MANUAL DE JAVA

Enrique Sánchez Acosta

No tiene nada de particular.
Basta con pulsar las teclas adecuadas
en el momento oportuno, y el instrumento toca solo.
Johann Sebastian Bach.

Autor: Enrique Sánchez Acosta
Fecha: Febrero de 1.997

1. ALGORITMOS Y PROGRAMAS 6

1.1 INTRODUCCIÓN 6

2 LOS LENGUAJES DE PROGRAMACIÓN 8

2.1 TRADUCTORES DE LENGUAJES 8

3 TIPOS DE DATOS 9

3.1 DATOS NUMÉRICOS 9

4 CONSTANTES Y VARIABLES 10

5 EXPRESIONES 11

5.1 EXPRESIONES ARITMÉTICAS 11
5.2 EXPRESIONES LÓGICAS 12

6 PSEUDOCÓDIGO 13

7 ESTRUCTURA GENERAL DE UN PROGRAMA 14

7.1 TIPO DE INSTRUCCIONES 14
7.1.1 INSTRUCCIONES DE ASIGNACIÓN 14
7.1.2 INSTRUCCIONES DE LECTURA DE DATOS 14
7.1.3 INSTRUCCIONES DE ESCRITURA DE RESULTADOS 14
7.1.4 INSTRUCCIONES DE BIFURCACIÓN 14

8 EJEMPLOS DE PROGRAMACIÓN ESTRUCTURADA 16

9 SUBPROGRAMAS, PROCEDIMIENTOS FUNCIONES 17

9.1 FUNCIONES 17
9.2 PROCEDIMIENTOS 18
9.3 OPERACIONES CON VECTORES 20
9.4 ARRAYS BIDIMENSIONALES 20

10 ARCHIVOS 22

11 ALGUNOS ALGORITMOS IMPORTANTES 23

11.1 ORDENACIÓN 23
11.2 BÚSQUEDA 23
11.3 INTERCALACIÓN 24

12 PROGRAMACIÓN ORIENTADA A OBJETOS 26

12.1 ¿QUÉ ES UN OBJETO? 26
12.2 ¿QUÉ ES UN MENSAJE? 26
12.3 ¿QUÉ ES UNA CLASE? 27
12.4 ¿QUÉ ES HERENCIA? 27

13 ¿QUÉ ES UN OBJETO? 28

14 ¿QUÉ ES UN MENSAJE? 30

15 ¿QUÉ ES UNA CLASE? 31

16 PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA 32

16.1 CREACIÓN DE OBJETOS 34
16.2 VARIABLES ESTÁTICAS 35
16.3 MÉTODOS DE CLASE 37
16.4 DESTRUCCIÓN DE OBJETOS 37
16.5 HERENCIA 37
16.6 OCULTACIÓN Y ENCAPSULAMIENTO DE LOS DATOS 38
16.7 CLASES ABSTRACTAS 39
16.8 REDEFINICIÓN DE MÉTODOS 40

17 INTRODUCCIÓN AL LENGUAJE JAVA 42

17.1 EL NACIMIENTO DEL LENGUAJE JAVA 42
17.2 CARACTERÍSTICAS DEL LENGUAJE JAVA 42
17.2.1 SIMPLICIDAD 42
17.2.2 ORIENTADO A OBJETOS 43
17.2.3 DISTRIBUIDO 44
17.2.4 ARQUITECTURA NEUTRA, INTERPRETADA, PORTABLE Y SEGURA. 44
17.2.5 CONCURRENCIA 45
17.3 JAVA Y EL WORLD WIDE WEB 45
17.4 EL FUTURO DE JAVA 46

18 REFERENCIA BÁSICA DEL LENGUAJE JAVA 48

18.1 GENERALIDADES SOBRE TIPOS, VALORES Y VARIABLES 48
18.2 TIPOS PRIMITIVOS Y SUS VALORES 49
18.2.1 TIPOS ENTEROS Y SUS VALORES 50
18.3 TIPOS DE REFERENCIA Y SUS VALORES 50

18.4 IDENTIFICADORES 51
18.5 VARIABLES 51
18.5.1 ÁMBITO DE LAS VARIABLES 51
18.5.2 VALORES INICIALES DE LAS VARIABLES 52
18.6 PALABRAS RESERVADAS 52
18.6.1 COMENTARIOS 53
18.7 UNICODE Y SECUENCIAS DE ESCAPE 54
18.7.1 SECUENCIAS DE ESCAPE PARA LITERALES DE CARACTERES Y CADENAS 54
18.8 OPERADORES 54
18.9 CONTROL DE FLUJO 55
18.9.1 ESTRUCTURAS DE CONTROL EN JAVA 56
18.10 ARRAYS Y CADENAS 57
18.10.1 ARRAYS 57
18.10.2 CADENAS (STRING Y STRINGBUFFER) 58
18.10.3 CREACIÓN DE UN STRING 59

19 REFERENCIA DE CARACTERÍSTICAS AVANZADAS EN JAVA 60

19.1 PROGRAMACIÓN CON HILOS DE EJECUCIÓN (THREADS) 60
19.1.1 UN SENCILLO PROGRAMA MULTIHILO 61
19.1.2 ATRIBUTOS DE LOS HILOS DE EJECUCIÓN 62
19.1.3 SINCRONIZACIÓN DE HILOS DE EJECUCIÓN 65
19.2 TRATAMIENTO DE EXCEPCIONES 66
19.2.1 EXCEPCIONES EN JAVA 66
19.2.2 GESTIÓN DE EXCEPCIONES TRY, CATCH Y FINALLY 67

20 APPLETS JAVA 69

20.1 CREACIÓN DE APPLETS 69
20.2 INTEGRACIÓN DE APPLETS EN DOCUMENTOS HTML 71

21 PROGRAMANDO EN JAVA 74

22 PASOS INICIALES CON JAVA 74

22.1 COMO SE HACE PARA IMPRIMIR TEXTO 74
22.2 COMO SE HACE PARA ACEPTAR UNA ENTRADA DE TECLADO 75
22.3 COMO SE HACE PARA CONVERTIR CADENAS A NÚMEROS 76
22.4 COMO SE HACE PARA CONVERTIR NÚMEROS A CADENAS 77
22.5 COMO SE HACE PARA ACCEDER A LA LINEA DE MANDATOS 78
22.6 COMO SE HACE PARA LEER DESDE UN ARCHIVO 80
22.7 COMO SE HACE PARA GRABAR EN UN ARCHIVO 81
22.8 COMO SE HACE PARA UTILIZAR MATRICES 83
22.9 COMO SE HACE PARA ANALIZAR UNA CADENA 85
22.10 COMO SE HACE PARA UTILIZAR FUNCIONES MATEMÁTICAS 87

23 GRÁFICOS BÁSICOS 89

23.1 DIBUJAR UNA LINEA 89
23.2 AÑADIR COLOR 91

23.3 DIBUJAR FORMAS 93
23.4 RELLENAR FORMAS 98
23.5 DIBUJAR TEXTO 103
23.6 UTILIZAR FUENTES 105
23.7 ACTUALIZACIONES DE PANTALLA 107

24 TRAMAS 109

25 EVENTOS Y PARÁMETROS 113

25.1 PARA ACCEDER A LOS EVENTOS DE TECLADO 113
25.2 EVENTOS DEL RATÓN 122
25.3 OTROS EVENTOS 126

26 INTERFAZ DE USUARIO 131

26.1 COMO CREAR BOTONES 131
26.2 COMO HACER LISTAS DE ELECCIÓN 138
26.3 DESLIZADORES Y AREAS DE TEXTO 140
26.4 COMO USAR LISTAS 143
26.5 CREAR VENTANAS 147
26.6 USAR DIALOGOS DE ARCHIVOS. 151

27 ALGUNOS PROGRAMAS AVANZADOS 157

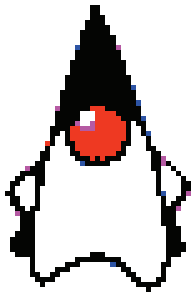
27.1 ANIMACIÓN Y SONIDO 157
27.2 REDES 166
27.3 PREVENIR EL ROBO DE APPLETS 174

28 REFERENCIA A LA API DE JAVA 185

29 BIBLIOGRAFÍA 186

30 ÍNDICE ANALÍTICO 187

PROGRAMACIÓN BÁSICA



1. Algoritmos y programas[1]

1.1 Introducción

Un algoritmo es un método para resolver un problema. El término algoritmo proviene de mohamed al-Khowârizmî, matemático persa que vivió en el siglo IX y alcanzó gran reputación por el enunciado de las reglas paso a paso para sumar, restar, multiplicar y dividir números decimales. El otro padre de la ciencia de los algoritmos o algoritmia es Euclides que inventó un método para encontrar el máximo común divisor de dos números.

En otras palabras un algoritmo es una fórmula para la resolución de un problema.

La resolución de un problema se base en identificar y definir los siguientes pasos:

1. Definición o análisis del problema
2. Diseño del algoritmo
3. Transformación del algoritmo en un programa
4. Ejecución y validación del problema

Cuando el algoritmo se realiza sobre una computadora, el algoritmo ha de expresarse de una forma que recibe el nombre de programa.

Existen dos conceptos fundamentales en el procesamiento de la información: Hardware y Software. Hardware es el conjunto de componentes físicos de una computadora y Software es el conjunto de programas que controlan el funcionamiento de una computadora.

Características de los algoritmos

Todo algoritmo debe ser:
- preciso
- definido
- finito.

La definición de un algoritmo debe describir tres partes:
- Entrada
- Proceso
- Salida.

Ejemplo de un algoritmo:

[1] Fundamentos de programación de Luis Joyanes

Realizar la suma de todos lo números pares entre 2 y 1000.

Utilizaremos las palabras SUMA y NUMERO para representar las sumas sucesivas.

1. Establecer SUMA a 2.
2. Establecer NUMERO a 4.
3. Sumar NUMERO a SUMA. El resultado será el nuevo valor de la suma (SUMA).
4. Incrementar NUMERO en 2 unidades
5. Si NUMERO =< 1000 bifurcar el paso 3; en caso contrario, escribir el último valor de la SUMA y terminar el proceso.

2 Los lenguajes de programación[2]

Como hemos dicho antes, cuando el que procesa el algoritmo es una computadora el formato se llama programa. Un programa se escribe en un lenguaje de programación y las operaciones que conducen a expresar un algoritmo en forma de programa se llaman programación.

Existen 3 tipos fundamentales de lenguajes de programación:

- Lenguaje máquina
- Lenguaje de bajo nivel (ensamblador)
- Lenguaje de alto nivel (C, Basic, Java)

2.1 Traductores de lenguajes

Los traductores de lenguajes son programas que traducen a su vez los programas fuente en lenguajes de alto nivel a código máquina.

Los traductores se dividen en:

- Compiladores
- Intérpretes

Un intérprete es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta.

Un compilador es un programa que traduce los programas fuente escritos en lenguaje de alto nivel a lenguaje máquina.

[2] Fundamentos de programaci

3 Tipos de datos[3]

Un dato es una expresión general que describe los objetos con los cuales opera una computadora.

Los tipos de datos más simples son:

- Numéricos (integer, real)
- lógicos (boolean)
- carácter (char, string)

De todos estos tipos básicos después algunos lenguajes tienen otros que no son mas que extensiones de estos.

3.1 Datos numéricos

El tipo numérico es el conjunto de los valores numéricos. Estos se pueden representar de dos formas:

- Entero (integer) –32768 a +32767
- Real (real)

Datos lógicos

El tipo lógico es aquel que solo puede tomar unos de los valores:

- Cierto
- Falso

Este tipo de datos se utiliza para representar las alternativas sí o no.

Datos de tipo carácter

Es el conjunto finito y ordenado de caracteres que la computadora reconoce.

[3] Fundamentos de programación de Luis Joyanes.

4 Constantes y variables[4]

Un programa de una computadora contiene ciertos valores que no deben cambiar durante la ejecución del programa. Tales valores se denominan constantes. De igual forma existen otros valores que cambiaran durante la ejecución del programa; a estos valores se les llama variables.

Tanto las constantes como las variables deben de ser de un determinado tipo comentado en el apartado anterior, y tendrán un ámbito.

El ámbito de una variable es la zona del programa en la cual esta variable o contante tendrá validez, puede ser en todo el programa (global) o en determinadas zonas del mismo, normalmente dentro de subprogramas que veremos mas adelante.

[4] Fundamentos de programación de Luis Joyanes

5 Expresiones[5]

Una expresión consta de operandos y operadores, según sea el tipo de objeto que manipulan, se clasifican las expresiones en:

- aritméticas
- relacionales
- lógicas
- carácter

5.1 Expresiones aritméticas

Son análogas a las fórmulas matemáticas. Las operaciones aritméticas mas comunes son:

- + suma
- - resta
- * multiplicación
- / división
- **, ^ exponenciación
- div división entera
- mod modulo (resto).

Las expresiones aritméticas tienen una serie de reglas de prioridad. Las expresiones que tienen dos o mas operandos requieren unas reglas matemáticas que permitan determinar el orden de las operaciones, estas son las que se denominan reglas de prioridad.

a) las operaciones que están entre paréntesis se evalúan primero
b) las operaciones aritméticas dentro de una expresión suelen seguir el siguiente orden de prioridad

   1. Operador exponencial **, ^
   2. Operadores *, /
   3. Operadores +, -
   4. Operadores div y mod

[5] Fundamentos de programación de Luis Joyanes.

Ejemplo:

```
3 + 6 + 15 * 6 - 2 = 3 + 6 + 90 - 2 = 97

3 + ( 6 + 15 ) * ( 6 - 2 ) = 3 + 21 * 4 = 3 + 84 = 87
```

5.2 Expresiones lógicas

Una expresión lógica es una expresión que sólo puede tomar estos dos valores:

- Verdadero
- Falso

Se denominan también expresiones booleanas en honor al matemático George Boole que desarrolló el álgebra lógica o de Boole.

Algunos de los operadores para expresiones lógicas son:

- < menor que
- > mayor que
- = igual que
- no (not) negación
- y (and) intersección
- o (or) unión

6 Pseudocódigo[6]

El pseudocódigo es un lenguaje de especificación de algoritmos. El uso de tal lenguaje hace el paso de codificación final relativamente fácil.

El pseudocódigo nació como un lenguaje similar al ingles y era un medio de representar básicamente las estructuras de control de programación estructurada.

El pseudocódigo utiliza para representar las acciones sucesivas palabras reservadas en inglés o en español similares a sus homónimas en los lenguajes de programación

- Start Inicio
- End Fin
- Stop Parar
- if-then-else Si-entonces-sino
- While-wend Mientras
- repeat-until Hasta

[6] Fundamentos de programación de Luis Joyanes.

7 Estructura general de un programa[7]

Un programa de una computadora es un conjunto de instrucciones que producirán la ejecución de una determinada tarea. En esencia, un programa es un medio para conseguir un fin. El fin será normalmente definido como la información necesaria para solucionar un problema.

7.1 Tipo de instrucciones

7.1.1 Instrucciones de asignación

A ← 4+5
B ← 6+7

7.1.2 Instrucciones de lectura de datos

Leer del terminal, normalmente el teclado, los valores que se especifiquen a continuación.

Leer NUMERO, FECHA

7.1.3 Instrucciones de escritura de resultados

Escribir en el terminal, normalmente la pantalla, los valores que se especifiquen a continuación.

Escribir NUMERO, FECHA

7.1.4 Instrucciones de bifurcación

El desarrollo lineal de un programa se interrumpe cuando se ejecuta una bifurcación. Las bifurcaciones pueden ser según el punto del programa a sonde se bifurca:

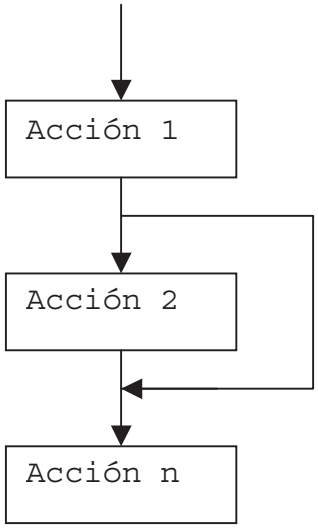
- hacia delante
- hacia atrás

[7] Fundamentos de programación de Luis Joyanes.

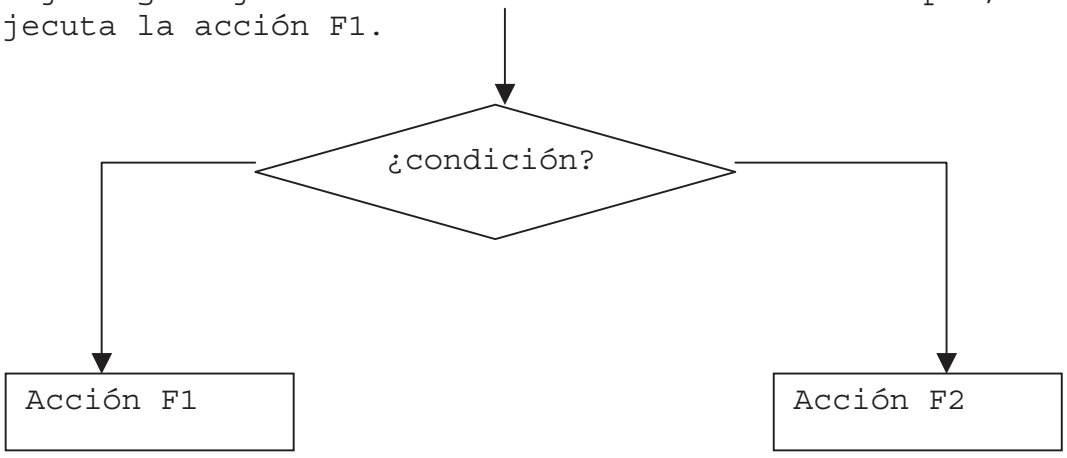
Existen dos importantes tipos de bifurcaciones:

- Condicionales
- Incondicionales

La bifurcación incondicional se realiza siempre que el flujo del programa pase por la instrucción sin necesidad del cumplimiento de ninguna condición.



La bifurcación condicional depende del cumplimiento de una determinada condición. Si se cumple la condición, el flujo sigue ejecutando la acción F2. Si no se cumple, se ejecuta la acción F1.



8 Ejemplos de programación estructurada[8]

Calcular la media de un conjunto de notas de alumnos. Pondremos un valor centinela de -99 que detecte el fin del bucle.

```
Inicio
     Total ← 0
     Media ← 0
     N = 0 {número de alumnos}
     Leer nota
     Mientras nota <> -99 hacer
          Total ← total + nota
          C ← C + nota
          Leer nota
     Fin_mientras
     Escribir 'la media es', total/C
Fin
```

Calcular el factorial de n números leídos del terminal.

El problema consistirá en realizar una estructura repetitiva de n iteraciones del algoritmo del problema ya conocido de cálculo del factorial de un entero

```
Inicio
     {lectura de la cantidad de números}
     leer n
     desde i = 1 hasta n hacer
          leer NUMERO
          FACTORIAL ← 1
          Desde j =1 hasta NUMERO hacer
               FACTORIAL ← FACTORIAL * j
          Fin_desde
          Escribir ' el factorial del número', NUMERO,
          'es', FACTORIAL
     Fin_desde
Fin
```

[8] Fundamentos de programación de Luis Joyanes.

9 Subprogramas, procedimientos funciones[9]

La resolución de problemas complejos se facilita considerablemente si se dividen los programas en problemas mas pequeños. La resolución de estos subproblemas se realiza mediante subalgoritmos.

Los subalgoritmos pueden ser de dos tipos:

- funciones
- procedimientos

9.1 Funciones

Matemáticamente una función es una operación que toma uno o más valores llamados argumentos y produce un valor denominado resultado. Todos los lenguajes de programación tienen funciones incorporadas o intrínsecas, y otras que puede definir el usuario.

```
Función nombre_funcion (param1, param2, ...)
     Acciones
Fin_función o Return
```

Por ejemplo:

```
Función tan(x)
     Tan ← seno(x) / coseno(x)
Return tan
```

[9] Fundamentos de programación de Luis Joyanes

Otro ejemplo:

Diseñar un algoritmo que contenga un subprograma de cálculo del factorial de un número y una llamada al mismo.

```
Función FACTORIAL (n)
Inicio
     F ← 1
     Desde i =1 hasta n hacer
          F ← f* i
     Fin_desde
     FACTORIAL ← F
Fin
Algoritmo factorial
Inicio
     Leer número
     X ← FACTORIAL (número)
     Y ← FACTORIAL (25)
     Escribir x, y
Fin
```

9.2 Procedimientos

Aunque las funciones son herramientas de programación muy útiles para la resolución de problemas, su alcance está muy limitado. Con frecuencia se requieren subprogramas que calculen varios resultados en vez de uno solo, o que realicen la ordenación de una serie de números, etc.

La declaración de un procedimiento es similar al de las funciones.

Las funciones y los procedimientos se diferencias en los siguientes aspectos:

1. Un procedimiento es llamado desde un algoritmo o programa principal mediante su nombre y una lista de parámetros actuales, o bien con la instrucción llamar (call). Al llamar al procedimiento se detiene momentáneamente el programa que se estuviese realizando y el control para al procedimiento llamado. Después que las acciones del mismo se ejecutan, se regresa a la acción inmediatamente siguiente a la que se llamó.
2. Las funciones devuelven un valor, las subrutinas pueden devolver 0,1 o n valores y en forma de lista de parámetros.

3. El procedimiento se declara igual que la función, pero su nombre no está asociado a ninguno de los resultados que obtiene.

Ejemplo:

Realizar un algoritmo simple de ordenación, utilizando un procedimiento llamado intercambio. Para realizar un intercambio es necesario lo siguiente:

```
AUX ← A
A ← B
B ← AUX

Procedimiento intercambio (A, B)
Inicio
     AUX ← A
     A ← B
     B ← AUX
Fin

Algoritmo ordenar
Inicio
     Leer X, Y, Z
     Si X>Y
          Entonces
               Intercambio (X, Y)
     Fin_si
     Si Y>Z
          Entonces
               Intercambio (Y, Z)
          Si X>Y
               Entonces
                    Intercambio (X, Y)
          Fin_si
     Fin_si
     Escribir A, B, C
Fin
```

Estructuras de datos (Arrays)[10]

Una estructura de datos es una colección de datos que pueden ser caracterizados por su organización y las operaciones que se definen en ella.
Arrays unidimensionales o vectores

Un array es un conjunto finito y ordenado de elementos homogéneos. Los elementos de un array son homogéneos, es decir del mismo tipo de datos.

Luis Francisco	1
Enrique	2
Alberto	3
Juan	4
Elena	5
Rosa	6
Ana	7

9.3 Operaciones con vectores

Los vectores son como las variables y las constantes, se pueden asignar, escribir o leer, pero con una determinada cualidad y es que hay que referirse a ellos con la posición que ocupan en la lista o array.

Z ← A(3) (Alberto en la tabla anterior)

9.4 Arrays bidimensionales

El array bidimensional se puede considerar como un vector de vectores. Es, por consiguiente, un conjunto de elementos todos de mismo tipo en el cual el orden de los componentes es significativo y en el que se necesitan especificar dos subíndices para poder identificar a cada elemento del array.

Si se visualiza un array unidimensional, se puede considerar como una columna de datos; un array bidimensional es un grupo de columnas como se ilustra en la figura.

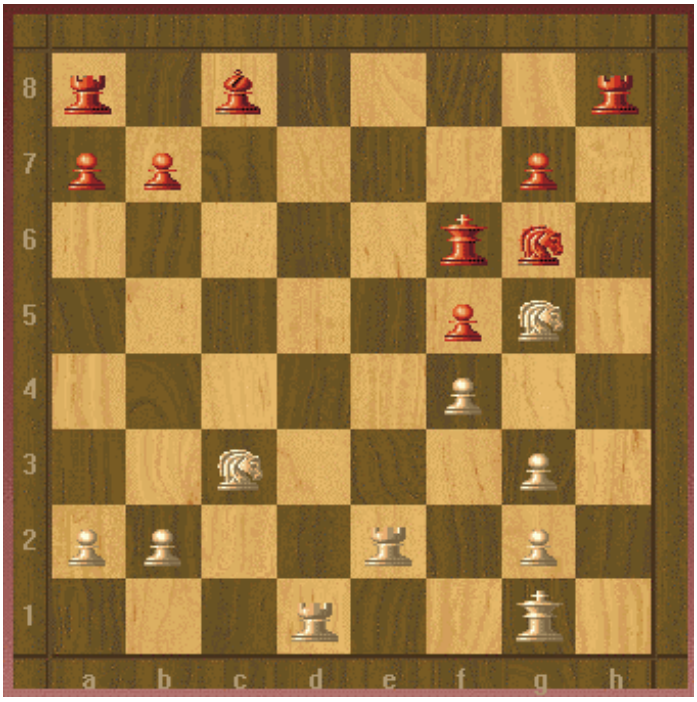
[10] Fundamentos de programación de Luis Joyanes.

		(i, j)			

Siendo i la fila y j la columna

Ejemplo

Un ejemplo típico de un array bidimensional es un tablero de ajedrez. Se puede representar cada posición o casilla del tablero mediante un array, en el que cada elemento es una casilla y en el que su valor será un código representativo de cada figura del juego.



Los diferentes elementos serán:

- elemento (i, j) = 0 no hay nada
- elemento (i, j) = 1 un peón blanco
- elemento (i, j) = 2 un caballo blanco
- Y así sucesivamente.

10 Archivos[11]

Las estructuras de datos de los enunciados anteriores se encuentran almacenadas en la memoria central. Sin embargo no siempre se puede almacenar los datos en la memoria central de la computadora debido a una serie de limitaciones:

- El tamaño de la memoria es escaso.
- Cuando el ordenador de apaga los datos se pierden.

Un archivo o fichero es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas registros o artículos que son de igual tipo y que constan, a su vez, de diferentes entidades de nivel mas bajo denominadas campos.

Un campo es un ítem o elemento de datos elementales, tales como un nombre, número de empleados, ciudad, etc.

Un registro es una colección de información, normalmente relativa a una entidad particular. Un registro es una colección de campos lógicamente relacionados que pueden ser tratados como una unidad por algún programa.

Un fichero o archivo es una colección de registros relacionados entre sí con aspectos en común y organizados para un propósito específico

Una base de datos es una colección de archivos a los que puede accederse por medio de un conjunto de programas y que contienen todos ellos datos relacionados, constituye una base de datos. Así una base de datos de una universidad puede contener archivos de estudiantes, archivos de nóminas, inventarios de equipos, etc.

[11] Fundamentos de programación de Luis Joyanes.

11 Algunos algoritmos importantes[12]

11.1 Ordenación

La ordenación o clasificación es el proceso de organizar datos en algún orden o secuencia específica tal como creciente o decreciente para datos numéricos o alfabéticamente para datos de caracteres.

Burbuja por el método 2.

```
Algoritmo burbuja2
Inicio
     Desde i = 1 hasta n-1 hacer
          Desde j = 1 hasta n-1 hacer
               Si X(J) > X(J+1)
                    Entonces
                         AUX ← X(J)
                         X(J) ← X(j+1)
                         X(J+1) ← AUX

               Fin_si
          Fin_desde
     Fin-desde
Fin
```

11.2 Búsqueda

La recuperación de información es una de las aplicaciones más importantes de las computadoras.

El algoritmo mas sencillo de búsqueda es el de búsqueda secuencial, no por ello el mas eficiente, pero servirá como ejemplo:

```
Algoritmo búsqueda
Inicio
     I ← 1
     Mientras (A[i] <> t) y (i<n) hacer
          I ← i+1
     Fin_mientras
     Si A[i] = t
     Entonces escribir ' El número buscado está en', i
     Sino escribir t 'no existe el vector'
Fin
```

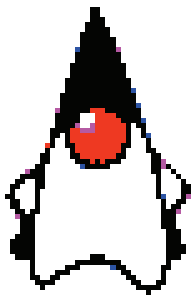
[12] Fundamentos de programación de Luis Joyanes.

11.3 Intercalación

La intercalación es el proceso de mezclar dos vectores ordenados y producir un nuevo vector ordenado.

```
Algoritmo intercalación
Inicio
     Leer A, B
     I ← 1
     J ← 1
     K ← 0
     Mientras i<=M y j<=N hacer
          K ← k+1
          Si A[i] < B[j]
               Entonces
                    C[k] ← A[i]
                    I ← i+1
               Sino
                    C[k]← B[j]
                    J ← j+1
          Fin_si
     Fin_mientras
     Si i <=M
          Entonces
               Desde r ← i hasta M hacer
                    K ← k+1
                    C[k] ← A[r]
               Fin_desde
          Sino
               Desde r ← j hasta N hacer
                    K ← k+1
                    C[k] ← B[r]
               Fin_desde
     Fin_si
     Escribir C
Fin
```

PROGRAMACIÓN ORIENTADA A OBJETOS

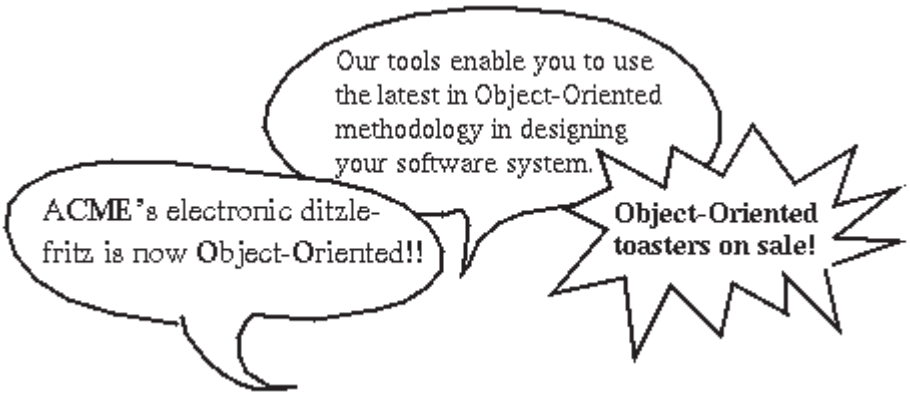


12 Programación Orientada a Objetos[13]

La programación orientada a objetos trae muchos beneficios a los desarrolladores de software y a sus productos. Sin embargo, hay un montón de exageraciones que rodean a esta tecnología creando confusión entre los programadores.

Muchas compañías son víctimas de estos problemas y reclaman que sus productos tienen tecnología orientada a objetos, cuando en realidad no es así. Esta falsa pretensión confunde a los consumidores causando una amplia desinformación y desconfianza hacia esta tecnología.

Sin embargo, a pesar del empleo excesivo y del abuso del termino orientado a objetos, la industria informática está comenzando a introducirse en este aspecto, a entender esta tecnología y a utilizar sus beneficios.



12.1 ¿Qué es un objeto?

Un objeto es una serie de variables relacionadas con una métodos o procedimientos.

A menudo se utiliza como un modelo del mundo real.

12.2 ¿Qué es un mensaje?

Los objetos interactuan con otros objetos utilizando mensajes.

[13] The Java Tutorial. Sun MicroSystem

12.3 ¿Qué es una clase?

Es un diseño o prototipo que define las variable y métodos comunes a todos los objetos de un cierto tipo.

12.4 ¿Qué es herencia?

Una clase hereda un estado y comportamiento de otra clase superior a esta.

13 ¿Qué es un objeto?

Como la misma palabra indica, objeto es una clave para entender la tecnología orientada a objetos.

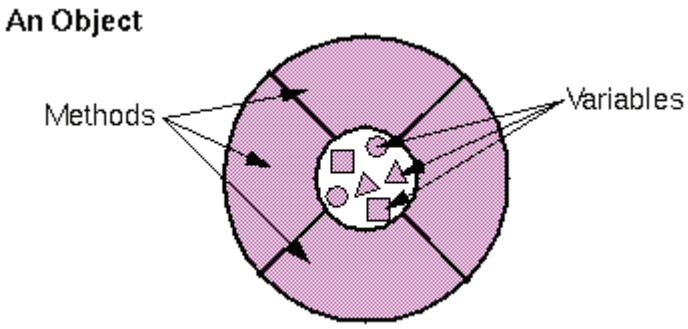
Una puede mirar a su alrededor y ver muchos ejemplos de objetos, como un perro, un televisor, un escritorio, una bicicleta, etc.

Estos objetos del mundo real comparten dos características:

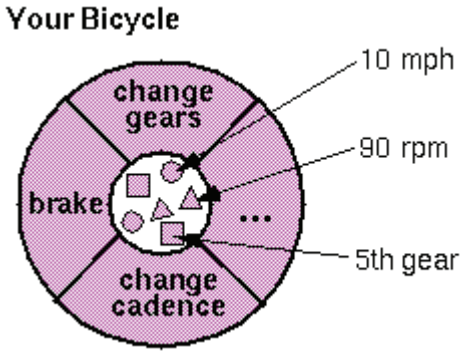
- Estado
- Comportamiento

Por ejemplo los perros tienen unos estados o atributos como son el color, el nombre, si tiene o no hambre, etc. Y además poseen unos comportamientos o métodos, como ladrar, ir a traer algo, babear etc.

