Programas

Estructura de un programa simple

Los programas más simples escritos en lenguajes imperativos suelen realizar tres tareas de forma secuencial:

- Entrada de datos
- Procesamiento de los datos
- Salida de resultados

El punto de entrada de un programa en Java es la función `main`:

```
public static void main (String[] args)
{
    Declaraciones y sentencias escritas en Java
}
```

En realidad, Java es un lenguaje de programación orientada a objetos y todo debe estar dentro de una clase, incluida la función `main`, tal como muestra el siguiente programa

```
public class MiPrimerPrograma
{
  public static void main (String args[])
  {
    System.out.println("Mensaje por pantalla”);
  }
}
```

- Las llaves `{}` delimitan bloques en Java (conjuntos de elementos de un programa).

- La máquina virtual Java ejecuta el programa invocando a la función `main`.

Mostrar resultados con la función System.out.println

La función `System.out.println` nos permite mostrar una línea de texto en la pantalla cuando ejecutamos el programa:

```
int edad = 25;
System.out.println("Tengo " + edad + " años");

final double pi = 3.1415927;
System.out.println("Valor de PI = " + pi);
```

- En la función `println` se suele utilizar el operador + para concatenar cadenas de caracteres.
- Cualquier cosa en Java se puede convertir en una cadena.

Obtener datos de entrada a través de los parámetros de main

La función `main` puede recibir parámetros:

```
public class Eco
{
  public static void main (String[] args)
  {
    System.out.println(args[0]);
    System.out.println(args[1]);
  }
}
```

Si ejecutamos el programa `Eco`

```
java Eco dato1 dato2
```

obtenemos como salida

```
dato1
dato2
```

- Los parámetros de la función main son de tipo `String`, por lo que deberemos convertir las cadenas en datos del tipo deseado.
- Con `args.length` podemos saber cuántos parámetros se le han pasado a nuestro programa.

Leer datos desde el teclado en Java

En Java, la realización de operaciones de entrada de datos no es inmediata, por lo que utilizaremos una clase auxiliar que se encargará de realizar las operaciones necesarias:

```
b = TextIO.getByte();    // byte
s = TextIO.getShort();   // short
i = TextIO.getInt();     // int
k = TextIO.getLong();    // long

x = TextIO.getFloat();   // float
y = TextIO.getDouble();  // double

a = TextIO.getBoolean(); // boolean

c = TextIO.getChar();    // char

s = TextIO.getln();      // String
```

Internamente, la implementación de la clase auxiliar `TextIO` realiza algo similar a lo siguiente:

```
InputStreamReader input;
BufferedReader lector;
String cadena;

// Secuencia de bytes → Secuencia de caracteres
input = new InputStreamReader(System.in);
// Secuencia de caracteres → Secuencia de líneas
lector = new BufferedReader(input);

try {
    cadena = lector.readLine();
} catch (Exception e) {
    cadena = "";
}

// Y ahora hacemos lo que queramos con la cadena
```

Cuando estudiemos el uso de ficheros en Java comprenderemos exactamente qué es lo que se hace al leer datos desde el teclado.

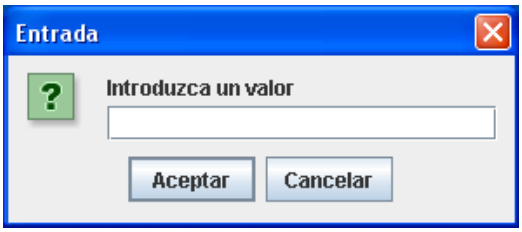
E/S con interfaces gráficas de usuario (GUIs): JOptionPane

La biblioteca de clases estándar de Java incluye una amplia gama de componentes para la construcción de interfaces gráficas de usuario.

