Aprende Java con Ejercicios

Un manual para aprender a programar en Java desde cero, con multitud de ejemplos y más de 200 ejercicios resueltos

Luis José Sánchez González

Este libro está a la venta en http://leanpub.com/aprendejava

Esta versión se publicó en 2016-01-25

ISBN 978-84-608-2372-8

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2014 - 2016 Luis José Sánchez González

Índice general

Sobre este libro

"Aprende Java con Ejercicios" es un manual práctico para aprender a programar en Java desde cero.

No es necesario tener conocimientos previos de programación. La dificultad del libro es gradual,

empieza con conceptos muy básicos y ejercicios muy sencillos y va aumentando en complejidad y dificultad a medida que avanzan los capítulos.

La práctica hace al maestro. Como de verdad se aprende a programar es programando desde el principio y esa es la filosofía que sigue este libro. En cada capítulo se explican una serie de conceptos de programación y se ilustran con ejemplos. Al final de cada uno de los capítulos se plantean ejercicios de diferente dificultad.

Este libro contiene más de 200 ejercicios. Tanto las soluciones a los ejercicios como los ejemplos están disponibles en GitHub: https://github.com/LuisJoseSanchez/aprende-java con-ejercicios y tanto los unos como los otros siguen los estándares de programación de Google para el código fuente escrito en el lenguaje de programación Java[1].

"Aprende Java con Ejercicios" es un libro hecho casi a medida de la asignatura *"Progra mación"* que forma parte del currículo del primer curso de los ciclos formativos DAW (Desarrollo de Aplicaciones Web) y DAM (Desarrollo de Aplicaciones Multiplataforma) pero igualmente puede ser utilizado por estudiantes de Ingeniería Informática, Ingenie ría de Telecomunicaciones o Ciencias Matemáticas en la asignatura de *"Programación"* de los primeros cursos.

Si tienes alguna duda o quieres hacer algún comentario en relación a este libro, tienes a tu disposición el grupo de Google https://groups.google.com/d/forum/aprende-java-con ejercicios. Cualquier sugerencia para mejorar este manual será bienvenida.

[1] http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

Sobre el autor

Luis José Sánchez González es Ingeniero Técnico en Informática de Gestión por la Universidad de Málaga (España) y funcionario de carrera de la Junta de Andalucía desde 1998. En su trabajo de profesor de Informática combina sus dos pasiones: la enseñanza y la programación.

En el momento de publicar este libro, es profesor del **I.E.S. Campanillas (Málaga)** e imparte clases en el **Ciclo Superior de Desarrollo de Aplicaciones Web**.

Puedes ponerte en contacto con el autor mediante la dirección de correo electró nico luisjoseprofe@gmail.com o mediante LinkedIn (https://es.linkedin.com/pub/luis-josé sánchez/86/b08/34).

Centros en los que se usa este libro

- IES Campanillas, Málaga (España)
- IES Polígono Sur, Sevilla (España)
- IES Murgi, El Ejido - Almería (España)
- IES Punta del Verde, Sevilla (España)

Si eres profesor y utilizas este libro para enseñar Java, pídele al autor que añada en esta sección el nombre de tu centro educativo o empresa, escribiendo un correo a la dirección luisjoseprofe@gmail.com indicando la localidad y el país del centro.

Introducción

Java es un lenguaje de programación; es de hecho, desde hace más de 10 años, el lenguaje de programación más utilizado en el mundo según el índice PYPL[2](PopularitY of Programming Language index).

Los ordenadores no entienden - por ahora - el español, ni el inglés, ni ningún otro idioma natural. De una forma muy simplificada podríamos decir que un lenguaje de programación es un idioma que entiende el ordenador[3]. Cualquier aplicación - proce sador de textos, navegador, programa de retoque fotográfico, etc. - está compuesta de una serie de instrucciones convenientemente empaquetadas en ficheros que le dicen al ordenador de una manera muy precisa qué tiene que hacer en cada momento.

Java es un lenguaje de programación estructurado y, como tal, hace uso de variables, sentencias condicionales, bucles, funciones… Java es también un lenguaje de progra mación orientado a objetos y, por consiguiente, permite definir clases con sus métodos correspondientes y crear instancias de esas clases. Java no es un lenguaje de marcas como HTML o XML aunque puede interactuar muy bien con ellos[4].

Las aplicaciones Java se suelen compilar a *bytecode*, que es independiente del sistema

operativo, no a binario que sí depende del sistema operativo. De esta forma, el *bytecode* generado al compilar un programa escrito en Java debería funcionar en cualquier sistema operativo que tenga instalada una máquina virtual de java (JVM).

Cualquier editor simple como **Nano** o **GEdit** es suficiente para escribir código en Java aunque se recomiendan IDEs[5] como **NetBeans** o **Eclipse** ya que tienen algunas características que facilitan mucho la programación como el chequeo de errores mientras se escribe, el autocompletado de nombres de variables y funciones y mucho más.

[2] http://pypl.github.io/PYPL.html

[3] En realidad un ordenador no entiende directamente los lenguajes de programación como Java, C, Ruby, etc. Mediante una herramienta que se llama compilador o traductor según el caso, lo que hay escrito en cualquiera de estos lenguajes se convierte en código máquina - secuencias de unos y ceros - que es el único lenguaje que de verdad entiende el ordenador.

[4] El lector deberá tener unos conocimientos básicos de HTML para entender bien y sacar provecho del capítulo relativo a JSP de este mismo libro. Una web excelente para aprender HTML es http://w3schools.com

[5] IDE: Integrated Development Enviroment (Entorno de Desarrollo Integrado)

Instalación y configuración del entorno de programación Java

Todos los ejemplos que contiene este libro así como las soluciones a los ejercicios se han escrito sobre **Linux**, en concreto sobre **Ubuntu** (http://www.ubuntu.com/). No obstante, todo el código debería compilar y funcionar sin ningún problema en cualquier otra plataforma.

El software necesario para seguir este manual es el siguiente[6]:

OpenJDK

El **Open Java Development Kit** (OpenJDK) contiene una serie de componentes que permiten desarrollar y ejecutar aplicaciones escritas en Java. Incluye la má quina virtual de Java (JVM) que permite ejecutar *bytecode*. Como mencionamos en el apartado Introducción, el *bytecode* es el ejecutable en Java. El **OpenJDK** también incluye el compilador `javac` que permite compilar el código fuente para generar el *bytecode*. En **Ubuntu**, basta ejecutar una línea en la ventana de terminal para instalar el **OpenJDK**:

```
$ sudo apt-get install openjdk-8-jdk
```

Para los sitemas operativos **Mac OS X** y **Windows**, se encuentran disponibles para su descarga los archivos de instalación del **OpenJDK** en la página Unofficial OpenJDK installers for Windows, Linux and Mac OS X[7]. Otra opción consiste en utilizar el *JDK* oficial de Oracle, que se puede descargar en el apartado Java SE Development Kit 8 - Downloads[8] de la web de esta misma empresa. No vamos a explicar las diferencias entre el **OpenJDK** y el **JDK** ya que se escapa al ámbito de este libro. A efectos prácticos, todos los ejemplos contenidos en este manual funcionan tanto con en el **OpenJDK** como con el **JDK** y los ejercicios se pueden realizar sin problema sea cual sea el kit de desarrollo

instalado.

Editor de textos simple

Para seguir este manual recomendamos utilizar al principio un editor de textos sencillo para probar los primeros programas y realizar los primeros ejercicios, compilando en línea de comandos. De esta manera, el lector se va familiarizando con el proceso de edición de código, compilación y ejecución y va entendiendo lo que sucede en cada paso. En **Ubuntu** viene instalado por defecto el editor **GEdit**, el sistema operativo **Mac OS X** viene con **TextEdit** y en **Windows** podemos usar el programa **Bloc de notas**.

[6]En la página http://helloworldcollection.de/ se muestra el programa ***Hola mundo*** escrito en más de 400 lenguajes de programación.

[7]https://github.com/alexkasko/openjdk-unofficial-builds

[8]http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Geany

Una vez que el lector ha superado la barrera de escribir sus primeros programas, recomendamos utilizar el entorno de programación **Geany** (por ejemplo a partir del capítulo 2). Se trata de un IDE muy ligero que permite realizar aplicaciones sencillas con rapidez. Mediante **Geany** se puede escribir el código, chequear si contiene errores, compilar y ejecutar el *bytecode* generado. Para programas pequeños - por ej. calcular la media de varios números - es recomendable utilizar **Geany** en lugar de un IDE más complejo como **Netbeans**. Mientras con **Geany** da tiempo a escribir el código, compilar y ejecutar, **Netbeans** todavía estaría arrancando.

Instalación de **Geany** (y los *plugins* adicionales) en **Ubuntu**:

```
$ sudo apt-get install geany
$ sudo apt-get install geany-plugins
```

En la sección de descargas de la página oficial de Geany[9] se encuentran los ficheros de instalación de este programa tanto para **Mac OS X** como para **Windows**.

Una vez instalado **Geany** es conveniente realizar algunos ajustes en la configu ración. Presiona **Control + Alt + P** para abrir la ventana de preferencias. Haz clic en la pestaña **Editor** y luego en la pestaña **Sangría**. Establece las opciones tal y como se indican a continuación.