Un objeto por lo tanto es una serie de instrucciones que contiene variable y métodos relacionados con el objeto.



Un ejemplo sería la bicicleta:



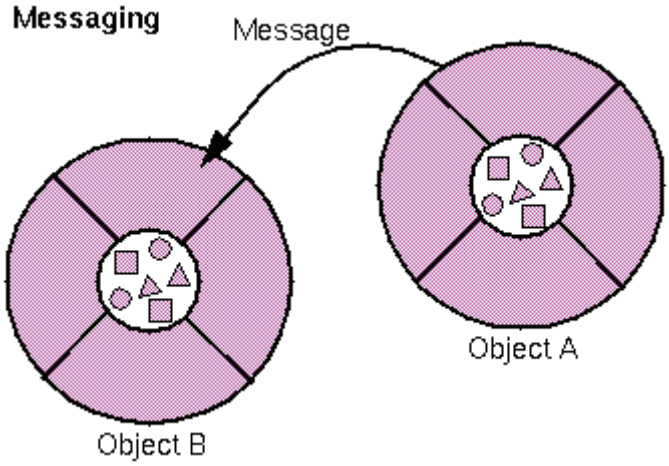
Como puede verse en los diagramas las variables o atributos están en el centro o núcleo del objeto. Los métodos o procedimientos están alrededor.

En muchos lenguajes, incluyendo JAVA un objeto puede permitir a otros que utilicen sus atributos, que modifiquen sus variables. También un objeto puede determinar que métodos serán utilizados por el resto de objetos y cuales no.

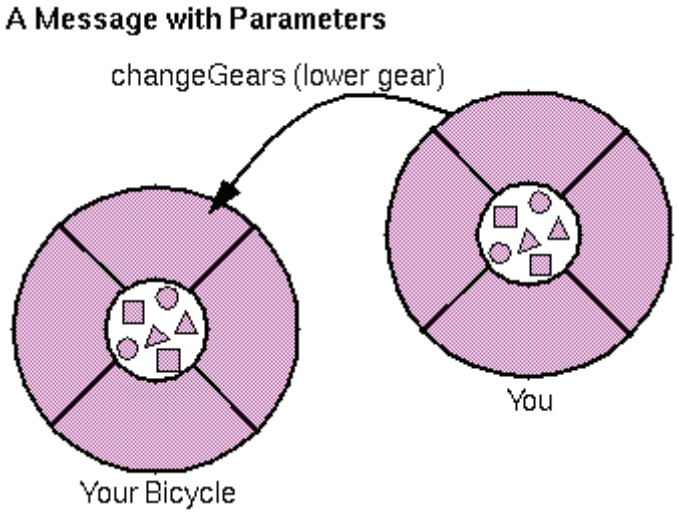
14 ¿Qué es un mensaje?

Un objeto aislado generalmente no tiene mucha utilidad, y normalmente suelen aparecer como componentes de otros o dentro de una gran estructura que contenga varios objetos. Por ejemplo una bicicleta es incapaz de hacer nada por si sola, necesita otro objeto persona que interactue con ella y que comience a pedalear.

Para comunicar estos dos objetos es por lo que se utilizan mensajes.



A veces un objeto desconoce ciertos parámetros que deben pasársele como mensaje al objeto, por ejemplo la bicicleta sabe cambiar una rueda y tiene en sus atributos rueda delantera y rueda trasera, pero como sabe el objeto que rueda cambiar, eso se lo podemos pasar como atributo.



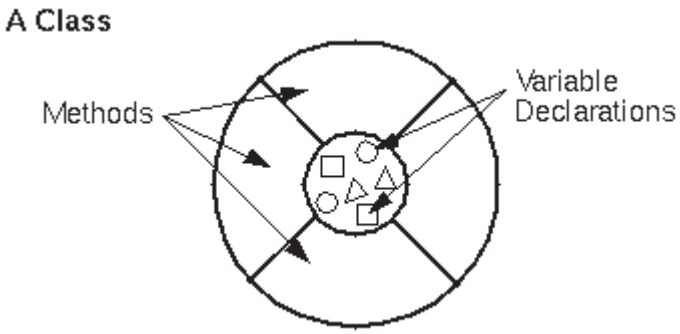
15 ¿Qué es una clase?

En el mundo real suele haber muchos objetos del mismo tipo, por ejemplo las bicicletas no son todas iguales, hay muchos tipos de bicicletas en el mundo.

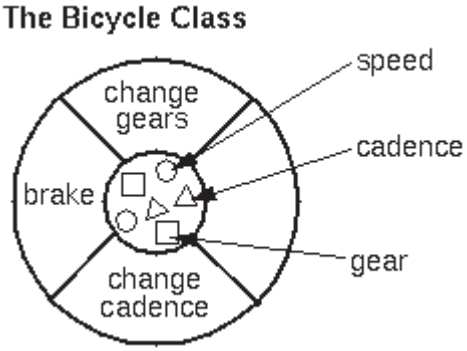
Utilizando la terminología orientada a objetos nosotros podemos decir que un objeto es una instancia de una clase.

Por ejemplo una clase denominada bicicleta donde haya diversos tipos de bicicletas, de 2 ruedas, triciclos, motorizadas, etc.

En pocas palabras, una clase es un diseño o prototipo que define variables y métodos comunes a una serie de objetos de un cierto tipo.



El ejemplo que estamos tratando de la bicicleta:



16 Programación Orientada a Objetos en JAVA[14]

Como se ha indicado anteriormente Java es un lenguaje orientado a objetos.

A grandes rasgos, la programación orientada a objetos se base en el desarrollo de componentes software autosuficientes, denominados objetos.

Estos objetos se definen en base a la información que contienen así como a las operaciones que ofrecen para manipular dicho datos.

Una clase es una estructura en la que define una colección de datos y métodos que operan con dichos datos. De alguna forma, una clase es un compendio de las características y propiedades del elemento modelado, a partir del cual crearemos objetos particulares sobre los que trabajar.

Por ejemplo, un rectángulo queda totalmente definido a partir de las coordenadas x e y de su esquina inferior izquierda, así como su anchura y su altura.

Estos serían los datos intrínsecos de todo rectángulo; sobre ellos se pueden aplicar diversas funciones, como por ejemplo, mover su origen, cambiar sus dimensiones, calcular su área, etc.

El conjunto de datos y funciones que operan sobre los mismos son comunes a todos los rectángulos, lo que define una clase:

Clase rectángulo

```
public class rectángulo{
     public double x,y;
     public double ancho,alto;
     public void mover(int a, int b){
          x=a;
          y=b;
     }
     public void redimensionar (int a, int b){
          ancho=a;
          ancho=b;
     }
     public double área() {return x*y}
}
```

[14] Java. El lenguaje de Internet. Sergio Rios Aguilar

Sin embargo esta clase no es mas que la definición de lo se entiende por rectángulo. Es necesario ahora poder trabajar con entidades individuales, concretas, pertenecientes a esa clase: los objetos.

Un objeto es la instancia de una clase.

Para crear un objeto de la clase rectángulo, se utilizarían las sentencias:

```
rectángulo miRectangulo;
miRectangulo= new rectángulo();
```

miRectangulo es simplemente un nombre que permite hacer referencia a un objeto de tipo rectángulo. Para poder crear el objeto en sí, es preciso asignarle dinámicamente espacio de almacenamiento, lo que se consigue con el operador new.

Hecho esto, el objeto miRectangulo puede usar sus datos tal y como se muestra en el siguiente ejemplo:

```
rectángulo miRectangulo = new rectángulo();
miRectangulo.x=10.0;
miRectangulo.y=15.0;
miRectangulo.ancho=70,0;
miRectangulo.alto=25.0;
```

Así mismo, se puede acceder directamente a los métodos del objeto.

```
double valor1;
valor1 = miRectangulo.area();
miRectangulo.mover(24.0,16.0);
```

Obsérvese con especial detenimiento como se ha calculado el área del objeto miRectangulo: se ha aplicado un método que opera directamente sobre el objeto. En programación clásica, se hubiera usado una función, a la que se pasa como argumento los datos del rectángulo.

16.1 Creación de objetos

Retornando el ejemplo anterior, se observa un detalle interesante: al crear el objeto con el operador new se utiliza algo que parece ser una función:

```
rectángulo miRectangulo = new rectángulo();
```

En efecto rectángulo es una función muy especial, denominada "método constructor". En Java, todas las clases disponen de al menos un constructor, que se encarga de realizar todas las operaciones de inicialización necesarias para crear el nuevo objeto.

En la clase rectángulo no se ha definido ningún constructor, por lo que Java utiliza un constructor por defecto, que no toma argumentos ni realiza operación.

Para definir un constructor se utiliza un método que tenga el mismo nombre de la clase y cualquier número de parámetros.

Por ejemplo, podemos modificar la clase rectángulo para suministrar un constructor, incluyendo un nuevo método:

```
public rectángulo(double a, double b, double c, double d) {
     x=a;
     y=b;
     ancho=c;
     alto=d;
}
```

En este caso para inicializar el objeto miRectangulo con los mismos valores que en el ejemplo anterior, bastaría con crearlo como sigue:

```
rectángulo miRectangulo= new rectángulo(10,15,70,25);
```

También es posible el uso de varios constructores que permitan inicializar el objeto de distintas maneras. Por ejemplo, suele ser interesante poder inicializar un objeto con los valores de otro ya existente.

En este caso, se añade un nuevo constructor:

```
public rectángulo (rectángulo otro) {
     x=otro.x;
     y=otro.y;
     ancho=otro.ancho;
     alto=otro.alto;
}
```

Ahora se puede crear el objeto dos a partir del objeto uno:

```
rectángulo uno=new rectángulo(10,15,70,25);
rectángulo dos=new rectángulo(uno);
```

En este ejemplo se plantea una pregunta interesante: si ambos constructores tienen el mismo nombre ¿cómo sabe Java de cual de ellos se trata?.

La respuesta apunta a una de las características mas importantes de la programación orientada a objetos: el polimorfismo, es decir, la capacidad de usar varios métodos con el mismo nombre y distintos argumentos.

16.2 Variables estáticas

En el ejemplo que nos ocupa, las variables "x", "y", "ancho" y "alto" existen de forma específica para cada objeto creado.

De forma complementaria, existe posibilidad de crear una variable común a toda una clase, esto es, una única variable que existe con independencia del número de instancias creadas a partir de la clase (existe incluso si no se crea ninguna instancia).

Este tipo de variables se denomina variables estáticas, y se definen usando la palabra reservada static.

un uso muy común de este tipo de variables consiste en almacenar información sobre el número de instancias (objetos) creadas a partir de una clase.

Se puede modificar la clase rectángulo añadiendo información de modo que queda así:

```
public class rectángulo {

     static int num_rects = 0;
     public double x,y;
     public ancho, alto;
     public rectángulo(double a, double b, double c,
     double d) {
          x=a;
          y=b;
          ancho=c;
          alto=d;
          num_rects++;
     }

     public rectángulo (rectángulo otro) {
          x=otro.x;
          y=otro.y;
          ancho=otro.ancho;
          alto=otro.alto;
          num_rects++;
     }
     public void mover(int a,int b) {
          x=a;
          y=b;
     }

     public void redimensionar(int a, int b) {
          ancho=a;
          alto=b;
     }

     public double área() {return x*y; }

}
```

Para acceder a una variable estática se especifica directamente:

```
int valor;
valor=rectangulo.num_rects;
```

16.3 Métodos de clase

Al igual que existen variables estáticas comunes a toda una clase, en Java también está permitido el uso de métodos estáticos

16.4 Destrucción de objetos

Java no utiliza destructores como sucede en C++, ya que dispone de un sofisticado mecanismo para recuperar automáticamente el espacio de almacenamiento reservado por todos lo objetos no usados, esto es, los que se salen del ámbito de ejecución.

Este mecanismo está implementado por un programa conocido con el nombre de "recolector automático de basura", que se ejecuta como un hilo de baja prioridad de forma concurrente con los programas normales de usuario.

16.5 Herencia

Es el mecanismo por el cual se puede definir una nueva clase en función de otras existentes, añadiendo nuevas posibilidades que hacen de la nueva clase una más especializada que las que hereda su estructura.

Por ejemplo, podríamos crear una denominada "Ordenador", en la que se definen datos y métodos genéricos aplicables a todo tipo de ordenadores (tipo y número de CPUs, cantidad de memoria principal y de almacenamiento secundario...).

A partir de la clase Ordenador se podrán derivar nuevas clases como OrdenadorPersonal y EstaciónTrabajo, que heredan todas las propiedades de un Ordenador genérico y además añaden nuevas características especializadas.

Según la terminología oficial de la programación orientada a objetos, OrdenadorPersonal y EstaciónTrabajo son subclases o clases derivadas de Ordenador. Análogamente Ordenador es una Superclase de las otras.

En Java la palabra reservada extends permite crear clases derivadas de otras.

A modo de ejemplo, se puede crear una clase derivada de la ya conocida clase rectángulo, que llamaremos rectColor y que incluya información (no pública) sobre el color, así como también un método para modificar el mencionado color:

```
public class rectColor extends rectangulo {

     Color relleno;
     public void cambiar_color(Color nuevo){
          relleno=nuevo;
     }
}
```

La clase rectColor hereda todos los métodos de la clase rectángulo, por lo que se ha creado un objeto unRectColor a partir de rectColor es factible usar la sentencia:

```
unRectColor.redimensionar(23.4, 45.1);
```

En principio, no hay límite en cuanto a niveles de derivación de clases que se pueden utilizar. No es aconsejable exagerar, porque se podría perder el control.

16.6 Ocultación y encapsulamiento de los datos

En programación orientada a objetos se utiliza una técnica muy importante que permite incrementar notablemente la fiabilidad y las posibilidades de reutilización del código: el encapsulamiento de información.

Se trata de lograr que los datos de una clase queden ocultos al exterior, de modo que la única forma de acceder a los mismos sea a través de métodos diseñados específicamente para tal propósito.

Cuando se declara un método, dentro de una clase Java, es posible permitir o bloquear el acceso al mismo desde otras clases y objetos.

Este mecanismo de control de acceso se implementa gracias a los especificadores de acceso definidos en Java:

- private
- private protected
- protected
- public
- friendly

Para interpretar esta tabla, suponemos que se define en una clase un método y se le aplican lo distintos especificadores de acceso.

En cada columna de la tabla se indica un ámbito del programa: desde la propia clase, desde una subclase de la misma, desde el paquete en que se encuentra la clase y, por último, desde cualquier punto del programa.

	Clase	Subclase	Paquete	Todos
private	Si	No	No	No
private protected	Si	Si	No	No
protected	Si	Si	Si	No
public	Si	Si	Si	Si
friendly	Si	No	Si	No

16.7 Clases abstractas

Se define una clase abstracta como la que tiene al menos un método abstracto.

Un método abstracto es el que ha sido definido usando la palabra reservada abstract, pero no implementado.

Por tanto, la implementación de los métodos abstractos de una clase abstracta deberá realizarse mediante el mecanismo de redefinición de métodos en subclases.

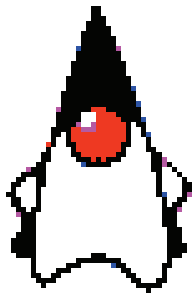
16.8 Redefinición de métodos

Una subclase puede definir un método heredado de su superclase, de tal manera de tal manera que cuando se invoque este método sobre un objeto de la subclase, siempre se realizará una llamada al nuevo método.

Si desde una subclase se quiere acceder a los datos o métodos de su superclase y que han sido redefinidos, se puede utilizar la palabra reservada super, como se muestra a continuación:

```
class principal  {
     double valor1=1.0;
     int a() {return valor1;}
}

class derivada extends principal {
     int a() { return (super.a() + 5.0);
}
```

JAVA



17 Introducción al lenguaje Java[15]

17.1 El nacimiento del lenguaje Java

El proyecto Java denominado en un principio proyecto OAK, tenía como objetivo la creación de un lenguajes que permitiera solucionar los problemas inherentes del desarrollo de aplicaciones distribuidas en redes con sistemas heterogéneos.

Inicialmente se eligió el lenguaje C++ como punto de partida, pero las dificultades fueron tales que se prefirió crear un entorno de programación totalmente nuevo. En este se incluirían los aspectos mas sobresalientes de varios lenguajes como Eiffel, SmallTalk, etc. utilizando la sintaxis mas conocida hoy en día C y C++.

El resultado ha siso lo que hoy se conoce como lenguaje Java.

James Gosling, socio y vicepresidente de Sun Microsystem, es el creador de Java y uno de los mejores programadores del planeta.

17.2 Características del lenguaje Java

A continuación se realizará un breve recorrido por las características mas sobresalientes del lenguaje Java, para lo que se presupone un cierto conocimiento de la terminología y técnicas usadas en la programación orientada a objetos, por ello se han visto en los capítulos anteriores del manual.

17.2.1 Simplicidad

Hoy en día la mayoría de los programadores utilizan en sus desarrollos los lenguajes C y C++. La sintaxis de Java es muy similar a la de C++, con lo que se consigue que su programación no resulta demasiado complicada ni implique grandes cambios en los hábitos de programación.

[15] Java. El lenguaje de Internet. Sergio Rios Aguilar

Java simplifica en gran medida el desarrollo de aplicaciones, pues con él se reduce parte de las características más conflictivas del lenguaje C++.

A continuación se fijan algunas de las principales diferencias con C++.

- Se ha eliminado el preprocesador, con lo que desaparecen las directivas del tipo #include. Ello también implica la desaparición de los archivos de cabecera *.h. En efecto, éstos han sido sustituidos por estructuras denominadas Interfaces, que permiten definir los puntos de entrada a las clases sin ofrecer información detallada sobre la implementación de las mismas.
- No existe herencia múltiple aunque se pueda lograr indirectamente usando interfaces.
- Se ha eliminado la sentencia goto, que ya era hora, aunque siguen pudiéndose utilizar saltos y etiquetas con otra estructura.
- No existe la sobrecarga de operadores.
- Se incluido un recolector de basura, lo que garantiza que el número de errores de compilación disminuya enormemente.
- No existe conversión implícita de tipos, habitual en otros lenguajes.
- Incluye mecanismos de concurrencia basados en threads o hilos de ejecución, no en procesos como sucede con ADA 95.

Este conjunto de características posibilitan que el aprendizaje de Java se consiga en pocos días, sobretodo para los que ya conocen C++.

17.2.2 Orientado a objetos

Java cumple con todos los requisitos necesarios para ser considerado un lenguaje de programación orientada a objetos:

- Encapsulación
- Herencia
- Polimorfismo
- Enlace dinámico

Suele ser muy habitual comenzar con la programación en Java con el programa HolaMundo:

```
class HolaMundo {
     static public void main(String args[]) {
          System.out.println("Hola Mundo!!!");
          }
}
```

En la clase HolaMundo se ha definido el método main() que llama al método println del objeto out. Este objeto out es una variable de la clase System, y se encarga de las operaciones de salida relacionadas con ficheros.

17.2.3 Distribuido

Java dispone de una extensa librería de rutinas que permiten utilizar diversos protocolos basados en el stack TCP/IP, como es el caso de HTTP o FTP.

Las aplicaciones Java pueden acceder a objetos de la red mediante URLs, con la misma facilidad con que normalmente se accede a los ficheros en un sistema local.

17.2.4 Arquitectura neutra, interpretada, portable y segura.

Para solucionar el problema de la distribución de aplicaciones en un mundo heterogéneo de diferentes máquinas y sistemas operativos se optado por usar una arquitectura neutra y portable, basándose en el concepto de "máquina virtual".

Los programas Java se compilan y se obtiene como resultado código binario en un formato especial para una máquina virtual Java, independiente de cualquier sistema.

En cuanto a la seguridad, uno de los puntos fuertes de este lenguajes lo constituye su modelo de memoria. Éste es

totalmente opaco al programador, ya que las decisiones de gestión de memoria no se realizan en tiempo de compilación, sino en tiempo de ejecución. Además, el sistema run-time de Java desconfía del código binario que recibe y lo somete a un proceso de verificación.

Por último cuando se importa una clase desde la red, esta queda confinada en una espacio de nombres asociado a su origen, por lo que no es posible la suplantación de una clase local.

17.2.5 Concurrencia

Java permite la concurrencia basada en hilos, lo que supone una poderosa herramienta para la mejora de la ejecución interactiva de las aplicaciones gráficas.

En las librerías de Java está disponible la clase Thread, que permite un completo tratamiento de los hilos de ejecución: inicio, ejecución, detención y consulta de estados.

Se usa un sofisticado conjunto de primitivas de sincronización basadas en el conocido paradigma de monitores y variables.

Los hilos de Java usan mecanismos de requisa y dependiendo de la plataforma en la que se ejecute el intérprete, estos también pueden implementarse usando intervalos de tiempo.

Una última observación: en tiempo de ejecución, siempre se reserva una hilo de ejecución de baja prioridad para activar en segundo plano todas las funciones del recolector automático de basura antes mencionado.

17.3 Java y el World Wide Web

Una de las principales limitaciones del WWW es su carácter eminentemente estático. En efecto, pese a que el usuario puede apreciar un mayor o menos grado de

interactividad (navegando por las distintas páginas, comunicándose con los servidores, usando formularios, etc, lo cierto es que una vez que se ha cargado una página en el cliente WEB, esta permanecerá estática.

Hasta ahora, incluso utilizando las técnicas mas sofisticadas de codificación en HTML, no era posible obtener resultados similares a los logrados en la producción de títulos multimedia basados en CD-ROM.

Se crearon dos mecanismos de actualización de páginas, denominados Server Push y Client Pull, que podrían solucionar de alguna manera la estaticidad de las páginas. Sin embargo, estos mecanismos son claramente ineficientes y no constituyen una buena opción para el problema.

Afortunadamente, Sun MicroSystems presentó oficialmente el lenguaje Java en el verano de 1995, que ha supuesto una verdadera revolución en el diseño de sistemas de información basados en el World Wide Web.

Ahora es posible ejecutar pequeños programas escritos en Java (que en algún caso reciben el nombre de Applets) dentro del cliente Web. La salida de estos programas puede aparecer integrada en una página, como si de una imagen se tratara, o bien en una ventana creada para tal efecto, sin que haya necesidad de salirse del cliente Web.

17.4 El futuro de Java

En numerosas ocasiones se ha dicho que Java es mucho mas sencillo que C++. Probablemente sea cierto, pero en la mayoría de los casos, la dificultad no provienen del propio lenguaje, sino que mas bien estriba en la comprensión de la programación orientada a objetos.

Tal como se ha comentado en el apartado anterior, Java ha supuesto toda una revolución en el Web, pero aún estamos asistiendo a los inicios de esta nueva era en Internet, y Java tiene que demostrar su madurez.

Ciertamente, ya existen numerosos ejemplos de lo que se puede realizar con Java. Sin embargo, la mayor parte de los applets escritos en Java disponibles hoy en día son miniaplicaciones o applets que generan todo tipo de efectos especiales. En cualquier caso se espera que un corto

periodo de tiempo se comiencen a realizar aplicaciones de utilidad real.

18 Referencia básica del lenguaje Java[16]

18.1 Generalidades sobre tipos, valores y variables

Java es un lenguaje en el que todas y cada una de las variables y expresiones propias de lenguajes son de un tipo específico, conocido en tiempo de compilación.

Estos tipos restringen todos los valores que puede tomar una variable determinada, o los valores generados como resultado de una expresión. Asimismo, dichos tipos pueden limitar y dar un significado concreto a las operaciones aplicadas a los valores en cuestión.

La comprobación estricta de tipos empleada en Java resulta de grana ayuda en el desarrollo de programas, puesto que permite detectar errores en tiempo de compilación.

Los tipos del lenguaje Java se dividen en dos categorías:

- primitivos
- de referencia

Se consideran primitivos el tipo booleano y los diferentes tipos numéricos.
Los de referencia son los relativos a arrays, clases e interfaces. También se debe añadir un tipo especial, el nulo (null).

Por otra parte, cabe señalar que en Java, un objeto es una instancia generada dinámicamente a partir de un tipo clase o de un array creado dinámicamente.
Los valores de un tipo de referencia son "referencias" a objetos. Todos ellos, incluyendo los arrays, soportan métodos de la clase Object. Dos tipos se consideran idénticos si poseen nombres similares totalmente calificados y si son utilizados por el mismo cargador de clases.

[16] Java. El lenguaje de Internet. Sergio Rios Aguilar

Los nombres de los tipos se usan en declaraciones, conversiones, en expresiones intanceof y en las que se crean arrays. Una variable es un espacio de almacenamiento (normalmente una posición de memoria). En Java, una variable de un tipo primitivo solo puede almacenar un valor de ese tipo concreto.

En lo que se refiere a los tipos de referencia, hay que señalar que se denominan así porque el valor de una de estas variables es un puntero, referente al valor o conjunto representados por dicha variable.

Sobre los tipos de referencia, es preciso diferenciar los valores que pueden tomar sus variables:

- Una variable de tipo clase puede contener una referencia nula o una referencia a una instancia de esa misma clase o de cualquier subclase de la misma.
- Una variable de tipo interfaz puede contener una referencia nula u otra a cualquier instancia de cualquier clase que implemente su interfaz.
- Una variable de tipo objeto puede contener una referencia nula u otra a cualquier objeto, pudiendo ser éste una instancia de una clase o un array.

Como ya se ha adelantado, existe también un tipo muy especial: el tipo nulo, que no tiene nombre. de este modo, resulta imposible declarar una variable especificando que es de tipo nulo. Por supuesto, tampoco se puede efectuar una conversión explícita de un tipo cualquiera al nulo.

18.2 Tipos primitivos y sus valores

Los tipos primitivos de los que se disponen en el lenguaje Java son los siguientes:

- Tipos numéricos
  - enteros: byte, short, int, long, char
  - coma flotante: float, double
- Tipo booleano: boolean

No es posible intercalar valores de distintos tipos primitivos, por lo que una variable de un determinado tipo siempre guarda un valor del mismo.

Como se puede apreciar en la tabla adjunta, los tipos enteros son de 8, 16, 32 y 64 bits, que el codificador de Java codifica internamente en complemento a dos.

Por su parte, el tipo char es de código Unicode usando 16 bits.

Los valores de coma flotante son de 32 y 64 bits, codificados según la norma IEEE754.

por último, el tipo boolean puede tomar únicamente dos valores: verdadero o falso. Hay que destacar que no es posible convertir una variable tipo boolean en ningún otro tipo numérico.

18.2.1 Tipos enteros y sus valores

Los tipos enteros pueden tomar como valores números enteros comprendidos entre:

- byte: -128 a 127
- short: -32768 a 32767
- int –2147483648 a 2147483647
- long: -9223372036854775808 a 9223372036854775807
- char: \u0000 a \uffff (0-65535)

Es posible realizar conversiones explícitas entre un valor de tipo entero y cualquier otro de tipo numérico.

Si no está familiarizado con los caracteres Unicode no debe preocuparse.
Java no dispone de ningún mecanismo para calcular el tamaño de una variable y tampoco permite ninguna clase de operaciones de aritmética de punteros.
Esto significa que si se utiliza caracteres ASCII o ISO-Latin1, no se puede distinguir un char Java de uno del lenguaje C.

18.3 Tipos de referencia y sus valores

Como ya se ha adelantado los tipos no primitivos se denominan tipos de referencia por que se gestionan mediante referencias. Por el contrario los tipos primitivos se gestionan por valor, lo que significa que los valores primitivos reales se almacenan en variables o se envían a los métodos.

En C y C++ es posible manipular un valor por referencia usando su dirección, que se obtienen aplicando el operador &. Una vez conseguida la dirección, se accede al valor directamente gracias a los operadores * y ->.

Este tipo de operadores no existen en Java, por lo que se aplica una regla general: los tipos primitivos siempre se pasan por valor, mientras que los arrays y los objetos se pasan por referencia.

18.4 Identificadores

Un identificador es una secuencia de longitud ilimitada compuesta por letras y dígitos válidos en Java, siendo el primer elemento de la misma una letra.

Las letras y los dígitos pueden ser cualquiera de los de Unicode, que permite codificar hasta 34.168 caracteres distintos, prácticamente todos los alfabetos existentes en el mundo. Esto permite usar identificadores escritos en su propio idioma.

En este lenguaje se incluyen las letras ASCII/LATIN. y por motivos históricos también se han mantenido el subrayado _ y el dólar $.

18.5 Variables

Un programa hace referencia al valor de una variable a través del identificador asociado a la misma.

Por convenio, los nombres de las variables en Java empiezan por una letra minúscula. Si su nombre consta de varias palabras, estas se unen de forma que la segunda y sucesivas empiecen por mayúscula.

Pueden existir variables con el mismo nombre siempre que hayan sido declaradas en ámbitos diferentes.

18.5.1 Ámbito de las variables

El ámbito de una variable es el bloque de código dentro del cual se encuentra dicha variable.

Por otro lado el ámbito determina cuando se crea y se destruye una variable.

Los ámbitos pueden ser:

- variable local
- variable miembro(de una clase)
- parámetro de un método
- parámetro de un gestor de excepciones
- Etc.

18.5.2 Valores iniciales de las variables

Toda variable de un programa en Java deberá tener un valor antes de poder ser usada.
las variables de una clase, de instancia o componentes de un array se inicializan con un valor por defecto, el 0 en numéricos, el false boolean y el \u0000 en char.

18.6 Palabras reservadas

Las siguientes secuencias de caracteres son palabras reservadas en Java, es decir no pueden utilizarse como identificadores de ningún tipo:

- abstract
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- extends
- final
- finally
- float
- for
- goto
- if
- throws
- import

- return
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- try
- short
- static
- super
- switch
- synchronized
- this
- throw
- implements
- trasient
- instanceof
- void
- volatile
- while

Como puede verse las palabras const y goto están reservadas aunque no se usan en Java. Esto permite a los compiladores detectar con mayor facilidad el uso erróneo de estas palabras reservadas de C/C++ en programas Java.

Pese a que true y false podrán parecer palabras reservadas, técnicamente con literales booleanos. De la misma forma, null es técnicamente el literal nulo.

18.6.1 Comentarios

Java define tres tipos de comentarios válidos para documentar y comentar los programas:

```
/* comentario de tipo 1 */
```

Son los comentarios tradicionales de C y C++.

```
// Comentario de tipo 2
```

Comentario de una sola línea, idéntico al de C++

```
/** Documentación */
```

Comentario especial para generación automatizada de documentos. Este texto es procesado por JavaDoc para crear una documentación automáticamente en HTML.

18.7 Unicode y secuencias de escape

Los programas Java se escriben con caracteres Unicode versión 2.0. La codificación de los 128 primeros caracteres coincide con los ASCII.

En un programa Java, se traducen las secuencias de caracteres ASCII \u seguidas de cuatro dígitos hexadecimales.

18.7.1 Secuencias de escape para literales de caracteres y cadenas

Java utiliza secuencias de escape especiales para la representación de algunos caracteres no gráficos, así como las comillas simples, las dobles y la barra invertida en literales de caracteres y cadenas.

En la siguiente tabla se enumeran las distintas secuencias de escape:

```
\b   \u0008: Retroceso
\t   \u0009: Tabulación horizontal
\n   \u000a: Salto de línea
\f   \u000c: Salto de página
\r   \u000d: Retorno de carro
\”   \u0022: Comillas dobles
\’   \u0027: Comillas simples
\\   \u005c: Barra invertida
\xxx \u0000 a \u00ff: Valores octales
\uxxxx    Caracteres Unicode
```

18.8 Operadores

Los operadores son elementos léxicos que permiten indicar determinadas operaciones sobre uno o dos operandos.

Los operadores que sólo requieren un operando se denominan unarios como por ejemplo ++.

En Java estos pueden aplicarse siguiendo una notación prefija o postfija, es decir delante o detrás.

Los operadores que necesitan dos operandos se denominan binarios, como por ejemplo =, >, >, etc.

Denotaremos ahora la tabla con los 37 operadores que existen en Java, el uso de ellos ya ha sido expuesto en apartados anteriores de este manual.

=	>	<	!	~	?	:
==	<=	>=	!=	&&	||	++
--	+	-	*	/	&	|
^	%	<<	>>	>>>	+=	-=
*=	/=	&=	|=	^=	%=	<<=
>>=	>>>=					

18.9 Control de flujo

Muchas de las sentencias de control de Java son prácticamente idénticas a C y C++.

El único detalle diferenciador que se debe tener muy en cuenta es que en el lenguaje Java los tipos boolean no se pueden convertir a otros tipos.

Por tanto, es necesario tener cuidado a la hora de codificar las expresiones que se utilizan como condición de salida en las diferentes estructuras de control, ya que solo se admitirán expresiones que den como resultado un booleano.

En Java los valores 0 y null no equivalen a false, por lo tanto los que no lo son tampoco equivalen a true.

