Capítulo 5
Las tablas

1. Presentación

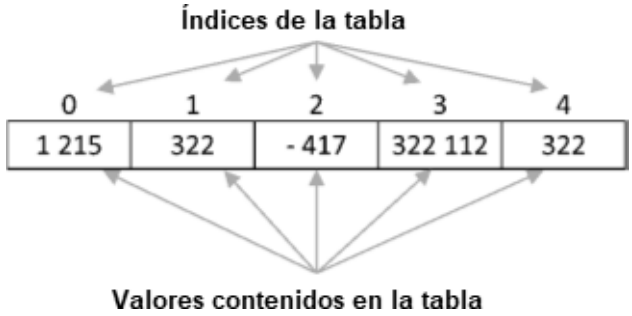
Con los conocimientos adquiridos durante la lectura de los capítulos anteriores, solo es posible asociar un valor a una variable. Pero eso cambiará gracias a las tablas.

Observación

Atención, si ya ha visto el concepto de tabla en lenguajes de programación como JavaScript, PHP o Python, por ejemplo, tenga cuidado, porque estos lenguajes en realidad usan tablas asociativas, que son estructuras de datos diferentes de tablas que se presentan en este capítulo.

Por tanto, una tabla es un tipo particular de variable capaz de almacenar un conjunto de valores, todos del mismo tipo. Estos valores se colocan en lo que se denominan celdas de la tabla. Para poder navegar en todas estas celdas, están numeradas con un entero llamado índice.

A continuación, se muestra un ejemplo de una tabla con 5 celdas, donde cada una contiene un valor entero:



Los índices se numeran comenzando desde cero. Así, la primera celda de la tabla está en el índice cero, la segunda tiene el índice uno y así sucesivamente. Por tanto, una tabla de N celdas, tiene sus celdas numeradas con índices que van desde cero hasta N-1.

Observación

La mayoría de lenguajes de programación, numeran los índices de las celdas de una tabla comenzando desde cero, ya que esto simplifica los cálculos para acceder al valor. Sin embargo, en otras publicaciones es posible ver índices numerados a partir de uno, lo que parece más natural para un humano. Esta elección es arbitraria y, como un algoritmo no está destinado a ser interpretado por una máquina, coexisten las dos formas de proceder. En este libro, los índices comenzarán desde cero para que pueda adquirir este hábito desde ahora, ya que esta es la forma de numeración que usa Java.

2. La declaración de una tabla

Las tablas son variables especiales ya que pueden almacenar a su vez varias variables, pero aun así son variables. Por lo tanto, se deben declarar en la sección de declaración como las otras variables. Aquí está la sintaxis para declarar una tabla:

```
Variable nombre_de_la_tabla: tipo[tamaño_de_la_tabla]
```

© Editions ENI - All rights reserved

La declaración es bastante similar a la declaración de otras variables. Esta declaración también comienza con la palabra clave `Variable`, seguida por el nombre elegido para la tabla. Después del carácter dos puntos, se indica el tipo de valor que se almacenará en las celdas de la tabla (recuerde que todos los valores contenidos en cada celda de la tabla deben ser del mismo tipo). Para terminar, y esta es la única diferencia, se indica entre paréntesis el tamaño de la tabla, es decir, el número de celdas que debe contener esta tabla.

Ejemplos:

```
# declaración de una tabla de diez enteros:
Variable te: entero[10]

# declaración de una tabla de 3 números reales:
Variable tr: real[3]

# declaración de una tabla de 30 caracteres:
Variable tc: caracter[30]

# declaración de una tabla de 6 cadenas de caracteres:
Variable tt: texto[6]
```

Observación

Los corchetes son característicos de las tablas. Cuando vea corchetes, puede pensar que seguramente tiene que ver con tablas. Esto es cierto en algoritmia, pero también en muchos lenguajes de programación, y en particular, en Java.

El tamaño de una tabla debe ser fijo y conocido inicialmente: no es posible poner una variable entre los corchetes utilizados para fijar el tamaño de la tabla. Solo se autoriza el uso de números o el uso de una constante.

También es posible declarar una tabla sin indicar un valor entre los corchetes. Esto es necesario para recuperar una tabla que se ha creado dentro de una función, por ejemplo. Esto se explica en el capítulo siguiente, que trata de los procedimientos y funciones.