[9]http://www.geany.org/Download/Releases

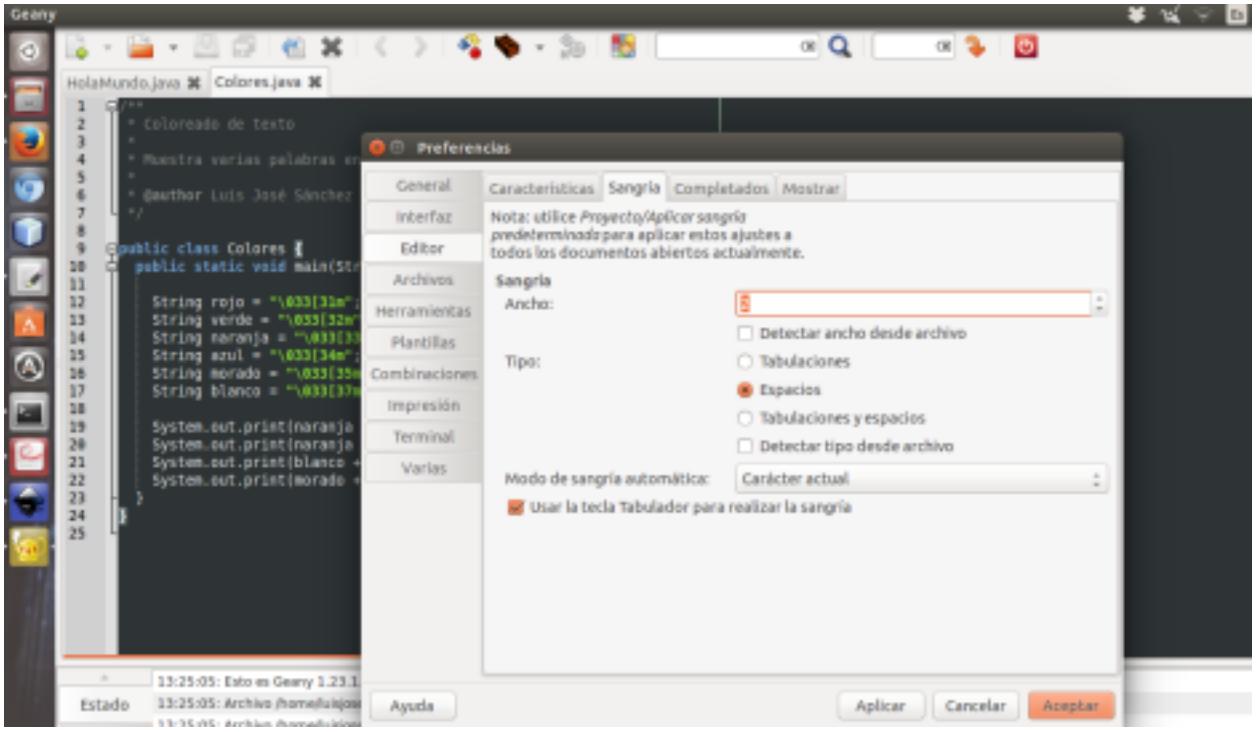

Figura 0.1: Configuración de Geany (sangría)

También es recomendable marcar las casillas **Mostrar guías de sangría** y **Mostrar números de línea**. Estas opciones se activan en la pestaña **Mostrar**.

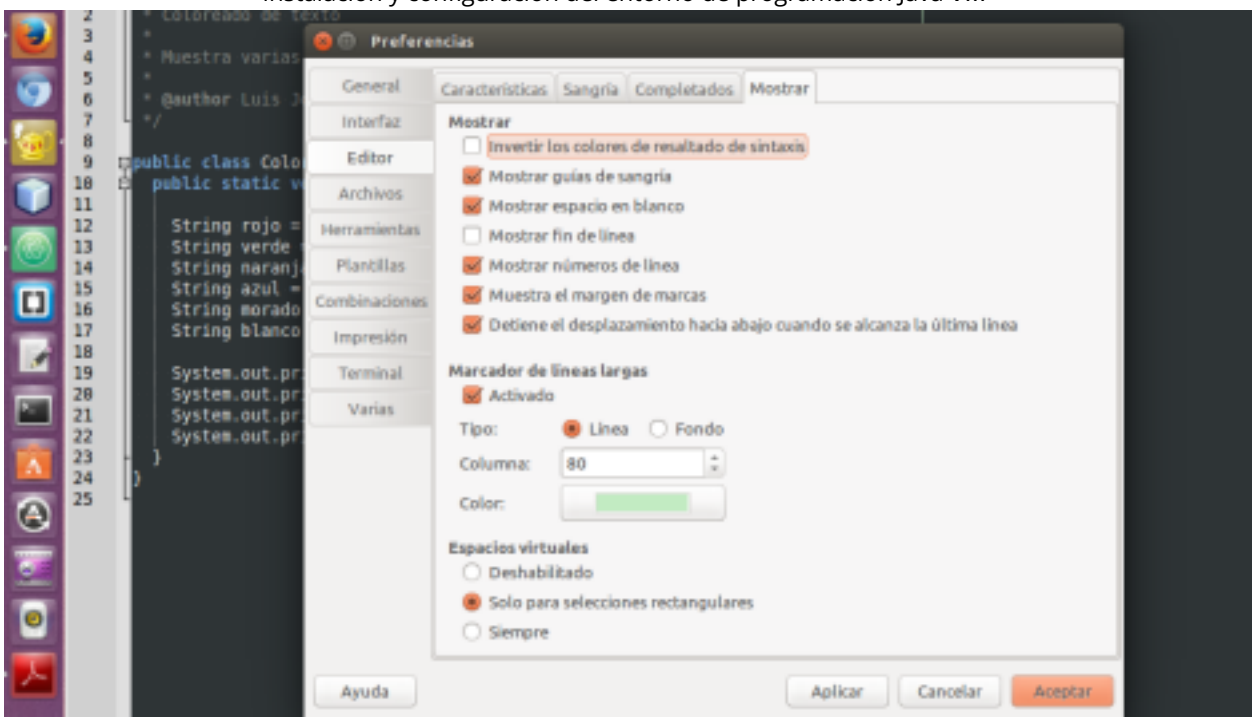

Figura 0.2: Configuración de Geany (guías de sangría y números de línea)

Netbeans

Cuando las aplicaciones a desarrollar son más complejas y, sobre todo, cuando éstas constan de varios ficheros, se recomienda utilizar **Netbeans** de la empresa **Oracle**. En el capítulo 9, el lector ya podría sacarle mucho partido al uso de este IDE. Se trata de un entorno integrado muy potente y muy usado a nivel profesional que incluye el autocompletado de sintaxis y permite una depuración avanzada paso a paso. **Netbeans** se puede descargar de forma gratuita y para cualquier plataforma desde https://netbeans.org/downloads/.

Después de la instalación de **Netbeans**, procedemos a configurar el editor. Para ello seleccionamos **Herramientas** → **Opciones** → **Editor**. A continuación seleccionamos la pestaña **Formato** y en el menú desplegable correspondiente al lenguaje de programación seleccionamos **Java**. La configuración debería quedar como se indica en la imagen.

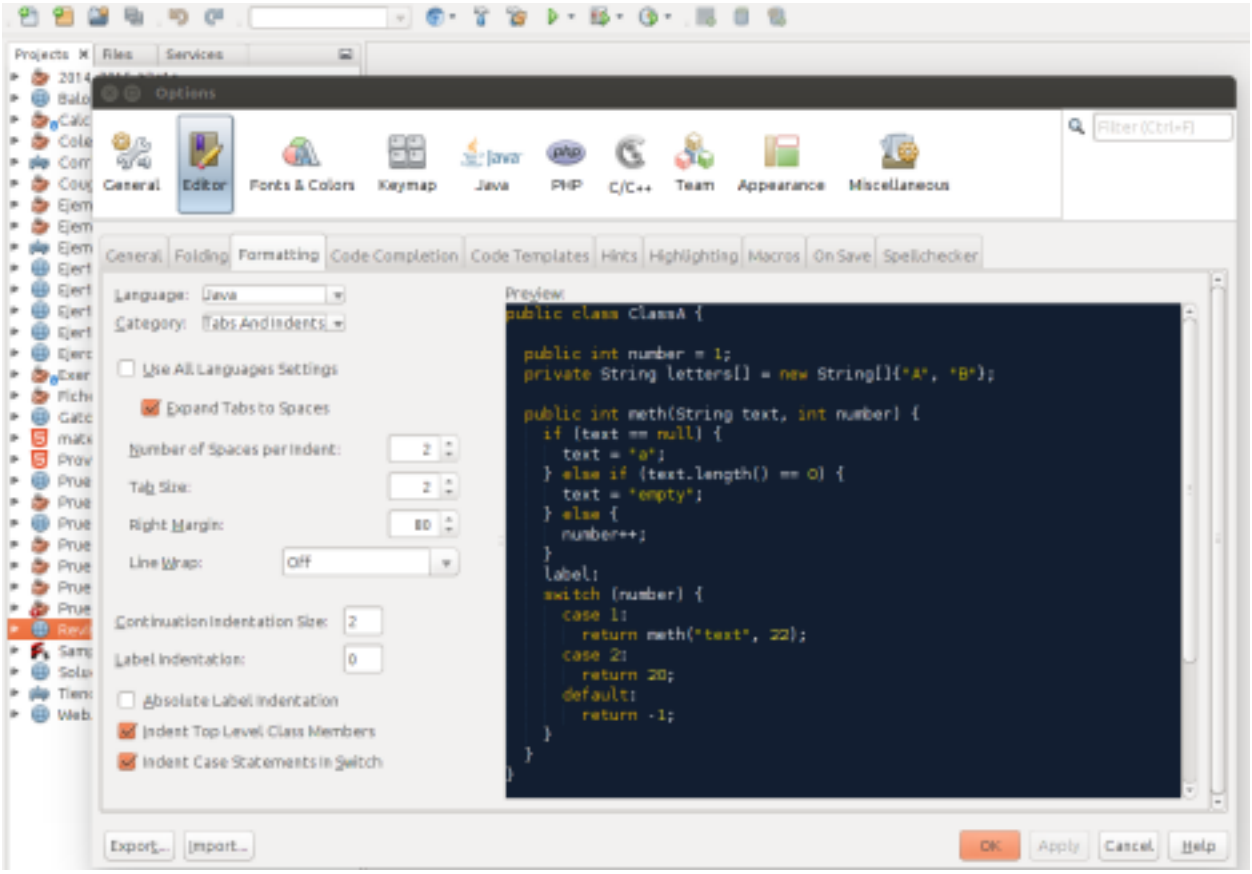

Figura 0.3: Configuración de Netbeans

Es muy importante que los editores/IDEs que se utilicen para esribir código en Java se configuren adecuadamente. Hemos visto cómo configurar **Geany** y **Netbeans**, pero si el lector decide utilizar otras herramientas deberá configurarlas igualmente. En definitiva, la configuración/preferencias de las herramientas que se utilicen deben establecerse de tal forma que se cumplan estos requisitos:

- La tecla `TAB` debe insertar espacios, no debe insertar el carácter de tabulación.
- La indentación debe ser de 2 caracteres (por defecto suele estar a 4). • La condificación de caracteres debe ser UTF-8.