18.9.1 Estructuras de control en Java

if / else

```
     if (bolean) {
          sentencias;
          }
     else {
          sentencias;
          }
```

Switch

```
     switch (expresion1) {
          case expresion2:
               sentencias;
               break;
          case expresion3:
               sentencias;
               break;
          default:
               sentencias;
               break;
          }
```

for

```
     for ( expr1 inicio; expr2 test_salida; espr3 incr) {
          sentencias;
          }
```

while

```
     while (bolean) {
          sentencias;
          }
```

do/while

```
     do {
          sentencias;
          } while (bolean);
```

En lo que se refiere al bucle for, es preciso hacer dos observaciones de particular interés:

El bucle for de Java no permite el uso del operador [,] del lenguaje C que posibilita la unión de diversas expresiones en una única expresión. En cualquier caso, Java intenta simular esta característica permitiendo la inclusión de múltiples expresiones separadas por comas en las secciones de inicialización y de incremento.

Por otra parte, Java admite la declaración de variables locales en la sección de inicialización de bucles for. El ámbito de estas variables es el propio bucle for.

Por último, se indica que en Java también disponemos de una serie de sentencias complementarias a las anteriores que permiten efectuar al flujo de ejecución de un programa de forma general. Su significado es prácticamente idéntico al que tienen en C, con la salvedad que en Java es posible usar una etiqueta.

```
etiqueta: sentencia;
          break [etiqueta]
          continue [etiqueta]
          return expr;
```

18.10 Arrays y cadenas

Como sucede en la mayoría de los lenguajes de programación, Java le permite agrupar sentencias de valores de un tipo en una construcción de almacenamiento que recibe el nombre de array.

En el caso de que todos los valores sean caracteres esto se llama String.

18.10.1 Arrays

Es preciso en Java declarar un array antes de usarlo. La declaración consta de dos elementos:

- array
- identificador

```
int[] arrayEnteros;
```

Es de suma importancia entender que la declaración de un array no implica ninguna asignación de memoria para almacenar sus elementos.

Si un programa intenta acceder al valor de una elemento en un array antes de que se le haya asignado espacio el compilador abortará el proceso de compilación.

Por ejemplo, cuando desee asignar espacio de almacenamiento en memoria para un array de entero de cien elementos, habría que utilizar la siguiente sentencia:

```
int [] arrayEnteros = new int[100]
```

Una vez hecho esto se pueden manipular sus elementos sin ningún problema excepto el no pasarse del límite del array (los arrays en Java son fijos), esto daría una excepción que trataremos mas adelante.

```
for (int = 0; i < arrayEnteros.length; i++) {
     arrayEnteros[i]=i;
     }
```

En realidad los arrays no son mas que objetos de la clase array y que por tanto tienen una serie de métodos como es length que nos da la longitud del mismo.

18.10.2 Cadenas (String y StringBuffer)

Una cadena es una secuencia de caracteres. Es importante diferenciar que las cadenas en Java no tienen caracteres finalizados con nulo como sucede en C o C++.

Una característica importante de los objetos String es que son invariables, esto es, no existen métodos que permitan alterar su contenido. Las cadenas String son constantes.

En el lenguaje Java está disponible una clase específica para cadenas no constantes o modificables. Se trata de StringBuffer.

Por tanto, si necesita modificar contenido de un String deberá crear un objeto StringBuffer.

18.10.3 Creación de un String

Muchos Strings se crean a partir de literales de cadenas. Cuando el compilador se encuentra con una secuencia de caracteres entrecomillados, crea un objeto String cuyo valor es el texto que existe entre las comillas.

```
new String ("Soy el valor de un String");

String[] arrayStrings = new String[20];
```

Los elementos de este último tipo de array son tipos de referencia, esto es, cada uno contiene una referencia a un objeto String. Es importante comprender que en esta declaración se ha asignado espacio de almacenamiento en memoria suficiente para almacenar las referencias a los objetos String, no para los propios objetos.

Si llegamos a este punto se intenta acceder a uno de los elementos de arrayString, se obtendrá un error de tipo NullPointerException, ya que el array está vacío y no contiene objetos String. Para asignar los verdaderos Strings:

```
for (int i=0; i < arrayString.Length; i++) {
     arrayString[i] = new String("Cadena num." + i );
     }
```

19 Referencia de características avanzadas en Java[17]

19.1 Programación con hilos de ejecución (Threads)

La mayor parte de los programas de alto nivel desarrollados por programadores suelen ser de tipo secuencial, esto es: tienen un principio, una secuencia de ejecución y un fin claramente definidos.

En Java es posible aplicar un nuevo modelo de programación con unas posibilidades muy superiores a las de la simple programación secuencial con un solo flujo de ejecución: la programación con múltiples hilos de ejecución.

Un hilo de ejecución es muy similar a los programas secuenciales antes descritos: también tienen un principio, un flujo de ejecución y un fin definidos, pero no es una entidad independiente, sino que debe ejecutarse en el contexto de un programa.

Se puede decir que en Java, un programa consta de uno o más hilos de ejecución, que constituyen varios flujos de secuencias independientes.

Por tanto, estos hilos no aportan casi nada nuevo, conceptualmente hablando. Lo que suscita un gran interés en torno a los hilos es la posibilidad de ejecutar varios de ellos de forma simultánea e independientemente dentro de un programa, pudiendo realizar varias actividades en paralelo.

Si ha utilizado un navegador con soporte de Java, habrá podido observar el uso de múltiples hilos de ejecución: dos applets se pueden ejecutar al mismo tiempo, y además es posible desplazar la página del navegador mientras el applet continúa.

[17] Java. El lenguaje de Internet. Sergio Rios Aguilar

19.1.1 Un sencillo programa multihilo

```
import java.lang.Thread;
import java.lang.System;
import java.lang.Math;
import java.lang.InterrupedException;

class EjemploHilos {
     public static void main (String args[]) {
          Hilo hilo1 = new Hilo("Hilo1: ");
          Hilo hilo2 = new Hilo("Hilo2: ");
          hilo1.start();
          hilo2.start();
          boolean Hilo1Vivo = true;
          boolean Hilo2Vivo = true;
          do {
               if (Hilo1Vivo &&!hilo1.isAlive() ) {
                    Hilo1Vivo =false;
                    System.out.println("El hilo1 ha
                    finalizado.");
                    }
               if (Hilo2Vivo &&!hilo2.isAlive() ) {
                    Hilo2Vivo =false;
                    System.out.println("El hilo2 ha
                    finalizado.");
                    }
          } while (Hilo1Vivo || Hilo2Vivo );
     }
}

class Hilo extends Thread {
     static String
     msg[]={"Java","es","el","lenguaje","del","futuro"};
     public Hilo(String id) {super (id); }
     public void run() {
          String nombre=getName();
          for (int i=0; i <msg.length; ++i) {
               esperar();
               System.out.println(nombre+msg[i]);
          }
     }

     void esperar() {
          try {
               sleep((long) (4500*Math.random() ));
               } catch (interruptedException x) {
               System.out.println ("Error capturado");
               }
     }
}
```

Para ejecutar este programa, primero grábelo con un editor de textos, luego:

```
javac EjemploHilos.java

Java EjemploHilos
```

19.1.2 Atributos de los hilos de ejecución

Los hilos de ejecución de Java están implementados en la clase thread, que forma parte del paquete java.lang.

La clase thread implementa el concepto de ejecución multihilo de una forma independiente de la plataforma utilizada realmente para ejecutar los hilos de ejecución del lenguaje Java. Sin embargo, en la última instancia, la implementación concreta de la concurrencia de operaciones dependerá fuertemente del sistema fina usado.

En cualquier caso, el programador deberá ignorar esta circunstancia y programar directamente usando la API que ofrece Java para el tratamiento de hilos de ejecución.

Nos ocuparemos ahora de estudiar concierto detalle los distintos atributos de un hilo de ejecución: cuerpo, estado, prioridad y grupo.

Cuerpo de un hilo de ejecución

La mayor parte de la actividad de procesamiento de un hilo de ejecución se produce en su cuerpo: el método run().

Con mucha frecuencia el método run() es un bucle. Por ejemplo, un hilo de ejecución que implementa una animación itera visualizando una secuencia de imágenes.

Existen dos posibilidades para crear el cuerpo:

- Usar una subclase derivada de thread y redefinir su método run()
- Utilizar una clase que implemente la interfaz runnable, también definida en el paquete java.lang. En este caso cuando se instancia un hilo de ejecución a partir de la clase thread o de una clase derivada de la misma, se suministra a dicho hilo o manejador (handle) de una instancia de la clase runnable, que suministra el método run() al hilo de ejecución.

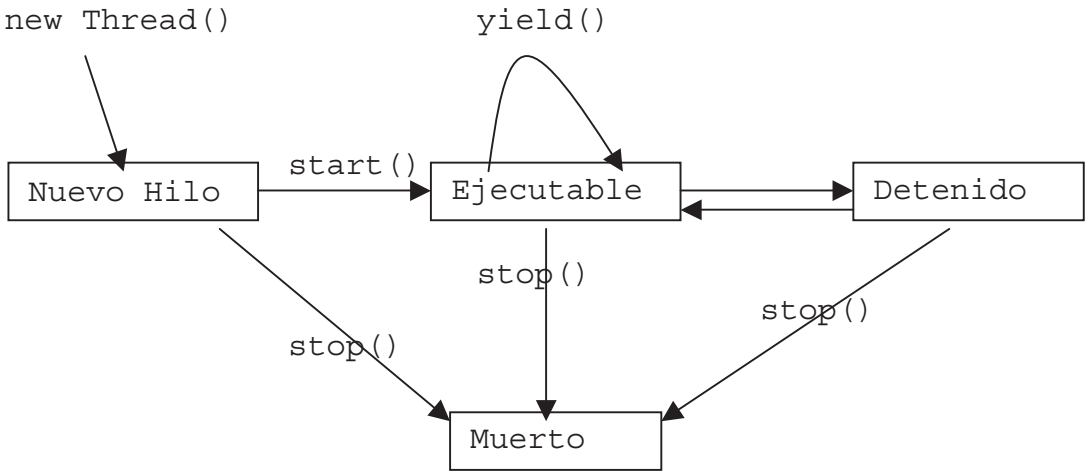
Otro ejemplo sería un reloj en un applet que podemos incluir en nuestras páginas Web.

```
import java.awt.Graphics;
import java.util.Date;

public class Reloj extends java.applet.Applet implements
Runnable {
     Thread hiloReloj = null;
     public void start() {
          if (hiloReloj == null) {
               hiloReloj = new Thread(this, "Reloj");
               hiloReloj.start();
          }
     }

     public void run() {
          while (Thread.currentThread() == hiloReloj) {
               repaint();
               try {
                    hiloReloj.sleep(1000);
                    } catch (InterruptedException e){}
          }
     }

     public void paint (Graphics g) {
          Date hora = new Date();
          g.drawString (hora.getHours() + ":" +
          hora.getMinutes() + ":" + hora.getSeconds(), 5,
          10);
     }

     public void stop() {
          hiloReloj =null;
     }
}
```

Estados de un hilo de ejecución

Durante su ciclo de vida, un hilo de ejecución Java se encuentra en un estado concreto entre los posibles.

Este diagrama no es en absoluto un diagrama de estados completo y riguroso; mas bien se trata de una representación de los hitos más importantes del ciclo de vida de un hilo de ejecución.



Prioridad de un hilo de ejecución

Como ya se ha mencionado, la mayoría de los sistemas personales hoy en día sólo disponen de una CPU, por lo que los hilos de ejecución deben compartir este recurso siguiendo unas determinadas reglas.

El sistema de tiempo de ejecución de Java implementa un sencillo algoritmo de planificación conocido con el nombre de planificación de prioridad fija.

Cada hilo de ejecución tiene asociada una prioridad, expresada mediante un valor numérico entre MIN_PRIORITY y MAX_PRIORITY. Cuanto mayor sea el valor entero, mayor será el nivel de prioridad que indica.

Cuando crea un hilo esta hereda la prioridad del que lo creó. Posteriormente se podrá modificar dicha prioridad usando el método setPriority().

La planificación de la CPU en Java usa mecanismos de requisa. Esto significa que en determinadas circunstancias, un hilo puede arrebatar la CPU a otro hilo actualmente en ejecución.

En Java no está implementado el reparto de tiempo de CPU entre hilos de idéntica prioridad. Esto no quiere decir que la implementación de hilos en la plataforma concreta usada por la máquina virtual Java no admita el uso de “rodajas de tiempo”.

Hilos de ejecución tipo demonio

Se dice que un hilo es de tipo demonio si su único propósito consiste en ofrecer servicios a otros hilos de ejecución existentes dentro del mismo proceso.

Para especificar un hilo tipo demonio se realiza con setDaemon() poniéndolo a true.

Grupos de hilos de ejecución

El grupo de hilos es una constitución que permite manipular y gestionar de forma colectiva un conjunto de hilos, como si de una única entidad se tratase.

De esta forma es perfectamente posible detener un conjunto de hilos con la llamada a suspend().

Java dispone de una clase ThreadGroup que define e implementa las características y posibilidades de un grupo de hilos relacionados entre si.

19.1.3 Sincronización de hilos de ejecución

Todos los hilos de ejecución considerados hasta el momento son de naturaleza asíncrona; esto es, cada hilo es totalmente independiente de los demás.

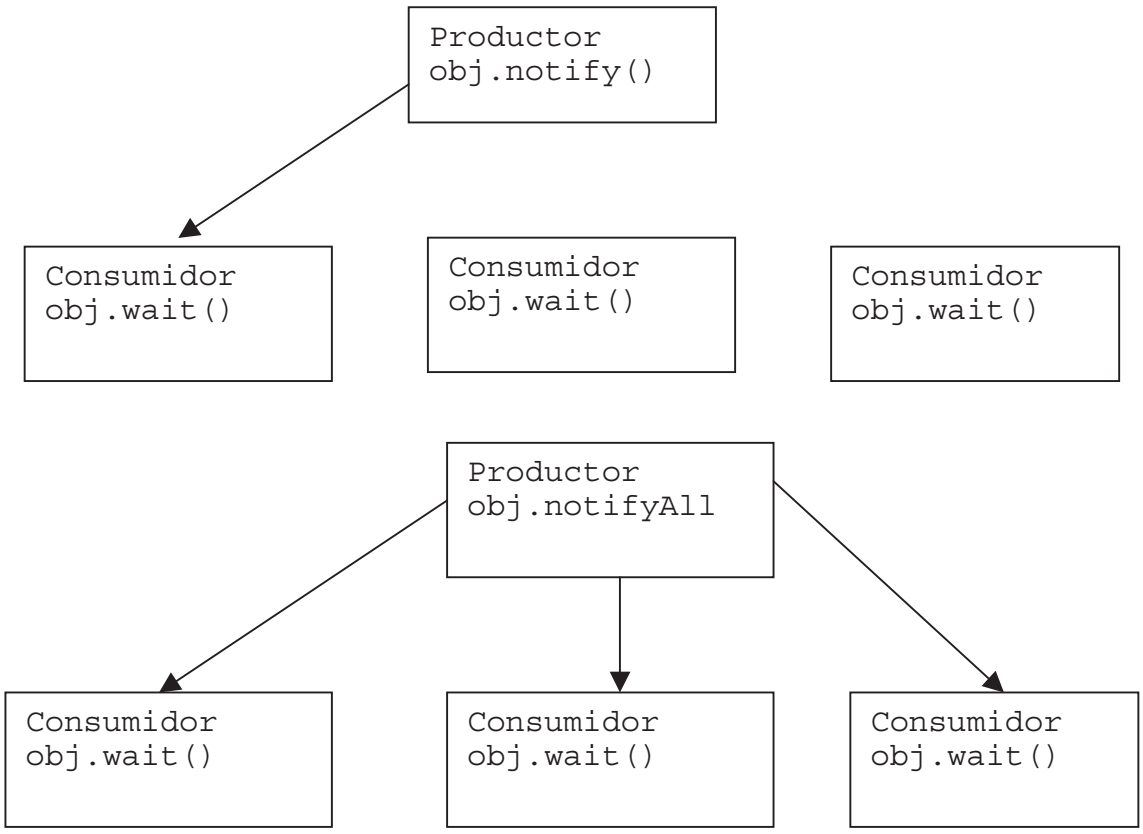
Tal independencia es posible gracias a que los hilos contienen todos lo datos y métodos necesarios para su ejecución y no necesitan acceder a ningún otro recurso externo.

En ocasiones se deberá considerar otro tipo de situaciones, en las que se está ejecutando un conjunto de hilos que comparten datos. En estas circunstancias, es del todo necesario que cada hilo pueda conocer de algún modo el estado y las actividades de los demás.

Tanto el propio lenguaje Java como su sistema interno de tiempo de ejecución permiten la sincronización de diferentes hilos para el acceso a una variable condición usando monitores.

El concepto de monitor fue presentado por primera vez en ámbitos académicos por C.A.R. Hoare en su ya famoso artículo Communicating Sequential Processes, 1978. En términos generales, un monitor está asociado a un elemento de datos concreto y actúa como un cerrojo que controla a dichos datos.

Cuando un hilo de ejecución adquiere un monitor de algún elemento de datos, los demás dejan de tener acceso a dichos datos.



19.2 Tratamiento de excepciones

19.2.1 Excepciones en Java

Las excepciones en Java tienen como propósito último, al igual que en el resto de los lenguajes que las soportan, la detección y corrección de errores.

Una excepción es una señal que indica que se ha producido algún error de situación excepcional, como por ejemplo, un error del código detectado en tiempo de ejecución.

Lanzar (to throw) una excepción es señalar una circunstancia excepcional.

Capturar (to catch) una excepción significa gestionarla, esto es, realizar todas aquellas acciones que permitan reducir o eliminar el impacto del error y que conduzcan a la posibilidad de reanudar normalmente la ejecución del código.

En Java una excepción es un objeto instanciado a partir de alguna clase derivada de java.lang.Throwable. Esta tiene dos subclases: java.lang.Error y java.lang.Exception.

Las excepciones que son subclases de Error generalmente indican problemas de enlazado relativos al sistema de carga dinámica, o bien son señal de que hay problemas en la máquina virtual Java, como puede ser falta de memoria.

En principio, este tipo de excepciones deben considerarse como irrecuperables, por lo que no es necesario molestarse en intentar capturarlas.

Por el contrario las excepciones que son subclases de Exception generalmente indican situaciones de error reversibles, por lo que será muy conveniente tratar de capturarlas.

19.2.2 Gestión de excepciones try, catch y finally.

Las sentencias try, catch y finally constituyen el núcleo de la gestión de excepciones en Java.

try indica el bloque de código cuyas excepciones y situaciones anómalas van a ser gestionadas. Este bloque va seguido de la cláusula catch que permiten capturar y gestionar tipos específicos de excepciones.

Las cláusulas catch pueden ir opcionalmente seguidas de un bloque finally que efectúa labores adicionales de limpieza.

Estructura de una secuencia try/catch/finally.

```
try {
     // Código
}
catch (Excepcion1 exc1) {
     // se gestiona el objeto-excepcion1 exc1
     // de tipo Excepcion1 o de una clase derivada
}
     .
     .
     .

}
finally{
     // Siempre este bloque al salir del try

}
```

Si se produce una excepción en el bloque try y existe un bloque catch que permita gestionarla, se transfiere el control a dicho bloque para hacerlo después al finally.

Si, por el contrario, no existe ningún bloque catch local para gestionar ese tipo concreto de excepción, el control se transfiere directamente a finally. Inmediatamente después, la excepción se propaga hacia ámbitos superiores en busca de un bloque catch que permita tratarla.

Algunos tipos de excepciones predefinidas:

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ClassCastException
- IncompatibleClassChangeException
- InternalException
- NegativeArraySizeException
- NoClassDefFoundException
- NullPointerException
- OutOfMemoryException

20 Applets Java[18]

20.1 Creación de Applets

Hoy día la definición mas extendida de una applet es la ofrecida por Patrick Naughton: " Un applet es una pequeña aplicación accesible en un servidor de Internet, que se transporta por la red, se instala automáticamente y se ejecuta en el sistema final como parte de un documento HTML dentro de un navegador".

Por ejemplo:

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

public class HolaMundo extends Applet {
     public void paint(Graphics g) {
          g.drawString("Hola Mundo!",50,25);
     }
}
```

Como se puede comprobar la clase pública deberá derivarse de la clase Applet.

El ciclo de vida de un applet es el siguiente:

1. Se crea una instancia de la clase que éste controla
2. Se inicializa
3. Inicia su ejecución

Cuando un usuario abandona el documento HTML en el que se encuentra el applet, éste tiene la opción de detener su ejecución. Si el usuario vuelve de nuevo al documento HTML, el applet puede reanudar su ejecución.

Esta misma secuencia de operaciones tiene lugar cuando el usuario minimiza y maximiza la ventana del cliente Web en la que se encuentra el applet.

Cuando el usuario quita el cliente Web, el applet tiene la opción de detener su ejecución y efectuar una últimas tareas de limpieza antes de que este cliente Web termine su ejecución.

[18] Java. El lenguaje de Internet. Sergio Rios Aguilar

Con el fin de poder dar respuesta a posibles eventos, un applet Java puede redefinir cuatro métodos de la clase Applet, tal como se muestra en el siguiente ejemplo:

```
public class miApplet extends Applet {

     public void init() { . . . }
     public void start() { . . . }
     public void stop()  { . . . }
     public void destroy() { . . . }
}
```

A continuación se define el propósito de cada uno de estos métodos.

- init()
  Permite iniciar el applet cada vez que se carga.
- start()
  Permite iniciar la ejecución de un applet, bien cuando se carga, bien cuando el usuario vuelve a visitar la página Web que lo contiene.
- stop()
  Detiene la ejecución del applet, por ejemplo, cuando el usuario abandona la página en la que éste se encuentra o tras quitar el cliente Web.
- destroy()
  Para realizar tareas de limpieza finales, antes de que se descargue.

No es necesario que un applet redefina estos cuatro métodos. De hecho, es habitual encontrar que en aplicaciones muy sencilla, como por ejemplo HolaMundo, no ocurra con ninguno de estos métodos.

Podrá observar que en el anterior ejemplo se refine un método hasta ahora desconocido: se trata de paint().

El método paint() implementa los mecanismos básicos de visualización por pantalla dentro de un cliente Web. Este método se suele usar junto con otro muy relacionado, update(), por motivos de eficiencia.

Los applets heredan sus métodos paint() y update() de la clase Applet, que, a su vez, lo hereda de la case Component del Abstract Window Toolkit (AWT).

Por otra parte, desde la clase Component, los applets heredan un grupo de métodos para la gestión de eventos.

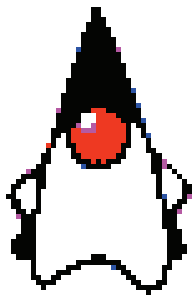
Como habrá podido deducir de lo anterior, su desarrollo implica conocimientos de aspectos muy diversos del lenguaje Java, y muy en especial sobre el AWT, que sin duda, daría de si para otro libro completo.

20.2 Integración de applets en documentos HTML

Para integrar un applet en una página Web se debe usar el código <APPLET> del lenguaje HTML, que tiene la siguiente sintaxis.

```
<APPLET
     [CODEBASE = URL del applet]
     CODE = nombre del archivo de bytecodes del applet
     WIDTH = anchura en pixeles
     HEIGHT = altura en pixeles
     [ ALT = texto alternativo para clientes sin Java]
     [ NAME = nombre de la instancia del applet]
     [ ALIGN = justificación ( top middle botton )]
     [ VSPACE = margen vertical en pixeles]
     [ HSPACE = margen horizontal en pixeles]
>

     [<PARAM NAME= parametro VALUE=valor>]
     [<PARAM NAME= parametro VALUE=valor>]
     [ . . . ]
</APPLET>
```

PROGRAMANDO EN JAVA



21 Programando en Java[19]

22 Pasos iniciales con Java

22.1 Como se hace para imprimir texto

Interesan los métodos print y println. Por ejemplo un programa llamado Convert.java que escribe factores de conversión.

```
/*
 * Convert.class imprime una tabla reducida de factores de conversión
comunes
 */

class Convert {

/*
 * Nótese que el método principal debe ser definido como
 * public static void main (String []) para que sea llamado desde el
intérprete Java
 */

public static void main (String args[]) {

      double mi_to_km = 1.609;
      double gals_to_l = 3.79;
      double kg_to_lbs = 2.2;

      System.out.print ("1 Milla igual a\t");// aquí no va una nueva
línea al final
      System.out.print (mi_to_km);
      System.out.println ("\tKilometros");     // imprimir nueva línea
al final

      System.out.print ("1 Galon igual a\t");
      System.out.print (gals_to_l);
      System.out.print ("\tlitros\n");    // \n también funciona como
nueva línea

      System.out.print ("1 Kilogramo igual a\t");
      System.out.print (kg_to_lbs);
      System.out.println ("\tlibras");
      double dia;
      dia = 1;
      while (dia < 300000) {
            dia = dia + 0.01;
      }
}
}
```

[19] Como se hace con Java. Siddalingaiah y Lockwood

22.2 Como se hace para aceptar una entrada de teclado

Ahora se utilizarán los métodos readLine que toma la entrada desde un DataInputStream y devuelve un String.

Por ejemplo un programa llamado Input.java que lee una línea de texto, determina la longitud y la escribe.

```
import java.io.*;

/*
 * Input.class lee una línea de texto desde la entrada estándar,
* determina la longitud de la línea y la imprime.
 */

class Input {
public static void main (String args[]) {
      String input = "";
      boolean error;

/*
 * DataInputStream contiene el método readLine.
 * Crear una nueva instancia para la entrada estándar System.in
 */
      DataInputStream in = new DataInputStream(System.in);

/*
 * Este bucle se usa para detectar errores de E/S que puedan ocurrir
 */
      do {
            error = false;
            System.out.print ("Introducir la cadena > ");

/*
* se necesita depurar la salida porque no hay nueva línea al final
 */
            System.out.flush ();

            try {
                  input = in.readLine ();       // lee la entrada
            } catch (IOException e) {
                  System.out.println (e);       // imprime error
                  System.out.println ("Se detectó un error de
entrada");
                  error = true;
            }
      } while (error);

      System.out.print ("Se ha introducido \"");
      System.out.print (input);     // readLine no guarda  \n ni \r
      System.out.println ("\"");
      System.out.print ("La longitud es ");
      System.out.println (input.length ());



} // final de main ()
}
```

22.3 Como se hace para convertir cadenas a números

Se verá muy bien con el siguiente ejemplo que utiliza el método ValueOf para calcular la propina.

```
import java.io.*;

/*
 * Tip.class calcula la propina de una cuenta, dado el importe de la
factura
 * y el porcentaje de la propina.
 */

class Tip {
public static void main (String args[]) {
      String input = "";
      int tip_percent=0;
      double bill=0, tip=0;
      boolean error;

      DataInputStream in = new DataInputStream(System.in);

      do {
            error = false;
            System.out.print ("Entrar el total de la cuenta > ");
            System.out.flush ();
            try {
                  input = in.readLine ();
            } catch (IOException e) {
                  System.out.println (e);
                  System.exit (1);
            }

/*
 * Convertir la cadena a double, con detección de
NumberFormatException
 */
            try {
                  bill = Double.valueOf (input).doubleValue();
            } catch (NumberFormatException e) {
                  System.out.println (e);
                  System.out.println ("Inténtelo nuevamente");
                  error = true;
            }
      } while (error);

      do {
            error = false;
            System.out.print ("Entrar el porcentaje de la propina > ");
            System.out.flush ();
            try {
                  input = in.readLine ();
            } catch (IOException e) {
                  System.out.println (e);
                  System.exit (1);
            }

/*
 * Esta vez la conversión es a integer
 */
```

```
            try {
                  tip_percent = Integer.valueOf (input).intValue();
            } catch (NumberFormatException e) {
                  System.out.println (e);
                  System.out.println ("Inténtelo nuevamente ");
                  error = true;
            }
      } while (error);

      System.out.print ("El total es ");
      tip = bill * ((double) tip_percent)/100.0;
      System.out.println (bill + tip);
} // final de main ()
}
```

22.4 Como se hace para convertir números a cadenas

Exactamente igual al anterior utilizando ValueOf podemos hacer el efecto contrario.

```
import java.util.Date;

/*
 * Debt.class calcula la deuda nacional aproximada, la población y lo
que debe cada uno
*/
class Debt {

public static void main (String args[]) {
      Date now = new Date ();              // fecha actual
      Date debtDate = new Date (92, 5, 29);     // última información
de la deuda: 29/6/92
// (codificación de la fecha, véase comentarios)
      Date popDate = new Date (90, 6, 1); // último censo:1/7/90
      long milliPerDay = 24*60*60*1000;   // milisegundos por día

/*
 *las unidades de  now.getTime () son milisegundos, convertir a
segundos para la deuda y
 * a días para la población
 */
      long debtSecs = (now.getTime () - debtDate.getTime ())/1000;
      long popDays = (now.getTime () - popDate.getTime
())/milliPerDay;

/*
 * Por suerte los long son de 64 bits!  El promedio de crecimiento de
la deuda el 29/6/92 era de
 * $14,132/segundo.  La tasa de crecimiento de la población el 1/7/90
era de
 * 6170 personas/día.  Para simplificar el cálculo se usan
aproximaciones de primer orden.
 */
      long debt = 3965170506502L + (debtSecs * 14132);
      long pop = 250410000 + (popDays * 6170);

      print ("La deuda nacional es $", debt);
      print ("La población es de ", pop);
      print ("Cada persona debe $", debt/pop);
```

```
}

/*
 * el método print convierte un long a string y los imprime con puntos
separadores de los miles.
 *    String str  etiqueta precedente
 *    long n            long para el  formateo e impresión
 */
public static void print (String str,long n) {
      System.out.print (str);            // impresión de la etiqueta
      String buf = String.valueOf (n);         // Integer a String

      int start, end, ncommas, i;
      int buflen = buf.length ();

/*
 * Es un algoritmo alocado, pero funciona. Funciona de izquierda a
derecha.
 */
      ncommas = buflen/3;
      if (ncommas * 3 == buflen) ncommas -= 1;
      start = 0;
      end = buflen-(3*ncommas);
      System.out.print (buf.substring (start, end));
      for (i=0; i<ncommas; i+=1) {
            start = end;
            end = start+3;
            System.out.print (".");             // punto separador
            System.out.print (buf.substring (start, end));
      }
      System.out.println ("");                  // la nueva línea final
}
}
```

22.5 Como se hace para acceder a la linea de mandatos

Es decir poder ver los argumentos pasados al programa y manipularlos a nuestro gusto.

Simplemente utilizaremos el parametro de la función main (args).

Hay que fijarse en una cosa muy importante y es que no se hace igual que en C o C++.

```
import java.util.Date;

/* Calendar.class imprime un calendario dados como argumentos el año y
el mes como
 */
class Calender {

public static void main (String args[]) {

/*
 * Java, a diferencia de C/C++, no necesita cuenta de argumento (argc)
 */
      if (args.length < 2) {
```

```
            System.out.println ("uso: Calendario <mes> <año>");
            System.exit (1);
      }

/*
 * A diferencia de C/C++ args[0] es el PRIMER argumento. El nombre
 * de la aplicación no está disponible como argumento
 */
      int mes = Integer.valueOf (args[0]).intValue();
      int anio = Integer.valueOf (args[1]).intValue();
      if (mes < 1 || mes > 12) {
            System.out.println ("El mes debe ser entre 1 y 12");
            System.exit (1);
      }
      if (anio < 1970) {
            System.out.println ("El año debe ser mayor a 1969");
            System.exit (1);
      }

/*
 * Esta versión de Date necesita año, mes y día.
 * Nótese que la base de mes es cero (0-11), pero el día está basado
en 1 (1-31)
 */
      Date date = new Date (anio-1900, mes-1, 1);
      System.out.println ("Dom\tLun\tMar\tMie\tJue\tVie\tSab");
      int i, day, ndays=0, lastday=28;

      switch (mes) {
      case 1:  // Enero
      case 3:  // Marzo
      case 5:  // Mayo
      case 7:  // Julio
      case 8:  // Agosto
      case 10: // Octubre
      case 12: // Diciembre
            lastday = 31;
            break;

/*
 * Tratamiento especial del año bisiesto, es más complicado que el
resto de
 * dividir por 4
 */
      case 2:  // febrero
            lastday = (anio%4 == 0 && anio%100 != 0) ||
                  (anio%400 == 0) ? 29 : 28;
            break;

      case 4:  // Abril
      case 6:  // Junio
      case 9:  // Septiembre
      case 11: // Noviembre
            lastday = 30;
            break;
      }

/*
 * Primero se imprimen espacios (tabs) para alinear los días de la
semana
 */
```

```
      for (i=0; i<date.getDay(); i+=1) {
            System.out.print ("\t");
            ndays += 1;
            if (ndays % 7 == 0) System.out.println ("");
      }

/*
 * Ahora se imprimen los días
 */
      for (day=1; day<=lastday; day+=1) {
            System.out.print (day);
            System.out.print ("\t");
            ndays += 1;
            if (ndays % 7 == 0) System.out.println ("");
      }
      System.out.println ("");
}
}
```

22.6 Como se hace para leer desde un archivo

Los archivos se abren mediante la creación de un FileInputStream con el nombre de archivo en cada argumento. También es muy importante utilizar siempre con archivos el tratamiento de excepciones tal y como se ve en el ejemplo:

```
import java.io.*;

/*
 * More.class similar al programa de utilidad More de UNIX
 */
class More {

public static void main (String args[])
{
      String buf;
      FileInputStream fs=null;
      int nlines;

      if (args.length != 1) {
            System.out.println ("uso: java More <archivo>");
            System.exit (1);
      }

/*
 * Intenta abrir el nombre de archivo especificado por args[0]
 */
      try {
            fs = new FileInputStream (args[0]);
      } catch (Exception e) {
            System.out.println (e);
            System.exit (1);
      }

/*
 * Crea un DataInputStream asociado con FileInputStream fs
 */
      DataInputStream ds = new DataInputStream (fs);
```

```
      DataInputStream keyboard = new DataInputStream (System.in);
      nlines = 0;
      while (true) {
            try {
                  buf = ds.readLine ();         // lee 1 línea
                  if (buf == null) break;
            } catch (IOException e) {
                  System.out.println (e); // no hay nuevas líneas en el
búfer
                  break;
            }
            System.out.println (buf);
            nlines += 1;
            if (nlines % 23 == 0) {       // 23 líneas por página vt100
                  System.out.print ("--More--");
                  System.out.flush ();
                  try {
                        keyboard.readLine ();
                  } catch (IOException e) {
                  }
            }
      }
/*
 * close también puede provocar una condición de error
 */
      try {
            fs.close ();
      } catch (IOException e) {
            System.out.println (e);
      }
}
}
```

22.7 Como se hace para grabar en un archivo

Como en el ejemplo anterior, para leer un archivo se debe abrir un FileInputStream y un DataInputStream. Para grabar un archivo, se usa FileOutputStream y PrintStream.