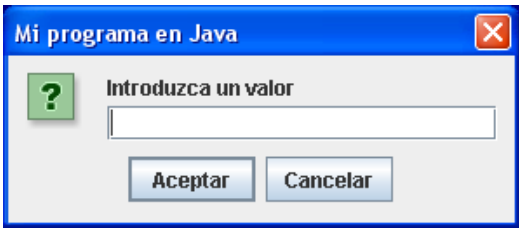
El componente `javax.swing.JOptionPane` se puede emplear para obtener datos de entrada y mostrar mensajes de salida:

Entrada de datos con showInputDialog

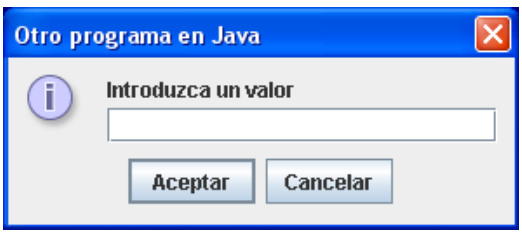
```
String entrada;

entrada = JOptionPane.showInputDialog
          ( "Introduzca un valor" );
```




```
entrada = JOptionPane.showInputDialog ( null,
            "Introduzca un valor",
            "Mi programa en Java",
            JOptionPane.QUESTION_MESSAGE );
```

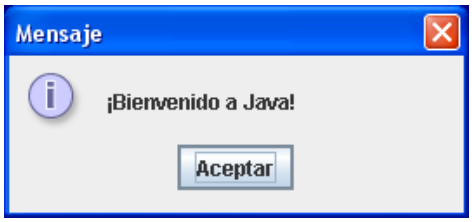



```
entrada = JOptionPane.showInputDialog ( null,
            "Introduzca un valor",
            "Otro programa en Java",
            JOptionPane.INFORMATION_MESSAGE );
```

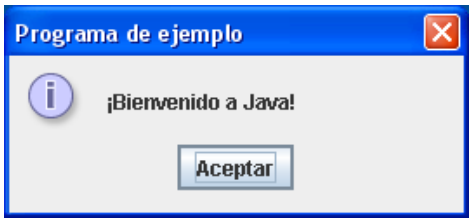


Salida de datos con showMessageDialog

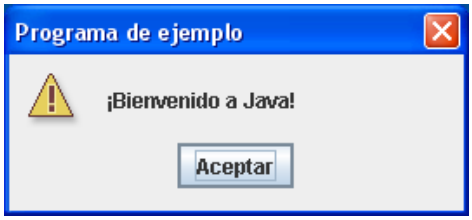
```
JOptionPane.showMessageDialog ( null,
            “¡Bienvenido a Java!” );
```



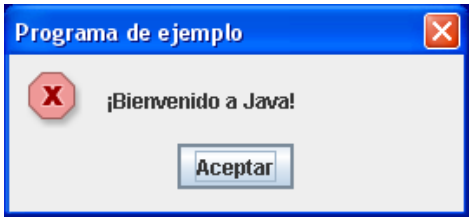

```
JOptionPane.showMessageDialog ( null,
            “¡Bienvenido a Java!”,
            “Programa de ejemplo”,
            JOptionPane.INFORMATION_MESSAGE );
```




```
JOptionPane.showMessageDialog ( null,
            “¡Bienvenido a Java!”,
            “Programa de ejemplo”,
            JOptionPane.WARNING_MESSAGE );
```




```
JOptionPane.showMessageDialog ( null,
            “¡Bienvenido a Java!”,
            “Programa de ejemplo”,
            JOptionPane.ERROR_MESSAGE );
```



Cuando se emplea `JOptionPane` es aconsejable llamar a `System.exit(0);` para terminar la ejecución del programa.

Ejemplos

Calificación final de la asignatura

```
public class Asignatura
{
  public static void main (String args[])
  {
    // Declaraciones
    // Constante
    final double PORCENTAJE_LABORATORIO = 0.5;
    // Variables
    String nombre;
    double notaExamen;
    double notaLaboratorio;
    double notaFinal;
    // Entrada de datos
    nombre = args[0];
    notaExamen = Double.parseDouble(args[1]);
    notaLaboratorio = Double.parseDouble(args[2]);
    // Cálculo
    notaFinal =
        (1-PORCENTAJE_LABORATORIO)*notaExamen
        + PORCENTAJE_LABORATORIO*notaLaboratorio;
    // Salida de resultados
    System.out.println (nombre
        + " ha obtenido una nota final de "
        + notaFinal);
  }
}
```

Ejecución del programa

```
java Asignatura Pepe 6 9
```

```
Pepe ha obtenido una nota final de 7.5
```

```
java Asignatura “Juan Nadie” 8 9.5
```

```
Juan Nadie ha obtenido una nota final de 8.75
```