1. ¡Hola mundo! - Salida de datos por pantalla

1.1 ¡Hola mundo! - Mi primer programa

El primer programa que aprende a hacer cualquier aspirante a programador es un **Hola mundo** [1]. Es seguramente el programa más sencillo que se puede escribir. Se trata de un programa muestra el mensaje “Hola mundo” por pantalla. Abre el programa **GEdit** que, como hemos comentado en la sección Instalación y configuración del entorno de programación Java, viene instalado por defecto en Ubuntu y teclea el siguiente código:

```
/**
 * Muestra por pantalla la frase "¡Hola mundo!"
```

```
 *
 * @author Luis J. Sánchez
 */

public class HolaMundo { // Clase principal
  public static void main(String[] args) {
    System.out.println("¡Hola mundo!");
  }
}
```

Verás que, en principio, el texto del código no aparece en colores. El coloreado de sintaxis es muy útil como comprobarás más adelante, sobre todo para detectar errores. Graba el programa como `HolaMundo.java`. Al grabar con la extensión `.java`, ahora el programa **GEdit** sí sabe que se trata de un programa escrito en Java y reconoce la sintaxis de este lenguaje de forma que puede colorear los distintos elementos que aparecen en el código.

[1]En la página http://helloworldcollection.de/ se muestra el programa ***Hola mundo*** escrito en más de 400 lenguajes de programación.

¡Hola mundo! - Salida de datos por pantalla 2

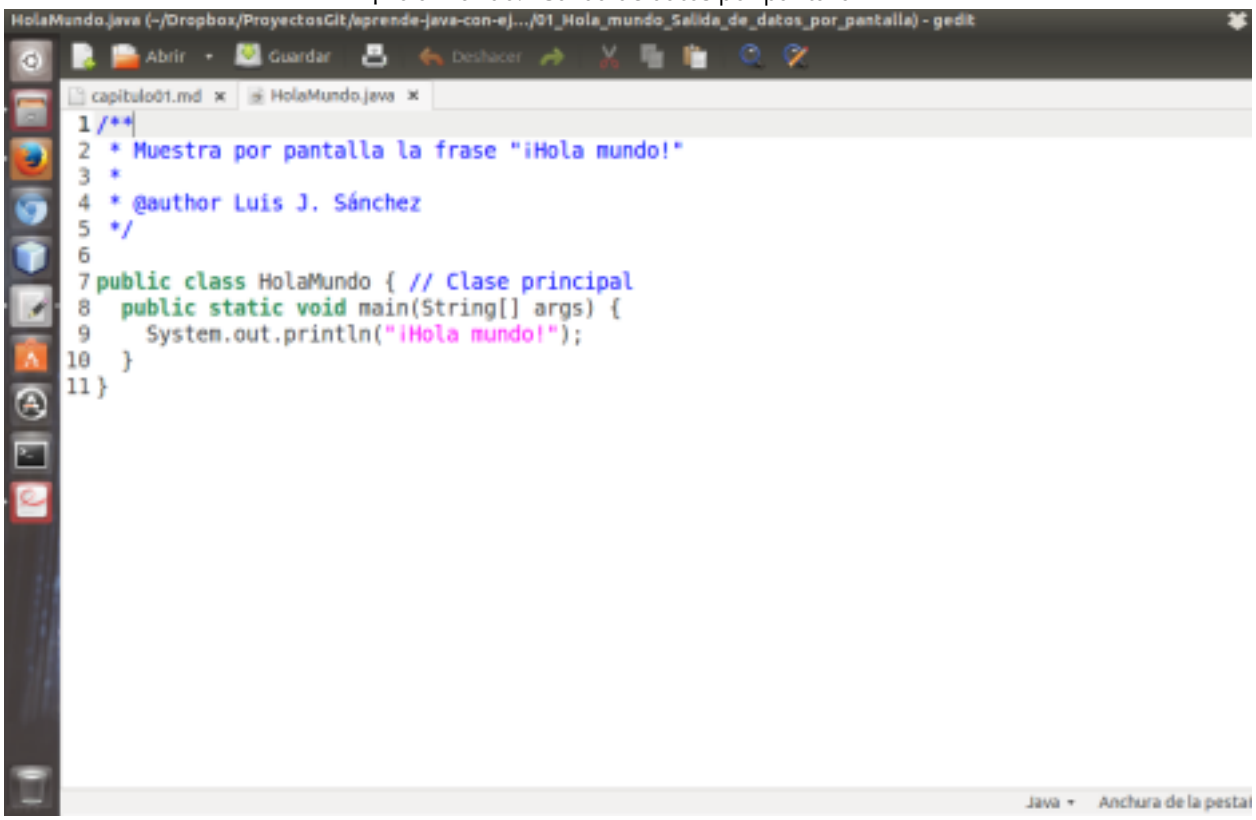

Figura 1.1: Programa `Hola mundo` escrito en el editor GEdit

Fíjate que el nombre que le damos al fichero es exactamente igual que la clase principal seguido de la extensión `.java`. Abre una ventana de terminal, entra en el directorio donde se encuentra el fichero y teclea lo siguiente:

```
$ javac HolaMundo.java
```

Este comando crea `HolaMundo.class` que es el *bytecode*. Para ejecutar el programa, teclea:

```
$ java HolaMundo
```

¡Enhorabuena! Acabas de realizar tu primer programa en Java.

En la siguiente captura de pantalla puedes ver la ventana de terminal y los comandos que acabamos de describir.

¡Hola mundo! - Salida de datos por pantalla 3

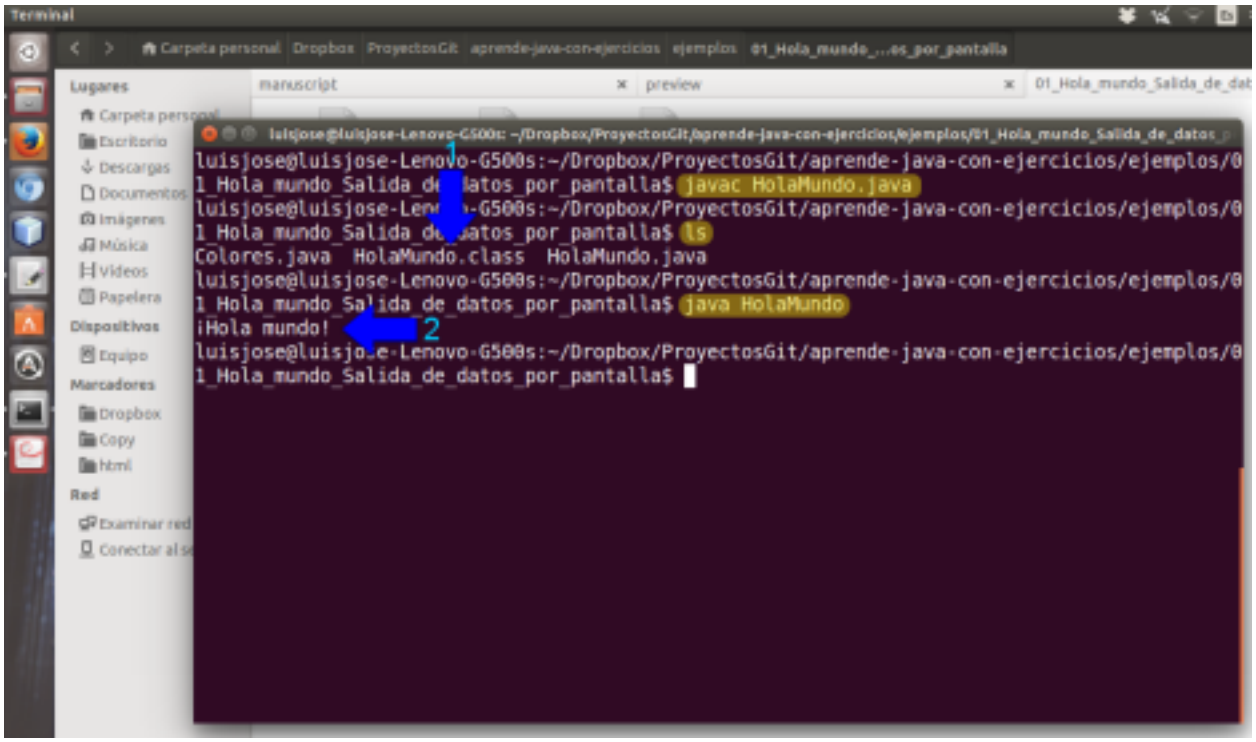

Figura 1.2: `HolaMundo.class` es el *bytecode* generado (1). El resultado del programa es una línea de texto escrita en pantalla (2).

La instrucción que hemos utilizado para mostrar una frase por pantalla es `Sys tem.out.println()`, colocando la frase entre paréntesis. También se puede volcar texto por pantalla mediante `System.out.print()`. La única diferencia radica en que esta última no añade un salto de línea al final, pruébalo en `HolaMundo.java` y compruébalo por ti mismo. Si lo que quieres mostrar es una palabra o una frase, como en este ejemplo, es importante que el texto esté entrecomillado.