```
import java.io.*;

/*
 * Format.class traduce un archivo de texto al formato DOS, Mac o
UNIX.
 * La diferencia está en la línea de terminación.
 */
class Format {

static final int TO_DOS = 1;
static final int TO_MAC = 2;
static final int TO_UNIX = 3;
static final int TO_UNKNOWN = 0;

static void usage () {
      System.out.print ("uso: java Format -dmu <archivo-entrada> ");
      System.out.println ("<archivo-salida>");
```

```
      System.out.println ("\t-d convierte <archivo-entrada> a DOS");
      System.out.println ("\t-m convierte <archivo-entrada> a MAC");
      System.out.println ("\t-u convierte <archivo-entrada> a UNIX");
}

public static void main (String args[])
{
      int format=TO_UNKNOWN;
      String buf;
      FileInputStream fsIn = null;
      FileOutputStream fsOut = null;

      if (args.length != 3) {       // se deben especificar formatos
            usage ();
            System.exit (1);
      }

/*
 * args[0] es un String, por lo que para las comparaciones se puede
usar el método
 * equals  (de la clase String)
 */
      if (args[0].equals ("-d")) format = TO_DOS; else
      if (args[0].equals ("-m")) format = TO_MAC; else
      if (args[0].equals ("-u")) format = TO_UNIX; else {
            usage ();
            System.exit (1);
      }

      try {
            fsIn = new FileInputStream (args[1]);
      } catch (Exception e) {
            System.out.println (e);
            System.exit (1);
      }

/*
 * FileOutputStream es el complemento de FileInputStream
 */
      try {
            fsOut = new FileOutputStream (args[2]);
      } catch (Exception e) {
            System.out.println (e);
            System.exit (1);
      }
      DataInputStream dsIn = new DataInputStream (fsIn);
      PrintStream psOut = new PrintStream (fsOut);
      while (true) {
            try {
                  buf = dsIn.readLine ();
                  if (buf == null) break;       // termina con el EOF
            } catch (IOException e) {
                  System.out.println (e);
                  break;
            }
            psOut.print (buf);
            switch (format) {
                  case TO_DOS:
                  psOut.print ("\r\n");
                  break;
```

```
                case TO_MAC:
                psOut.print ("\r");
                break;

                case TO_UNIX:
                psOut.print ("\n");
                break;
            }
      }

/*
 *  No es absolutamente detectar los errores individualmente, pero se
 *  impide un aviso del compilador
 */
      try {
            fsIn.close ();
            fsOut.close ();
      } catch (IOException e) {
            System.out.println (e);
      }
}
}
```

22.8 Como se hace para utilizar matrices

En este ejemplo repasaremos todo lo visto anteriormente y además utilizaremos los arrays.

```
import java.io.*;

/*
 * Sort.class lee un archivo de texto especificado en el argumento
args[0] y
 * realiza la clasificación previamente a la impresión
 */
class Sort {

static final int NMaxLines = 128;   // un límite arbitrario

public static void main (String args[]) {
/*
 * reserva una nueva matriz de cadenas. Aquí es donde se lee el
archivo
 * y se clasifica (en el mismo lugar)
 */
      String sortArray[] = new String [NMaxLines];
      FileInputStream fs=null;
      int nlines;

      if (args.length != 1) {
            System.out.println ("uso: java Sort <archivo>");
            System.exit (1);
      }
      try {
            fs = new FileInputStream (args[0]);
      } catch (Exception e) {
            System.out.println ("No se puede abrir "+args[0]);
            System.exit (1);
```

```
      }
      DataInputStream ds = new DataInputStream (fs);
      DataInputStream keyboard = new DataInputStream (System.in);
      for (nlines=0; nlines<NMaxLines; nlines += 1) {
            try {
                  sortArray[nlines] = ds.readLine ();
                  if (sortArray[nlines] == null) break;
            } catch (IOException e) {
                  System.out.println ("Error detectado en la lectura");
                  break;
            }
      }
      try {
            fs.close ();
      } catch (IOException e) {
            System.out.println ("Error detectado en el cierre.");
      }

/*
 * clasificación sobre el mismo sitio e impresión
 */
      QSort qsort = new QSort ();
      qsort.sort (sortArray, nlines);
      print (sortArray, nlines);
}

/*
 * el método print imprime una matriz de Strings de n elementos
 *    String a[]  matriz de Strings que se quiere imprimir
 *    int n       número de elementos
 */
private static void print (String a[], int n) {
      int i;

      for (i=0; i<n; i+=1) System.out.println (a[i]);
      System.out.println ("");
}
}

/*
 * QSort.class usa el algoritmo quicksort
 *
 */
class QSort {

/*
 * Esto se usa internamente, por lo que se hace  private
 */
private void sort (String a[], int lo0, int hi0) {
      int lo = lo0;
      int hi = hi0;

      if (lo >= hi) return;
      String mid = a[(lo + hi) / 2];
      while (lo < hi) {
            while (lo<hi && a[lo].compareTo (mid) < 0) lo += 1;
            while (lo<hi && a[hi].compareTo (mid) > 0) hi -= 1;
            if (lo < hi) {
                  String T = a[lo];
                  a[lo] = a[hi];
                  a[hi] = T;
```

```
            }
      }
      if (hi < lo) {
            int T = hi;
            hi = lo;
            lo = T;
      }
      sort(a, lo0, lo); // sí, es recursivo
      sort(a, lo == lo0 ? lo+1 : lo, hi0);
}

/*
 * método llamado para iniciar la clasificación
 *    String a[]  una matriz de String para clasificar en el mismo
sitio
 *    int n       número de elementos de la matriz
 */
public void sort (String a[], int n) {
      sort (a, 0, n-1);
}
}
```

22.9 Como se hace para analizar una cadena

Utilizaremos para ello la clase StringTokenizer del java.util que toma una cadena y devuelve los substrings que están separados por caracteres arbitrarios.

```
import java.io.*;
import java.util.StringTokenizer;

/*
 * Phone.class implementa una agenda telefónica muy simple con
búsqueda por
 * nombres.  El archivo de la agenda se puede crear con un editor de
texto
 * o con una hoja de cálculo utilizando texto separado con
tabuladores.
 */
class Phone {

public static void main (String args[])
{
      String buf;
      FileInputStream fs=null;

      if (args.length != 1) {
            System.out.println ("uso: java Phone <nombre-buscado>");
            System.exit (1);
      }

/*
 * PhoneBook.txt es el nombre del archivo-agenda.
 */
      try {
            fs = new FileInputStream ("PhoneBook.txt");
      } catch (Exception e) {
```

```
            System.out.println ("No es posible abrir PhoneBook.txt");
            System.exit (1);
      }
      DataInputStream ds = new DataInputStream (fs);
      DataInputStream keyboard = new DataInputStream (System.in);
      while (true) {
            try {
                  buf = ds.readLine ();
                  if (buf == null) break;
            } catch (IOException e) {
                  System.out.println ("Error en lectura de archivo.");
                  break;
            }

/*
 * Crea un StringTokenizer nuevo para cada línea leída.
 * Especificar explícitamente los delimitadores.
 */
            StringTokenizer st = new StringTokenizer (buf, ":\t");

            String name = st.nextToken ();
            if (contains (name, args[0])) {
                  System.out.println (name);
                  System.out.println (st.nextToken ());
                  System.out.println (st.nextToken () + "\n");
            }
      }
      try {
            fs.close ();
      } catch (IOException e) {
            System.out.println ("Error detectado en cierre de
archivo.");
      }
}

/*
 * el método contains realiza una comparación de cadenas que devuelve
true
 * si la cadena está contenida en alguna de las cadenas del archivo
 *    String s1, s2     compara dos cadenas
 */
static boolean contains (String s1, String s2) {

      int i;
      int l1 = s1.length ();
      int l2 = s2.length ();

      if (l1 < l2) {
            for (i=0; i<=l2-l1; i+=1)
                  if (s1.regionMatches (true, 0, s2, i, l1))
                        return true;
      }
      for (i=0; i<=l1-l2; i+=1)
            if (s2.regionMatches (true, 0, s1, i, l2))
                  return true;

      return false;
}
}
```

22.10 Como se hace para utilizar funciones matemáticas

Es muy simple, los métodos que debemos utilizar son los de la clase Math.

```
import java.io.*;

/*
 * Ammortize.class calcula el pago mensual de un préstamo, dada la
 * cantidad prestada, el interés y la cantidad de años del préstamo.
 */
class Ammortize {
public static void main (String args[]) {
      double loanAmount=0, interest=0, years=0;

      DataInputStream in = new DataInputStream(System.in);

      loanAmount = inputDouble ("Introduzca la cantidad del préstamo
en pesetas > ", in);

      interest = inputDouble ("Introduzca el porcentaje de interés >
", in);

      years = inputDouble ("Introduzca la cantidad de años > ", in);

      System.out.print ("El pago mensual es de ptas.");
      System.out.println (payment (loanAmount, interest, years));
} // final de main ()

/*
 * el método inputDouble method imprime un aviso y lee un
 * DataInputStream
 */
static double inputDouble (String prompt, DataInputStream in) {
      boolean error;
      String input="";
      double value=0;

      do {
            error = false;
            System.out.print (prompt);
            System.out.flush ();
            try {
                  input = in.readLine ();
            } catch (IOException e) {
                  System.out.println ("Se detectó un error de
entrada");
                  System.exit (1);
            }
            try {
                  value = Double.valueOf (input).doubleValue();
            } catch (NumberFormatException e) {
                  System.out.println ("Por favor, vuélvalo a
intentar");
                  error = true;
            }
      } while (error);
      return value;
} // final de inputDouble ()
```

```
/*
 * el método payment realiza el cálculo
 *    double A    cantidad
 *    double I    interés
 *    double Y    cantidd de años
 */
static double payment (double A, double I, double Y) {

/*
 * llamada a las funciones de exponenciación y logaritmos como métodos
estáticos
 * en la clase Math
 */
      double top = A * I / 1200;
      double bot = 1 - Math.exp (Y*(-12) * Math.log (1 + I/1200));

      return top / bot;
} //final de payment ()
}
```

23 Gráficos básicos

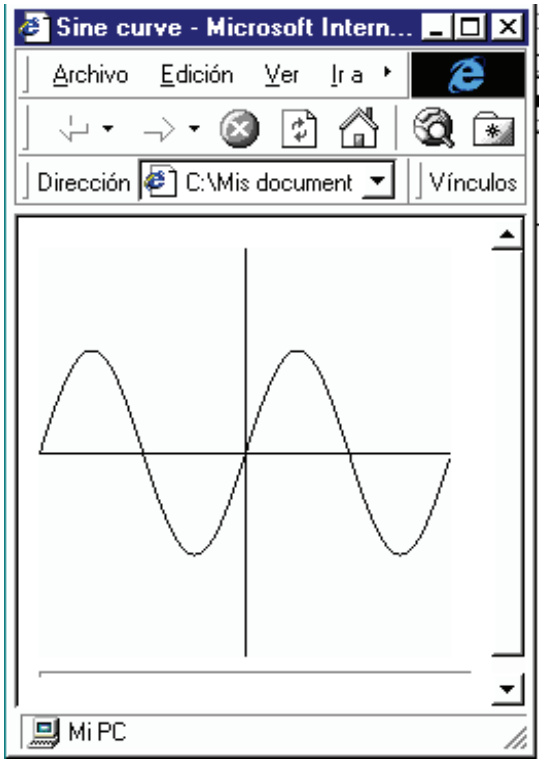
Este apartado lo veremos muy rápidamente ya que con gráficos se pueden hacer muchísimas cosas, y lo importante no es que métodos se utilizan sino como se utilizan.

Utilizaremos en todo momento los métodos del Java AWT, así se enumerarán una serie de ejemplos para ir poco a poco en gráficos, muy bien comentados y que no necesitan otro tipo de documentación adicional.

23.1 Dibujar una linea

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

/**
 * aplicación/applet de la curva seno
 * dibuja un ciclo de la curva seno
 */
public class Sine extends Applet {

/*
 * ancho y altura del panel del applet
 */
int width, height;

/*
 * init(): es llamado cuando se carga el applet
 * simplemente guárdese el ancho y la altura
 */
public void init () {

      Dimension d = size ();

      width = d.width;
      height = d.height;
}

/**
 * paint(): realiza el dibujo de los ejes y la curva seno
 * @param g - objeto gráfico de destino
 */
public void paint (Graphics g) {

      int x, x0;
      double y0, y, A, f, t, offset;

      A = (double) height / 4;
      f = 2;
      offset = (double) height / 2;
      x0 = 0;
      y0 = offset;

      g.drawLine (x0, (int) y0, width, (int) y0);
      g.drawLine (width/2, 0, width/2, height);
```

```
      for (x=0; x<width; x+=1) {
            t = (double) x / ((double) width);
            y = offset - A * Math.sin (2 * Math.PI * f * t);
            g.drawLine (x0, (int) y0, x, (int) y);
            x0 = x;
            y0 = y;
      }
}

/**
 * main(): punto de entrada de la aplicación
 * main() no se usa cuando se ejecuta como applet
 * crea un marco de ventana y añade adentro al applet
 * @param args[] - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      Frame f = new Frame ("Curva seno");
      Sine sine = new Sine ();

      f.resize (200, 200);
      f.add ("Center", sine);
      f.show ();
      sine.init ();
}
}
```



23.2 Añadir color

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

/**
 * clase LineColors contiene valores de 24 colores
 */
class LineColors {

/**
 * matriz color[] contiene los colores que se utilizan
 */
Color color[];

/**
 * constructor de la clase
 * inicializa la matriz de colores utilizando un algoritmo arbitrario
 */
public LineColors () {

      color = new Color[24];
      int i, rgb;

      rgb = 0xff;
      for (i=0; i<24; i+=1) {
            color[i] = new Color (rgb);
            rgb <<= 1;
            if ((rgb & 0x1000000) != 0) {
                  rgb |= 1;
                  rgb &= 0xffffff;
            }
      }
}
}

/**
 * clase segment describe una línea
 */
class Segment {

/*
 * x1, y1 - coordenadas de comienzo del segmento
 * x2, y2 -coordenadas de final del segmento
 * dx1,...dy2 - velocidades
 * whichcolor - índice actual en la matriz de colores
 * width, height - ancho y altura del panel limitante
 * LC - instancia de la clase LineColors
 */
double x1, y1, x2, y2;
double dx1, dy1, dx2, dy2;
int whichcolor, width, height;
LineColors LC;

/**
 * constructor de la clase
 * inicializa puntos finales y velocidades con valores al azar
 * @param w - ancho del panel limitante
 * @param h - altura del panel limitante
 * @param c - índice del color de inicio
 * @param lc - instancia de la clase LineColors
```

```
 */
public Segment (int w, int h, int c, LineColors lc) {

      whichcolor = c;
      width = w;
      height = h;
      LC = lc;
      x1 = (double) w * Math.random ();
      y1 = (double) h * Math.random ();
      x2 = (double) w * Math.random ();
      y2 = (double) h * Math.random ();

      dx1 = 5 - 10 * Math.random ();
      dy1 = 5 - 10 * Math.random ();
      dx2 = 5 - 10 * Math.random ();
      dy2 = 5 - 10 * Math.random ();
}

/*
 * incrementa el índice del color
 * calcula la siguiente posición final del segmento
 */
void compute () {

      whichcolor += 1;
      whichcolor %= 24;

      x1 += dx1;
      y1 += dy1;
      x2 += dx2;
      y2 += dy2;

      if (x1 < 0 || x1 > width) dx1 = -dx1;
      if (y1 < 0 || y1 > height) dy1 = -dy1;
      if (x2 < 0 || x2 > width) dx2 = -dx2;
      if (y2 < 0 || y2 > height) dy2 = -dy2;
}

/**
 * dibuja el segmento con el color actual
 * @param g - objeto de destino
 */
void paint (Graphics g) {

      g.setColor (LC.color [whichcolor]);
      g.drawLine ((int) x1, (int) y1, (int) x2, (int) y2);
}
}

/**
 * applet/aplicación
 */
public class Lines extends Applet {

/*
 * width, height - ancho y altura del panel limitante
 * Nlines - número de segmentos de línea que se visualizarán
 * lines - matriz de instancias de la clase Segment
 * LC - instancia de la clase LineColors
 */
int width, height;
```

```
final int NLines = 4;
Segment lines[] = new Segment[NLines];
LineColors LC = new LineColors ();

/**
 * cuando se carga el applet, se llama a  init
 * guarda el ancho y la altura
 * crea instancias de la clase Segment
 */
public void init () {

      Dimension d = size ();

      width = d.width;
      height = d.height;

      int i;
      for (i=0; i<NLines; i+=1)
            lines[i] = new Segment (width, height, (2*i) % 24, LC);
}

/**
 * recalcula el siguiente punto final para cada línea
 * llama al método paint() para cada línea
 * llama a repaint() para forzar el repintado 50ms más tarde
 * @param g - objeto de destino
 */
public void paint (Graphics g) {

      int i;
      for (i=0; i<NLines; i+=1) {
            lines[i].compute ();
            lines[i].paint (g);
      }
      repaint (50);
}

/**
 * punto de entrada de la aplicación. No se usa cuando se ejecuta como
applet
 * crea un marco ventana y añade un applet
 * @param args[] - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      Frame f = new Frame ("Líneas coloreadas");
      Lines lines = new Lines ();

      f.resize (200, 200);
      f.add ("Center", lines);
      f.show ();
      lines.init ();
}
}
```

23.3 Dibujar formas

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

```
import java.io.*;
import java.util.StringTokenizer;

/**
 * clase que describe una forma
 */
class Shape {

/**
 * constantes para el tipo de forma
 */
static final int rectType = 1;
static final int ovalType = 2;
static final int arcType = 3;
static final int polyType = 4;

/*
 * tipo de forma
 */
int type;

/*
 * color de la forma elegida
 */
Color color;
static final int MaxPoints = 10;

/*
 * matrices de puntos x and y points para esta forma
 */
int xp[] = new int[MaxPoints];
int yp[] = new int[MaxPoints];

/*
 * cantidad de puntos de esta forma
 */
int npoints;

/**
 * constructor de shape
 * parámetros
 * @param tp - tipo de forma
 * @param n - número de puntos
 * @param pts[] - matriz de puntos finales
 * @param c - color de la forma
 */
public Shape (int tp, int n, int pts[], Color c) {

      int i;
      type = tp;
      color = c;
      npoints = n < MaxPoints ? n : MaxPoints;
      if (type == polyType) {
            npoints >>= 1;
            for (i=0; i<npoints; i+=1) {
                  xp[i] = pts[i << 1];
                  yp[i] = pts[(i << 1) +1];
            }
      } else {
            for (i=0; i<npoints; i+=1)
                  xp[i] = pts[i];
```

```
      }
}

/**
 * dibujo de la forma
 * @param g - objeto de destino
 */
void paint (Graphics g) {

      g.setColor (color);
      switch (type) {

      case rectType:
            g.drawRect (xp[0], xp[1], xp[2], xp[3]);
            break;

      case ovalType:
            g.drawOval (xp[0], xp[1], xp[2], xp[3]);
            break;

      case arcType:
            g.drawArc (xp[0], xp[1], xp[2], xp[3], xp[4], xp[5]);
            break;

      case polyType:
            g.drawPolygon (xp, yp, npoints);
            break;
      }
}
}

/**
 * clase de la aplicación
 */
public class DrawApp extends Panel {

/*
 * máxima cantidad de puntos de la forma
 */
static final int MaxShapes = 25;

/*
 * nshapes - número de formas leídas
 * nlines -  número de línea en el archivo de entrada
 */
static int nshapes, nlines = 0;

/*
 * matriz de instancias de la clase shape
 */
static Shape shapes[] = new Shape[MaxShapes];

/**
 * llamada al método paint() method para cada forma
 * @param g -objeto de destino
 */
public void paint (Graphics g) {

      int i;
      for (i=0; i<nshapes; i+=1)
            shapes[i].paint (g);
```

```
}

/**
 * punto de entrada de la aplicación
 * @param args - argumentos en la línea de mandatos
 */
public static void main (String args[]) {

      String buf;
      FileInputStream fs=null;
      int i, type = 0;

      if (args.length != 1) {
            System.out.println ("uso: java DrawApp <archivo>");
            System.exit (1);
      }

/*
 * Intenta abrir el archivo especificado en el argumento argsŠ0‹
 */
      try {
            fs = new FileInputStream (args[0]);
      } catch (Exception e) {
            System.out.println (e);
            System.exit (1);
      }

/*
 * Crea un DataInputStream asociado con el FileInputStream fs
 */
      DataInputStream ds = new DataInputStream (fs);
      String token;
      Color color = Color.white;
      int pts[] = new int[2 * Shape.MaxPoints];

/*
 * bucle hasta el final del archivo o hasta detectar un error
 * lee una línea y la analiza
 */
      while (true) {
            try {
                  buf = ds.readLine ();   // lee una línea
                  if (buf == null) break;
            } catch (IOException e) {
                  System.out.println (e); // no hay línea en el búfer
                  break;
            }
            nlines += 1;
            StringTokenizer st = new StringTokenizer (buf);
            token = st.nextToken ();
            if (token.equals ("white")) {
                  color = Color.white;
                  token = st.nextToken ();
            } else if (token.equals ("lightgray")) {
                  color = Color.white;
                  token = st.nextToken ();
            } else if (token.equals ("gray")) {
                  color = Color.gray;
                  token = st.nextToken ();
            } else if (token.equals ("darkgray")) {
                  color = Color.darkGray;
```

```
                token = st.nextToken ();
            } else if (token.equals ("black")) {
                color = Color.black;
                token = st.nextToken ();
            } else if (token.equals ("red")) {
                color = Color.red;
                token = st.nextToken ();
            } else if (token.equals ("pink")) {
                color = Color.pink;
                token = st.nextToken ();
            } else if (token.equals ("orange")) {
                color = Color.orange;
                token = st.nextToken ();
            } else if (token.equals ("yellow")) {
                color = Color.yellow;
                token = st.nextToken ();
            } else if (token.equals ("green")) {
                color = Color.green;
                token = st.nextToken ();
            } else if (token.equals ("magenta")) {
                color = Color.magenta;
                token = st.nextToken ();
            } else if (token.equals ("cyan")) {
                color = Color.cyan;
                token = st.nextToken ();
            } else if (token.equals ("blue")) {
                color = Color.blue;
                token = st.nextToken ();
            } else {
                System.out.println ("Color desconocido: "+token);
                System.out.println ("líneas "+nlines);
                System.exit (1);
            }

            int npoints = 0;
            if (token.equals ("rect")) {
                npoints = getInt (st, pts, 4);
                type = Shape.rectType;
            } else if (token.equals ("oval")) {
                npoints = getInt (st, pts, 4);
                type = Shape.ovalType;
            } else if (token.equals ("arc")) {
                npoints = getInt (st, pts, 6);
                type = Shape.arcType;
            } else if (token.equals ("poly")) {
                npoints = getInt (st, pts, Shape.MaxPoints);
                type = Shape.polyType;
            } else {
                System.out.println ("Forma desconocida: "+token);
                System.out.println ("líneas "+nlines);
                System.exit (1);
            }
            shapes[nshapes++] = new Shape (type, npoints, pts, color);
        }
/*
 * close también puede provocar alguna condición de error
 */
        try {
            fs.close ();
        } catch (IOException e) {
            System.out.println (e);
```

```
      }

      Frame f = new Frame ("Dibujo de formas");
      DrawApp drawApp = new DrawApp ();

      f.resize (410, 430);
      f.add ("Center", drawApp);
      f.show ();
}

/**
 * puntos de análisis
 * @param st - StringTokenizer para la línea actual
 * @param pts[] - matriz de puntos que se devuelve
 * @param nmax - número máximo de puntos que se aceptan
 */
static int getInt (StringTokenizer st, int pts[], int nmax) {

      int i;
      String token;

      for (i=0; i<nmax; i+=1) {
            if (st.hasMoreTokens () == false) break;
            token = st.nextToken ();
            try {
                  pts[i] = Integer.valueOf (token).intValue ();
            } catch (NumberFormatException e) {
                  System.out.println (e);
                  System.out.println ("línea "+nlines);
                  System.exit (1);
            }
      }
      return i;
}
}
```

23.4 Rellenar formas

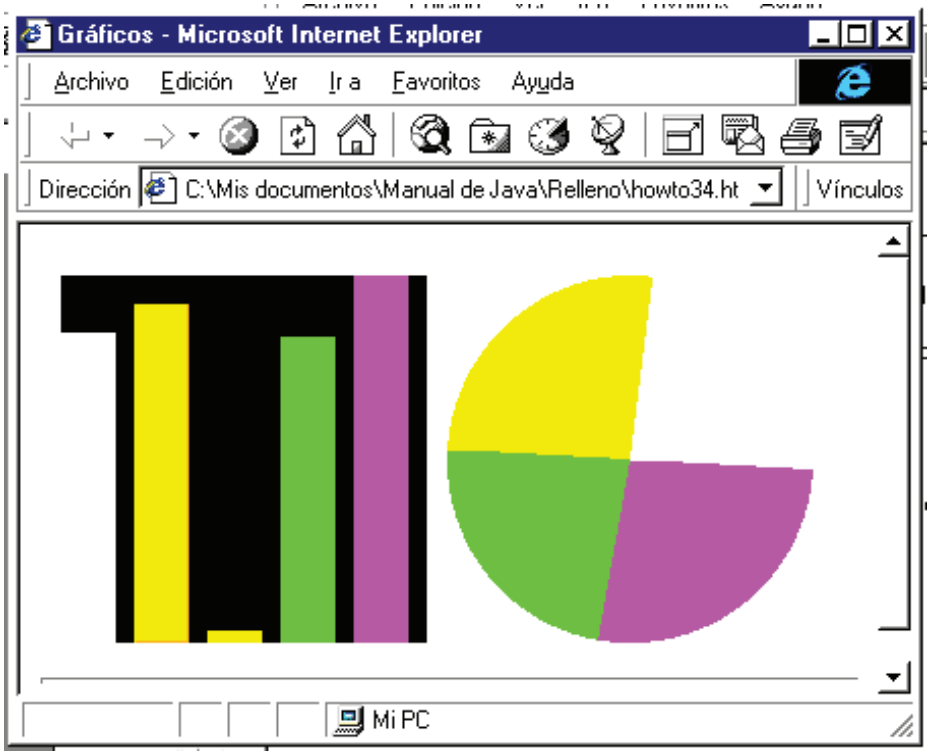
```
import java.applet.Applet;
import java.awt.*;

/**
 * clase padre
 */
class Chart {

/*
 * posiciones x e y positions de la parte superior izquierda del
gráfico
 * nvalues - número de valores de este gráfico
 */
int xpos, ypos, nvalues;

/*
 * ancho y altura de este gráfico
 */
int width, height;
```

```
/*
 * máximo número de valores permitidos
 */
final int MaxValues = 10;

/*
 * valores de los datos de este gráfico
 */
double values[] = new double[MaxValues];

/*
 * color asociado con cada valor
 */
Color colors[] = new Color[MaxValues];

/*
 * suma total de valores, usado para fines de cambio de escala
 */
double total;

/**
 * constructor de la clase
 * guarda los valores y los normaliza, de manera que el valor máximo
es 1.0
 * @param x, y - coordenada  superior izquierda
 * @param w, h - ancho y altura
 * @param n - número de puntos
 * @param val[] - matriz de valores
 * @param c[] - matriz de colores que corresponden a los valores
 */
public Chart (int x, int y, int w, int h, int n, double val[], Color
c[]) {

      int i;
      double extreme;

      xpos = x;
      ypos = y;
      width = w;
      height = h;
      nvalues = n;
      if (nvalues > MaxValues) nvalues = MaxValues;
      extreme = 0.0;
      for (i=0; i<nvalues; i+=1) {
            if (Math.abs (val[i]) > extreme)
                  extreme = Math.abs (val[i]);
            colors[i] = c[i];
      }
      extreme = 1/extreme;
      total = 0;
      for (i=0; i<nvalues; i+=1) {
            values[i] = extreme * val[i];
            total += values[i];
      }
}
}

/**
 * implementa la clase del gráfico de barras
 */
class BarChart extends Chart {
```

```
/**
 * el constructor simplemente llama al constructor Chart
 * @param x, y - corrdenada superior izquierad
 * @param w, h - ancho y altura
 * @param n - número de puntos
 * @param val[] - matriz de valores
 * @param c[] - matriz de colores relacionadas con los puntos
 */
public BarChart (int x, int y, int w, int h, int n, double val[],
Color c[]) {

      super (x, y, w, h, n, val, c);
}

/**
 * se necesita agregar un método para pintar
 * dibuja el gráfico de barras usando fill3DRect
 * @param g - objeto de destino
 */
void paint (Graphics g) {

      int i;
      int barwidth = 3 * width / (4 * nvalues);
      int bardx = width / nvalues;
      int x, y, h;

      g.setColor (Color.black);
      g.fillRect (xpos, ypos-height, width, height);
      for (i=0; i<nvalues; i+=1) {
            g.setColor (colors[i]);
            x = xpos + bardx*i;
            h = (int) (values[i] * height);
            y = ypos - h;
            g.fill3DRect (x, y, barwidth, h, true);
      }
}
}

/**
 * implementa la clase del gráfico de pastel
 */
class PieChart extends Chart {

/**
 * el constructor simplemente llama al constructor Chart
 * @param x, y - corrdenada superior izquierad
 * @param w, h - ancho y altura
 * @param n - número de puntos
 * @param val[] - matriz de valores
 * @param c[] - matriz de colores relacionadas con los puntos
 */
public PieChart (int x, int y, int w, int h, int n, double val[],
Color c[]) {

      super (x, y, w, h, n, val, c);
}

/**
 * se necesita agregar un método para pintar
 * dibuja el gráfico de barras usando fillArc
```

```
 * @param g - objeto de destino

 */
void paint (Graphics g) {

      int i, y;
      int startAngle, arcAngle;

      startAngle = 0;
      y = ypos - height;
      for (i=0; i<nvalues; i+=1) {
            arcAngle = (int) (360.0 * values[i] / total);
            g.setColor (colors[i]);
            g.fillArc (xpos, y, width, height, startAngle, arcAngle);
            startAngle += arcAngle;
      }
}
}

/**
 * applet/aplicación
 */
public class ChartApp extends Applet {

/*
 * ancho y altura del panel limitante
 */
int width, height;

/*
 * instancias de BarChart y PieChart
 */
BarChart bc1;
PieChart pc1;

/*
 * se llama cuando carga al applet
 * genera valores al azar y los traza
 */
public void init () {

      int i;
      Dimension d = size ();
      double values[] = new double[5];
      Color colors[] = new Color[5];

      width = d.width;
      height = d.height;
      colors[0] = Color.white;
      colors[1] = Color.orange;
      colors[2] = Color.yellow;
      colors[3] = Color.green;
      colors[4] = Color.magenta;

      for (i=0; i<5; i+=1) values[i] = Math.random () + 0.001;
      int w = (width-40)/2;
      int h = height-20;
      bc1 = new BarChart (10, height-10, w, h, 5, values, colors);
      pc1 = new PieChart (width/2, height-10, w, h, 5, values,
colors);
}
```