En Java, para declarar una tabla, debe usar la siguiente sintaxis:

```
// declaración de la tabla
tipo [] nombre_de_la_tabla;
// reserva de las celdas de la tabla
nombre_de_la_tabla = new tipo[tamaño_de_la_tabla];
```

Estas dos declaraciones se pueden combinar en una, con la siguiente sintaxis:

```
tipo [] nombre_de_la_tabla = new tipo [tamaño_de_la_tabla];
```

El tipo que se indica para declarar la tabla y el de la reserva de las celdas de ésta, debe ser obligatoriamente el mismo. La palabra clave `new`, que revisaremos más adelante en este libro, significa reservar espacio en la memoria.

Aquí hay algunos ejemplos de declaraciones de tablas:

```
// declaración de una tabla de diez números enteros:
int[] te = new int[10];

// declaración de una tabla de 3 números reales:
double[] tr = new double[3];

// declaración de una tabla de 30 caracteres:
char[] tc = new char[30];

// declaración de una tabla de 6 cadenas de caracteres:
String[] tt = new String[6];
```

En Java, es posible crear una tabla inicializada con valores utilizando la siguiente sintaxis:

```
tipo[] nombre_de_la_tabla = { valor1, valor2, ... , valorN };
```

El tamaño de la tabla se corresponde con el número de elementos presentes entre las llaves.

Ejemplo:

```
// declaración de una tabla de números enteros, directamente
inicializada:
int[] tab = {17, 51, 43, 22, 7, 91, 17, 34, 4, 12};
```

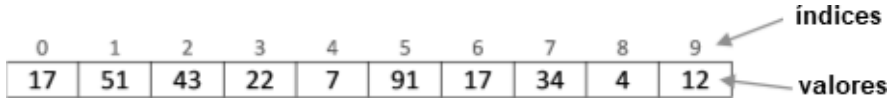
© Editions ENI - All rights reserved

En este ejemplo, se crea una tabla de diez celdas que contiene los diez valores indicados entre las llaves.

Hay otras formas de inicializar una tabla y se presentan en los siguientes apartados.

3. El uso de una tabla

Para acceder a una de las celdas de la tabla, debe especificar cuál de todas ellas. Esto se hace gracias al índice de la celda. Por ejemplo, para indicar la celda de índice tres de la tabla (es decir la cuarta celda de la tabla), es necesario indicar nombre_de_la_tabla [3]. Una vez especificada una de las celdas de la tabla, es posible realizar las mismas operaciones como si se tratara de una simple variable. Por ejemplo, es posible almacenar un valor dentro: número_de_la_tabla [3] <- 22 o mostrar su valor: escribir(número_de_la_tabla [3]).



Observación

Preste atención cuando especifique el índice para acceder a una casilla de la tabla, para que este índice tenga un valor aceptable, es decir, que su valor esté entre cero y el tamaño de la tabla menos uno.

A continuación se muestra un ejemplo de algoritmo que usa una tabla:

```
Algo Tablas
Variable tab: real[3]
Inicio
 tab[0] <- 1,61803398875
 tab[1] <- 2,71828182846
 tab[2] <- 3,14159265359
 escribir("número áureo = " & tab[0])
 escribir("e = " & tab[1])
 escribir("pi = " & tab[2])
Fin
```

En este algoritmo, se declara una tabla de tres celdas. Por lo tanto, las celdas de esta tabla tienen índices cero, uno y dos. La primera instrucción asigna un valor a la primera celda de la tabla (la que tiene índice cero). Las siguientes dos instrucciones hacen lo mismo para las siguientes dos celdas de la tabla. Las últimas tres instrucciones consultan los valores contenidos en estas celdas para visualizar estos valores.

En Java es igual para acceder a una celda de una tabla. También es necesario especificar su índice entre corchetes. A continuación se muestra el algoritmo anterior programado en Java:

```
public class Tablas {

   public static void main(String[] args) {
       double[] tab = new double[3];
       tab[0] = 1.61803398875;
       tab[1] = 2.71828182846;
       tab[2] = 3.14159265359;
       System.out.println("número áureo = " + tab[0]);
       System.out.println("e = " + tab[1]);
       System.out.println("pi = " + tab[2]);
   }

}
```

4. Recorrer una tabla

Es bastante común tener que repetir una operación en todas las celdas de la tabla. Para ello, se usa un bucle `For` y el valor de la variable de este bucle se usa como valor de índice para acceder a cada celda de la tabla.