- El programa propiamente dicho está dentro de lo que llamamos “clase principal”.
- El fichero debe tener el mismo nombre que la clase principal seguido por la extensión `.java`
- Los comentarios de varias líneas se colocan entre `/*` y `*/`

- Los comentarios de línea se indican mediante `//`
- Para mostrar información por pantalla se utilizan `System.out.println()` y `System.out.print()`.

1.2 Coloreado de texto

Mostrar siempre texto blanco sobre fondo negro puede resultar muy aburrido. El texto que se muestra por pantalla se puede colorear; para ello es necesario insertar unas secuencias de caracteres - que indican el color con el que se quiere escribir - justo antes del propio texto.

Prueba el siguiente programa y no te preocupes si todavía no lo entiendes del todo, en el próximo capítulo se explican los diferentes tipos de datos que se pueden utilizar en Java, entre estos tipos está la cadena de caracteres o `String`.

```
/**
 * Coloreado de texto
 *
 * Muestra varias palabras en el color que corresponde.
 *
 * @author Luis José Sánchez
 */

public class Colores {
  public static void main(String[] args) {

    String rojo = "\033[31m";
    String verde = "\033[32m";
    String naranja = "\033[33m";
    String azul = "\033[34m";
    String morado = "\033[35m";
    String blanco = "\033[37m";

    System.out.print(naranja + "mandarina" + verde + " hierba");
    System.out.print(naranja + " saltamontes" + rojo + " tomate");
    System.out.print(blanco + " sábanas" + azul + " cielo");
    System.out.print(morado + " nazareno" + azul + " mar");
  }
}
```

Como puedes ver en los dos ejemplos que hemos mostrado, algunas líneas están ligeramente desplazadas hacia la derecha, es lo que se llama **sangría** o **indentación**. En programación es muy importante sangrar (indentar) bien porque da una idea de qué partes del código son las que contienen a otras. En este último ejemplo tenemos un programa que tiene unos comentarios al principio y luego la clase principal marcada por la línea `public class Colores {`. Dentro de la clase `Colores` está el `main` o bloque del programa principal que tiene sangría por estar dentro de la definición de la clase `Colores`. A su vez, dentro del `main` hay una serie de líneas de código que igualmente tienen sangría.

En este libro, tanto el código de los ejemplos como el de las soluciones a los ejercicios siguen el estándar de Google para el código fuente escrito en el lenguaje de programación Java[2], por tanto, cada vez que se aplique la sangría será exactamente de dos espacios.

[2] http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

1.3 Ejercicios

Ejercicio 1

Escribe una programa que muestre tu nombre por pantalla.

Ejercicio 2

Modifica el programa anterior para que además se muestre tu dirección y tu número de teléfono. Asegúrate de que los datos se muestran en líneas separadas.

Ejercicio 3

Escribe un programa que muestre por pantalla 10 palabras en inglés junto a su correspondiente traducción al castellano. Las palabras deben estar distribuidas en dos columnas y alineadas a la izquierda. Pista: Se puede insertar un tabulador mediante `\t`.

Ejercicio 4

Escribe un programa que muestre tu horario de clase. Puedes usar espacios o tabuladores para alinear el texto.

Ejercicio 5

Modifica el programa anterior añadiendo colores. Puedes mostrar cada asig natura de un color diferente.

Ejercicio 6

Escribe un programa que pinte por pantalla una pirámide rellena a base de asteriscos. La base de la pirámide debe estar formada por 9 asteriscos.

Ejercicio 7

Igual que el programa anterior, pero esta vez la pirámide estará hueca (se debe ver únicamente el contorno hecho con asteriscos).

Ejercicio 8

Igual que el programa anterior, pero esta vez la pirámide debe aparecer invertida, con el vértice hacia abajo.

Ejercicio 9

Escribe un programa que pinte por pantalla alguna escena - el campo, la habitación de una casa, un aula, etc. - o algún objeto animado o inanimado - un coche, un gato, una taza de café, etc. Ten en cuenta que puedes utilizar caracteres como *, +, <, #, @, etc. ¡Échale imaginación!

2. Variables

2.1 Definición y tipos de variables

Una variable es un contenedor de información. Imagina una variable como una cajita con una etiqueta que indica su nombre, una cajita en la que se puede introducir un va lor. Las variables pueden almacenar valores enteros, números decimales, caracteres, cadenas de caracteres (palabras o frases), etc. El contenido de las variables puede cambiar durante la ejecución del

programa, de ahí viene el nombre de “variable”.

Java es un lenguaje fuertemente tipado, es decir, es necesario declarar todas las variables que utilizará el programa, indicando siempre el nombre y el tipo de cada una.

El nombre que se le da a una variable es muy importante; intenta usar siempre nombres significativos que, de alguna forma, identifiquen el contenido. Por ejemplo, si usas una variable para almacenar el volumen de una figura, una buena opción sería llamarla `volumen`; si tienes una variable que almacena la edad de una persona, lo mejor es llamarla `edad`, y así sucesivamente.

Un programa se lee y se depura mucho mejor si los nombres de las variables se han elegido de forma inteligente.

¿Podrías averiguar qué hace la siguiente línea de código?

```
x = vIp3 * Weerty - zxc;
```

Ahora observa el mismo código pero con otros nombres de variables.

```
precioTotal = cantidad * precio - descuento;
```

Se entiende mucho mejor ¿verdad?

Escribiremos los nombres de variables en formato *lowerCamelCase*. La primera letra se escribe en minúscula y, a continuación, si se utiliza más de una palabra, cada una de ellas empezaría con mayúscula. Por ejemplo, `edadMin` es un buen nombre para una variable que almacena la edad mínima a la que se puede acceder a un sitio web. Observa que hemos usado una mayúscula para diferenciar dos partes (edad y Min). Puede que en algún libro también encuentres nombres de variables con el carácter de subrayado (_) de tal forma que el nombre de la variable sería `edad_min`, pero como hemos comentado anteriormente, en este libro nos ceñimos al estándar de Google[1], que exige el formato *lowerCamelCase*.

[1] http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

No se permiten símbolos como $, %, @, +, -, etc. Puedes usar números en los nombres de variables pero nunca justo al principio; `5x` no es un nombre válido pero `x5` sí lo es.

No se debe poner una letra mayúscula al comienzo del nombre de una variable para no confundirla con una clase (los nombres de las clases comienzan por mayúscula).

2.1.1 Enteros (`int` y `long`)

Las variables que van a contener números enteros se declaran con `int`. Veamos un ejemplo.

```
/**
 * Uso de variables enteras
 *
 * @author Luis J. Sánchez
 */

public class VariablesEnteras {
  public static void main(String[] args) {
```

```
    int x; // Se declara la variable x como entera

    x = 5; // Se asigna el valor 5 a la variable x
    System.out.println("El valor actual de mi variable es " + x);

    x = 7; // Se asigna el valor 7 a la variable x
    System.out.println("y ahora es " + x);
  }
}
```

Si pretendemos almacenar valores muy grandes en una variable, usaremos el tipo `long` en lugar de `int`.

2.1.2 Números decimales (`double` y `float`)

Usamos los tipos `double` o `float` cuando queremos (o esperamos) almacenar números con decimales en las variables.

Ciñéndonos al estándar de Google, no daremos por válidas definiciones de variables como la siguiente (aunque funcionaría sin ningún problema).

```
    double x, y;
```

Cada variable se debe definir en una línea diferente. Lo siguiente sí sería correcto.


```
    double x;
    double y;
```

A continuación tienes un ejemplo completo.

```
/**
 * Uso de variables que contienen números decimales
 *
 * @author Luis J. Sánchez
 */

public class VariablesConDecimales {
  public static void main(String[] args) {
    double x; // Se declaran las variables x e y
    double y; // de tal forma que puedan almacenar decimales.

    x = 7;
    y = 25.01;

    System.out.println(" x vale " + x);
    System.out.println(" y vale " + y);
  }
}
```

Como puedes ver, también se pueden almacenar números enteros en variables de tipo `double`.

2.1.3 Cadenas de caracteres (`String`)

Las cadenas de caracteres se utilizan para almacenar palabras y frases. Todas las cadenas de caracteres deben ir entrecomilladas.

```
/**
 * Uso del tipo String
 *
 * @author Luis J. Sánchez
 */

public class UsoDeStrings {
  public static void main(String[] args) {
    String miPalabra = "cerveza";
    String miFrase = "¿dónde está mi cerveza?";

    System.out.println("Una palabra que uso con frecuencia: " + miPalabra);
```



```
    System.out.println("Una frase que uso a veces: " + miFrase);
  }
}
```

Como puedes ver, en una cadena de caracteres se pueden almacenar signos de puntuación, espacios y letras con tildes.

2.2 Operadores artiméticos

En Java se puede operar con las variables de una forma muy parecida a como se hace en matemáticas. Los operadores aritméticos de Java son los siguientes:

OPERADOR	NOMBRE	EJEMPLO	DESCRIPCIÓN
+	suma	20 + x	suma dos números
-	resta	a - b	resta dos números
*	multiplicación	10 * 7	multiplica dos números
/	división	altura / 2	divide dos números
%	resto (módulo)	5 % 2	resto de la división entera
++	incremento	a++	incrementa en 1 el valor de la variable
- -	decremento	a- -	decrementa en 1 el valor de la variable

A continuación tienes un programa que ilustra el uso de los operadores aritméticos.

```
/**
 * Uso de los operadores aritméticos
 *
 * @author Luis J. Sánchez
 */

public class UsoDeOperadoresAritmeticos {
```

```
  public static void main(String[] args) {
    int x;
    x = 100;

    System.out.println(x + " " + (x + 5) + " " + (x - 5));
    System.out.println((x * 5) + " " + (x / 5) + " " + (x % 5));
  }
}
```

2.3 Asignación de valores a variables

La sentencia de asignación se utiliza para dar un valor a una variable. En Java (y en la mayoría de lenguajes de programación) se utiliza el símbolo igual (=) para este cometido. Es importante recalcar que una asignación no es una ecuación. Por ejemplo `x = 7 + 1` es una asignación en la cual se evalúa la parte derecha `7 + 1`, y el resultado de esa evaluación se almacena en la variable que se coloque a la izquierda del igual, es decir, en la `x`, o lo que es lo mismo, el número 8 se almacena en `x`. La sentencia `x + 1 = 23 * 2` no es una asignación válida ya que en el lado izquierdo debemos tener únicamente un nombre de variable.