```
/**
 * llamada a los métodos paint
 * @param g - objeto de destino
 */
public void paint (Graphics g) {

      bc1.paint (g);
      pc1.paint (g);
}

/**
 * punto de entrada de la aplicación
 * crea  un marco de ventana y le coloca dentro un applet
 * @param args[] - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      Frame f = new Frame ("Gráficos");
      ChartApp chart = new ChartApp ();

      f.resize (410, 230);
      f.add ("Center", chart);
      f.show ();
      chart.init ();
      chart.start ();
}
}
```



23.5 Dibujar texto

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

/**
 * una clase que manipula texto desplazable
 */
class Scroll {

/*
 * coordenadas x e y del punto de inicio
 */
int xstart, ystart;

/*
 * ancho y altura del panel limitante
 */
int width, height;

/*
 * texto
 */
String text;

/*
 * velocidades de x  e  y
 */
int deltaX, deltaY;

/*
 * posición actual de l texto en coordenadas x e y
 */
int xpos, ypos;

/*
 * color del texto
 */
Color color;

/**
 * el constructor de la clase simplemente guarda los argumentos
 * @param x, y -coordenadas de inicio
 * @param dx, dy - velocidades x e y
 * @param w, h - ancho y altura del panel limitante
 * @param t - cadena de texto
 * @param c - color de texto
 */
public Scroll (int x, int y, int dx, int dy, int w, int h, String t,
Color c) {

      xstart = x;
      ystart = y;
      width = w;
      height = h;
      text = t;
      deltaX = dx;
      deltaY = dy;
      color = c;
      xpos = xstart;
      ypos = ystart;
```

```
}

/*
 * dibujar el texto en la posición actual
 * avanzar la posición y reinicializar el panel limitante
 * @param g - objeto de destino
 */
void paint (Graphics g) {

      g.setColor (color);
      g.drawString (text, xpos, ypos);
      xpos += deltaX;
      ypos += deltaY;

      FontMetrics fm = g.getFontMetrics ();
      int textw = fm.stringWidth (text);
      int texth = fm.getHeight ();
      if (deltaX < 0 && xpos < -textw) xpos = xstart;
      if (deltaX > 0 && xpos > width) xpos = xstart;
      if (deltaY < 0 && ypos < 0) ypos = ystart;
      if (deltaY > 0 && ypos > height+texth) ypos = ystart;
}
}

/**
 * applet/aplicación
 */
public class ScrollApp extends Applet {

/*
 * ancho y altura del panel limitante
 */
int width, height;

/*
 * instancias de Scroll
 */
Scroll left, right, up, down, diag;

/*
 * se llama al cargarse el applet
 * crea  nuevas instancias de Scroll
 */
public void init () {

      Dimension d = size ();

      width = d.width;
      height = d.height;

      left = new Scroll (400, 50, -5, 0, width, height,
            "Movimiento hacia la izquierda", Color.red);
      right = new Scroll (0, 150, 5, 0, width, height,
            "Movimiento hacia la derecha", Color.green);
      up = new Scroll (100, 200, 0, -5, width, height,
            "Movimiento hacia arriba", Color.blue);
      down = new Scroll (200, 0, 0, 5, width, height,
            "Movimiento hacia abajo", Color.cyan);
      diag = new Scroll (0, 0, 7, 3, width, height,
            "Movimiento en diagonal", Color.magenta);
}
```

```
/*
 * llamada al método paint method de cada instancia de texto
desplazable
 * fuerza un repinta 50ms más tarde
 */
public void paint (Graphics g) {

      left.paint (g);
      right.paint (g);
      up.paint (g);
      down.paint (g);
      diag.paint (g);
      repaint (50);
}

/*
 * punto de entrada de la aplicación
 * @param args - argumentos de línea de mandatos
 */
public static void main (String args[]) {

      Frame f = new Frame ("Texto desplazable");
      ScrollApp scrollApp = new ScrollApp ();

      f.resize (410, 230);
      f.add ("Center", scrollApp);
      f.show ();
      scrollApp.init ();
}
}
```

23.6 Utilizar fuentes

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

/**
 * clase que determina qué fuentes hay disponibles
 */
public class Fonts extends Applet {

/*
 * máximo número de fuentes que se pueden visualizar
 */
final int MaxFonts = 10;

/*
 * ancho y altura del panel limitante
 */
int width, height;

/*
 * matriz de nombres de fuentes
 */
String fontName[];
```

```
/*
 * matriz de fuentes
 * contiene  los estilos normal, negrita y cursiva de cada una
 */
Font theFonts[] = new Font[3 * MaxFonts];

/*
 * cantidad de fuentes encontrada
 */
int nfonts = 0;

/*
 * punto de entrada del applet
 */
public void init () {

      int i;
      Dimension d = size ();

      width = d.width;
      height = d.height;
      fontName = Toolkit.getDefaultToolkit().getFontList ();

      nfonts = fontName.length;
      if (nfonts > MaxFonts) nfonts = MaxFonts;
      for (i=0; i<nfonts; i+=1) {
            theFonts[3*i + 0] = new Font (fontName[i], Font.PLAIN, 12);
            theFonts[3*i + 1] = new Font (fontName[i], Font.BOLD, 12);
            theFonts[3*i + 2] = new Font (fontName[i], Font.ITALIC,
12);
      }
}

/*
 * dibujo del nombre de las fuentes
 * @param g objeto de destino
 */
public void paint (Graphics g) {

      int i;

      for (i=0; i<nfonts; i+=1) {
            g.setFont (theFonts[3*i + 0]);
            g.drawString (fontName[i], 10, 20*i+30);
            g.setFont (theFonts[3*i + 1]);
            g.drawString ("Bold", 70, 20*i+30);
            g.setFont (theFonts[3*i + 2]);
            g.drawString ("Italic", 150, 20*i+30);
      }
}

/*
 * punto de entrada de la aplicación
 * crea un marco de ventana y le coloca el applet
 * @param args[] - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      Frame f = new Frame ("Fuentes");
      Fonts fonts = new Fonts ();
```

```
      f.resize (200, 200);
      f.add ("Center", fonts);
      f.show ();
      fonts.init ();
}
}
```

23.7 Actualizaciones de pantalla

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

/*
 * clase aplicación/applet
*/
public class CrazyText extends Applet {

String text = "Java";  // texto que se visualizará
int delta = 5;      // grado de locura: desplazamiento máximo de
píxeles
String fontName = "TimesRoman";
int fontSize = 36;

char chars[];        // caracteres individuales en 'text'
int positions[];    // posición horizontal base para cada carácter
FontMetrics fm;

/*
 * Es llamado en el momento de la carga del applet
 * crea una fuente e inicializa las posiciones de caracteres
 */
public void init() {

      int fontStyle = Font.BOLD + Font.ITALIC;
      setFont(new Font(fontName, fontStyle, fontSize));
      fm = getFontMetrics(getFont());

      chars = new char[text.length()];
      text.getChars(0, text.length(), chars, 0);

      positions = new int[text.length()];
      for (int i = 0; i < text.length(); i++) {
            positions[i] = fm.charsWidth(chars, 0, i) + 20;
      }
}

/*
 * dibuja los caracteres y obliga el repinta 100ms más tarde
 * @param g - objeto gráfico de destino
 */
public void paint (Graphics g) {

      int x, y;
      g.setColor (new Color((float) Math.random(),
                  (float) Math.random(),
                  (float) Math.random()));
      for (int i = 0; i < text.length(); i++) {
            x = (int)(Math.random() * delta * 2) + positions[i];
```

```
            y = (int)(Math.random() * delta * 2) + fm.getAscent() - 1;
            g.drawChars (chars, i, 1, x, y);
      }
      repaint (100);
}

/*
 * modifica el método update() para eliminar el boorado del panel
*/
public void update (Graphics g) {
      paint (g);
}

/*
 * punto de entrada de la aplicación
 * crea un marco de ventana y le coloca el applet
 * @param args[] - argumentos de la línea de mandato
 */
public static void main (String args[]) {

      Frame f = new Frame ("Crazy");
      CrazyText crazy = new CrazyText ();

      f.resize (130, 80);
      f.add ("Center", crazy);
      f.show ();
      crazy.init ();
}
}
```

24 Tramas

En lo que se refiere a las tramas o hilos de ejecución podrían dedicarse libros enteros, sin embargo nosotros nos limitaremos a dar algún ejemplo de su utilización real para ver su funcionamiento sin pararnos demasiado en la sincronización o en las prioridades etc.

Como ya hemos visto, para crear una trama una clase debe implementar la interfaz ejecutable (Runnable) o ampliar la clase Thread, la cual ya implementa por si misma la interfaz ejecutable. En este ejemplo se implementa la interfaz runnable.

Se debe creare una instancia de la clase Thread con la clase que contiene la trama como argumento. Si la clase que contiene la trama crea la instancia de la trama, se puede usar la autoreferencia.

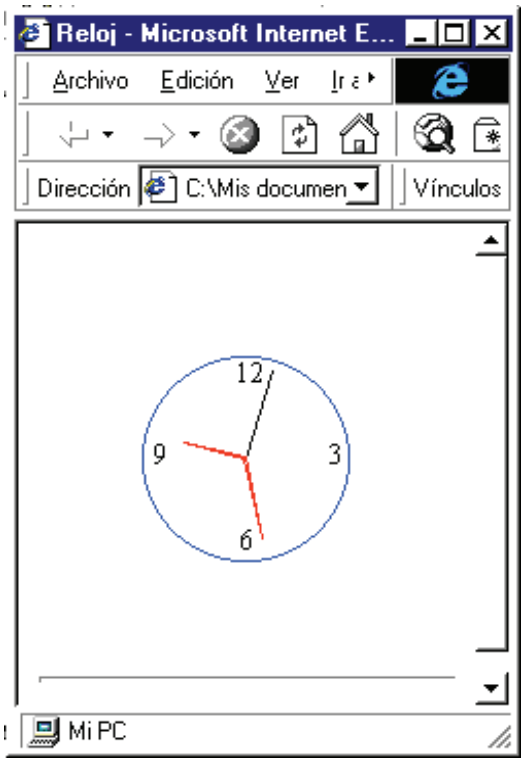
Las clases que contienen una trama separada deben también crear un método run() sin argumentos y que devuelva void. Cuando arranque la trama se llamará al método run(). Una trama se arranca con la llamada al método Thread.stop() o mediante el retorno del método run().

```
import java.awt.*;
import java.applet.*;
import java.util.Date;

/*
 * clase para el applet/aplicación
 */
public class Clock extends Applet implements Runnable {

/*
 * instancia de Thread para la verificación periódica del tiempo
 */
Thread thread = null;

/*
 * valores guardados usados para dibujar, sólo cuando hubo cambios
 */
int lastxs=0;
int lastys=0;
int lastxm=0;
int lastym=0;
int lastxh=0;
int lastyh=0;

/**
  * establece el gris claro como color de fondo
  */
public void init(){
        this.setBackground(Color.lightGray);
}

/**
```

```
 * dibujo de la cara del reloj
 * @param g - objeto de destino
 */
public void paint (Graphics g) {

      int xh, yh, xm, ym, xs, ys, s, m, h, xcenter, ycenter;
      Date dat = new Date();
      Dimension dim = size();


      s = dat.getSeconds();
      m = dat.getMinutes();
      h = dat.getHours();

      xcenter=dim.width >> 1;
      ycenter=dim.height >> 1;

      xs = (int)(Math.cos(s * 3.14f/30 - 3.14f/2) * 45 + xcenter);
      ys = (int)(Math.sin(s * 3.14f/30 - 3.14f/2) * 45 + ycenter);
      xm = (int)(Math.cos(m * 3.14f/30 - 3.14f/2) * 40 + xcenter);
      ym = (int)(Math.sin(m * 3.14f/30 - 3.14f/2) * 40 + ycenter);
      xh = (int)(Math.cos((h*30 + m/2) * 3.14f/180 - 3.14f/2) * 30
            + xcenter);
      yh = (int)(Math.sin((h*30 + m/2) * 3.14f/180 - 3.14f/2) * 30
            + ycenter);

// dibujo del círculo y de los números
      g.setFont(new Font("TimesRoman", Font.PLAIN, 14));
      g.setColor(Color.blue);
      g.drawOval (xcenter-50, ycenter-50, 100, 100);
      g.setColor(Color.darkGray);
      g.drawString("9",xcenter-45,ycenter+3);
      g.drawString("3",xcenter+40,ycenter+3);
      g.drawString("12",xcenter-5,ycenter-37);
      g.drawString("6",xcenter-3,ycenter+45);

// si es necesario, boora y redibuja
      g.setColor(Color.lightGray);
      if (xs != lastxs || ys != lastys) {
            g.drawLine(xcenter, ycenter, lastxs, lastys);
      }
      if (xm != lastxm || ym != lastym) {
            g.drawLine(xcenter, ycenter-1, lastxm, lastym);
            g.drawLine(xcenter-1, ycenter, lastxm, lastym);
      }
      if (xh != lastxh || yh != lastyh) {
            g.drawLine(xcenter, ycenter-1, lastxh, lastyh);
            g.drawLine(xcenter-1, ycenter, lastxh, lastyh);
      }

      g.setColor(Color.darkGray);
      g.drawLine(xcenter, ycenter, xs, ys);
      g.setColor(Color.red);
      g.drawLine(xcenter, ycenter-1, xm, ym);
      g.drawLine(xcenter-1, ycenter, xm, ym);
      g.drawLine(xcenter, ycenter-1, xh, yh);
      g.drawLine(xcenter-1, ycenter, xh, yh);
      lastxs=xs; lastys=ys;
      lastxm=xm; lastym=ym;
      lastxh=xh; lastyh=yh;
}
```

```
/*
 * método llamado cuando se carga el applet
 * crea una nueva instancia de Thread y comienza
 */
public void start() {

      if(thread == null) {
            thread = new Thread(this);
            thread.start();
      }
}

/*
 * método llamado cuando el applet se detiene
 * detiene la trama
 */
public void stop() {

      thread = null;
}

/*
 * trama
 * duerme 100ms y fuerza el repintado
 */
public void run() {

      while (thread != null) {
            try {
                  Thread.sleep(100);
            } catch (InterruptedException e) { }
            repaint();
      }
      thread = null;
}

/**
 * reemplaza el método update() para evitar el parpadeo causado por el
borrado
 * innecesario del panel del applet
 * @param g - objeto de destino
 */
public void update(Graphics g) {
      paint(g);
}

/**
 * punto de entrada de la aplicación
 * no se usa cuando se ejecuta como applet
 * crea un nuevo marco de ventana y coloca el applet
 * @param args[] - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      Frame f = new Frame ("Reloj analógico");
      Clock clock = new Clock ();

      f.resize (210, 230);
      f.add ("Center", clock);
      f.show ();
```

```
        clock.init ();
        clock.start ();
}
}
```



25 Eventos y parámetros

Es manejo de los eventos es una parte importantisima de la programación. Los veremos en dos grupos muy diferenciados y con dos ejemplos muy curiosos.

- Eventos del teclado (Tetris)
- Eventos del ratón (Paint)

25.1 Para acceder a los eventos de teclado

En Java los eventos del teclado se manipulan mediante el reemplazo del método KeyDown(), que se activa cada vez que ocurre un evento de teclado. Este lo recibe, lo comprueba y realiza una serie de acciones.

El ejemplo del tetris no solo nos servirá para ver los eventos del teclado sino para repasar todos los aspectos de gráficos vistos anteriormente.

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

/**
 *
 * La clase Square manipula un cuadrado individual que forma parte de
una pieza que cae.
 *
 */

class Square {

/**
 * La columna ocupada por el cuadrado
 */
int column;

/**
 * La fila que ocupa el cuadrado
 */
int row;

/**
 * El color del cuadrado
 */
int color;

/**
 * Construcción de un cuadrado
 *
 * @param column  La columna ocupada por el cuadrado
 * @param row           La fila que ocupa el cuadrado
 * @param color   El color del cuadrado
 */
Square (int column, int row, int color) {
```

```
      this.column = column;
      this.row = row;
      this.color = color;
}

/**
 * Verificación si el cuadrado está dentro de los límtes del tablero
 *
 * @return        True si está dentro
 */
boolean InBounds () {

      return (column >= 0 && column < Tetris.cols &&
            row >= 0 && row < Tetris.rows+4);
}

/**
 * Verificación de la equivalencia entre dos cuadrados (this y s).
 *
 * @param s Square que se compara contra la instancia this
 * @return  True si ambos cuadrados son equivalentes.
 */
boolean IsEqual (Square s) {
      return column == s.column && row == s.row && color == s.color;
}
}

/**
 * La clase Tetris implementa el applet.
 *
 * @see java.applet.Applet
 * @see java.lang.Runnable
 */
public class Tetris extends Applet implements Runnable {

static            int   sqlength;   // longitud fel cuadrado
static final      int   xoffset = 200;    // distancia desde la
izquierda
static            int   cols;       // columnas
static            int   rows;       // filas

/**
 * field es una matriz de dos dimensiones que contiene
 * los valores del índice de color de los cuadrados. Un valor cero
indica
 * que ningún cuadrado ocupa esa posición.
 */
int         field[][];

/**
 * oldField se usa para determinar qué piezas han cambiado
 * sólo se dibujan las posiciones que han cambiado
 */
int         oldField[][];

/**
 * curPiece es una matriz de cuadrados que describe la pieza que está
en juego
 */
Square            curPiece[] = new Square[4];
```

```
/**
 * la variable gameInPlay está en True mientras sigue el juego
*/
boolean           gameInPlay;

/**
 * needNewPiece está en True si se debe añadir una nueva pieza
 * al tablero de juego
 */
boolean           needNewPiece;

/**
 * theThread es una trama separada que hace que las piezas caigan
 * normalmente en el tablero de juego
 */
Thread            theThread = null;

/**
 * color es una matriz que contiene los valores de los colores usados
en el juego

 */
Color       colors[];
int         pieceValue, theScore=0;
int         playLevel;
int         totalPieces;
boolean     justupdating = false;

/**
 * mueve una matriz de cuadrados desde una posición a otra, si
 * esa posición está dentro del tablero y no está ocupada por otro
cuadrado.
 *
 * @param from[]  Matriz de Squares fuente
 * @param to[]          Matriz de Squares destino
 */
boolean moveSquares (Square from[], Square to[]) {

      outerlabel:
      for (int i=0; i<to.length; i++) {
            if (to[i].InBounds () == false) return false;
            if (field[to[i].column][to[i].row] != 0) {
                  for (int j=0; j<from.length; j++)
                        if (to[i].IsEqual (from[j]))
                              continue outerlabel;
                  return false;
            }
      }

/*
 * Si se ha llegado hasta aquí, significa que los cuadrados se pueden
mover
 * por lo que primero son borrados del campo mediante la asignación
 * del color blanco en cada posición de los cuadrados
 */
      for (int i=0; i<from.length; i++)
            if (from[i].InBounds ())
                  field[from[i].column][from[i].row] = 0;

/*
```

```
 * Los cuadrados son visibles en la nueva posición estableciendo los
colores
 * de esas posiciones a color no blanco.
 */
      for (int i=0; i<to.length; i++)
            field[to[i].column][to[i].row] = to[i].color;

      return true;
}

/**
 * Crea una nueva pieza al azar y la coloca en la parte superior del
área
 * de juego. Hay siete tipos de piezas, todas de diferentes tamaños y
forma.
 * Siempre se diseñan con cuatro cuadrados.
*/
void newPiece () {

/*
 * la matriz old se usa para determinar si se sigue el juego
 */
      Square  old[] = new Square[4];
      old[0] = old[1] = old[2] = old[3] = new Square (-1, -1, 0);

      int middle = cols/2;
      int top = rows;

      switch ((int) (Math.random ()*7)) {
          case 0:
            // XXXX           rojo
            pieceValue = 100;
            curPiece[0] = new Square (middle-1, top-1, 1);
            curPiece[1] = new Square (middle-2, top-1, 1);
            curPiece[2] = new Square (middle,   top-1, 1);
            curPiece[3] = new Square (middle+1, top-1, 1);
            break;

          case 1:
            //  X       naranja
            // XXX
            pieceValue = 120;
            curPiece[0] = new Square (middle  , top-2, 5);
            curPiece[1] = new Square (middle  , top-1, 5);
            curPiece[2] = new Square (middle-1, top-2, 5);
            curPiece[3] = new Square (middle+1, top-2, 5);
            break;

          case 2:
            // XX       verde
            //  XX
            pieceValue = 180;
            curPiece[0] = new Square (middle  , top-2, 2);
            curPiece[1] = new Square (middle-1, top-1, 2);
            curPiece[2] = new Square (middle  , top-1, 2);
            curPiece[3] = new Square (middle+1, top-2, 2);
            break;

          case 3:
            //  XX            azul
            // XX
```

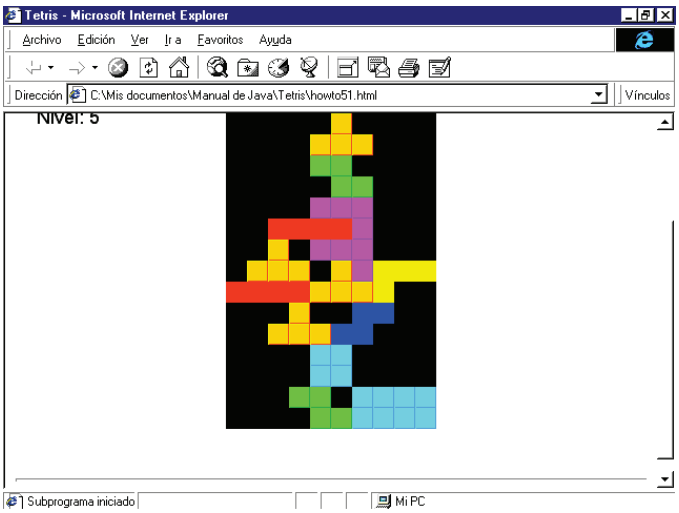
```
            pieceValue = 180;
            curPiece[0] = new Square (middle  , top-2, 7);
            curPiece[1] = new Square (middle+1, top-1, 7);
            curPiece[2] = new Square (middle  , top-1, 7);
            curPiece[3] = new Square (middle-1, top-2, 7);
            break;

          case 4:
            //  XX            azul claro
            //  XX
            pieceValue = 100;
            curPiece[0] = new Square (middle-1, top-1, 3);
            curPiece[1] = new Square (middle  , top-1, 3);
            curPiece[2] = new Square (middle-1, top-2, 3);
            curPiece[3] = new Square (middle  , top-2, 3);
            break;

          case 5:
            //  XXX           violeta
            //    X
            pieceValue = 120;
            curPiece[0] = new Square (middle  , top-1, 6);
            curPiece[1] = new Square (middle-1, top-1, 6);
            curPiece[2] = new Square (middle+1, top-1, 6);
            curPiece[3] = new Square (middle+1, top-2, 6);
            break;

          case 6:
            //  XXX           amarillo
            //  X
            pieceValue = 120;
            curPiece[0] = new Square (middle  , top-1, 4);
            curPiece[1] = new Square (middle+1, top-1, 4);
            curPiece[2] = new Square (middle-1, top-1, 4);
            curPiece[3] = new Square (middle-1, top-2, 4);
            break;
      }
      gameInPlay = moveSquares (old, curPiece);
}

/**
 * Mover la pieza actual por byx columnas y byy filas
 * La pieza se rota 90 grados si rotate está puesto en True
 * returna True si la pieza se movió  y False  en caso contrario.
 */
private synchronized boolean movecurPiece (int byx, int byy, boolean
rotate) {

      Square newpos[] = new Square[4];

      for (int i=0; i<4; i++) {
            if (rotate) {
                  int dx = curPiece[i].column - curPiece[0].column;
                  int dy = curPiece[i].row - curPiece[0].row;

                  newpos[i] = new Square (curPiece[0].column - dy,
                              curPiece[0].row + dx,
                              curPiece[i].color);
            } else {
                  newpos[i] = new Square (curPiece[i].column + byx,
                              curPiece[i].row + byy,
```

```
                              curPiece[i].color);
                }
        }
        if (moveSquares (curPiece, newpos) == false) return false;

        curPiece = newpos;
        return true;
}

/**
 * busca filas completamente llenas con cuadrados y las borra
 */
void removelines () {

        outerlabel:
        for (int j=0; j<rows; j++) {
                for (int i=0; i<cols; i++)
                        if (field[i][j] == 0)
                                continue outerlabel;

/*
 * la fila j está totalmente llena con cuadrados,
 * por lo que se mueven todos una fila más abajo.
 */
                for (int k=j; k<rows-1; k++)
                        for (int i=0; i<cols; i++)
                                field[i][k] = field[i][k+1];

// vuelve a verificar la fila j
                j -= 1;
        }
}

/**
 * punto de entrada del Applet
 */
public void init () {

        sqlength = 20;
        cols = 10;
        rows = 20;

        field = new int[cols][rows+4];
        oldField = new int[cols][rows+4];

        resize (sqlength*cols+xoffset*2, sqlength* (rows+3));

/*
 * reserva de los colores usados en el juego
 * el color cero es especial; significa que no hay cuadrado en esa
posición.
 */
        colors = new Color[8];
        colors[0] = new Color (40,40,40);       // color cero
        colors[1] = new Color (255,0,0);        // rojo, por defecto
        colors[2] = new Color (0,200,0);        // verde
        colors[3] = new Color (0,200,255);      // azul claro
        colors[4] = new Color (255,255,0);      // amarillo
        colors[5] = new Color (255,150,0);      // naranja
        colors[6] = new Color (210,0,240);      // violeta
        colors[7] = new Color (40,0,240);       // azul oscuro
```

```
}

/**
 * arranque del applet
 */
public void start () {

      for (int i=0; i<cols; i++) {
            for (int j=0; j<rows+4; j++) {
                  field[i][j] = 0;
                  oldField[i][j] = -1;
            }
      }

      playLevel = 5;
      theScore = 0;
      totalPieces= 0;
      needNewPiece = true;
      gameInPlay = true;

      theThread = new Thread (this);
        theThread.start ();
      requestFocus ();
}

/**
 * detiene el applet
 */
public synchronized void stop () {

      if (theThread != null)
          theThread.stop ();
      theThread = null;
}

/**
 * valores de demora en milisegundos. Indicación del nivel del juego:
 * niveles altos => demora corta => juego veloz
 */
static int delay_map[] = {
      600, 600, 600, 600, 500, 400, 300, 250, 200, 150, 100
};

/**
 * bucle principal del juego
 */
public void run () {

      while (gameInPlay) {
            try {
                  int t;
                  if (playLevel > 10) t = 75;
                  else t = delay_map[playLevel];
                  Thread.sleep (t);
            } catch (InterruptedException e) {}
            if (needNewPiece) {
                  if (pieceValue > 0) {
                        theScore += pieceValue;
                        totalPieces += 1;
                        playLevel = 5 + totalPieces/30;
                  }
```

```
                removelines ();
                newPiece ();
                needNewPiece = false;
            } else {
                needNewPiece = !movecurPiece (0, -1, false);
                if (!needNewPiece) pieceValue -= 5;
            }
            repaint ();
      }
      theThread = null;
}

/**
 * se pulsó una tecla
 */
public boolean keyDown (Event evt, int key)  {

      if (key != 's' && !gameInPlay)
          return true;

      switch (key) {
          case 's':
            stop ();    // detener juego anterior
            start ();   // comienza nuevo juego
            break;

          case 'h':           // izquierda
          case Event.LEFT:
            pieceValue -= 5;
            movecurPiece (-1, 0, false);
            needNewPiece = false;
            repaint ();
            break;

          case 'k':           // derecha
          case Event.RIGHT:
            pieceValue -= 5;
            movecurPiece (1, 0, false);
            needNewPiece = false;
            repaint ();
            break;

          case 'j':           // rotación
          case Event.UP:
            pieceValue -= 5;
            movecurPiece (0, 0, true);
            repaint ();
            break;

          case ' ':           // soltar
          case Event.DOWN:
            while (movecurPiece (0, -1, false));
            repaint ();
            break;
      }
        return true;
}

/**
 * se debe reemplazar este método, en caso contrario, repaint ()
 * borrará toda la pantalla, lo que provoca una gran parpadeo
```

```
 */
public void update (Graphics g)  {

      justupdating = true;
      // se le dice a paint() que dibuje sólo lo que haya cambiado
      paint (g);
}

/**
 * pinta gráficos en la pantalla
 */
public synchronized void paint (Graphics g) {

      g.setFont (new Font ("Helvetica", 0, 18));
      int  gx = sqlength;
      int  gy = sqlength*rows/4;

      g.clearRect (gx, gy-25, xoffset-gx, 25);
      g.drawString ("Resultado: " + theScore, gx, gy);

      gy += 30;
      g.clearRect (gx, gy-25, xoffset-gx, 25);
      g.drawString ("Nivel: " + playLevel, gx, gy);

      for (int i=0; i<cols; i++)
          for (int j=0; j<rows; j++) {
            if (!justupdating ||
                oldField[i][rows-1-j] == -1 ||
                oldField[i][rows-1-j] != field[i][rows-1-j]) {
                  g.setColor (colors[field[i][rows-1-j]]);
                  g.fill3DRect (
                        xoffset+sqlength*i,
                        sqlength+sqlength*j,
                        sqlength, sqlength, true);
            }
            oldField[i][rows-1-j] = field[i][rows-1-j];
          }
      justupdating = false;
}
} // final del applet Tetris
```



25.2 Eventos del ratón

Una manera de acceder a los eventos del ratón usando AWT es mediante el reemplazo del método mouseDown(). Este método se define en la clase Component, por lo que cada subclase de Component hereda el método mouseDown(). Cuando el usuario pulse el botón del ratón, se producirá la llamada a mouseDown() con las coordenadas de la posición del ratón como argumentos. Existen otros métodos para manipular el arrastre del ratón y la liberación del botón del mismo. Estos son respectivamente mouseDrag() y mouseUp().

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

/**
 * La clase ColorBar visualiza una barra de colores para selección de
color
 */
class ColorBar {

/*
 * coordenada superior e izquierda de la barra de color
 */
int xpos, ypos;

/*
 * ancho y altura de la barra de color
 */
int width, height;

/*
 * índice de selección de color actual en la matriz de color
 */
int selectedColor = 3;

/*
 * matriz de colres disponibles
 */
static Color colors[] = {
      Color.white, Color.gray, Color.red, Color.pink,
      Color.orange, Color.yellow, Color.green, Color.magenta,
      Color.cyan, Color.blue
};

/**
 * Crea la barra de color
 */
public ColorBar (int x, int y, int w, int h) {

      xpos = x;
      ypos = y;
      width = w;
      height = h;
}

/**
```

```
 * pinta la barra de color
 * @param g - objeto gráfico de destino
 */
void paint (Graphics g) {

      int x, y;   // posición de cada casilla de color
      int w, h;   // tamaño de cada casilla de color

      for (int i=0; i<colors.length; i+=1) {
            w = width;
            h = height/colors.length;
            x = xpos;
            y = ypos + (i * h);
            g.setColor (Color.black);
            g.fillRect (x, y, w, h);
            if (i == selectedColor) {
                  x += 5;
                  y += 5;
                  w -= 10;
                  h -= 10;
            } else {
                  x += 1;
                  y += 1;
                  w -= 2;
                  h -= 2;
            }
            g.setColor (colors[i]);
            g.fillRect (x, y, w, h);
      }
}

/**
 * verifica si el ratón está dentro de la paleta de colores
 * si lo está, establece selectedColor y retorna True,
 * en caso contrario, retorna False.
 * @param x, y - posición x e y del ratón
 */
boolean inside (int x, int y) {

      int i, h;

      if (x < xpos || x > xpos+width) return false;
      if (y < ypos || y > ypos+height) return false;

      h = height/colors.length;
      for (i=0; i<colors.length; i+=1) {
            if (y < (i+1)*h+ypos) {
                  selectedColor = i;
                  return true;
            }
      }
      return false;
}

}

/**
 * el applet Doodle implementa una superficie dibujable
 * con una lista de colores permitidos para dibujar en ella.
 */
public class Doodle extends Applet {
```

```
/*
 * máxima cantidad de puntos que se pueden guardar en las matrices
 * xpoints, ypoints y color
 */
static final int MaxPoints = 1000;

/*
 * matrices para contener los puntos en donde dibuja el usuario
 */
int xpoints[] = new int[MaxPoints];
int ypoints[] = new int[MaxPoints];

/*
 * color de cada punto
 */
int color[] = new int[MaxPoints];

/*
 * usado para controlar el clic previo del ratón
 * y evitar llenar las matrices con la repetición del mismo punto
 *
 */
int lastx;
int lasty;

/*
 * cantidad de puntos de la matriz
 */
int npoints = 0;
ColorBar colorBar;
boolean inColorBar;

/**
 * Inicializa el espacio de dibujo
 */
public void init () {

      setBackground(Color.white);
      colorBar = new ColorBar (10, 10, 30, 200);
}

/**
 * revisualización del espacio de dibujo
 * @param g - objeto gráfico de destino
 */
public void update (Graphics g) {

      int i;

      for (i=0; i<npoints; i+=1) {
            g.setColor (colorBar.colors[color[i]]);
            g.fillOval (xpoints[i]-5, ypoints[i]-5, 10, 10);
      }
      colorBar.paint (g);
}

/**
 * repintado del espacio de dibujo cuando se requiera
 * @param g - objeto gráfico de destino
 */
```

```
public void paint (Graphics g) {

      update (g);
}

/**
 * controla los clic del ratón y cada elección de color desde la
paleta
 * o registra el punto para repintar luego
 */
public boolean mouseDown (Event evt, int x, int y) {

      if (colorBar.inside (x, y)) {
            inColorBar = true;
            repaint ();
            return true;
      }
      inColorBar = false;
      if (npoints < MaxPoints) {
            lastx = x;
            lasty = y;
            xpoints[npoints] = x;
            ypoints[npoints] = y;
            color[npoints] = colorBar.selectedColor;
            npoints += 1;
      }
      return true;
}

/**
 * controla el arrastre del ratón
 * o registra el punto para repintar luego
 */
public boolean mouseDrag (Event evt, int x, int y) {

      if (inColorBar) return true;
      if ((x != lastx || y != lasty) && npoints < MaxPoints) {
            lastx = x;
            lasty = y;
            xpoints[npoints] = x;
            ypoints[npoints] = y;
            color[npoints] = colorBar.selectedColor;
            npoints += 1;
            repaint ();
      }
      return true;
}

/**
 * El método main permite que esta clase se procese como aplicación
 * además de poder ser usada como applet.
 * @param args - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      Frame f = new Frame ("Doodle");
      Doodle doodle = new Doodle ();

      f.resize (410, 430);
      f.add ("Center", doodle);
      f.show ();
```

```
      doodle.init ();
}
}
```