Ejemplo:

```
Algo Tablas2
Constante TAMAÑO: entero <- 10
Variable tab: entero[TAMAÑO]
Variable i: entero
Inicio
 Para i <- 0 a TAMAÑO - 1
   tab[i] <- i + 1
 FPara
Fin
```

© Editions ENI - All rights reserved

Durante la primera iteración del bucle, la variable `i` es igual a cero, por lo que `tab[i]` permite acceder a la celda de índice cero de la tabla, es decir, la primera celda de la tabla. Esta se inicializa con el valor uno. Durante la segunda iteración, la variable `i` es igual a uno, por lo que la `tab[i]` se corresponde con `tab[1]` y esta segunda celda se inicializa con el valor dos. Así sucesivamente, hasta que la última celda de la tabla, la celda con el índice nueve, se inicialice con el valor diez. Así, las celdas de la tabla se han inicializado de la siguiente manera:

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

Este bucle `Para` se traduce por un bucle `for` en Java. Esto da por ejemplo:

```
final int TAMAÑO = 10;
int[] tab = new int[TAMAÑO];
for (int i = 0; i < TAMAÑO; i++) {
   tab[i] = i+1;
}
```

En Java, es posible prescindir de la constante `TAMAÑO`, porque el tamaño de una tabla está directamente integrado en la tabla. Para eso, es necesario recurrir a:

```
nombre_de_la_tabla.length
```

Esto permite simplificar el código anterior de la siguiente manera:

```
int[] tab = new int[10];
for (int i = 0; i < tab.length; i++) {
   tab[i] = i+1;
}
```

En Java, hay un cuarto bucle que no tiene equivalente en algoritmia. Este es el bucle `foreach`, es decir, "para cada uno". Aunque se llama `foreach`, se escribe con la palabra clave `for` para reducir el número de palabras clave. La siguiente sintaxis ayuda a distinguirlo de un bucle `for` clásico:

```
for(tipo_de_una_celda nombre_variable: tabla)
```

Capítulo 5
Aplicaciones gráficas

1. Introducción

Hasta ahora, todos los ejemplos de código que hemos realizado funcionan exclusivamente en modo texto. La información se visualiza en una consola y también se informa desde dicha consola. La sencillez de este modo de funcionamiento supone una ventaja innegable para el aprendizaje de un lenguaje. Sin embargo, la mayoría de los usuarios de las futuras aplicaciones seguramente esperan disponer de una interfaz un poco menos austera que una pantalla en modo texto. En este capítulo vamos a estudiar cómo funcionan las interfaces gráficas con Java. Se dará cuenta enseguida de que el diseño de interfaces gráficas en Java no es tan sencillo y requiere escribir muchas líneas de código. En la práctica, contará con varias herramientas de desarrollo, capaces de encargarse de la generación de una gran parte de este código según el diseño gráfico de la aplicación que esté dibujando. Sin embargo, es importante entender correctamente los principios de funcionamiento de este código para intervenir en él y eventualmente optimizarlo. En este capítulo, no emplearemos ninguna herramienta específica.

1.1 Las bibliotecas gráficas

El lenguaje Java propone dos bibliotecas dedicadas al diseño de interfaces gráficas: la biblioteca AWT y la biblioteca SWING. Los fundamentos de uso son casi idénticos en ambas bibliotecas. El uso simultáneo de las dos bibliotecas en una misma aplicación puede provocar problemas de funcionamiento y por ello debería evitarse. Hay una tercera biblioteca más reciente llamada Java FX. Sin embargo, ya no forma parte de la plataforma estándar Java SE. Por lo tanto, este capítulo no cubrirá esta biblioteca.

1.1.1 La biblioteca AWT

Esta biblioteca es la primera disponible para el desarrollo de interfaces gráficas. Contiene una multitud de clases e interfaces que permiten la definición y la gestión de interfaces gráficas. En realidad, esta biblioteca utiliza las funcionalidades gráficas del sistema operativo. Por lo tanto, no es el código presente en esta biblioteca el que asegura el resultado gráfico de los diferentes componentes. Este código hace de intermediario con el sistema operativo. Su uso ahorra bastante recursos, pero presenta varios inconvenientes:

- Al estar relacionado el aspecto visual de cada componente con la representación que el sistema operativo hace de él, puede resultar delicado desarrollar una aplicación que tenga una apariencia coherente en todos los sistemas. El tamaño y la posición de los diferentes componentes son los dos elementos que se ven principalmente afectados por este problema.
- Para que esta biblioteca sea compatible con todos los sistemas operativos, los componentes que contiene están limitados a los más corrientes (botones, zonas de texto, listas…).