Veamos un ejemplo con algunas operaciones y asignaciones.


```
/**
 * Operaciones y asignaciones
 *
 * @author Luis J. Sánchez
 */

public class Asignaciones {
  public static void main(String[] args) {

    int x = 2;
    int y = 9;

    int sum = x + y;
    System.out.println("La suma de mis variables es " + sum);

    int mul = x * y;
    System.out.println("La multiplicación de mis variables es " + mul);
  }
}
```

2.4 Conversión de tipos (*casting*)

En ocasiones es necesario convertir una variable (o una expresión en general) de un tipo a otro. Simplemente hay que escribir entre paréntesis el tipo que se quiere obtener. Experimenta con el siguiente programa y observa los diferentes resultados que se obtienen.

```
/**
 * Conversión de tipos
 *
```

```
 * @author Luis J. Sánchez
 */

public class ConversionDeTipos {
  public static void main(String[] args) {

    int x = 2;
    int y = 9;
    double division;

    division = (double)y / (double)x;
    //division = y / x; // Comenta esta línea y
              // observa la diferencia.
```



```
    System.out.println("El resultado de la división es " + division);
  }
}
```


2.5 Ejercicios

Ejercicio 1

Escribe un programa en el que se declaren las variables enteras `x` e `y`. Asignales los valores 144 y 999 respectivamente. A continuación, muestra por pantalla el valor de cada variable, la suma, la resta, la división y la multiplicación.

Ejercicio 2

Crea la variable nombre y asígnale tu nombre completo. Muestra su valor por pantalla de tal forma que el resultado del programa sea el mismo que en el ejercicio 1 del capítulo 1.

Ejercicio 3

Crea las variables nombre, direccion y telefono y asígnale los valores corres pondientes. Muestra los valores de esas variables por pantalla de tal forma que el resultado del programa sea el mismo que en el ejercicio 2.

Ejercicio 4

Realiza un conversor de euros a pesetas. La cantidad en euros que se quiere convertir deberá estar almacenada en una variable.

Ejercicio 5

Realiza un conversor de pesetas a euros. La cantidad en pesetas que se quiere convertir deberá estar almacenada en una variable.

Ejercicio 6

Escribe un programa que calcule el total de una factura a partir de la base imponible (precio sin IVA). La base imponible estará almacenada en una variable.

3. Lectura de datos desde teclado

Hemos visto en los capítulos anteriores cómo mostrar información por pantalla y cómo usar variables. Veremos ahora algo muy importante que te permitirá realizar programas algo más funcionales, que tengan una utilidad real; aprenderás a leer la información que introduce un usuario mediante el teclado.

El funcionamiento de casi todos los programas se podría resumir en los siguientes puntos:

1. Entrada de datos desde teclado (o desde cualquier otro dispositivo de entrada) 2. Procesamiento de los datos de entrada para producir un resultado 3. Visualización de los resultados por pantalla

De momento, hemos visto cómo realizar los puntos 2 y 3. Vamos a ver a continuación cómo se recoge la información desde el teclado.

3.1 Lectura de texto

Para recoger datos por teclado usamos `System.console().readLine()`. Cuando llegamos a esta sentencia, el programa se detiene y espera que el usuario introduzca información mediante el teclado. La introducción de datos termina con la pulsación de la tecla INTRO. Una vez que el usuario presiona INTRO, todo lo que se ha tecleado se almacena en una variable, en el siguiente ejemplo esa variable es `nombre`.

```
/**
* Lectura de datos desde teclado
*
* @author Luis J. Sanchez
*/

public class DimeTuNombre {
  public static void main(String[] args) {
    String nombre;
    System.out.print("Por favor, dime cómo te llamas: ");
    nombre = System.console().readLine();
    System.out.println("Hola " + nombre + ", ¡encantado de conocerte!");
  }
}
```


Mediante `System.console().readLine()` se recoge una línea de texto introducida por

teclado.

Cuando se piden datos por teclado es importante que el programa especifique claramente cuál es exactamente la información que requiere por parte del usuario. Date cuenta que este programa muestra el mensaje "`Por favor, dime cómo te llamas:`". Imagina que se omite este mensaje, en la pantalla aparecería únicamente un cursor parpadeando y el usuario no sabría qué tiene que hacer o qué dato introducir.

También es importante reseñar que los datos introducidos por teclado se recogen como una cadena de caracteres (un `String`).

3.2 Lectura de números

Si en lugar de texto necesitamos datos numéricos, deberemos convertir la cadena introducida en un número con el método adecuado. Como se muestra en el ejemplo, `Integer.parseInt()` convierte el texto introducido por teclado en un dato numérico, concretamente en un número entero.

```
/**
* Lectura de datos desde teclado
*
* @author Luis J. Sánchez
*/

public class LeeNumeros {
  public static void main(String[] args) {

    String linea;

    System.out.print("Por favor, introduce un número: ");
    linea = System.console().readLine();
    int primerNumero;
    primerNumero = Integer.parseInt( linea );

    System.out.print("introduce otro, por favor: ");
    linea = System.console().readLine();
    int segundoNumero;
    segundoNumero = Integer.parseInt( linea );

    int total;
    total = (2 * primerNumero) + segundoNumero;

    System.out.print("El primer número introducido es " + primerNumero);
```



```
    System.out.println(" y el segundo es " + segundoNumero);
    System.out.print("El doble del primer número más el segundo es ");
    System.out.print(total);
  }
}
```

Este último programa se podría acortar un poco. Por ejemplo, estas dos líneas

```
int total;
total = (2 * primerNumero) + segundoNumero;
```

se podrían quedar en una sola línea

```
int total = (2 * primerNumero) + segundoNumero;
```

De igual modo, estas tres líneas

```
linea = System.console().readLine();
int primerNumero;
primerNumero = Integer.parseInt( linea );
```

también se podrían reducir a una sola tal que así

```
int primerNumero = Integer.parseInt( System.console().readLine() );
```

Es muy importante que el código de nuestros programas sea limpio y legible. A veces, abreviando demasiado el código se hace más difícil de leer; es preferible tener unas líneas de más y que el código se entienda bien a tener un código muy compacto pero menos legible.

3.3 La clase `Scanner`

El método `System.console().readLine()` funciona bien en modo consola (en una ventana de terminal) pero puede provocar problemas cuando se trabaja con IDEs como **Eclipse**, **Netbeans**, **JavaEdit**, etc. Para evitar estos problemas puedes usar la clase `Scanner` cuando necesites recoger datos desde teclado. La clase `Scanner` funciona tanto en entornos integrados como en una ventana de terminal.


```
/**
* Lectura de datos desde teclado usando la clase Scanner
*
* @author Luis J. Sánchez
*/

import java.util.Scanner;

public class LeeDatosScanner01 {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);

    System.out.print("Introduce tu nombre: ");
    String nombre = s.nextLine();

    System.out.print("Introduce tu edad: ");
```

```
    int edad = Integer.parseInt(s.nextLine());

    System.out.println("Tu nombre es " + nombre + " y tu edad es " + edad);
  }
}
```

Fíjate que en el programa anterior la sentencia

```
s.nextLine()
```

sería el equivalente a

```
System.console().readLine()`
```

Mediante el uso de la clase `Scanner` es posible leer varios datos en una misma línea. En el programa anterior se pedía un nombre y una edad, en total dos datos que había que introducir en líneas separadas. Observa cómo en el siguiente ejemplo se piden esos dos datos en una sola línea y separados por un espacio.

```
/**
* Lectura de datos desde teclado usando la clase Scanner
*
* @author Luis J. Sánchez
*/

import java.util.Scanner;

public class LeeDatosScanner02 {
```



```
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);

    System.out.print("Introduce tu nombre y tu edad separados por un espacio:
    "); String nombre = s.next();
    int edad = s.nextInt();

    System.out.println("Tu nombre es " + nombre + " y tu edad es " + edad);
  }
}
```

Fíjate cómo se ha utilizado `s.next()` para leer una cadena de caracteres y `s.nextInt()` para leer un número entero, todo ello en la misma línea.

El siguiente programa de ejemplo calcula la media de tres números decimales. Para leer cada uno de los números en la misma línea se utiliza `s.nextDouble`.

```
/**
* Lectura de datos desde teclado usando la clase Scanner
*
* @author Luis J. Sánchez
```

```
*/

import java.util.Scanner;

public class LeeDatosScannerMedia {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);

    System.out.print("Introduce tres números (pueden contener decimales) separados por esp\
acios: ");
    double x1 = s.nextDouble();
    double x2 = s.nextDouble();
    double x3 = s.nextDouble();

    double media = (x1 + x2 + x3) / 3;

    System.out.println("La media de esos tres números es " + media);
  }
}
```


Recuerda que **¡a programar se aprende programando!** Es muy importante que pruebes los ejemplos - si los modificas y experimentas, todavía mejor - y que realices los ejercicios. Para aprender a programar no basta con leer el libro y entenderlo, es una condición necesaria pero no suficiente. Igual que para ser piloto hacen falta muchas horas de vuelo, para ser programador hacen falta muchas horas picando código, probando, corrigiendo errores… ¡Ánimo! ¡El esfuerzo merece la pena!

3.4 Ejercicios

Ejercicio 1

Realiza un programa que pida dos números y que luego muestre el resultado de su multiplicación.