25.3 Otros eventos

En este ejemplo se reemplazará el método handleEvent() y se ilustrará como manipular distintos eventos. La técnica utilizada aquí se puede usar para cualquier clase de eventos.

```
import java.awt.*;

/**
 * clase aplicación
 * lista eventos
 */
public class EventApp extends Frame {

/*
 * TextField usado sólo para la demostración
 */
TextField location;

/*
 * área de visualización para el evento de impresión
 */
TextArea content;

/*
 * constructor
 * establece el título de la ventana de la aplicación y crea la
interfaz de usuario
 */
public EventApp () {

      int i;

      setTitle ("EventApp");
      setLayout(new BorderLayout());

      location = new TextField ("", 20);
      content = new TextArea ("", 24, 80);
      content.setEditable (false);

      Panel mypanel = new Panel ();
      mypanel.setLayout (new FlowLayout ());
      mypanel.add (location);
      mypanel.add (new Button ("Button"));
      mypanel.add (new Checkbox("Checkbox"));
      mypanel.add (new Checkbox("Checkbox"));

      add ("North", mypanel);
      add ("Center", content);

      resize (400, 300);
      show ();
```

```
}

/*
 * llamado en respuesta a eventos
 * imprime el tipo de evnto en el área textarea
 */
public boolean handleEvent(Event evt) {
      switch (evt.id) {

            case Event.KEY_ACTION:
                  content.appendText("evento de teclado acción en
tecla.\n");


                  switch (evt.key) {
                            case Event.DOWN:
                              content.appendText("tecla flecha hacia
abajo.\n");
                              return true;
                  case Event.END:
                        content.appendText("tecla end.\n");
                        return true;
                  case Event.F1:
                        content.appendText("tecla de función F1.\n");
                        return true;
                  case Event.F10:
                        content.appendText("tecla de función F10.\n");
                        return true;
                  case Event.F11:
                        content.appendText("tecla de función F11.\n");
                        return true;
                  case Event.F12:
                        content.appendText("tecla de función F12.\n");
                        return true;
                  case Event.F2:
                        content.appendText("tecla de función F2.\n");
                        return true;
                  case Event.F3:
                        content.appendText("tecla de función F3.\n");
                        return true;
                  case Event.F4:
                        content.appendText("tecla de función F4.\n");
                        return true;
                  case Event.F5:
                        content.appendText("tecla de función F5.\n");
                        return true;
                  case Event.F6:
                        content.appendText("tecla de función F6.\n");
                        return true;
                  case Event.F7:
                        content.appendText("tecla de función F7.\n");
                        return true;
                  case Event.F8:
                        content.appendText("tecla de función F8.\n");
                        return true;
                  case Event.F9:
                        content.appendText("tecla de función F9.\n");
                        return true;
                  case Event.HOME:
                        content.appendText("tecla Home.\n");
                        return true;
```

```
                case Event.LEFT:
                      content.appendText("tecla flecha hacia la
izquierda.\n");
                      return true;
                case Event.PGDN:
                      content.appendText("tecla página hacia
abajo.\n");
                      return true;
                case Event.PGUP:
                      content.appendText("tecla página hacia
arriba.\n");
                      return true;
                case Event.RIGHT:
                      content.appendText("tecla flecha hacia la
derecha.\n");
                      return true;
                    case Event.UP:
                      content.appendText("tecla flecha hacia
arriba.\n");
                      return true;
           }

                return true;
          case Event.KEY_ACTION_RELEASE:
                content.appendText("evento de teclado liberación de
acción.\n");

                switch (evt.key) {
                case Event.DOWN:
                            content.appendText("tecla flecha hacia
abajo.\n");
                            return true;
                case Event.END:
                      content.appendText("tecla end.\n");
                      return true;
                case Event.F1:
                      content.appendText("tecla de función F1.\n");
                      return true;
                case Event.F10:
                      content.appendText("tecla de función F10.\n");
                      return true;
                case Event.F11:
                      content.appendText("tecla de función F11.\n");
                      return true;
                case Event.F12:
                      content.appendText("tecla de función F12.\n");
                      return true;
                case Event.F2:
                      content.appendText("tecla de función F2.\n");
                      return true;
                case Event.F3:
                      content.appendText("tecla de función F3.\n");
                      return true;
                case Event.F4:
                      content.appendText("tecla de función F4.\n");
                      return true;
                case Event.F5:
                      content.appendText("tecla de función F5.\n");
                      return true;
                case Event.F6:
                      content.appendText("tecla de función F6.\n");
```

```
                        return true;
                  case Event.F7:
                        content.appendText("tecla de función F7.\n");
                        return true;
                  case Event.F8:
                        content.appendText("tecla de función F8.\n");
                        return true;
                  case Event.F9:
                        content.appendText("tecla de función F9.\n");
                        return true;
                  case Event.HOME:
                        content.appendText("tecla Home.\n");
                        return true;
                  case Event.LEFT:
                        content.appendText("tecla flecha hacia la
izquierda.\n");
                        return true;
                  case Event.PGDN:
                        content.appendText("tecla página hacia
abajo.\n");
                        return true;
                  case Event.PGUP:
                        content.appendText("tecla página hacia
arriba.\n");
                        return true;
                  case Event.RIGHT:
                        content.appendText("tecla flecha hacia la
derecha.\n");
                        return true;
                   case Event.UP:
                        content.appendText("tecla flecha hacia
arriba.\n");
                        return true;
                  }
                  return true;
            case Event.GOT_FOCUS:
                  content.appendText("Un componente obtuvo el
foco.\n");
                  return true;
            case Event.KEY_PRESS:
                  content.appendText("evento de teclado. Pulsado de
tecla.\n");
                  return true;
            case Event.KEY_RELEASE:
                  content.appendText("evento de tecla. Liberación de
tecla.\n");
                  return true;

            case Event.LIST_DESELECT:
                  content.appendText("Deja de seleccionar ítem.\n");
                  return true;

            case Event.LOST_FOCUS:
                  content.appendText("Un componente perdió el
foco.\n");
                  return true;

            case Event.MOUSE_DOWN:
                  content.appendText("evento de ratón, botón hacia
abajo. " +
                        evt.x + ", " + evt.y + "\n");
```

```
                return true;
            case Event.MOUSE_DRAG:
                content.appendText("evento de arrastre de ratón. " +
                    evt.x + ", " + evt.y + "\n");
                return true;
            case Event.MOUSE_ENTER:
                content.appendText("evento de entrada de ratón. " +
                    evt.x + ", " + evt.y + "\n");
                return true;
            case Event.MOUSE_EXIT:
                content.appendText("evento de salida de ratón. " +
                    evt.x + ", " + evt.y + "\n");
                return true;
            case Event.MOUSE_MOVE:
                content.appendText("evento de movimiento de ratón. "
+
                    evt.x + ", " + evt.y + "\n");
                return true;
            case Event.MOUSE_UP:
                content.appendText("evento de ratón, botón hacia
arriba. " +
                    evt.x + ", " + evt.y + "\n");
                return true;
            case Event.WINDOW_DEICONIFY:
                content.appendText("evento de ventana. De icono a
normal.\n");
                return true;
            case Event.WINDOW_DESTROY:
                content.appendText("evento de destrucción de
ventana.\n");
                return true;
            case Event.WINDOW_EXPOSE:
                content.appendText("evento de exposición de
ventana.\n");
                return true;
            case Event.WINDOW_ICONIFY:
                content.appendText("evento de ventana. De normal a
icono.\n");
                return true;
            case Event.WINDOW_MOVED:
                content.appendText("evento movimiento de
ventana.\n");
                return true;

        }
        return false;
}

/*
 * punto de entrada de la aplicación
 * crea una instancia de EventApp
 */
public static void main (String args[]) {

        new EventApp ();
}
}
```

26 Interfaz de usuario

la interfaz gráfica del usuario es otra de las partes importantes de la programación. En este apartado aprenderemos a realizar una serie de aspectos con unos cuantos ejemplos que englobarán la mayoría de ellos.

Los aspectos fundamentales en una interfaz de usuario en Java son:

- botones
- casillas de verificación
- menús
- listas de elección
- campos y áreas de texto
- deslizadores
- listas
- ventanas
- diálogos
- diálogos de archivos

26.1 Como crear botones

La calculadora del ejemplo usará la clase Button para crear los botones.

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

/**
 * Calculadora sencilla
 */
public class Calc extends Applet {

    Display  display = new Display();

/**
 *  inicializa el applet Calc
 */
    public void init () {

        setLayout(new BorderLayout());
        Keypad   keypad = new Keypad();

        add ("North", display);
        add ("Center", keypad);
    }

/**
 * manipulador para las funciones de la calculadora.
 */
    public boolean action (Event ev, Object arg) {
```

```
         if (ev.target instanceof Button) {

              String label = (String)arg;
              if (label.equals("C")) {
                display.Clear ();
                return true;
              }
              else if (label.equals(".")) {
                 display.Dot ();
                 return true;
              }
              else if (label.equals("+")) {
                 display.Plus ();
                 return true;
              }
              else if (label.equals("-")) {
                 display.Minus ();
                 return true;
              }
              else if (label.equals("x")) {
                 display.Mul ();
                 return true;
              }
              else if (label.equals("/")) {
                 display.Div ();
                 return true;
              }
              else if (label.equals("+/-")) {
                 display.Chs ();
                 return true;
              }
              else if (label.equals("=")) {
                 display.Equals ();
                 return true;
              }
              else {
                  display.Digit (label);
                  return true;
              }
         }
         return false;
    }


/**
 * Esto permite que la clase se use como applet o como aplicación
independiente
 */
public static void main (String args[]) {

      Frame f = new Frame ("Calculator");
      Calc calc = new Calc ();

      calc.init ();

      f.resize (210, 200);
      f.add ("Center", calc);
      f.show ();
}
}
```

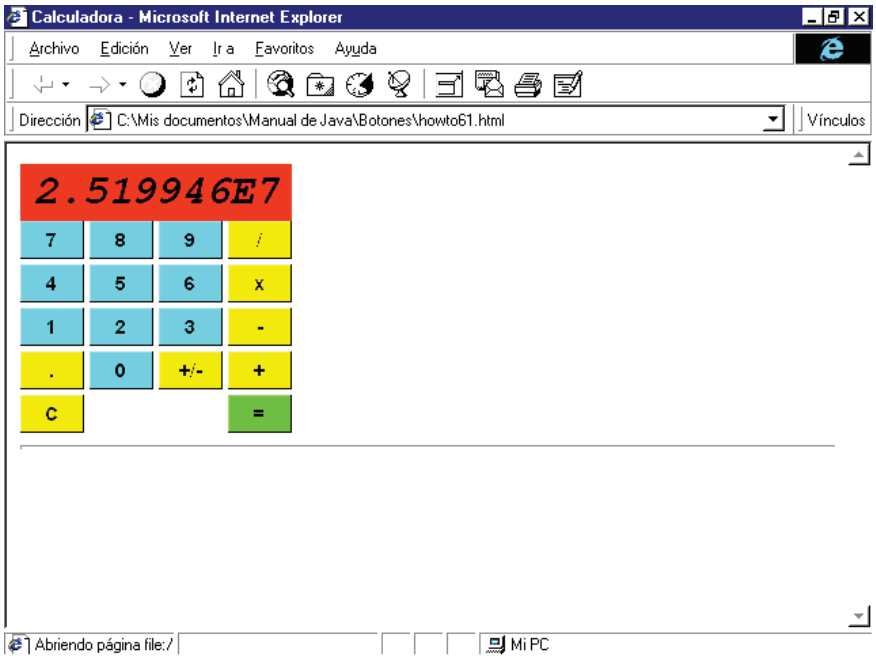
```
/* ------------------------------------------------ */

/**
 * Keypad manipula la entrada de datos y la visualización de
resultados
 */
class Keypad extends Panel {

/**
 * inicializa keypad, añade los botones, establece los colores, etc.
 */
    Keypad (){

          Button  b = new Button();
          Font    font = new Font ("Times", Font.BOLD, 14);
          Color   functionColor = new Color (255, 255, 0);
          Color   numberColor = new Color (0, 255, 255);
          Color   equalsColor = new Color (0, 255, 0);
          setFont (font);
          b.setForeground (Color.black);

          add (b = new Button ("7"));
          b.setBackground (numberColor);
          add (b = new Button ("8"));
          b.setBackground (numberColor);
          add (b = new Button ("9"));
          b.setBackground (numberColor);
          add (b = new Button ("/"));
          b.setBackground (functionColor);

          add (b = new Button ("4"));
          b.setBackground (numberColor);
          add (b = new Button ("5"));
          b.setBackground (numberColor);
          add (b = new Button ("6"));
          b.setBackground (numberColor);
          add (b = new Button ("x"));
          b.setBackground (functionColor);

          add (b = new Button ("1"));
          b.setBackground (numberColor);
          add (b = new Button ("2"));
          b.setBackground (numberColor);
          add (b = new Button ("3"));
          b.setBackground (numberColor);
          add (b = new Button ("-"));
          b.setBackground (functionColor);

          add (b = new Button ("."));
          b.setBackground (functionColor);
          add (b = new Button ("0"));
          b.setBackground (numberColor);
          add (b = new Button ("+/-"));
          b.setBackground (functionColor);
          add (b = new Button ("+"));
          b.setBackground (functionColor);

          add (b = new Button ("C"));
          b.setBackground (functionColor);
          add (new Label (""));
```

```
         add (new Label (""));
         add (b = new Button ("="));
         b.setBackground (equalsColor);

         setLayout (new GridLayout (5, 4, 4, 4));
    }

}

/* ------------------------------------------------ */

/**
 * la clase Display gestiona la visualización del resultado calculado
y también
 * implementa las tecla de fucnión de la calculadora
 */
class Display extends Panel{

    double     last = 0;
    int        op = 0;
    boolean    equals = false;
    int        maxlen = 10;
    String     s;
    Label      readout = new Label("");

/**
 * Inicializa display
 */
    Display () {

      setLayout(new BorderLayout());
      setBackground (Color.red);
        setFont (new Font ("Courier", Font.BOLD + Font.ITALIC, 30));
        readout.setAlignment(1);
        add ("Center",readout);
      repaint();
        Clear ();
    }

/**
 * manipulación del pulsado de un dígito
 */
    void Digit (String digit) {
         checkEquals ();

            /*
             *    quita los ceros precedentes
             */
         if (s.length () == 1 && s.charAt (0) == '0' && digit.charAt
(0) != '.')
                  s = s.substring (1);

         if (s.length () < maxlen)
                  s = s + digit;
         showacc ();
    }

/**
 * manipulación del punto decimal
 */
    void Dot () {
```

```
         checkEquals ();

            /*
             *    ya tiene un '.'
             */
         if (s.indexOf ('.') != -1)
                  return;

         if (s.length () < maxlen)
                  s = s + ".";
         showacc ();
    }

/**
 * si el usuario pulsa = sin haber pulsado antes un operador
 * (+,-,x,/), pone cero en la visualización
 */
    private void checkEquals () {
         if (equals == true) {
                        equals = false;
                  s = "0";
         }
    }

/**
 * operador suma para uso posterior.
 */
    void Plus () {
         op = 1;
         operation ();
    }

/**
 * operador resta para uso posterior.
 */
    void Minus () {
         op = 2;
         operation ();
    }

/**
 * operador multiplicación para uso posterior.
 */
    void Mul () {
         op = 3;
         operation ();
    }

/**
 * operador división para uso posterior.
 */
    void Div () {
         op = 4;
         operation ();
    }

/**
 * Interpreta el valor de la visualización como double  y lo  almacena
 * para uso posterior
 */
    private void operation () {
```

```
         if (s.length () == 0) return;

         Double xyz = Double.valueOf (s);
         last = xyz.doubleValue ();

         equals = false;
         s = "0";
    }
/**
 * invalida el valor actual y revisualiza.
 */
    void Chs () {
         if (s.length () == 0) return;

         if (s.charAt (0) == '-') s = s.substring (1);
         else s = "-" + s;

         showacc ();
    }

/**
 * termina el último cálculo y visualiza el resultado
 */
    void Equals () {
         double acc;

         if (s.length () == 0)  return;
         Double xyz = Double.valueOf (s);
         switch (op)  {
              case 1:
                  acc = last + xyz.doubleValue ();
                  break;

              case 2:
                  acc = last - xyz.doubleValue ();
                  break;

              case 3:
                  acc = last * xyz.doubleValue ();
                  break;

              case 4:
                  acc = last / xyz.doubleValue ();
                  break;

              default:
                  acc = 0;
                  break;
         }

         s = new Double (acc).toString ();
         showacc ();
         equals = true;
         last = 0;
         op = 0;
    }

/**
 * limpia la visualización y el último valor interno
 */
    void Clear () {
```

```
            last = 0;
            op = 0;
            s = "0";
            equals = false;
            showacc ();
        }

/**
 * pide que se repinte el resultado
 */
    private void showacc () {
        readout.setText(s);
        repaint ();
    }

}
```



26.2 Como hacer listas de elección

Se ve con claridad en el siguiente Applet

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

/*
 * la clase applet
 */
public class Converter extends Applet {

/*
 * índice de unidad de medida "desde"
 */
int fromindex = 0;

/*
 * índice de unidad de medida "hacia"
 */
int toindex = 0;

/*
 * un lugar para imprimir el factor de conversión
 */
TextField textfield = new TextField(12);

/*
 * lugar de visualización de las listas
 */
Panel listpanel = new Panel();

/*
* lugar de visualización del texto
 */
Panel textpanel = new Panel();
Choice unit1 = new Choice();
Choice unit2 = new Choice();

/*
 * matriz de factores de conversión
 */
String values[][] = {
      {"1.000", "1.000 E-2", "1.000 E-5", "3.397 E-1", "3.937 E-2",
"6.214 E-6"},
      {"1.000 E+2", "1.000", "1.000 E-3", "39.37", "3.28", "6.214 E-
4"},
      {"1.000 E+5", "1.000 E+3", "1.000", "3.937 E+4", "3.281 E+3",
"6.214 E-1"},
      {"2.54", "0.0254", "2.54 E-5", "1.000", "12.0", "1.578 E-5"},
      {"30.48", "0.3048", "3.048 E-4", "12.0", "1.000", "1.894 E-4"},
      {"1.609 E+5", "1.609 E+3", "1609", "6.336 E+4", "5280", "1.000"}
};

/*
 * método llamado en la carga del applet
 * crea la interfaz de usuario
 */
public void init() {
```

```
      textfield.setText(values[fromindex][toindex]);
      textfield.setEditable (false);

      this.setLayout(new BorderLayout());
      listpanel.setLayout(new FlowLayout());
      add("North", listpanel);
      add("South", textpanel);

      Label fromlabel = new Label ("Para convertir desde  ",1);
      listpanel.add(fromlabel);
      unit1.addItem("Centímetros");
        unit1.addItem("Metros");
        unit1.addItem("Kilómetros");
        unit1.addItem("Pulgadas");
        unit1.addItem("Pie");
        unit1.addItem("Millas");
        listpanel.add(unit1);

        Label tolabel = new Label ("  hacia    ",1);
      listpanel.add(tolabel);
      unit2.addItem("Centímetros");
        unit2.addItem("Metros");
        unit2.addItem("Kilómetros");
        unit2.addItem("Pulgadas");
        unit2.addItem("Pie");
        unit2.addItem("Millas");
      listpanel.add(unit2);

      Label multlabel = new Label ("Multiplicar por ",1);
      textpanel.add(multlabel);
      textpanel.add(textfield);
}

/**
 * llamada cuando ocurre un vento de acción
 * @param evt - objeto evento
 * @param arg - objetivo
 */
public boolean action(Event evt, Object arg) {

      if (evt.target.equals (unit1)) {
            fromindex = unit1.getSelectedIndex();
            textfield.setText(values[fromindex][toindex]);
            repaint();
      } else if (evt.target.equals (unit2)) {
            toindex = unit2.getSelectedIndex();
            textfield.setText(values[fromindex][toindex]);
            repaint();
      }
      return true;
}

/**
 * punto de entrada de la aplicación
 * @param args - argumentos de la línea de mandatos
 */
public static void main(String args[]) {

        Frame f = new Frame("Convertidor ");
        Converter converter = new Converter();
        converter.init();
```

```
        converter.start();
        f.add("Center", converter);
        f.resize(500, 100);
      f.show();
}
}
```

26.3 Deslizadores y areas de texto

Java contiene una clase ScrollBar que se usa para crear barras de desplazamiento y deslizadores.

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


/*
 * la clase applet
 */
public class ColorPicker extends Applet {

/*
 * panel que contiene los deslizadores que controlan el color RGB
 */
Panel controls = new Panel();

/*
 * panel que contiene el color de ejemplo
 */
Panel sample = new Panel();

/*
 * barra de desplazamiento para el rojo
 */
Scrollbar sbRed;

/*
 * barra de desplazamiento para el verde
 */
Scrollbar sbGreen;

/*
 * barra de desplazamiento para el azul
 */
Scrollbar sbBlue;

/*
 * TextField para el componente rojo
 */
TextField tfRed;

/*
* TextField para el componente verde
 */
TextField tfGreen;

/*
 * TextField para el componente azul
```

```
 */
TextField tfBlue;
int min = 0;
int max = 255;

/*
 * inicializa el applet
 */
public void init () {

      this.setLayout(new BorderLayout());
      controls.setLayout (new GridLayout(3,3,5,5));
      this.add ("South",controls);
      this.add ("Center",sample);

      tfRed = new TextField (5);
      tfGreen = new TextField (5);
      tfBlue = new TextField (5);
      controls.add (new Label ("Rojo",1));
      controls.add (new Label ("Verde",1));
      controls.add (new Label ("Azul",1));
      controls.add (tfRed);
      controls.add (tfGreen);
      controls.add (tfBlue);

      sbRed = new Scrollbar (Scrollbar.HORIZONTAL, 0, 1, min, max);
      //establece los valores de la barra de desplazamiento
      // valor, tamaño de la página, mínimo, máximo
      controls.add (sbRed);

      sbGreen = new Scrollbar (Scrollbar.HORIZONTAL, 0, 1, min, max);
      //establece los valores de la barra de desplazamiento
      // valor, tamaño de la página, mínimo, máximo

      controls.add (sbGreen);

      sbBlue = new Scrollbar (Scrollbar.HORIZONTAL, 0, 1, min, max);
      //establece los valores de la barra de desplazamiento
      // valor, tamaño de la página, mínimo, máximo
      controls.add (sbBlue);

      // establece los campos de texto según el valor del deslizador
      tfRed.setText(String.valueOf(sbRed.getValue()));
      tfGreen.setText(String.valueOf(sbGreen.getValue()));
      tfBlue.setText(String.valueOf(sbBlue.getValue()));

      changecolor();
}

/** Obtiene el valor vigente en el campo de texto
 * @param textField -
 */
double getValue(TextField textField) {

      double f;
      try {
            f = Double.valueOf(textField.getText()).doubleValue();
      } catch (java.lang.NumberFormatException e) {
            f = 0.0;
      }
      return f;
```

```
}

/*
 * rspuesta a las acciones de usuario
 * @param evt - objeto evento
 */
public boolean handleEvent(Event evt) {

      if (evt.target.equals (sbRed)) {
            tfRed.setText(String.valueOf(sbRed.getValue()));
            changecolor();
      } else if (evt.target.equals (sbGreen)) {
            tfGreen.setText(String.valueOf(sbGreen.getValue()));
            changecolor();
      } else if (evt.target.equals (sbBlue)) {
            tfBlue.setText(String.valueOf(sbBlue.getValue()));
            changecolor();
      } else if ((evt.target.equals (tfRed))
            && (evt.id == Event.ACTION_EVENT)) {
            setSliderValue(sbRed, getValue(tfRed));
            tfRed.setText(String.valueOf(sbRed.getValue()));
            changecolor();
      } else if ((evt.target.equals (tfGreen))
            && (evt.id == Event.ACTION_EVENT)) {
            setSliderValue(sbGreen, getValue(tfGreen));
            tfGreen.setText(String.valueOf(sbGreen.getValue()));
            changecolor();
      } else if ((evt.target.equals (tfBlue))
            && (evt.id == Event.ACTION_EVENT)) {
            setSliderValue(sbBlue, getValue(tfBlue));
            tfBlue.setText(String.valueOf(sbBlue.getValue()));
            changecolor();
      }
      return false;
}

/*
 * establece el valor del deslizador
 * @param slider - instancia de Scrollbar
 * @param f - valor a establecer en el deslizador
 */
void setSliderValue(Scrollbar slider, double f) {

      int sliderValue = (int)f;

      if (sliderValue > max)
            sliderValue = max;
      if (sliderValue < min)
            sliderValue = min;
      slider.setValue(sliderValue);
}

/*
 * cambia el color según la definición RGB
 */
public void changecolor () {

      int i;

      sample.setBackground(new Color((int)getValue(tfRed),
            (int)getValue(tfGreen), (int)getValue(tfBlue)));
```

```
      repaint();
}

public void update (Graphics g) {

      paintAll(g);
}

/*
 * punto de entrada de la aplicación
 * no se usa cuando se ejecuta como applet
 * @param args - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      Frame f = new Frame ("Selector de colores");
      ColorPicker colorPicker = new ColorPicker ();

      colorPicker.init ();
        f.resize (300, 200);
        f.add ("Center", colorPicker);
      f.show ();
}
}
```

26.4 Como usar listas

El conjunto de herramientas AWT dan soporte a las listas. La clase List contiene los métodos para crear, añadir, borrar, seleccionar y anular la selección de items, entre tantos otros útiles métodos. Se puede usar el método getSelectedIndex() para determinar que item se ha seleccionado.

```
import java.applet.Applet;
import java.awt.*;
import java.io.*;
import java.util.StringTokenizer;

/**
 * clase de la aplicación de la agenda telefónica
 * muestra el uso de una lista
 * no se puede ejecutar como applet porque
 * se necesita leer un  archivo local
 */
public class PhoneBook extends Frame {

/*
 * cantidad máxima de números telefónicos
 */
final int MaxNumbers = 25;

/*
 * una instancia de la lista
 */
List phoneList;
```

```
/*
 * instancias de botón
 */
Button addButton;
Button deleteButton;
Button findButton;
Button saveButton;

/*
 * instancia de campo de texto
 */
TextField nameField;
TextField phoneField;
TextField emailField;

/*
 * una matriz de cadenas contiene todos los números telefónicos
 */
String data[] = new String[MaxNumbers];
int NNumbers = 15;

/*
 * constructor
 * añade todos los componentes de la interfaz de usuario
 * lee en el archivo de datos
 * visualiza los nombres en la lista
 */
public PhoneBook () {

      int i;

      setTitle ("Agenda telefónica");
      setLayout (new BorderLayout ());

      Panel buttonPanel = new Panel ();
      buttonPanel.setLayout (new FlowLayout ());
      buttonPanel.add (findButton = new Button ("Encontrar nombre"));
      buttonPanel.add (addButton = new Button ("Añadir"));
      buttonPanel.add (deleteButton = new Button ("Borrar"));
      buttonPanel.add (saveButton = new Button ("Guardar"));

      Panel fieldPanel = new Panel ();
      fieldPanel.setLayout (new GridLayout (6, 1, 5, 5));
      fieldPanel.add (new Label ("Nombre"));
      fieldPanel.add (nameField = new TextField ("", 32));
      fieldPanel.add (new Label ("Número telefónico"));
      fieldPanel.add (phoneField = new TextField ("", 32));
      fieldPanel.add (new Label ("Dirección electrónica"));
      fieldPanel.add (emailField = new TextField ("", 32));

      phoneList = new List (5, false);

      FileInputStream fs = null;

        try {
                fs = new FileInputStream ("PhoneBook.txt");
        } catch (Exception e) {
                System.out.println ("No se puede abrir
PhoneBook.txt");
                System.exit (1);
        }
```

```
        DataInputStream ds = new DataInputStream (fs);

      NNumbers = 0;
      for (i=0; i<MaxNumbers; i+=1) {
                try {
                        data[i] = ds.readLine ();
                        if (data[i] == null) break;
                  NNumbers += 1;
                } catch (IOException e) {
                        System.out.println ("Detecta error en lectura
de archivo.");
                        break;
                }
            StringTokenizer st = new StringTokenizer (data[i], ":\t");
            phoneList.addItem (st.nextToken ());
      }
        try {
                fs.close ();
        } catch (IOException e) {
                System.out.println ("Detecta error en cierre de
archivo.");
        }

      add ("North", buttonPanel);
      add ("South", fieldPanel);
      add ("Center", phoneList);

      resize (300, 400);
      show ();
}

/*
 * manipula la pulsaciones de botón y los eventos sobre la lista
 */
public boolean action (Event evt, Object arg) {

      int i;
      StringTokenizer st;
      String dname;
      String tname;

      tname = nameField.getText ();
      if (evt.target == findButton || evt.target == nameField) {
            for (i=0; i<NNumbers; i+=1) {
                  st = new StringTokenizer (data[i], ":\t");
                  dname = st.nextToken ();
                  if (contains (tname, dname)) {
                        phoneList.select (i);
                        nameField.setText (dname);
                        phoneField.setText (st.nextToken ());
                        emailField.setText (st.nextToken ());
                        break;
                  }
            }
            return true;
      }

      if (evt.target == addButton) {
            if (NNumbers < MaxNumbers) {
                  tname = nameField.getText ();
                  dname = tname + ":" + phoneField.getText() +
```