Ejemplos

Año bisiesto

Programa para comprobar si un año es bisiesto o no:
Un año es bisiesto si es divisible por 4 pero no por 100,
o bien es divisible por 400.

```
public class Bisiesto
{
  public static void main (String args[])
  {
    // Declaración de variables

    int     year;
    boolean bisiesto;

    // Entrada de datos

    year = Integer.parseInt(args[0]);

    // Cálculos

    bisiesto = ((year%4==0) && (year%100!=0))
             || (year%400==0);

    // Salida de resultados

    System.out.println(bisiesto);
  }
}
```

Ejecución del programa

```
java Bisiesto 2004
    true
java Bisiesto 2005
    false
java Bisiesto 2000
    true
java Bisiesto 2100
    false
```

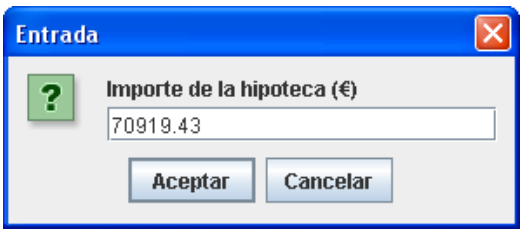
Ejemplos

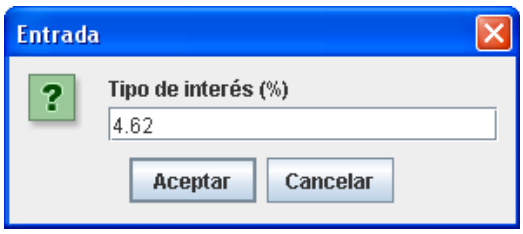
Cuota de una hipoteca

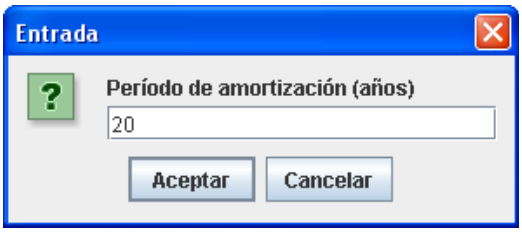
```
import javax.swing.JOptionPane;

public class Hipoteca
{
  public static void main (String args[])
  {
    double cantidad; // en euros
    double interes;  // en porcentaje (anual)
    int    tiempo;   // en años
    double cuota;    // en euros (mensual)
    double interesMensual; // en tanto por uno
    String entrada;        // variable auxiliar

    // Entrada de datos

    entrada = JOptionPane.showInputDialog
                ("Importe de la hipoteca (€)");
    cantidad = Double.parseDouble(entrada);

    entrada = JOptionPane.showInputDialog
                ("Tipo de interés (%)");
    interes = Double.parseDouble(entrada);

    entrada = JOptionPane.showInputDialog
                ("Período de amortización (años)");
    tiempo = Integer.parseInt(entrada);

    // Cálculo de la cuota mensual

    interesMensual = interes/(12*100);

    cuota = (cantidad*interesMensual)
      / (1.0–1.0/Math.pow(1+interesMensual,tiempo*12));

    cuota = Math.round(cuota*100)/100.0;

    // Resultado

    JOptionPane.showMessageDialog
       (null, "Cuota mensual = "+cuota+"€" );

    System.exit(0);
  }
}
```

Ejecución del programa

1. Datos de entrada







2. Cálculo de la cuota mensual

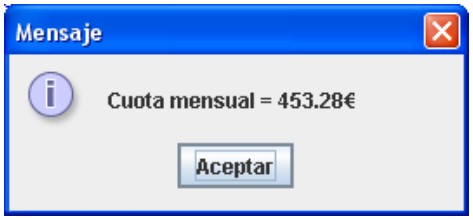
$$interésMensual\ (d) = interésAnual\ /\ 12$$

$$cuotaMensual = \frac{cantidad \times d}{1 - \frac{1}{(1+d)^{años \times 12}}}$$

Redondeo

Operación	Descripción	Resultado
`100*cuota`	Cuota en céntimos de euro	`double`
`Math.round(…)`	Redondeo al entero más cercano	`int`
`… / 100.0`	Cuota en euros (con céntimos)	`double`

3. Resultado final



Comentarios

Los comentarios sirven para incluir aclaraciones en el código.