Ejercicio 2

Realiza un conversor de euros a pesetas. La cantidad de euros que se quiere convertir debe ser introducida por teclado.

Ejercicio 3

Realiza un conversor de pesetas a euros. La cantidad de pesetas que se quiere convertir debe ser introducida por teclado.

Ejercicio 4

Escribe un programa que sume, reste, multiplique y divida dos números introducidos por teclado.

Ejercicio 5

Escribe un programa que calcule el área de un rectángulo.

Ejercicio 6

Escribe un programa que calcule el área de un triángulo.

Ejercicio 7

Escribe un programa que calcule el total de una factura a partir de la base imponible.

Ejercicio 8

Escribe un programa que calcule el salario semanal de un empleado en base a las horas trabajadas, a razón de 12 euros la hora.

Ejercicio 9

Escribe un programa que calcule el volumen de un cono según la fórmula $V = \frac{1}{3}\pi r^2 h$

Ejercicio 10

Realiza un conversor de Mb a Kb.

Ejercicio 11

Realiza un conversor de Kb a Mb.

Ejercicio 12

Escribe un programa que calcule el precio final de un producto según su base imponible (precio antes de impuestos), el tipo de IVA aplicado (general, reducido o superreducido) y el código promocional. Los tipos de IVA general, reducido y superreducido son del 21%, 10% y 4% respectivamente. Los códi gos promocionales pueden ser **nopro**, **mitad**, **meno5** o **5porc** que significan respectivamente que no se aplica promoción, el precio se reduce a la mitad, se descuentan 5 euros o se descuenta el 5%. El ejercicio se da por bueno si se muestran los valores correctos, aunque los números no estén tabulados.

Ejemplo:

```
Introduzca la base imponible: 25
Introduzca el tipo de IVA (general, reducido o superreducido): reducido
Introduzca el código promocional (nopro, mitad, meno5 o 5porc): mitad
Base imponible 25.00
```

```
IVA (10%) 2.50
Precio con IVA 27.50
Cód. promo. (mitad): -13.75
TOTAL 13.75
```


Ejercicio 13

Realiza un programa que calcule la nota que hace falta sacar en el segundo examen de la asignatura **Programación** para obtener la media deseada. Hay que tener en cuenta que la nota del primer examen cuenta el 40% y la del segundo examen un 60%.

Ejemplo 1:

```
Introduce la nota del primer examen: 7
¿Qué nota quieres sacar en el trimestre? 8.5
Para tener un 8.5 en el trimestre necesitas sacar un 9.5 en el segundo examen.
```

Ejemplo 2:

```
Introduce la nota del primer examen: 8
¿Qué nota quieres sacar en el trimestre? 7
Para tener un 7 en el trimestre necesitas sacar un 6.33 en el segundo examen.
```

4. Sentencia condicional (`if` y `switch`)

Una sentencia condicional permite al programa bifurcar el flujo de ejecución de instrucciones dependiendo del valor de una expresión.

4.1 Sentencia `if`

La sentencia `if` permite la ejecución de una serie de instrucciones en función del resultado de una expresión lógica. El resultado de evaluar una expresión lógica es siempre verdadero (`true`) o falso (`false`). Es muy simple, en lenguaje natural sería algo como ”si esta condición es verdadera entonces haz esto, sino haz esto otro”.

El formato de la sentencia `if` es el siguiente:

```
if (condición) {

    instrucciones a ejecutar si la condición es verdadera

} else {

    instrucciones a ejecutar si la condición es falsa

}
```

A continuación se muestra un ejempo del uso de la sentencia `if`.

```
/**
 * Sentencia if
 *
 * @author Luis J. Sánchez
```

```
 */

public class SentenciaIf01 {
  public static void main(String[] args) {
    System.out.print("¿Cuál es la capital de Kiribati? ");
    String respuesta = System.console().readLine();

    if (respuesta.equals("Tarawa")) {
      System.out.println("¡La respuesta es correcta!");
    } else {
      System.out.println("Lo siento, la respuesta es incorrecta.");
```



```
    }
  }
}
```

En el programa se le pregunta al usuario cuál es la capital de Kiriwati. La respuesta introducida por el usuario se almacena en la variable `respuesta`. A continuación viene la sentencia condicional `if`.

```
    if (respuesta.equals("Tarawa"))
```

Llegado a este punto, el programa evalúa la expresión `respuesta.equals("Tarawa")`. Observa que para comparar dos cadenas de caracteres se utiliza `equals()`. Ima ginemos que el usuario ha introducido por teclado `Madrid`; entonces la expresión `"Madrid".equals("Tarawa")` daría como resultado `false` (falso).

Si la expresión hubiera dado como resultado `true` (verdadero), se ejecutaria la línea

```
    System.out.println("¡La respuesta es correcta!");
```

pero no es el caso, el resultado de la expresión ha sido `false` (falso), todo el mundo sabe que la capital de Kiriwati no es Madrid, por tanto se ejecutaría la línea

```
    System.out.println("Lo siento, la respuesta es incorrecta.");
```

Vamos a ver otro ejemplo, esta vez con números. El usuario introducirá un número por teclado y el programa dirá si se trata de un número positivo o negativo.

```
/**
 * Sentencia if
 *
 * @author Luis J. Sánchez
 */

public class SentenciaIf02 {
  public static void main(String[] args) {
    System.out.print("Por favor, introduce un número entero: ");
    String linea = System.console().readLine();
    int x = Integer.parseInt( linea );

    if (x < 0) {
```

```
      System.out.println("El número introducido es negativo.");
    } else {
      System.out.println("El número introducido es positivo.");
    }
  }
}
```

El siguiente bloque de código


```
    if (x < 0)
      System.out.println("El número introducido es negativo.");
    else
      System.out.println("El número introducido es positivo.");
```

compilaría y funcionaría sin problemas - fíjate que hemos quitado las llaves - ya que antes y después del `else` hay una sola sentencia y en estos casos no es obligatorio poner llaves. Sin embargo, nosotros siempre usaremos llaves, es una exigencia del estándar de Google al que nos ceñimos en este manual.

Llaves egipcias (egyptian brackets)

Fíjate en la manera de colocar las llaves dentro del código de un programa en Java. La llave de apertura de bloque se coloca justo al final de la línea y la llave de cierre va justo al principio de la línea. Se llaman llaves egipcias[1] por la similitud entre las llaves y las manos de los egipcios que aparecen en los papiros.

[1] http://blog.codinghorror.com/new-programming-jargon/

4.2 Operadores de comparación

En el ejemplo anterior, usamos el operador `<` en la comparación `if (x < 0)` para saber si la variable x es menor que cero. Hay más operadores de comparación, en la siguiente tabla se muestran todos.

OPERADOR	NOMBRE	EJEMPLO	DESCRIPCIÓN
==	igual	a == b	a es igual a b
!=	distinto	a != b	a es distinto de b
<	menor que	a < b	a es menor que b
>	mayor que	a > b	a es mayor que b
<=	mayor o igual que	a > = b	a es mayor o igual que b

El siguiente ejemplo muestra el uso de uno de estos operadores, concretamente de `>=` (mayor o igual). El usuario introduce una nota; si esta nota es **mayor o igual a 5** se le mostrará un mensaje diciendo que ha aprobado y en caso de que no se cumpla la condición se mostrará un mensaje diciendo que está suspenso.

```
/**
 * Sentencia if
 *
 * @author Luis J. Sánchez
 *
```

```
 */

public class SentenciaIf03 {
  public static void main(String[] args) {
    System.out.print("¿Qué nota has sacado en el último examen? ");
    String line = System.console().readLine();
    double nota = Double.parseDouble( line );

    if (nota >= 5) {
      System.out.println("¡Enhorabuena!, ¡has aprobado!");
    } else {
      System.out.println("Lo siento, has suspendido.");
    }
  }
}
```


4.3 Operadores lógicos

Los operadores de comparación se pueden combinar con los operadores lógicos. Por ejemplo, si queremos saber si la variable `a` es mayor que `b` y además es menor que `c`, escribiríamos `if ((a > b) && (a < c))`. En la siguiente tabla se muestran los operadores lógicos de Java:

OPERADOR	NOMBRE	EJEMPLO	DEVUELVE VERDADERO CUANDO…
&&	y	(7 > 2) && (2 < 4)	las dos condiciones son verdaderas
||	o	(7 > 2) || (2 < 4)	al menos una de las condiciones es verdadera
!	no	!(7 > 2)	la condición es falsa

Vamos a ver cómo funcionan los operadores lógicos con un ejemplo. Mediante `if ((n < 1) || (n > 100))` se pueden detectar los números que no están en el rango de 1 a 100; literalmente sería “si n es menor que 1 o n es mayor que 100”.

```
/**
 * Operadores lógicos
 *
 * @author Luis J. Sánchez
 */

public class OperadoresLogicos01 {
  public static void main(String[] args) {
    System.out.println("Adivina el número que estoy pensando.");
    System.out.print("Introduce un número entre el 1 y el 100: ");
    String linea = System.console().readLine();
    int n = Integer.parseInt( linea );

    if ((n < 1) || (n > 100)) {
      System.out.println("El número introducido debe estar en el intervalo 1 –
      100."); System.out.print("Tienes otra oportunidad, introduce un número: ");
      linea = System.console().readLine();
```

```
      n = Integer.parseInt( linea );
    }

    if (n == 24) {
      System.out.println("¡Enhorabuena!, ¡has acertado!");
    } else {
      System.out.println("Lo siento, ese no es el número que estoy
    pensando."); }
```



```
  }
}
```

En el siguiente programa puedes ver el uso de operadores lógicos combinado con operadores relacionales (operadores de comparación). Intenta adivinar cuál será el resultado mirando el código.

```
/**
 *
 * Operadores lógicos y relacionales
 *
 * @author Luis J. Sánchez
 *
 */

public class OperadoresLogicos02 {
  public static void main(String[] args) {
    boolean a = true;
    boolean b = false;
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("a && b = " + (a && b));
    System.out.println("a || b = " + (a || b));
    System.out.println("!a = " + !a);
    System.out.println("a || (6 > 10) = " + (a || (6 > 10)));
    System.out.println("((4 <= 4) || false) && (!a) = " + (((4 <= 4) || false) && (!a)));
  }
}
```


4.4 Sentencia `switch` (selección múltiple)

A veces es necesario comparar el valor de una variable con una serie de valores concretos. La selección múltiple es muy parecida (aunque no es exactamente igual) a una secuencia de varias sentencias `if`.