```
                        ":" + emailField.getText();
                  data[NNumbers] = dname;
                  NNumbers += 1;
                  phoneList.addItem (tname);
            }
            return true;
      }

      if (evt.target == deleteButton) {
            i = phoneList.getSelectedIndex ();
            phoneList.delItem (i);
            for (; i<NNumbers-1; i+=1) data[i] = data[i+1];
            NNumbers -= 1;
            nameField.setText ("");
            phoneField.setText ("");
            emailField.setText ("");
            return true;
      }

      if (evt.target == saveButton) {
            FileOutputStream fs = null;

              try {
                  fs = new FileOutputStream ("PhoneBook.txt");
            } catch (Exception e) {
                  System.out.println ("No se puede abrir
PhoneBook.txt");
                  System.exit (1);
            }
            PrintStream ps = new PrintStream (fs);

            for (i=0; i<NNumbers; i+=1) {
                  ps.println (data[i]);
                }
            return true;
      }

      if (evt.target == phoneList) {
            i = phoneList.getSelectedIndex ();
            st = new StringTokenizer (data[i], ":\t");
            nameField.setText (st.nextToken ());
            phoneField.setText (st.nextToken ());
            emailField.setText (st.nextToken ());
            return true;
      }

      return false;
}

/*
 * el método contains compara cadenas, devuelve true
 * si una cadena está totalmente contenida en otra
 *      String s1, s2   las dos cadenas que se comparan
 */
boolean contains (String s1, String s2) {

        int i;
        int l1 = s1.length ();
        int l2 = s2.length ();

        if (l1 < l2) {
```

```
                for (i=0; i<=l2-l1; i+=1)
                        if (s1.regionMatches (true, 0, s2, i, l1))
                                return true;
        }
        for (i=0; i<=l1-l2; i+=1)
                if (s2.regionMatches (true, 0, s1, i, l2))
                        return true;

        return false;
}

/**
 * punto de entrada de la aplicación
 * @param args - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      new PhoneBook ();
}
}
```

26.5 Crear ventanas

En Java las ventanas se pueden crear usando la clase Frame. El título del marco se puede especificar mediante el uso del constructor de la clase Frame con un título o especificandolo luego con el método Frame.SetTitle().

Los componentes se añaden al marco mediante la llamada al método Frame.add() con el argumento que indica el componente que se añade. Existen varios eventos que son propios de las ventanas: Event.WINDOW_DESTROY, Event.WINDOW_ICONIFY y Event.WINDOW_MOVE. El ejemplo a continuación escrito se puede usar para determinar que eventos se producen ante distintas acciones.

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

/*
 * la clase del applet
 */
public class Fonts extends Applet {

/*
 * cantidad máxima de fuentes que se pueden buscar
 */
final int MaxFonts = 10;

/*
 * ancho y alto del panel
 */
int width, height;

/*
 * matriz para contener el nombre de las fuentes
 */
```

```
String fontName[];

/*
 * matriz de fuentes: versiones normal, cursiva y negrita
 */
Font theFonts[] = new Font[3 * MaxFonts];

/*
 * cantidad de fuentes encontradas
 */
int nfonts = 0;

/*
 * instancia de la ventana de depuración para la impresión de mensajes
 */
DebugWin debugWin;

/*
 * llamado cuando se carga el applet
 * crea la ventana de depuración y obtiene las fuentes del sistema
 */
public void init () {

      int i;
      Dimension d = size ();

      width = d.width;
      height = d.height;

      debugWin = new DebugWin ();

      debugWin.println ("Ancho = "+width);
      debugWin.println ("Alto = "+height);

      fontName = Toolkit.getDefaultToolkit().getFontList ();

      nfonts = fontName.length;

      debugWin.println ("Cantidad de fuentes = "+fontName.length);

      if (nfonts > MaxFonts) nfonts = MaxFonts;
      for (i=0; i<nfonts; i+=1) {

            debugWin.println ("La fuente "+i+" es "+fontName[i]);

            theFonts[3*i + 0] = new Font (fontName[i], Font.PLAIN, 12);
            theFonts[3*i + 1] = new Font (fontName[i], Font.BOLD, 12);
            theFonts[3*i + 2] = new Font (fontName[i], Font.ITALIC,
12);
      }
}

/**
 * pinta los nombres de fuentes
 * @param g - objeto gráfico de destino
 */
public void paint (Graphics g) {

      int i;

      for (i=0; i<nfonts; i+=1) {
```

```
            g.setFont (theFonts[3*i + 0]);
            g.drawString (fontName[i], 10, 20*i+30);
            g.setFont (theFonts[3*i + 1]);
            g.drawString ("Negrita", 70, 20*i+30);
            g.setFont (theFonts[3*i + 2]);
            g.drawString ("Cursiva", 150, 20*i+30);
      }
}

/**
 * punto de entrada de la aplicación
 * @param args - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      Frame f = new Frame ("Fuentes");
      Fonts fonts = new Fonts ();

      f.resize (200, 200);
      f.add ("Center", fonts);
      f.show ();
      fonts.init ();
}
}

import java.awt.*;
import java.io.*;

/*
 * una clase para implementar una salida de datos para una ventana de
depuración
*/
public class DebugWin extends Frame {

/*
 * punto final del área de texto, usado para inserción
 */
int endPosition;

/*
 * objeto TextArea usado para visualizar texto
 */
private     TextArea t;

/*
 * el constructor crea la ventana y añade el área de texto
 */
public DebugWin()
{
      setTitle ("Ventana para depuración");

      Font f = new Font("Courier", Font.PLAIN, 12);
      setFont(f);
      setBackground(Color.white);

      t = new TextArea ("", 24, 80);
      t.setEditable (true);
      endPosition = 0;
      add ("Center", t);

      resize(300,200);
```

```
      pack();
      show();
}

/**
 * un método para añadir texto al final del área de texto
 * @param s - texto que se introduce
 */
public void print (String s) {

      t.insertText (s, endPosition);
      endPosition += s.length ();
}

/**
 * un método para añadir texto y un indicador de nueva línea en el
área de texto
 * @param s - texto que se introduce
 */


public void println (String s) {

      t.insertText (s+"\n", endPosition);
      endPosition += s.length () + 1;
}

/**
 * graba el contenido del área de texto en un archivo
 * @param s - nombre del archivo de salida
 */
public void write (String s) {

      FileOutputStream out = null;

      try {
            out = new FileOutputStream(s);
      } catch (Exception e) {
            println ("Error al abrir el archivo " + s);
            return;
      }

      PrintStream psOut = new PrintStream (out);

      psOut.print (t.getText());

      try {
            out.close ();
      } catch (IOException e) {
            println ("No se puede cerrar el archivo "+s);
      }
}

/*
 * manipulación de los eventos
 * @param evt - evento
 */
public boolean handleEvent(Event evt) {

      String filename;
```

```
      switch(evt.id)
      {
            case Event.WINDOW_DESTROY:
                  dispose ();
                  return true;
      }

      return false;
}
}
```

26.6 Usar dialogos de archivos.

Java admite diálogos de archivo (File Dialog) de una manera similar a la utilizada por X Windows, Macintosh y Windows. De hecho, el sistema de diálogo de archivos soportados por Java comprende dos tipos: diálogos de apertura y dialogos de guardar. En realidad son de la misma clase. La única diferencia es que una constante le dice a Java qué tipo de diálogo se usará.

```
import java.awt.*;

/**
 * una clase que manipula el desplazamiento de texto
 */
class Scroll {

/*
 * coordenadas del punto de inicio
 */
int xstart, ystart;

/*
 * ancho y alto del panel limitante
 */
int width, height;

/*
 * texto que se desplaza
 */
String text;

/*
 * velocidades x e y
 */
int deltaX, deltaY;

/*
 * posición x e y del texto
 */
int xpos, ypos;

/*
 * color del texto
 */
Color color;
```

```
/**
 * constructor de la clase (simplemente guarda los argumentos)
 * @param x, y - coordenadas de inicio
 * @param dx, dy -velocidades x e y
 * @param w, h -ancho y alto del panel limitante
 * @param t - texto
 * @param c -color del texto
 */
public Scroll (int x, int y, int dx, int dy, int w, int h, String t,
Color c) {

      xstart = x;
      ystart = y;
      width = w;
      height = h;
      text = t;
      deltaX = dx;
      deltaY = dy;
      color = c;
      xpos = xstart;
      ypos = ystart;
}

/*
 * dibujo del texto en la posición actual
 * avanza la posición y reinicializa el panel limitante exterior
 * @param g - objeto gráfico de destino
 */
void paint (Graphics g) {

      g.setColor (color);
      g.drawString (text, xpos, ypos);
      xpos += deltaX;
      ypos += deltaY;

      FontMetrics fm = g.getFontMetrics ();
      int textw = fm.stringWidth (text);
      int texth = fm.getHeight ();
      if (deltaX < 0 && xpos < -textw) xpos = xstart;
      if (deltaX > 0 && xpos > width) xpos = xstart;
      if (deltaY < 0 && ypos < 0) ypos = ystart;
      if (deltaY > 0 && ypos > height+texth) ypos = ystart;
}
}


import java.awt.*;
import java.io.*;

/*
 * una aplicación de un editor de texto sencillo
 */
public class NotePad extends Frame {

/*
 * TextArea
 */
private     TextArea t;

/*
```

```
 * instancias de FileDialog
 */
private FileDialog openDialog;
private FileDialog saveDialog;

/*
 * barra de menú
 */
private MenuBar mbar;

/*
 * el constructor crea la ventana de la aplicación y la barra de menú
 */
public NotePad ()
{
      super ("NotePad Editor");

      Font f = new Font("Courier", Font.PLAIN, 12);
      setFont(f);
      setBackground(Color.white);

      mbar = new MenuBar();
      Menu m = new Menu("Archivo");
      m.add(new MenuItem("Nuevo"));
      m.add(new MenuItem("Abrir"));
      m.add(new MenuItem("Guardar"));
      m.add(new MenuItem("Guardar como"));
      m.addSeparator();
      m.add(new MenuItem("Salida"));
      mbar.add(m);

      m = new Menu("Help");
      m.add(new MenuItem("Ayuda!!!!"));
      m.addSeparator();
      m.add(new MenuItem("Acerca de..."));
      mbar.add(m);

      t = new TextArea ("", 24, 80);
      t.setEditable (true);
      add ("Center", t);

      openDialog = new FileDialog(this, "Abrir archivo...",
FileDialog.LOAD);
      saveDialog = new FileDialog(this, "Guardar archivo...",
FileDialog.SAVE);

      setMenuBar(mbar);
      resize(300,200);
      pack();
      show();
}

/**
 * lee el texto desde el archivo y lo introduce en el área de texto
 * @param s - nombre del archivo de entrada
 */
public void read (String s) {

      String line;
      int i =0;
      FileInputStream in = null;
```

```
      DataInputStream dataIn = null;

      try {
            in = new FileInputStream(s);
            dataIn = new DataInputStream (in);
      } catch(Throwable e) {
            System.out.println("Error al abrir el archivo");
            return;
      }

      try {
            while ((line = dataIn.readLine()) != null)
            {
                  line += "\n";
                  t.insertText (line, i);
                  i += line.length ();
            }
      } catch(IOException e) {
            System.out.println("Error al leer el archivo");
      }

      try {
            in.close ();
      } catch (IOException e) {
            System.out.println (e);
      }
}

/**
 * graba texto en el archivo desde el área de texto
 * @param s - nombre del archivo de salida
 */
public void write(String s) {

      FileOutputStream out = null;

      try {
            out = new FileOutputStream(s);
      } catch (Exception e) {
            System.out.println("Error al abrir el archivo");
            return;
      }

      PrintStream psOut = new PrintStream (out);

      psOut.print (t.getText());

      try {
            out.close ();
      } catch (IOException e) {
            System.out.println (e);
      }
}

/**
 * manipulación de eventos de usuario
 * @param evt - evento
 */
public boolean handleEvent (Event evt) {

      String filename;
```

```
      switch(evt.id)
      {
      //    System.out.println(evt.target.toString());
            case Event.WINDOW_DESTROY:
                  System.exit(0);
                  return true;

            case Event.ACTION_EVENT:
                  if(evt.target instanceof MenuItem)
                  {
                        if (evt.arg.equals("Abrir")) {

                              openDialog.show();
                              filename = openDialog.getFile();
                              read(filename);
                              return true;
                        }
                        if (evt.arg.equals("Guardar como")) {

                              saveDialog.show();
                              filename = saveDialog.getFile();
                              write(filename);
                              return true;
                        }
                        if(evt.arg.equals("Salida")) {
                              System.exit(0);
                        }
                        if (evt.arg.equals("Acerca de...")) {
                              AboutDialog ab = new AboutDialog(this);
                              ab.show();
                              return true;
                        }
                  }
                  break;
      }

      return false;
}

/**
 * punto de entrada en la aplicación
 * @param args - argumentos de la línea de mandatos
 */
public static void main(String args[])
{
      new NotePad();
}
}

/*
 * una clase para visualizar el diálogo de información acerca de la
aplicación
 */
class AboutDialog extends Dialog {

/*
 * instancias del texto desplazable
 */
Scroll left, right;
```

```
/**
 * el constructor crea el diálogo
 * @param parent - marco de ventana padre
 */
public AboutDialog(NotePad parent) {

      super(parent, "About Dialog", true);
      Panel p = new Panel();
      p.add(new Button("OK"));
      add("South", p);

      setFont (new Font ("Dialog", Font.BOLD + Font.ITALIC, 24));

      int w = 200;
      int h = 150;

      resize(w, h);

      left = new Scroll (w, 50, -5, 0, w, h, "About...", Color.red);
      right = new Scroll (0, 75, 5, 0, w, h, "Dialog!", Color.green);
}

/**
 * pinta el texto desplazable y fuerza el repintado cada 50 ms
 * @param g - contexto gráfico de destino
 */
public void paint (Graphics g) {

      left.paint (g);
      right.paint (g);
      repaint (50);
}

/**
 * destruye el diálogo al pulsar el botón OK
 * @param evt - evento
 */
public boolean handleEvent(Event evt) {

      switch(evt.id)
      {
            case Event.ACTION_EVENT:
            {
                  if("OK".equals(evt.arg))
                  {
                        dispose ();
                        return true;
                  }
            }
      }
      return false;
}
}
```

27 Algunos programas avanzados

Solo pondremos algunos ejemplos avanzados para que el usuario de este manual pueda utilizarlos para introducirse mucho mas en la programación en Java, ya que aún quedan muchos aspectos que este manual no cubre y que podría ser preciso escribir otro:

27.1 Animación y sonido

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

/**
 * Una clase que describe un objetivo (invasor)
 * contiene un invasor
 */
class Invader {

/*
 * posición y velocidad del invasor
 */
double x, y, dx, dy;

/*
 * posición y velocidad del misil del invasor
 * sólo se permite un misil por invasor
 */
int mx, my, mdy;

/*
 * imagen de un invasor
 */
Image img[] = new Image[4];

/*
 * inplay es True si el invasor está vivo
 */
boolean inplay;

/*
 * fired se pone en True cuando invasor dispara un misil
 */
boolean fired = false;

/*
 * state se usa para hacer ciclos entre las cuatro imágenes de un
invasor
 */
double state;

/*
 * value es el valor del resultado actual, depende de la velocidad
 */
int value;
```

```
/*
 * inicializa la posición y la velocidad del invasor
 */
void random (int speed, int w, int h) {

      x = 10 + (w-20)*Math.random ();
      y = 10 + ((h>>1)-20)*Math.random ();
      dx = (speed>>1) - speed*Math.random ();
      dy = (speed>>1) - speed*Math.random ();
      inplay = true;
      state = 3 * Math.random ();
      fired = false;
      mdy = 20;
      value = speed * 10;
}

/*
 * calcula la posición del invasor y del misil.
 * también dispara un misil al azar.
 * @param w ancho del panel
 * @param h altura del panel
 */
void compute (int w, int h) {

      if (x <= 0 || x > w) dx = -dx;
      if (y <= 0 || y > h>>1) dy = -dy;
      if (my > h-20) fired = false;
      if (fired) my += mdy;
      else my = 0;
      if (inplay && !fired && Math.random () > 0.99) {
            fired = true;
            mx = (int) x; my = (int) y+25;
      }
      x += dx; y += dy;
}

/*
 * pinta invasor y misil (si hubo disparo)
 * @param g objeto gráfico de destino
 * @param obs     observador asociado con este contexto gráfico
 */
public void paint (Graphics g, ImageObserver obs) {

      int whichImage;

      if (inplay) {
            whichImage = (int) state;
            g.drawImage (img[whichImage & 0x3], (int) x-25,
                  (int) y-25, obs);
            state += .25;
      }
      if (fired) {
            g.setColor (Color.green);
            g.drawLine ((int) mx, (int) my, (int) mx, (int) my-10);
      }
}

/*
 * verifica si los misiles del jugador han acertado con algún invasor
 * retorna True si el inavasor fue alcanzado
 * @param pmx     posición x del misil del jugador
```

```
 * @param pmy     posición y del misil del jugador
 */
boolean killer (int pmx, int pmy) {

      int deltaX, deltaY;

      if (!inplay) return false;
            deltaX = (int) Math.abs (x-pmx);
            deltaY = (int) Math.abs (y-pmy);
            if (deltaX < 20 && deltaY < 20) {
                  inplay = false;
                  return true;
            }
            return false;
}
}

/**
 * una clase para describir al jugador, muy similar al invasor
*/
class Player {

/*
 * posición del jugador
 */
int x, y=-100;

/*
 * posición del misil del jugador
 */
int mx, my, mdy = -20;

/*
 * dos imágenes del jugador
 */
Image img1, img2;

/*
 * fired es True si el jugador ha disparado un misil
 * inplay es true si el juego no ha terminado
 */
boolean fired = false, inplay=true;

/*
 * método llamado cuando un jugador dispara un misil
 */
void fire () {

      if (fired || !inplay) return;
      mx = x; my = y;
      fired = true;
}

/*
 * calcula la siguiente posición del misil
 */
void compute () {

      if (my < 0) fired = false;
      if (fired) my += mdy;
      else my = y;
```

```
}

/**
 * pinta al jugador y al misil
 * @param g - objeto gráfico de destino
 * @param obs - observador
 */
public void paint (Graphics g, ImageObserver obs) {

      if (fired) {
            if (inplay) g.drawImage (img2, x-25, y, obs);
            g.setColor (Color.white);
            g.drawLine (mx, my, mx, my+10);
      } else if (inplay) g.drawImage (img1, x-25, y, obs);
}

/**
 * devuelve True si han matado al jugador
 * @param bmx, bmy - posición del misil enemigo
 */
boolean killer (int bmx, int bmy) {

      int dx, dy;

      if (!inplay) return false;
      dx = (int) Math.abs (x-bmx);
      dy = (int) Math.abs (y-bmy);
      if (dx < 20 && dy < 20) {
            return true;
      }
      return false;
}
}

/*
 * gran parte de la lógica del juego está aquí
 */
class Playfield extends Panel implements Runnable {

static final int PLAYER_HIT = 1;
static final int INVADER_HIT = 2;

InvaderApp invaderApp;

/*
 * cantidad de invasores en juego
 */
int NInvaders=0;

/*
 * máxima cantidad de invasores en juego
 */
final int MaxInvaders = 32;

/*
 * matriz de invasores
 */
Invader invaders[] = new Invader[MaxInvaders];
Player player;

/*
```

```
 * imagen fuera de pantalla para el uso de doble búfer
 */
Image offscreen;

/*
 * dimensión de la imagen gráfica fuera de pantalla
 */
Dimension psize;

/*
 * objeto gráfico asociado con la imagen fuera de pantalla
 */
Graphics offgraphics;

/*
 * trama de acción del juego
 */
Thread theThread;

/*
 * color del fondo del juego
 */
Color bgcolor = new Color (51, 0, 153);
int score, playerLives, playLevel;
Font font;

/**
 * el constructor guarda la instancia del applet
 * @param invaderApp - instancia del applet
 */
public Playfield (InvaderApp invaderApp) {

      this.invaderApp = invaderApp;
}

/*
 * trama de acción del juego
 */
public void run() {

      psize = size();
      offscreen = createImage (psize.width, psize.height);
      offgraphics = offscreen.getGraphics ();
      font = new Font ("TimesRoman", Font.BOLD, 18);
      offgraphics.setFont (font);

      while (true) {
            compute ();
            repaint ();
            try {
                  Thread.sleep(25);
            } catch (InterruptedException e) { }
      }
}

/*
 * calcula las nuevas posiciones de todos los objetos
 */
synchronized void compute () {

      for (int i=0; i<NInvaders; i+=1) {
```

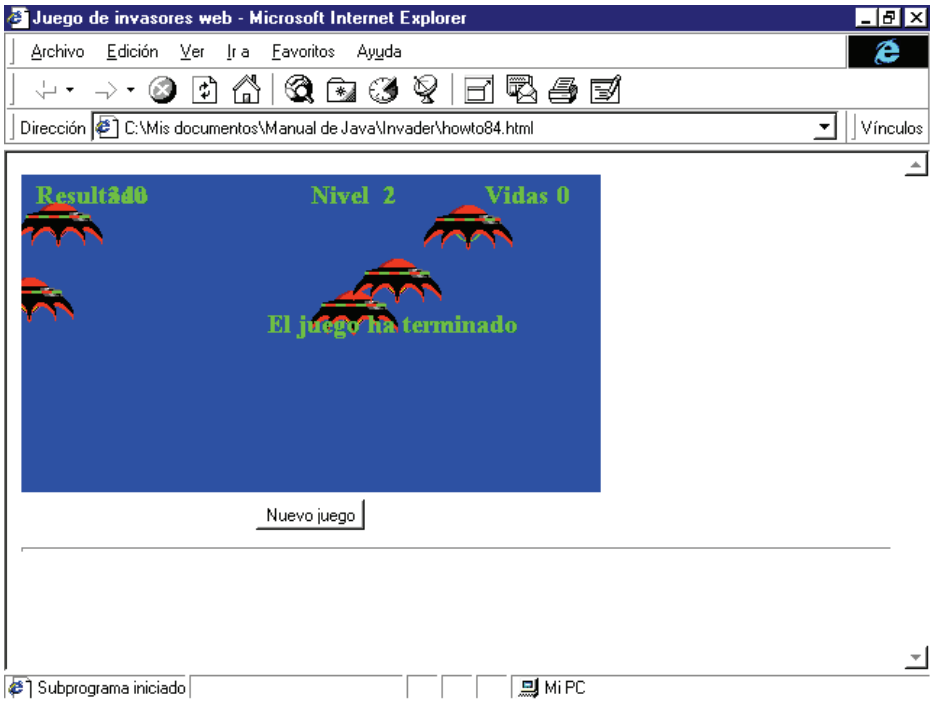
```
            invaders[i].compute (psize.width, psize.height);
            if (invaders[i].killer (player.mx, player.my)) {
                  invaderApp.hit (INVADER_HIT);
                  player.fired = false;
                  score += invaders[i].value;
            }
            if (player.killer (invaders[i].mx, invaders[i].my)) {
                  invaderApp.hit (PLAYER_HIT);
                  invaders[i].fired = false;
                  playerLives -= 1;
                  if (playerLives < 1) player.inplay = false;
            }
      }
      player.compute ();
}

/**
 * reemplaza el método update por defecto
 * dibuja en la imagen fuera de pantalla y luego la copia en la
pantalla
 * @param g - objeto gráfico de destino
 */
public synchronized void update(Graphics g) {

      offgraphics.setColor (bgcolor);
      offgraphics.fillRect (0, 0, psize.width, psize.height);

      for (int i = 0 ; i < NInvaders ; i++)
            if (invaders[i].inplay) invaders[i].paint (offgraphics,
this);
      player.paint (offgraphics, this);

      offgraphics.setColor (Color.green);
      offgraphics.drawString ("Resultado", 10, 20);
      offgraphics.drawString (Integer.toString (score), 60, 20);
      offgraphics.drawString ("Nivel", psize.width>>1, 20);
      offgraphics.drawString (Integer.toString (playLevel),
            (psize.width>>1)+50, 20);
      offgraphics.drawString ("Vidas", psize.width-80, 20);
      offgraphics.drawString (Integer.toString (playerLives),
            psize.width-30, 20);
      if (playerLives < 1) offgraphics.drawString ("El juego ha
terminado",
            (psize.width>>1)-30, psize.height>>1);
      g.drawImage (offscreen, 0, 0, null);
}

/**
 * dispara el misil del jugador
 * @param evt - evento
 * @param x, y - posición del ratón
 */
public synchronized boolean mouseDown(Event evt, int x, int y) {

      player.fire ();
      return true;
}

/**
 * mueve la posición del jugador
 * @param evt - evento
```

```
 * @param x, y - posición del ratón
 */
public boolean mouseMove (Event evt, int x, int y) {

      player.x = x;
      player.y = psize.height-45;
      if (player.x < 20) player.x = 20;
      if (player.x > psize.width-20) player.x = psize.width-20;
      return true;
}

/*
 * arranca la trama del juego
 */
public void start() {

      theThread = new Thread (this);
      theThread.start ();
}

/*
 * detiene la trama del juego
 */
public void stop() {

      theThread.stop ();
}
}

/*
 * la clase del applet
 */
public class InvaderApp extends Applet {

/*
 * instancia del campo de juego
 */
Playfield panel;

/*
 * almacenamiento temporario de las imágenes
 */
Image img[] = new Image[4];

/*
 * la velocidad del juego
 * cantidad de invasores en esta secuencia
 */
int speed, NInvadersInPlay;

/*
 * método llamado cuando se carga el applet
 * carga las imágenes
 */
public void init() {

      int i;
      MediaTracker tracker = new MediaTracker (this);

      setLayout(new BorderLayout());
```

```
      panel = new Playfield (this);
      add("Center", panel);
      Panel p = new Panel();
      add("South", p);
      p.add(new Button("Nuevo juego"));

      showStatus ("Obtención de las imágenes de los invasores...");
      for (i=0; i<4; i+=1) {
            img[i] = getImage (getDocumentBase(), "T"+(i+1)+".gif");
            tracker.addImage (img[i], 0);
      }

      try {
            tracker.waitForID(0);
      } catch (InterruptedException e) { }

      for (i=0; i<panel.MaxInvaders; i+=1) {
            panel.invaders[i] = new Invader ();
            panel.invaders[i].inplay = false;
            panel.invaders[i].img[0] = img[0];
            panel.invaders[i].img[1] = img[1];
            panel.invaders[i].img[2] = img[2];
            panel.invaders[i].img[3] = img[3];
      }
      panel.player = new Player ();

      showStatus ("Obtención de las imágenes del jugador...");
      panel.player.img1 = getImage (getDocumentBase(), "Player1.gif");
      panel.player.img2 = getImage (getDocumentBase(), "Player2.gif");

      tracker.addImage (panel.player.img1, 1);
      tracker.addImage (panel.player.img2, 1);
      try {
            tracker.waitForID (1);
      } catch (InterruptedException e) { }
      showStatus ("¡Listo para jugar!");
}

/*
 * arranca la trama de acción
 */
public void start() {

      panel.start();
}

/*
 *  detiene la trama de acción
 */
public void stop() {

      panel.stop();
}

/*
 * manipula los eventos de botón
 * @param evt - evento
 * @param arg - objetivo
 */
public boolean action(Event evt, Object arg) {
```

```
      if ("Nuevo juego".equals(arg)) {
            speed = 10;
            panel.player.inplay = true;
            panel.playerLives = 3;
            panel.score = 0;
            panel.playLevel = 1;
            NInvadersInPlay = 2 * panel.playLevel + 1;
            panel.NInvaders = NInvadersInPlay;
            for (int i=0; i<panel.NInvaders; i+=1)
                  panel.invaders[i].random (speed,
                        panel.psize.width, panel.psize.height);

            play (getCodeBase(), "gong.au");
            if (NInvadersInPlay >= panel.MaxInvaders)
                  NInvadersInPlay = panel.MaxInvaders;
            return true;
      }
      return false;
}

/**
 * reproduce el sonido adecuado cuando un misil alcanza algún objetivo
 * @param which - determina qué sonido se debe reproducir
 */
public void hit (int which) {

      switch (which) {
            case Playfield.INVADER_HIT:
            NInvadersInPlay -= 1;
            if (NInvadersInPlay < 1) {
                  play (getCodeBase(), "gong.au");
                  panel.playLevel += 1;
                  NInvadersInPlay = 2 * panel.playLevel + 1;
                  speed += 4;
                  panel.NInvaders = NInvadersInPlay;
                  for (int i=0; i<panel.NInvaders; i+=1)
                  panel.invaders[i].random (speed,
                        panel.psize.width, panel.psize.height);
            } else {
                  play (getCodeBase(), "drip.au");
            }
            break;

            case Playfield.PLAYER_HIT:
            play (getCodeBase(), "doh2.au");
            break;
      }
}
```

```
}
```



27.2 Redes

```
import java.applet.Applet;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

/**
 * clase de la aplicación phone book
 * muestra el uso de conector de servidor
 */
public class PhoneBook extends Frame {

/*
 * puerto para la lectura de la agenda telefónica
 */
final int InPort = 7000;

/*
 * puerto para la grabación de la agenda telefónica
 */
final int OutPort = 7001;

/*
 * máxima cantidad de números telefónicos
 */
final int MaxNumbers = 25;

/*
 * una instancia de la lista
```

```
 */
List phoneList;

/*
 * instancia del botón
 */
Button addButton;
Button deleteButton;
Button findButton;
Button saveButton;

/*
 * instancia del campo de texto
 */
TextField nameField;
TextField phoneField;
TextField emailField;

/*
 * matriz de cadena para contener todos los números telefónicos
 */
String data[] = new String[MaxNumbers];
int NNumbers = 15;

/*
 * nombre del host remoto que sirve los datos
 */
String hostName;

/*
 * constructor
 * añade todos los componentes de la interfaz de usuario
 * y lee datos del conector
 * visualiza los nombres en la lista
 */
public PhoneBook (String host) {

      int i;

      setTitle ("Agenda telefónica");
      setLayout (new BorderLayout ());

      Panel buttonPanel = new Panel ();
      buttonPanel.setLayout (new FlowLayout ());
      buttonPanel.add (findButton = new Button ("Encontrar nombre"));
      buttonPanel.add (addButton = new Button ("Agregar"));
      buttonPanel.add (deleteButton = new Button ("Eliminar"));
      buttonPanel.add (saveButton = new Button ("Guardar"));

      Panel fieldPanel = new Panel ();
      fieldPanel.setLayout (new GridLayout (6, 1, 5, 5));
      fieldPanel.add (new Label ("Nombre"));
      fieldPanel.add (nameField = new TextField ("", 32));
      fieldPanel.add (new Label ("Número telefónico"));
      fieldPanel.add (phoneField = new TextField ("", 32));
      fieldPanel.add (new Label ("Dirección electrónica"));
      fieldPanel.add (emailField = new TextField ("", 32));

      phoneList = new List (5, false);

      Socket fs = null;
```

```
      hostName = host;
      if (hostName == null) hostName = "synapses";
        try {
                fs = new Socket (hostName, InPort);
        } catch (Exception e) {
                System.out.println ("Imposible abrir el conector");
                System.exit (1);
        }

      DataInputStream ds = null;

        try {
            ds = new DataInputStream (fs.getInputStream ());
      } catch (IOException e) {
            System.out.println (e);
                System.exit (1);
        }

      NNumbers = 0;
      for (i=0; i<MaxNumbers; i+=1) {
                try {
                        data[i] = ds.readLine ();
                        if (data[i] == null) break;
                  NNumbers += 1;
                } catch (IOException e) {
                        System.out.println ("Error detectado al leer
el archivo.");
                        break;
                }
            StringTokenizer st = new StringTokenizer (data[i], ":\t");
            phoneList.addItem (st.nextToken ());
      }
        try {
                fs.close ();
        } catch (IOException e) {
                System.out.println ("Error detectado al cerrar el
conector.");
        }

      add ("North", buttonPanel);
      add ("South", fieldPanel);
      add ("Center", phoneList);

      resize (300, 400);
      show ();
}

/*
 * manipulación de los eventos de botón y de selección en la lista
 */
public boolean action (Event evt, Object arg) {

      int i;
      StringTokenizer st;
      String dname;
      String tname;

      tname = nameField.getText ();
      if (evt.target == findButton || evt.target == nameField) {
            for (i=0; i<NNumbers; i+=1) {
```

```
                st = new StringTokenizer (data[i], ":\t");
                dname = st.nextToken ();
                if (contains (tname, dname)) {
                      phoneList.select (i);
                      nameField.setText (dname);
                      phoneField.setText (st.nextToken ());
                      emailField.setText (st.nextToken ());
                      break;
                }
          }
          return true;
    }

    if (evt.target == addButton) {
          if (NNumbers < MaxNumbers) {
                tname = nameField.getText ();
                dname = tname + ":" + phoneField.getText() +
                      ":" + emailField.getText();
                data[NNumbers] = dname;
                NNumbers += 1;
                phoneList.addItem (tname);
          }
          return true;
    }

    if (evt.target == deleteButton) {
          i = phoneList.getSelectedIndex ();
          phoneList.delItem (i);
          for (; i<NNumbers-1; i+=1) data[i] = data[i+1];
          NNumbers -= 1;
          nameField.setText ("");
          phoneField.setText ("");
          emailField.setText ("");
          return true;
    }

    if (evt.target == saveButton) {
          Socket fs = null;

            try {
                fs = new Socket (hostName, OutPort);
          } catch (Exception e) {
                System.out.println ("No se puede abrir el conector");
                System.exit (1);
          }
          PrintStream ps = null;
          try {
                ps = new PrintStream (fs.getOutputStream ());
          } catch (Exception e) {
                System.out.println (e);
                System.exit (1);
          }

          for (i=0; i<NNumbers; i+=1) {
                ps.println (data[i]);
              }
          try {
                fs.close ();
          } catch (IOException e) {
                System.out.println (e);
          }
```

```
            return true;
      }

      if (evt.target == phoneList) {
            i = phoneList.getSelectedIndex ();
            st = new StringTokenizer (data[i], ":\t");
            nameField.setText (st.nextToken ());
            phoneField.setText (st.nextToken ());
            emailField.setText (st.nextToken ());
            return true;
      }

      return false;
}

/*
 * el método contains que compara cadenas y retorna True cuando
 * alguna cadena está contenida en la otra
 *      String s1 y s2 son las cadenas que se comparan
 */
boolean contains (String s1, String s2) {

        int i;
        int l1 = s1.length ();
        int l2 = s2.length ();

        if (l1 < l2) {
                for (i=0; i<=l2-l1; i+=1)
                        if (s1.regionMatches (true, 0, s2, i, l1))
                                return true;
        }
        for (i=0; i<=l1-l2; i+=1)
                if (s2.regionMatches (true, 0, s1, i, l2))
                        return true;

        return false;
}

/**
 * punto de entrada de la aplicaión
 * el primer argumento es el nombre del host remoto
 * @param args - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      if (args.length == 1) new PhoneBook (args[0]);
      else new PhoneBook (null);
}
}

import java.net.*;
import java.io.*;

