programa no tiene que ver con los detalles del menú de opciones, más allá de definir su contenido, conceptualmente hablando, y disponer las respuestas apropiadas a cada selección. El orden en que se dan valores a las opciones del menú, facilita el diseño del árbol y mejora la legibilidad del programa. Adicionalmente a estas ventajas cabe mencionar que al ser *MiMenu* un objeto abstracto, puede emplearse con muy poco esfuerzo extra tanto en aplicaciones con interfaz de escritorio (como el ejemplo mostrado), como en Applets e incluso instalándose en servidores de aplicación (J2EE), cuando se requiera programación a tres capas (Cliente-ServerApp-ServerBD).

Referencias

- ◆ Naughton, P. Schildt, H. "Java. Manual de Referencia". 1ª Edición. Osborne / McGraw - Hill, Madrid - España, 1997.
- ◆ Sun Microsystems, Inc. "Java 2 SDK, Standard Edition. Documentation. Version 1.4.2", EEUU, 2003.
- ◆ Sun Microsystems, Inc. "The Java Tutorial. A practical guide for programmers". <http://java.sun.com/docs/books/tutorial/>