El formato de `switch` es el que se muestra a continuación. En legunaje natural sería algo así como "Si `variable` vale `valor1` entonces entra por `case valor1:`, si `variable` vale `valor2` entonces entra por `case valor2:`,... si `variable` no vale ninguno de los valores que hay en los distintos `case` entonces entra por `default:`.

```
    switch(variable) {
      case valor1:
```

```
      sentencias
      break;

    case valor2:
      sentencias
      break;
       .
       .
       .

    default:
      sentencias
  }
```

A continuación tienes un ejemplo completo en Java. Se pide al usuario un número de mes y el programa da el nombre del mes que corresponde a ese número.

```java
/**
 * Sentencia múltiple (switch)
 *
 * @author Luis José Sánchez
 */

public class SentenciaSwitch {
  public static void main(String[] args) {

    System.out.print("Por favor, introduzca un numero de mes: ");
    int mes = Integer.parseInt(System.console().readLine());

    String nombreDelMes;

    switch (mes) {
```



```
      case 1:
        nombreDelMes = "enero";
        break;
      case 2:
        nombreDelMes = "febrero";
        break;
      case 3:
        nombreDelMes = "marzo";
        break;
      case 4:
        nombreDelMes = "abril";
        break;
      case 5:
        nombreDelMes = "mayo";
        break;
      case 6:
        nombreDelMes = "junio";
        break;
```

```
      case 7:
        nombreDelMes = "julio";
        break;
      case 8:
        nombreDelMes = "agosto";
        break;
      case 9:
        nombreDelMes = "septiembre";
        break;
      case 10:
        nombreDelMes = "octubre";
        break;
      case 11:
        nombreDelMes = "noviembre";
        break;
      case 12:
        nombreDelMes = "diciembre";
        break;
      default:
        nombreDelMes = "no existe ese mes";
    }

    System.out.println("Mes " + mes + ": " + nombreDelMes);
  }
}
```


`nombreDelMes`. En caso de no encontrarse el `break`, el programa continúa la ejecución en la línea siguiente.

El bloque que corresponde al `default` se ejecuta cuando la variable no coincide con ninguno de los valores de los `case`. Escribiremos siempre el `default` al final de la sentencia `switch` aunque no sea necesario.

La sentencia `switch` se utiliza con frecuencia para crear menús.

```
/**
 * Ejemplo de un menú con switch
 *
 * @author Luis José Sánchez
 */

public class MenuConSwitch {
  public static void main(String[] args) {

    System.out.println(" CÁLCULO DE ÁREAS");
    System.out.println(" ----------------");
    System.out.println(" 1. Cuadrado");
    System.out.println(" 2. Rectángulo");
    System.out.println(" 3. Triángulo");
    System.out.print("\n Elija una opción (1-3): ");
```

```
    int opcion = Integer.parseInt(System.console().readLine());

    double lado;
    double base;
    double altura;

    switch (opcion) {
      case 1:
        System.out.print("\nIntroduzca el lado del cuadrado en cm: ");
        lado = Double.parseDouble(System.console().readLine());
        System.out.println("\nEl área del cuadrado es " + (lado * lado) + " cm2");
        break;

      case 2:
        System.out.print("\nIntroduzca la base del rectángulo en cm: ");
        base = Double.parseDouble(System.console().readLine());
        System.out.print("Introduzca la altura del rectángulo en cm: ");
        altura = Double.parseDouble(System.console().readLine());
        System.out.println("El área del rectángulo es " + (base * altura) + " cm2");
        break;
```



```
      case 3:
        System.out.print("\nIntroduzca la base del triángulo en cm: ");
        base = Double.parseDouble(System.console().readLine());
        System.out.print("Introduzca la altura del triángulo en cm: ");
        altura = Double.parseDouble(System.console().readLine());
        System.out.println("El área del triángulo es " + ((base * altura) / 2) + "
        cm2"); break;

      default:
        System.out.print("\nLo siento, la opción elegida no es correcta.");
    }
  }
}
```


4.5 Ejercicios

Ejercicio 1

Escribe un programa que pida por teclado un día de la semana y que diga qué asignatura toca a primera hora ese día.

Ejercicio 2

Realiza un programa que pida una hora por teclado y que muestre luego buenos días, buenas tardes o buenas noches según la hora. Se utilizarán los tramos de 6 a 12, de 13 a 20 y de 21 a 5. respectivamente. Sólo se tienen en cuenta las horas, los minutos no se deben introducir por teclado.

Ejercicio 3

Escribe un programa en que dado un número del 1 a 7 escriba el correspon diente nombre del día de la semana.

Ejercicio 4

Vamos a ampliar uno de los ejercicios de la relación anterior para considerar las horas extras. Escribe un programa que calcule el salario semanal de un trabajador teniendo en cuenta que las horas ordinarias (40 primeras horas de trabajo) se pagan a 12 euros la hora. A partir de la hora 41, se pagan a 16 euros la hora.

Ejercicio 5

Realiza un programa que resuelva una ecuación de primer grado (del tipo $ax + b = 0$).

Ejercicio 6

Realiza un programa que calcule el tiempo que tardará en caer un objeto desde una altura h. Aplica la fórmula $t = \sqrt{\frac{2h}{g}}$ siendo $g = 9.81 m/s^2$

Ejercicio 7

Realiza un programa que calcule la media de tres notas.

Ejercicio 8

Amplía el programa anterior para que diga la nota del boletín (insuficiente, suficiente, bien, notable o sobresaliente).

Ejercicio 9

Realiza un programa que resuelva una ecuación de segundo grado (del tipo $ax^2 + bx + c = 0$).

Ejercicio 10

Escribe un programa que nos diga el horóscopo a partir del día y el mes de nacimiento.

Ejercicio 11

Escribe un programa que dada una hora determinada (horas y minutos), calcule los segundos que faltan para llegar a la medianoche.

Ejercicio 12

Realiza un minicuestionario con 10 preguntas tipo test sobre las asignaturas que se imparten en el curso. Cada pregunta acertada sumará un punto. El programa mostrará al final la calificación obtenida. Pásale el minicuestionario a tus compañeros y pídeles que lo hagan para ver qué tal andan de conoci mientos en las diferentes asignaturas del curso.

Ejercicio 13

Escribe un programa que ordene tres números enteros introducidos por teclado.

Ejercicio 14

Realiza un programa que diga si un número introducido por teclado es par y/o divisible entre 5.

Ejercicio 15

Escribe un programa que pinte una pirámide rellena con un carácter introdu cido por teclado que podrá ser una letra, un número o un símbolo como *, +, -, $, &, etc. El programa debe permitir al usuario mediante un menú elegir si el vértice de la pirámide está apuntando hacia arriba, hacia abajo, hacia la izquierda o hacia la derecha.

Ejercicio 16

Realiza un programa que nos diga si hay probabilidad de que nuestra pareja nos está siendo infiel. El programa irá haciendo preguntas que el usuario contestará con verdadero o falso. Cada pregunta contestada como verdadero sumará 3 puntos. Las preguntas contestadas con falso no suman puntos. Utili za el fichero `test_infidelidad.txt` para obtener las preguntas y las conclusiones del programa.

Ejercicio 17

Escribe un programa que diga cuál es la última cifra de un número entero introducido por teclado.

Ejercicio 18

Escribe un programa que diga cuál es la primera cifra de un número entero introducido por teclado. Se permiten números de hasta 5 cifras.

Ejercicio 19

Realiza un programa que nos diga cuántos dígitos tiene un número entero que puede ser positivo o negativo. Se permiten números de hasta 5 dígitos.

Ejercicio 20

Realiza un programa que diga si un número entero positivo introducido por teclado es capicúa. Se permiten números de hasta 5 cifras.

Ejercicio 21

Calcula la nota de un trimestre de la asignatura **Programación**. El programa pedirá las dos notas que ha sacado el alumno en los dos primeros controles. Si la media de los dos controles da un número mayor o igual a 5, el alumno está aprobado y se mostrará la media. En caso de que la media sea un número menor que 5, el alumno habrá tenido que hacer el examen de recuperación que se califica como **apto** o **no apto**, por tanto se debe preguntar al usuario **¿Cuál ha sido el resultado de la recuperación? (apto/no apto)**. Si el resultado de la recuperación es **apto**, la nota será un 5; en caso contrario, se mantiene la nota media anterior.

Ejemplo 1:

```
Nota del primer control: 7
Nota del segundo control: 10
Tu nota de Programación es 8.5
```

Ejemplo 2:

```
Nota del primer control: 6
Nota del segundo control: 3
¿Cuál ha sido el resultado de la recuperación? (apto/no apto): apto
Tu nota de Programación es 5
```

Ejemplo 3:

```
Nota del primer control: 6
Nota del segundo control: 3
¿Cuál ha sido el resultado de la recuperación? (apto/no apto): no apto
Tu nota de Programación es 4.5
```

Ejercicio 22

Realiza un programa que, dado un día de la semana (de lunes a viernes) y una hora (horas y minutos), calcule cuántos minutos faltan para el fin de semana. Se considerará que el fin de semana comienza el viernes a las 15:00h. Se da por hecho que el usuario introducirá un día y hora correctos, anterior al viernes a las 15:00h.