/*
 * lado servidor de la aplicación
 * crea  instancias de InServer y OutServer
 */
public class PhoneServer {

static final int OutPort = 7000;
static final int InPort = 7001;
```

```
/**
 * punto de entrada de la aplicación
 * @param args - argumentos de la línea de mandatos
 */
public static void main (String args[]) {

      OutServer outServer = new OutServer (OutPort);
      InServer inServer = new InServer (InPort);
}
}

/*
 * la clase crea un conector de servidor para escribir datos para el
cliente
 */
class OutServer implements Runnable {

/*
 * instancia de ServerSocket
 */
ServerSocket server = null;

/*
 * conector real usado para grabar
 */
Socket socket = null;

/*
 * para escribir se usa PrintStream
 */
PrintStream stream = null;
int thePort;

/*
 * se usa una trama separada para esperar la aceptación del conector
 */
Thread thread;

/**
 * el constructor arranca la trama
 * @param port - puerto
 */
public OutServer (int port) {

      thePort = port;
      thread = new Thread (this);
      thread.start ();
}

/*
 * la trama que espera la conexión  y graba los datos para el cliente
  */
public void run () {

      try {
            server = new ServerSocket (thePort);
      } catch (Exception e) {
            System.out.println (e);
            System.exit (1);
      }
```

```
      while (true) {
            try {
                  socket = server.accept ();
            } catch (Exception e) {
                  System.out.println (e);
                  System.exit (1);
            }
            try {
                  stream = new PrintStream (socket.getOutputStream ());
            } catch (Exception e) {
                  System.out.println (e);
                  System.exit (1);
            }

            FileInputStream fs = null;

            try {
                  fs = new FileInputStream ("PhoneBook.txt");
            } catch (Exception e) {
                  System.out.println (e);
                  System.exit (1);
            }
            DataInputStream ds = new DataInputStream (fs);
            while (true) {
                  try {
                        String s = ds.readLine ();
                        if (s == null) break;
                        stream.println (s);
                      } catch (IOException e) {
                        System.out.println (e);
                        break;
                  }
            }
            try {
                  fs.close ();
                  socket.close ();
            } catch (IOException e) {
                  System.out.println (e);
            }
      }
}
}

/*
 * la clase crea un conector de servidor para leer datos desde el
cliente
 */
class InServer implements Runnable {

/*
 * instancia de ServerSocket
 */
ServerSocket server = null;

/*
 * conector real usado para la lectura
 */
Socket socket = null;

/*
```

```
 * para leer se usa un DataInputStream
 */
DataInputStream stream = null;
int thePort;

/*
 * se crea una trama separada para esperar la aceptación del conector
 */
Thread thread;

/**
 * el constructor arranca la trama
 * @param port - puerto
 */
public InServer (int port) {

      thePort = port;
      thread = new Thread (this);
      thread.start ();
}

/*
 * la trama que espera la conexión y lee los datos desde el cliente
 */
public void run () {

      try {
            server = new ServerSocket (thePort);
      } catch (Exception e) {
            System.out.println (e);
            System.exit (1);
      }

      while (true) {
            try {
                  socket = server.accept ();
            } catch (Exception e) {
                  System.out.println (e);
                  System.exit (1);
            }
            try {
                  stream = new DataInputStream (socket.getInputStream
());
            } catch (Exception e) {
                  System.out.println (e);
                  System.exit (1);
            }

            FileOutputStream fs = null;

            try {
                  fs = new FileOutputStream ("PhoneBook.txt");
            } catch (Exception e) {
                  System.out.println (e);
                  System.exit (1);
            }
            PrintStream ds = new PrintStream (fs);
            while (true) {
                  try {
                        String s = stream.readLine ();
                        if (s == null) break;
```

```
                        ds.println (s);
                      } catch (IOException e) {
                        System.out.println (e);
                        break;
                  }
            }
            try {
                  fs.close ();
                  socket.close ();
            } catch (IOException e) {
                  System.out.println (e);
            }
      }
}
}
```

27.3 Prevenir el robo de applets

```
import java.awt.*;
import java.applet.Applet;
import java.awt.image.*;
import java.util.Date;

/**
 * Una clase que describe un objetivo (invasor)
 * contiene un invasor
 */
class Invader {

/*
 * posición y velocidad del invasor
 */
double x, y, dx, dy;

/*
 * posición y velocidad del misil del invasor
 * sólo se permite un misil por invasor
 */
int mx, my, mdy;

/*
 * imagen de un invasor
 */
Image img[] = new Image[4];

/*
 * inplay es True si el invasor está vivo
 */
boolean inplay;

/*
 * fired se pone en True cuando invasor dispara un misil
 */
boolean fired = false;

/*
 * state se usa para hacer ciclos entre las cuatro imágenes de un
invasor
```

```
 */
double state;

/*
 * value es el valor del resultado actual, depende de la velocidad
 */
int value;

/*
 * inicializa la posición y la velocidad del invasor
 */
void random (int speed, int w, int h) {

      x = 10 + (w-20)*Math.random ();
      y = 10 + ((h>>1)-20)*Math.random ();
      dx = (speed>>1) - speed*Math.random ();
      dy = (speed>>1) - speed*Math.random ();
      inplay = true;
      state = 3 * Math.random ();
      fired = false;
      mdy = 20;
      value = speed * 10;
}

/*
 * calcula la posición del invasor y del misil.
 * también dispara un misil al azar.
 * @param w ancho del panel
 * @param h altura del panel
 */
void compute (int w, int h) {

      if (x <= 0 || x > w) dx = -dx;
      if (y <= 0 || y > h>>1) dy = -dy;
      if (my > h-20) fired = false;
      if (fired) my += mdy;
      else my = 0;
      if (inplay && !fired && Math.random () > 0.99) {
            fired = true;
            mx = (int) x; my = (int) y+25;
      }
      x += dx; y += dy;
}

/*
 * pinta invasor y misil (si hubo disparo) i
 * @param g objeto gráfico de destino
 * @param obs     observador  asociado con este contexto gráfico
 */
public void paint (Graphics g, ImageObserver obs) {

      int whichImage;

      if (inplay) {
            whichImage = (int) state;
            g.drawImage (img[whichImage & 0x3], (int) x-25,
                  (int) y-25, obs);
            state += .25;
      }
      if (fired) {
            g.setColor (Color.green);
```

```
            g.drawLine ((int) mx, (int) my, (int) mx, (int) my-10);
      }
}

/*
 * verifica si los misiles del jugador han acertado con algún invasor
 * retorna True si el inavasor fue alcanzado
 * @param pmx     posición x del misil del jugador
 * @param pmy     posición y del misil del jugador
 */
boolean killer (int pmx, int pmy) {

      int deltaX, deltaY;

      if (!inplay) return false;
            deltaX = (int) Math.abs (x-pmx);
            deltaY = (int) Math.abs (y-pmy);
            if (deltaX < 20 && deltaY < 20) {
                  inplay = false;
                  return true;
            }
            return false;
}
}

/**
 * una clase para describir al jugador, muy similar al invasor
 */
class Player {

/*
 * posición del jugador
 */
int x, y=-100;

/*
 * posición del misil del jugador
 */
int mx, my, mdy = -20;

/*
 * dos imñagenes del jugador
 */
Image img1, img2;

/*
 * fired es True si el jugador ha disparado un misil
 * inplay es true si el juego no ha terminado
 */
boolean fired = false, inplay=true;

/*
 * método llamado cuando un jugador dispara un misil
 */
void fire () {

      if (fired || !inplay) return;
      mx = x; my = y;
      fired = true;
}
```

```
/*
 * calcula la siguiente posición del misil
 */
void compute () {

      if (my < 0) fired = false;
      if (fired) my += mdy;
      else my = y;
}

/**
 * pinta al jugador y al misil
 * @param g - objeto gráfico de destino
 * @param obs - observador
 */
public void paint (Graphics g, ImageObserver obs) {

      if (fired) {
            if (inplay) g.drawImage (img2, x-25, y, obs);
            g.setColor (Color.white);
            g.drawLine (mx, my, mx, my+10);
      } else if (inplay) g.drawImage (img1, x-25, y, obs);
}

/**
 * devuelve True si han matado al jugador
 * @param bmx, bmy - posición del misil enemigo
 */
boolean killer (int bmx, int bmy) {

      int dx, dy;

      if (!inplay) return false;
      dx = (int) Math.abs (x-bmx);
      dy = (int) Math.abs (y-bmy);
      if (dx < 20 && dy < 20) {
            return true;
      }
      return false;
}
}

/*
 * gran parte de la lógica del juego está aquí
 */
class Playfield extends Panel implements Runnable {

static final int PLAYER_HIT = 1;
static final int INVADER_HIT = 2;

InvaderApp invaderApp;

/*
 * cantidad de invasores en juego
 */
int NInvaders=0;

/*
 * máxima cantidad de invasores en juego
 */
final int MaxInvaders = 32;
```

```
/*
 * matriz de invasores
 */
Invader invaders[] = new Invader[MaxInvaders];
Player player;

/*
 * imagen fuera de pantalla para el uso de doble búfer
 */
Image offscreen;

/*
 * dimensión de la imagen gráfica fuera de pantalla
 */
Dimension psize;

/*
 * objeto gráfico asociado con la imagen fuera de pantalla
 */
Graphics offgraphics;

/*
 * trama de acción del juego
 */
Thread theThread;

/*
 * color del fondo del juego
 */
Color bgcolor = new Color (51, 0, 153);
int score, playerLives, playLevel;
Font font;

/**
 * el constructor guarda la instancia del applet
 * @param invaderApp - instancia del applet
 */
public Playfield (InvaderApp invaderApp) {

      this.invaderApp = invaderApp;
}

/*
 * trama de acción del juego
 */
public void run() {

      psize = size();
      offscreen = createImage (psize.width, psize.height);
      offgraphics = offscreen.getGraphics ();
      font = new Font ("TimesRoman", Font.BOLD, 18);
      offgraphics.setFont (font);

      while (true) {
            compute ();
            repaint ();
            try {
                  Thread.sleep(25);
            } catch (InterruptedException e) { }
      }
```

```
}

/*
 * calcula las nuevas posiciones de todos los objetos
 */
synchronized void compute () {

      for (int i=0; i<NInvaders; i+=1) {
            invaders[i].compute (psize.width, psize.height);
            if (invaders[i].killer (player.mx, player.my)) {
                  invaderApp.hit (INVADER_HIT);
                  player.fired = false;
                  score += invaders[i].value;
            }
            if (player.killer (invaders[i].mx, invaders[i].my)) {
                  invaderApp.hit (PLAYER_HIT);
                  invaders[i].fired = false;
                  playerLives -= 1;
                  if (playerLives < 1) player.inplay = false;
            }
      }
      player.compute ();
}

/**
 * reemplaza el método update por defecto
 * dibuja en la imagen fuera de pantalla y luego la copia en la
pantalla
 * @param g - objeto gráfico de destino
 */
public synchronized void update(Graphics g) {

      offgraphics.setColor (bgcolor);
      offgraphics.fillRect (0, 0, psize.width, psize.height);

      for (int i = 0 ; i < NInvaders ; i++)
            if (invaders[i].inplay) invaders[i].paint (offgraphics,
this);
      player.paint (offgraphics, this);

      offgraphics.setColor (Color.green);
      offgraphics.drawString ("Resultado", 10, 20);
      offgraphics.drawString (Integer.toString (score), 60, 20);
      offgraphics.drawString ("Nivel", psize.width>>1, 20);
      offgraphics.drawString (Integer.toString (playLevel),
            (psize.width>>1)+50, 20);
      offgraphics.drawString ("Vidas", psize.width-80, 20);
      offgraphics.drawString (Integer.toString (playerLives),
            psize.width-30, 20);
      if (playerLives < 1) offgraphics.drawString ("El juego ha
terminado",
            (psize.width>>1)-30, psize.height>>1);
      g.drawImage (offscreen, 0, 0, null);
}

/**
 * dispara el misil del jugador
 * @param evt - evento
 * @param x, y - posición del ratón
 */
public synchronized boolean mouseDown(Event evt, int x, int y) {
```

```
      player.fire ();
      return true;
}

/**
 * mueve la posición del jugador
 * @param evt - evento
 * @param x, y - posición del ratón
 */
public boolean mouseMove (Event evt, int x, int y) {

      player.x = x;
      player.y = psize.height-45;
      if (player.x < 20) player.x = 20;
      if (player.x > psize.width-20) player.x = psize.width-20;
      return true;
}

/*
 * arranca la trama del juego
 */
public void start() {

      theThread = new Thread (this);
      theThread.start ();
}

/*
 * detiene la trama del juego
 */
public void stop() {

      theThread.stop ();
}
}

/*
 * la clase del applet
 */
public class InvaderApp extends Applet {

/*
 * instancia del campo de juego
 */
Playfield panel;

/*
 * almacenamiento temporario de las imágenes
 */
Image img[] = new Image[4];

/*
 * la velocidad del juego
 * cantidad de invasores en esta secuencia
 */
int speed, NInvadersInPlay;

/*
 * puesta en True si pasa el control
 */
```

```
boolean canPlay;

/*
 * método llamado cuando se carga el applet
 * carga las imágenes
 */
public void init() {

      int i;
      MediaTracker tracker = new MediaTracker (this);
      canPlay = Validate();
      setLayout(new BorderLayout());

      panel = new Playfield (this);
      add("Center", panel);
      Panel p = new Panel();
      add("South", p);
      p.add(new Button("Nuevo juego"));

      showStatus ("Obtención de las imágenes de los invasores...");
      for (i=0; i<4; i+=1) {
            img[i] = getImage (getDocumentBase(), "T"+(i+1)+".gif");
            tracker.addImage (img[i], 0);
      }

      try {
            tracker.waitForID(0);
      } catch (InterruptedException e) { }

      for (i=0; i<panel.MaxInvaders; i+=1) {
            panel.invaders[i] = new Invader ();
            panel.invaders[i].inplay = false;
            panel.invaders[i].img[0] = img[0];
            panel.invaders[i].img[1] = img[1];
            panel.invaders[i].img[2] = img[2];
            panel.invaders[i].img[3] = img[3];
      }
      panel.player = new Player ();

      showStatus ("Obtención de las imágenes del jugador...");
      panel.player.img1 = getImage (getDocumentBase(), "Player1.gif");
      panel.player.img2 = getImage (getDocumentBase(), "Player2.gif");

      tracker.addImage (panel.player.img1, 1);
      tracker.addImage (panel.player.img2, 1);
      try {
            tracker.waitForID (1);
      } catch (InterruptedException e) { }
      showStatus ("¡Listo para jugar!");
}

/*
 * arranca la trama de acción
 */
public void start() {

      panel.start();
}

/*
 *  detiene la trama de acción
```

```
 */
public void stop() {

      panel.stop();
}

/*
 * manipula los eventos de botón
 * @param evt - evento
 * @param arg - objetivo
 */
public boolean action(Event evt, Object arg) {

      if ("Nuevo juego".equals(arg)) {
            speed = 10;
            panel.player.inplay = true;
            panel.playerLives = 3;
            panel.score = 0;
            panel.playLevel = 1;
            NInvadersInPlay = 2 * panel.playLevel + 1;
            panel.NInvaders = NInvadersInPlay;
            for (int i=0; i<panel.NInvaders; i+=1)
                  panel.invaders[i].random (speed,
                        panel.psize.width, panel.psize.height);

            play (getCodeBase(), "gong.au");
            if (NInvadersInPlay >= panel.MaxInvaders)
                  NInvadersInPlay = panel.MaxInvaders;
            return true;
      }
      return false;
}

/**
 * reproduce el sonido adecuado cuando un misil alcanza algún objetivo
 * @param which - determina qué sonido reproducir
 */
public void hit (int which) {

      switch (which) {
            case Playfield.INVADER_HIT:
            NInvadersInPlay -= 1;
            if (NInvadersInPlay < 1) {
                  play (getCodeBase(), "gong.au");
                  panel.playLevel += 1;
                  NInvadersInPlay = 2 * panel.playLevel + 1;
                  speed += 4;
                  panel.NInvaders = NInvadersInPlay;
                  for (int i=0; i<panel.NInvaders; i+=1)
                  panel.invaders[i].random (speed,
                        panel.psize.width, panel.psize.height);
            } else {
                  play (getCodeBase(), "drip.au");
            }
            break;

            case Playfield.PLAYER_HIT:
            play (getCodeBase(), "doh2.au");
            break;
      }
}
```

```
/**
 * algoritmo arbitrario de encriptación para verificación
 * @param minute - minuto actual
 * @param key - clave usada en el lanzamiento CGI
 */
String Crypt (int minute, int key) {

      int i;
      int code;
      String crypt = "";

      code = (key << 1) + 1;
      for (i=0; i<minute; i+=1) {
            if ((code & 0x80000000) != 0) {
                  code <<= 1;
                  code |= 1;
            } else code <<= 1;
            code += key;
      }
      for (i=0; i<8; i+=1) {
            crypt += code & 0xf;
            code >>= 4;
      }
      return crypt;
}

/*
 * retorna True si la verificación es correcta
 */
boolean Validate () {

      Date date = new Date ();
      String param = getParameter ("key");
      System.out.println (param);

      int minute = date.getMinutes ();

      if (param.equals(Crypt (minute, 0xc0defeed))) return true;
      minute -= 1;
      if (minute < 0) minute = 59;
      if (param.equals(Crypt (minute, 0xc0defeed))) return true;
      minute -= 1;
      if (minute < 0) minute = 59;
      if (param.equals(Crypt (minute, 0xc0defeed))) return true;
      return false;
}
}


import java.util.Date;

/*
 * una aplicación CGI que lanza el applet
 */
public class Launch {

/**
 * algoritmo de encriptación para generar un valor
 * @param minute - minuto actual
 * @param key - clave arbitraria para encriptación
```

```
 */
static String Crypt (int minute, int key) {

      int i;
      int code;
      String crypt = "";

      code = (key << 1) + 1;
      for (i=0; i<minute; i+=1) {
            if ((code & 0x80000000) != 0) {
                  code <<= 1;
                  code |= 1;
            } else code <<= 1;
            code += key;
      }
      for (i=0; i<8; i+=1) {
            crypt += code & 0xf;
            code >>= 4;
      }
      return crypt;
}

/**
 * punto de entrada de la aplicación
 * genera un valor encriptado y lo pasa al applet como parámetro
 * @param args - argumentos de la línes de mandatos
 */
public static void main (String args[]) {

      Date date = new Date ();
      String param;

      int minute = date.getMinutes ();

      param = Crypt (minute, 0xc0defeed);

      System.out.println
("<html>\n<head>\n<title>Invader</title>\n</head>");
      System.out.println ("<applet code=\"InvaderApp.class\" ");
      System.out.println ("width=600 height=400>\n");
      System.out.println ("<param name=key value=\""+param+"\">");
      System.out.println ("</applet><hr size=4></html>");
}
}
```

28 Referencia a la API de Java.

Debido a la gran extensión de todos los paquetes, clases y métodos de la API, se adjunta un programa HELP de Windows en el que pueden observarse todos los detalles. Esto aparecerá en los discos de este manual.

29 Bibliografía

- Java el lenguaje de Internet
  Sergio Rios Aguilar
- Como se hace con Java
  Madhu Siddalingaiah
  Stephen D.Lockwood
- The Java Tutorial
- El lenguaje de programación Java
  Arnold Gosling
- Fundamentos de programación
  Luis Joyanes
- Turbo C/C++
  Herbert Schildt

30 Índice analítico

A

abstract 38, 48
abstracta 37, 38
ADA 95 40
algoritmo ... 5, 6, 7, 15, 17, 18, 23, 60, 72, 78, 84, 175, 176
Algoritmos 5
ámbito 9, 35, 37, 48, 53
API 4, 57, 177
applet 56, 58, 64, 65, 66, 82, 83, 84, 85, 86, 87, 91, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 106, 107, 111, 112, 114, 115, 116, 118, 124, 125, 130, 131, 132, 133, 135, 136, 140, 141, 149, 154, 156, 158, 166, 171, 173, 176, 177
Applets 3, 64
archivo .. 3, 21, 22, 66, 74, 75, 77, 79, 80, 88, 89, 136, 137, 138, 143, 144, 146, 147, 160
ArithmeticException 63
ArrayIndexOutOfBoundsException 63
arrays 45, 47, 54, 77
Arrays 1, 3, 19, 53
ASCII 47, 50
atributos 26, 27, 29, 57

B

base de datos 22
bifurcación 1, 14
bits 46, 71
boolean 8, 46, 48, 51, 56, 69, 70, 80, 81, 106, 107, 108, 110, 112, 116, 117, 118, 119, 124, 127, 132, 134, 138, 139, 143, 147, 149, 150, 151, 152, 155, 157, 161, 162, 167, 168, 169, 172, 173, 174, 175
booleano 45, 46, 51
byte 46, 48

C

C 7, 15, 18, 24, 35, 39, 40, 43, 47, 49, 50, 51, 53, 54, 61, 72, 124, 126, 177
C++ 35, 39, 40, 43, 47, 49, 50, 51, 54, 72, 177
case ... 48, 52, 66, 73, 76, 88, 109, 110, 113, 119, 120, 121, 122, 123, 129, 143, 147, 149, 158, 175
catch .. 3, 49, 57, 58, 62, 63, 69, 70, 74, 75, 76, 77, 78, 79, 80, 81, 89, 91, 104, 112, 134, 137, 138, 139, 143, 146, 147, 154, 156, 157, 160, 161, 162, 164, 165, 166, 171, 173, 174
CD-ROM 43
char 8, 46, 47, 48, 49, 100
clase .. 2, 25, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 41, 42, 45, 47, 48, 54, 57, 58, 60, 62, 63, 65, 66, 76, 79, 80, 82, 84, 85, 86, 87, 88, 91, 92, 93, 96, 98, 100, 102, 106, 107, 114, 115, 118, 119, 124, 125, 127, 130, 132, 133, 136, 140, 142, 144, 148, 149, 151, 156, 159, 163, 165, 166, 168, 173
class. 31, 34, 36, 38, 41, 49, 56, 57, 58, 65, 68, 69, 70, 71, 72, 74, 75, 77, 78, 79, 80, 81, 82, 84, 85, 87, 88, 91, 93, 94, 96, 97, 98, 100, 102, 106, 107, 115, 116, 119, 124, 125, 127, 130, 133, 136, 140, 142, 144, 145, 148, 150, 151, 153, 156, 159, 163, 165, 166, 168, 170, 173, 176, 177
ClassCastException 63
Client Pull 43
comentarios 50, 71
compilación 40, 42, 44, 54
compilador 7, 54, 55, 77
Compiladores 7
const 49
constantes 9, 19, 54, 87
continue 49, 53, 108, 111
CPU 59, 60
CPUs 36

D

dato 8
default 49, 52, 129
demonio 60
do 49, 52, 56, 69, 70, 81
double ... 31, 32, 33, 34, 35, 38, 46, 49, 68, 70, 81, 82, 83, 84, 85, 92, 93, 94, 127, 128, 129, 134, 135, 150, 167

E

Ejecución 5
else 12, 49, 52, 76, 88, 89, 90, 110, 112, 116, 124, 125, 129, 132, 134, 135, 151, 152, 158, 163, 168, 169, 175, 176
Encapsulación 41
encapsulamiento 2, 36
Enlace dinámico 41
escape 3, 50
EstaciónTrabajo 36
Euclides 5
excepciones 3, 48, 62, 63, 74
expresión 8, 10, 11, 44, 53
Expresiones 1, 10, 11
extends .. 36, 38, 49, 58, 65, 82, 85, 88, 93, 94, 97, 98, 100, 102, 107, 116, 119, 124, 125, 127, 130, 133, 136, 140, 142, 145, 148, 153, 156, 159, 170, 173

F

FACTORIAL 15, 16, 17
final . 12, 49, 64, 68, 69, 70, 72, 75, 77, 81, 82, 84, 85, 86, 87, 88, 89, 92, 98, 107, 114, 116, 136, 140, 142, 153, 159, 163, 170
finally 3, 49, 63
float 46, 49, 100
flujo 3, 14, 51, 53, 55, 56
for 49, 52, 53, 54, 55, 57, 72, 73, 77, 78, 80, 83, 84, 86, 87, 88, 89, 91, 92, 93, 94, 99, 100, 108, 110, 111, 114, 115, 116, 117, 137, 138, 139, 141, 154, 155, 156, 157, 158, 160, 161, 162, 171, 172, 173, 174, 175, 176
friendly 37
FTP 41
funciones 1, 3, 16, 17, 31, 42, 80, 82, 124

G

George Boole 11
goto 40, 49

H

Hardware 5
herencia 2, 25, 40
Herencia 2, 35, 41
hilos 3, 40, 42, 55, 56, 57, 60, 61, 101
HTML 3, 43, 50, 64, 65, 66
HTTP 41

I

if 12, 49, 52, 56, 58, 72, 73, 74, 75, 76, 77, 78, 79, 80, 84, 85, 87, 89, 90, 91, 92, 97, 99, 103, 108, 110, 111, 112, 113, 114, 116, 117, 118, 124, 125, 127, 128, 129, 132, 134, 135, 137, 138, 139, 141, 145, 147, 148, 149, 151, 152, 153, 154, 155, 157, 158, 160, 161, 162, 163, 164, 166, 167, 168, 169, 170, 171, 172, 174, 175, 176
implements 49, 58, 102, 107, 153, 163, 165, 170
import 49, 56, 58, 64, 68, 69, 71, 72, 74, 75, 77, 79, 80, 82, 84, 87, 91, 96, 98, 100, 102, 106, 115, 119, 124, 130, 132, 136, 140, 142, 144, 145, 149, 158, 159, 163, 166, 176
IncompatibleClassChangeException 63
instanceof 49, 124, 147
instancias 34, 86, 88, 94, 97, 136, 145, 148, 163
int 31, 34, 35, 38, 46, 49, 53, 54, 55, 57, 70, 72, 73, 74, 75, 77, 78, 79, 80, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 96, 97, 98, 99, 100, 101, 102, 103, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 127, 130, 131, 133, 135, 136, 137, 138, 139, 140, 141, 142, 144, 145, 146, 148, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176
integer 8, 70
interfaces 40, 45
InternalException 63
intérprete 7, 42, 68
Ínterpretes 7

J

James Gosling 39
Java ... 2, 3, 4, 7, 24, 30, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 59, 60, 61, 62, 63, 64, 66, 67, 68, 72, 82, 100, 106, 123, 132, 140, 144, 149, 177
JAVA 2, 27, 30

L

Length 55
lenguajes 1, 7, 8, 12, 16, 27, 39, 40, 42, 44, 53, 62
librerías 42
long 46, 49, 57, 71

M

MAX_PRIORITY 60
mensaje 2, 25, 28, 29
mensajes 25, 28, 140
métodos ... 2, 25, 26, 27, 31, 32, 34, 35, 36, 37, 38, 45, 47, 54, 61, 65, 66, 67, 68, 80, 82, 95, 114, 136, 177
MIN_PRIORITY 60
mod 10, 11
mohamed al-Khowârizmî 5
multihilo 3, 56, 57

N

native 49
NegativeArraySizeException 64
new .. 32, 33, 49, 54, 55, 56, 58, 59, 69, 70, 71, 73, 74, 76, 77, 78, 79, 80, 81, 83, 84, 86, 87, 88, 89, 90, 91, 92, 94, 95, 97, 98, 99, 100, 101, 102, 103, 104, 107, 109, 110, 111, 112, 113, 116, 117, 118, 119, 123, 124, 125, 126, 127, 129, 131, 132, 133, 134, 135, 137, 138, 139, 140, 141, 142, 143, 146, 147, 148, 149, 150, 153, 154, 155, 156, 157, 159, 160, 161, 162, 163, 164, 165, 166, 167, 170, 171, 172, 173, 174, 175, 176
NoClassDefFoundException 64
null 45, 50, 52, 58, 59, 74, 75, 76, 77, 79, 89, 102, 103, 104, 108, 112, 137, 139, 143, 146, 147, 155, 160, 162, 163, 164, 165, 166, 172
NullPointerException 55, 64

O

objeto 2, 10, 25, 26, 27, 28, 29, 31, 32, 33, 34, 36, 38, 41, 45, 54, 55, 62, 63, 82, 85, 86, 88, 93, 94, 95, 97, 99, 100, 102, 104, 115, 117, 132, 134, 141, 142, 145, 151, 152, 153, 154, 168, 169, 170, 171
operadores 10, 11, 40, 47, 51
ordenar 18
orientado a objetos 24, 30
OutOfMemoryException 64

P

package 49
parámetros 4, 17, 29, 33, 87, 105
Polimorfismo 41
preprocesador 40
private 37, 49, 78, 110, 128, 129, 142, 145
procedimiento 17, 18
procedimientos 1, 16, 17, 25, 27
programa ... 1, 3, 5, 7, 9, 13, 14, 17, 22, 35, 37, 41, 48, 50, 53, 54, 56, 57, 67, 68, 72, 74, 177
programación 15
programas 1, 4, 5, 7, 16, 35, 42, 43, 44, 49, 50, 55, 149
protected 37, 49
pseudocódigo 12
public 31, 33, 34, 35, 36, 37, 41, 49, 56, 57, 58, 59, 65, 68, 69, 70, 71, 72, 74, 75, 77, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 107, 111, 112, 113, 115, 116, 117, 118, 119, 123, 124, 125, 130, 131,

132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 151, 152, 154, 155, 156, 157, 158, 159, 160, 161, 163, 164, 165, 168, 169, 171, 172, 173, 174, 175, 176

R

real 8, 25, 26, 29, 44, 101, 163, 165
recolector 35, 40, 42
rectangulo 31, 32, 33, 34, 35, 36
registro 21
return 31, 35, 38, 49, 53, 78, 80, 81, 82, 91, 106, 107, 108, 110, 113, 116, 118, 120, 121, 122, 123, 124, 125, 127, 128, 129, 132, 134, 135, 138, 139, 143, 146, 147, 148, 149, 151, 152, 153, 155, 157, 161, 162, 163, 168, 169, 170, 172, 174, 175, 176

S

Server Push 43
setDaemon 60
setPriority 60
short 46, 49
software 24, 31
Software 5
static 34, 41, 49, 56, 57, 68, 69, 70, 71, 72, 74, 75, 77, 78, 79, 80, 81, 83, 86, 87, 88, 89, 91, 95, 98, 99, 101, 104, 107, 112, 115, 116, 118, 123, 125, 132, 135, 139, 141, 148, 153, 163, 170, 176
string 8, 71
String 3, 41, 53, 54, 55, 56, 57, 68, 69, 70, 71, 72, 74, 75, 76, 77, 78, 79, 80, 81, 83, 86, 89, 91, 95, 96, 98, 99, 100, 101, 104, 118, 123, 124, 125, 127, 131, 132, 134, 135, 137, 138, 139, 140, 141, 142, 143, 144, 146, 147, 148, 159, 160, 161, 162, 163, 164, 166, 175, 176
StringBuffer 3, 54
subclases 36, 38, 62
subprogramas 9, 17
Sun Microsystem 39
Sun MicroSystems 43
super 38, 49, 57, 93, 145, 148
Superclase 36
switch 49, 52, 73, 76, 88, 109, 113, 119, 120, 121, 129, 143, 147, 149, 158, 175
Symantec 177
synchronized 49, 110, 112, 113, 154, 155, 171, 172

T

TCP/IP 41
this 49, 58, 102, 103, 106, 107, 112, 131, 133, 146, 148, 154, 155, 156, 164, 165, 171, 172, 173
thread 57, 58, 102, 103, 104, 164, 165
throw 49, 62
Throwable 62, 146
throws 49
Traductores 1, 7
trasient 49
try 3, 49, 57, 58, 63, 69, 70, 74, 75, 76, 77, 78, 79, 80, 81, 89, 91, 104, 112, 134, 137, 139, 143, 146, 147, 154, 156, 157, 160, 162, 164, 165, 166, 171, 173, 174

U

Unicode 3, 46, 47, 50

V

validación 5
variables 1, 3, 9, 19, 25, 27, 30, 34, 42, 44, 45, 47, 48, 53
vectores 1, 19, 23
Visual Café 177
void 31, 34, 35, 36, 41, 49, 56, 57, 58, 59, 65, 68, 69, 70, 71, 72, 74, 75, 77, 78, 79, 81, 82, 83, 85, 86, 88, 89, 93, 94, 95, 97, 98, 99, 100, 101, 102, 103, 104, 109, 110, 111, 112, 113, 115, 117, 118, 123, 124, 125, 127, 128, 129, 131, 132, 133, 135, 139, 141, 142, 143, 145, 146, 147, 148, 149, 150, 151, 152, 154, 155, 156, 157, 158, 163, 164, 165, 167, 168, 169, 171, 172, 173, 174, 175, 176
volatile 49

W

Web 43, 58, 65, 66
while 49, 52, 56, 58, 68, 69, 70, 74, 76, 78, 79, 81, 89, 104, 112, 113, 146, 154, 164, 165, 166, 171
World Wide Web 2, 43