1.1.2 La biblioteca Swing

Esta biblioteca se diseñó para resolver las principales carencias de la biblioteca AWT. Se obtuvo esta mejora al escribir completamente la biblioteca en Java sin apenas recurrir a los servicios del sistema operativo. Únicamente algunos elementos gráficos (ventanas y cuadros de diálogo) siguen relacionados con el sistema operativo. Para los demás componentes, es el código de la biblioteca Swing el encargado de determinar completamente su aspecto y su comportamiento.

© Editions ENI - All rights reserved

La biblioteca Swing contiene por lo tanto una cantidad impresionante de clases que sirven para redefinir los componentes gráficos. Sin embargo, no debemos pensar que la biblioteca Swing convierte la biblioteca AWT en algo completamente obsoleto. De hecho, Swing recupera muchos de los elementos de la biblioteca AWT. En el resto del capítulo, emplearemos esencialmente esta biblioteca.

1.2 Construcción de la interfaz gráfica de una aplicación

El diseño de la interfaz gráfica de una aplicación se fundamenta ante todo en crear instancias de las clases que representan los diferentes elementos necesarios, modificar las características de estas instancias de clase, agruparlas y prever el código de gestión de los diferentes eventos que pueden intervenir durante el funcionamiento de la aplicación. Así, una aplicación gráfica está compuesta por una multitud de elementos superpuestos o anidados. Entre estos elementos, uno toma un papel preponderante en la aplicación. Se le suele llamar contenedor de primer nivel. Es el encargado de interactuar con el sistema operativo y agrupar a los demás elementos. Este contenedor de primer nivel no suele contener directamente los componentes gráficos, sino otros contenedores en los cuales están ubicados los componentes gráficos. Para facilitar la disposición de estos elementos entre sí, vamos a utilizar la ayuda de un renderizador. Esta superposición de elementos podría asimilarse a un árbol, en su raíz principal tenemos el contenedor de primer nivel y sus diferentes ramas están constituidas por los demás contenedores. Las hojas del árbol corresponden a los componentes gráficos.

Dado que el contenedor de primer nivel es el elemento indispensable de cualquier aplicación gráfica, empezaremos estudiando con detalle sus características y utilización, para luego pasar a comprender los principales componentes gráficos.

2. Diseño de una interfaz gráfica

Hemos visto un poco más arriba que cualquier aplicación gráfica se compone de, al menos, un contenedor de primer nivel. La biblioteca Swing dispone de tres clases que permiten llevar a cabo este papel:

`JApplet`: representa una ventana gráfica incluida en una página html para que un navegador se haga cargo de ella. Esta clase ya no es útil desde la plataforma Java 11 porque la tecnología de applets ha sido abandonada.

`JWindow`: representa la ventana gráfica más rudimentaria que pueda existir. No dispone de ninguna barra de título, ningún menú de sistema, ningún borde: en realidad es un mero rectángulo. Esta clase se utiliza rara vez, excepto para la visualización de una pantalla de inicio en el momento del arranque de una aplicación (*splash screen*).

`JFrame`: representa una ventana gráfica completa y plenamente funcional. Dispone de una barra de título, de un menú de sistema y de un borde. Puede fácilmente contener un menú y, por supuesto, es el elemento que vamos a emplear en la mayoría de los casos.

2.1 Las ventanas

La clase `JFrame` es un elemento indispensable en cualquier aplicación gráfica. Como en el caso de una clase normal, debemos crear una instancia, modificar eventualmente las propiedades y utilizar los métodos. A continuación se muestra el código de la primera aplicación gráfica.

```
package es.eni.editions.ihml;
import javax.swing.JFrame;
public class Main {
    public static void main(String[] args)
    {
        JFrame ventana;
        // creación de la instancia de la clase JFrame
        ventana=new JFrame();
        // modificación de la posición y del
        // tamaño de la ventana
        ventana.setBounds(0,0,300,400);
        // modificación del título de la ventana
```

© Editions ENI - All rights reserved

```
        ventana.setTitle("Primera ventana en Java");
        // visualización de la ventana
        ventana.setVisible(true);
    }
```

y el resultado de su ejecución:

Para que este programa compile y se ejecute con el archivo `module-info.java` en su proyecto, es necesario agregar la referencia al módulo que contiene el paquete `java.swing`. Por lo tanto, la instrucción para agregar es la siguiente:

```
requires java.desktop;
```

Volviendo al programa, cabe admitir que es fácil de usar y muy eficaz. De hecho, es tan eficaz que no se puede parar la aplicación. En efecto, incluso si el usuario cierra la ventana, este cierre no provoca la supresión de la instancia de `JFrame` de la memoria. La única solución para detener la aplicación es apagar la máquina virtual Java con la combinación de teclas [Ctrl] **C**. Ante esto, se recomienda proporcionar otra solución para detener más fácilmente la ejecución de la aplicación, es decir, junto con el cierre de la ventana.

Una primera solución consiste en gestionar los eventos que se producen en el momento del cierre de la ventana y, en uno de ellos, provocar la detención de la aplicación. Se estudiará esta solución en la sección dedicada a la gestión de los eventos.

La segunda solución utiliza comportamientos predefinidos para el cierre de la ventana. Estos comportamientos están determinados por el método `setDefaultCloseOperation`. Se definen varias constantes para determinar la acción emprendida al cierre de la ventana.

`DISPOSE_ON_CLOSE`: esta opción provoca la detención de la aplicación en el momento del cierre de la última ventana controlada por la máquina virtual.

`DO_NOTHING_ON_CLOSE`: con esta opción, no ocurre nada cuando el usuario pide el cierre de la ventana. En este caso, es obligatorio gestionar los eventos para que la acción del usuario tenga algún efecto sobre la ventana o la aplicación.

`EXIT_ON_CLOSE`: esta opción provoca la detención de la aplicación incluso si otras ventanas siguen visibles.

`HIDE_ON_CLOSE`: con esta opción la ventana simplemente queda oculta como consecuencia de una llamada a su método `setVisible(false)`.

La clase `JFrame` se encuentra al final de una jerarquía de clases bastante importante e implementa numerosas interfaces. Por este motivo, dispone de varios métodos y atributos.

Module java.desktop
Package javax.swing

Class JFrame

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Window
                java.awt.Frame
                    javax.swing.JFrame
```

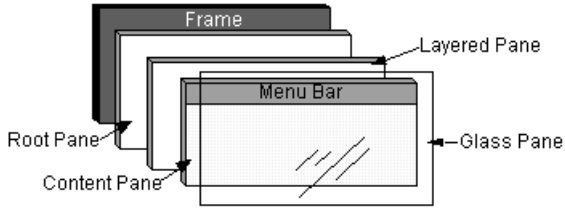
All Implemented Interfaces:

`ImageObserver, MenuContainer, Serializable, Accessible, RootPaneContainer, WindowConstants`

© Editions ENI - All rights reserved

La meta de este libro no es retomar toda la documentación del JDK, de modo que no recorreremos todos los métodos disponibles, sino sencillamente los más utilizados según las necesidades. Sin embargo, puede resultar interesante revisar la documentación antes de realizar el diseño de un método para determinar si lo que queremos diseñar no ha sido ya previsto por los diseñadores de Java. La javadoc del paquete `java.swing` está disponible en la dirección siguiente: https://docs.oracle.com/en/java/javase/18/docs/api/java.desktop/javax/swing/package-summary.html

Ahora que somos capaces de visualizar una ventana, el grueso del trabajo va a consistir en añadirle un contenido. Antes de poder añadir algo a una ventana, conviene entender bien su estructura, que resulta relativamente compleja. Un objeto `JFrame` se compone de varios elementos superpuestos, cada uno con un papel muy específico en la gestión de la ventana.



El elemento `RootPane` corresponde al contenedor de los otros tres elementos. El elemento `LayeredPane` es el responsable de la gestión de la posición de los elementos tanto en los ejes X e Y como en el eje Z, lo que permite la superposición de diferentes elementos. El elemento `ContentPane` es el contenedor básico de todos los elementos añadidos en la ventana. A él vamos a confiarle, por esta razón, los diferentes componentes de la interfaz de la aplicación. Por encima del `ContentPane` se superpone el `GlassPane`, como es posible hacer con un cristal sobre una foto. De hecho, presenta muchas similitudes con el cristal.

– Es transparente por defecto.
– Lo dibujado en el `GlassPane` esconde los demás elementos.
– Es capaz de interceptar los eventos relacionados con el ratón antes de que estos hayan alcanzado los demás componentes.