Java permite dos tipos de comentarios:

```
// Comentarios de una línea
/* Comentarios de varias líneas */
```

- Es bueno incluir comentarios que expliquen lo que hace el programa y sus características claves (p.ej. autor, fecha, algoritmos utilizados, estructuras de datos, peculiaridades…).

```
// Cálculo del MCD
// usando el algoritmo de Euclides
// © Fernando Berzal, 2004
```

- Los comentarios nunca han de limitarse a decir en lenguaje natural lo que ya está escrito en el código: Jamás se utilizarán para "parafrasear" el código y repetir lo que es obvio.

```
✗ i++;       // Incrementa el contador
```

- Los comentarios han de aclarar; esto es, ayudar al lector en las partes difíciles (y no confundirle). Si es posible, escriba código fácil de entender por sí mismo: cuanto mejor lo haga, menos comentarios necesitará.

```
✗ int mes;   // Mes
✓ int mes;   // Mes del año (1..12)
```

```
✗ ax = 0x723;   /* RIP L.v.B. */
```

NB: Beethoven murió en 1827 (0x723 en hexadecimal).

Sangrías

Conviene utilizar espacios en blanco o separadores para delimitar el ámbito de las estructuras de control de nuestros programas.

Líneas en blanco

Para delimitar claramente los distintos segmentos de código en nuestros programas dejaremos líneas en blanco entre ellos.

Identificadores

Los identificadores deben ser descriptivos (reflejar su significado).

- ✗ `p, i, s...`
- ✓ `precio, izquierda, suma...`

Declaraciones

- Usualmente, declararemos una única variable por línea.
- Nunca mezclaremos en una misma línea la declaración de variables que sean de distintos tipos o que se utilicen en el programa para distintos fines.

Constantes

- Se considera una mala costumbre incluir literales de tipo numérico (“números mágicos”) en medio del código. Se prefiere la definición de constantes simbólicas (declaraciones con `final`).

Expresiones

- Uso de paréntesis: Aunque las normas de precedencia de los operadores vienen definidas en el lenguaje, no abusaremos de ellas. Siempre resulta más fácil interpretar una expresión si ésta tiene los paréntesis apropiados. Además, éstos eliminan cualquier tipo de ambigüedad.

- Uso de espacios en blanco: Resulta más fácil leer una expresión con espacios que separen los distintos operadores y operandos involucrados en la expresión.

```
a%x*c/b-1    →   ( (a%x) * c ) / b - 1
```

- Expresiones booleanas: Es aconsejable escribirlas como se dirían en voz alta.

```
!(bloque<actual)   →   ( bloque >= actual )
```

- Expresiones complejas:
  Es aconsejable dividirlas para mejorar su legibilidad

- Claridad:
  Siempre buscaremos la forma más simple de escribir una expresión.

```
✗ key = key >> ( bits – ((bits>>3)<<3));

✓ key >>= bits & 0x7;
```

- Conversiones de tipo (castings):
  Evitaremos las conversiones implícitas de tipo. Cuando queramos realizar una conversión de tipo, lo indicaremos explícitamente.

```
i = (int) f;
```

- Siempre se han de evitar los efectos colaterales (modificaciones no deseadas pueden afectar a la ejecución del programa sin que nos demos cuenta de ello).

IDEA CLAVE

Escribimos código para que lo puedan leer otras personas,
no sólo para que lo traduzca el compilador (si no fuese así,
podríamos seguir escribiendo nuestros programas en binario).

- No comente el código “malo” (uso de construcciones extrañas, expresiones confusas, sentencias poco legibles...): Reescríbalo.

- No contradiga al código: Los comentarios suelen coincidir con el código cuando se escriben, pero a medida que se corrigen errores y el programa evoluciona, los comentarios suelen dejarse en su forma original y aparecen discrepancias. Si cambia el código, asegúrese de que los comentarios sigan siendo correctos.

El código bien escrito
es más fácil de leer, entender y mantener
(además, seguramente tiene menos errores)

Errores de programación

Errores sintácticos

Errores detectados por el compilador en tiempo de compilación.

Errores semánticos

Sólo se detectan en tiempo de ejecución: Causan que el programa finalice inesperadamente su ejecución (p.ej. división por cero) o que el programa proporcione resultados incorrectos.