5. Bucles

Los bucles se utilizan para repetir un conjunto de sentencias. Por ejemplo, imagina que es necesario introducir la notas de 40 alumnos con el fin de calcular la media, la nota máxima y la nota mínima. Podríamos escribir 40 veces la intrucción que pide un dato por teclado `System.console().readLine()` y convertir cada uno de esos datos a un número con decimales con 40 instrucciones `Double.parseDouble()`, no parece algo muy eficiente. Es mucho más práctico meter dentro de un bucle aquellas sentencias que queremos que se repitan.

Normalmente existe una condición de salida, que hace que el flujo del programa abandone el bucle y continúe justo en la siguiente sentencia. Si no existe condición de salida o si esta condición no se cumple nunca, se produciría lo que se llama un bucle infinito y el programa no terminaría nunca.

5.1 Bucle `for`

Se suele utilizar cuando se conoce previamente el número exacto de iteraciones (repeticiones) que se van a realizar. La sintaxis es la siguiente:

```
for (expresion1 ; expresion2 ; expresion3) {

    sentencias

}
```

Justo al principio se ejecuta `expresion1`, normalmente se usa para inicializar una variable. El bucle se repite mientras se cumple `expresion2` y en cada iteracción del bucle se ejecuta `expresion3`, que suele ser el incremento o decremento de una variable. Con un ejemplo se verá mucho más claro.

```
/**
 * Bucle for
 *
 * @author Luis José Sánchez
 */
public class EjemploFor {
  public static void main(String[] args) {

    for (int i = 1; i < 11; i++) {
      System.out.println(i);
```



```
    }
  }
}
```

En este ejemplo, `int i = 1` se ejecuta solo una vez, antes que cualquier otra cosa; como ves, esta expresión se utiliza para inicializar la variable `i` a 1. Mientra se cumpla la condición `i < 11` el contenido del bucle, o sea, `System.out.println(i);` se va a ejecutar. En cada iteración del bucle, `i++` hace que la variable `i` se incremente en 1. El resultado del ejemplo es la impresión

en pantalla de los números del 1 al 10.

Intenta seguir mentalmente el flujo del programa. Experimenta inicializando la variable `i` con otros valores, cambia la condición con `>` o `<=` y observa lo que sucede. Prueba también a cambiar el incremento de la variable `i`, por ejemplo con `i = i + 2`.

5.2 Bucle `while`

El bucle `while` se utiliza para repetir un conjunto de sentencias siempre que se cumpla una determinada condición. Es importante reseñar que la condición se comprueba al comienzo del bucle, por lo que se podría dar el caso de que dicho bucle no se ejecutase nunca. La sintaxis es la siguiente:

```
while (expresion) {

    sentencias

}
```

Las sentencias se ejecutan una y otra vez mientras `expresion` sea verdadera. El siguiente ejemplo produce la misma salida que el ejemplo anterior, muestra cómo cambian los valores de `i` del 1 al 10.

```
/**
 * Bucle while
 *
 * @author Luis José Sánchez
 */

public class EjemploWhile {
  public static void main(String[] args) {

    int i = 1;

    while (i < 11) {
      System.out.println(i);
```



```
      i++;
    }
  }
}
```

En el siguiente ejemplo se cuentan y se suman los números que se van introduciendo por teclado. Para indicarle al programa que debe dejar de pedir números, el usuario debe introducir un número negativo; esa será la condición de salida del bucle. Observa que el bucle se repite mientras el número introducido sea mayor o igual que cero.

```
/**
 * Bucle while que termina cuando se introduce por teclado un
 * número negativo.
```

```
 *
 * @author Luis José Sánchez
 */

public class CuentaPositivos {

  public static void main(String[] args) {

    System.out.println("Por favor, vaya introduciendo números y pulsando INTRO.");
    System.out.println("Para terminar, introduzca un número negativo.");

    int numeroIntroducido = 0;
    int cuentaNumeros = 0;
    int suma = 0;

    while (numeroIntroducido >= 0) {
      numeroIntroducido = Integer.parseInt(System.console().readLine());
      cuentaNumeros++; // Incrementa en uno la variable
      suma += numeroIntroducido; // Equivale a suma = suma + NumeroIntroducido
    }

    System.out.println("Has introducido " + (cuentaNumeros - 1) + " números
  positivos."); System.out.println("La suma total de ellos es " + (suma -
  numeroIntroducido)); }
}
```

5.3 Bucle `do-while`

El bucle `do-while` funciona de la misma manera que el bucle `while`, con la salvedad de que `expresion` se evalúa al final de la iteracción. Las sentencias que encierran el bucle `do-while`, por tanto, se ejecutan como mínimo una vez. La sintaxis es la siguiente:


```
    do {

        sentencias

    } while (expresion)
```

El siguiente ejemplo es el equivalente `do-while` a los dos ejemplos anteriores que cuentan del 1 al 10.

```
/**
 * Bucle do-while
 *
 * @author Luis José Sánchez
 */

public class EjemploDoWhile {
  public static void main(String[] args) {
```

```
    int i = 1;

    do {
      System.out.println(i);
      i++;
    } while (i < 11);
  }
}
```

Veamos otro ejemplo. En este caso se van a ir leyendo números de teclado mientras el número introducido sea par; el programa parará, por tanto, cuando se introduzca un número impar.

```
/**
 * Bucle do-while que termina cuando se introduce por teclado un
 * número impar.
 *
 * @author Luis José Sánchez
 */

public class TerminaCuandoEsImpar {

  public static void main(String[] args) {

    int numero;
```



```
    do {
      System.out.print("Dime un número: ");
      numero = Integer.parseInt(System.console().readLine());

      if (numero % 2 == 0) {// comprueba si el número introducido es
        par System.out.println("Qué bonito es el " + numero);
      } else {
        System.out.println("No me gustan los números impares, adiós.");
      }
    } while (numero % 2 == 0);
  }
}
```

Te invito a que realices una modificación sobre este último ejemplo. Después de pedir un número, haz que el programa diga `¿Quiere continuar? (s/n)`. Si el usuario introduce una `s` o una `S`, el programa deberá continuar pidiendo números.

Cualquier bucle que un programador quiera realizar en Java lo puede implementar con `for`, `while` o `do-while`; por tanto, con una sola de estas tres opciones tendría suficiente. No obstante, según el programa en cuestión, una u otra posibilidad se adapta mejor que las otras y resulta más elegante. Con la experiencia, te irás dando cuenta cuándo es mejor utilizar cada una.

5.4 Ejercicios

Ejercicio 1

Muestra los números múltiplos de 5 de 0 a 100 utilizando un bucle `for`.

Ejercicio 2

Muestra los números múltiplos de 5 de 0 a 100 utilizando un bucle `while`.

Ejercicio 3

Muestra los números múltiplos de 5 de 0 a 100 utilizando un bucle `do-while`.

Ejercicio 4

Muestra los números del 320 al 160, contando de 20 en 20 hacia atrás utilizando un bucle `for`.

Ejercicio 5

Muestra los números del 320 al 160, contando de 20 en 20 hacia atrás utilizando un bucle `while`.

Ejercicio 6

Muestra los números del 320 al 160, contando de 20 en 20 utilizando un bucle `do-while`.

Ejercicio 7

Realiza el control de acceso a una caja fuerte. La combinación será un número de 4 cifras. El programa nos pedirá la combinación para abrirla. Si no acertamos, se nos mostrará el mensaje “Lo siento, esa no es la combinación” y si acertamos se nos dirá “La caja fuerte se ha abierto satisfactoriamente”. Tendremos cuatro oportunidades para abrir la caja fuerte.

Ejercicio 8

Muestra la tabla de multiplicar de un número introducido por teclado.

Ejercicio 9

Realiza un programa que nos diga cuántos dígitos tiene un número introducido por teclado.

Ejercicio 10

Escribe un programa que calcule la media de un conjunto de números positivos introducidos por teclado. A priori, el programa no sabe cuántos números se introducirán. El usuario indicará que ha terminado de introducir los datos cuando meta un número negativo.

Ejercicio 11

Escribe un programa que muestre en tres columnas, el cuadrado y el cubo de los 5 primeros números enteros a partir de uno que se introduce por teclado.

Ejercicio 12

Escribe un programa que muestre los n primeros términos de la serie de Fibonacci. El primer término de la serie de Fibonacci es 0, el segundo es 1 y el resto se calcula sumando los dos anteriores, por lo que tendríamos que los términos son 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144… El número n se debe introducir por teclado.

Ejercicio 13

Escribe un programa que lea una lista de diez números y determine cuántos son positivos, y cuántos son negativos.

Ejercicio 14

Escribe un programa que pida una base y un exponente (entero positivo) y que calcule la potencia.

Ejercicio 15

Escribe un programa que dados dos números, uno real (base) y un entero positivo (exponente), saque por pantalla todas las potencias con base el numero dado y exponentes entre uno y el exponente introducido. No se deben utilizar funciones de exponenciación. Por ejemplo, si introducimos el 2 y el 5, se deberán mostrar 2^1, 2^2, 2^3, 2^4 y 2^5.

Ejercicio 16

Escribe un programa que diga si un número introducido por teclado es o no primo. Un número primo es aquel que sólo es divisible entre él mismo y la unidad.

Ejercicio 17

Realiza un programa que sume los 100 números siguientes a un número entero y positivo introducido por teclado. Se debe comprobar que el dato introducido es correcto (que es un número positivo).

Ejercicio 18

Escribe un programa que obtenga los números enteros comprendidos entre dos números introducidos por teclado y validados como distintos, el programa debe empezar por el menor de los enteros introducidos e ir incrementando de 7 en 7.

Ejercicio 19

Realiza un programa que pinte una pirámide por pantalla. La altura se debe pedir por teclado. El carácter con el que se pinta la pirámide también se debe pedir por teclado.

Ejercicio 20

Igual que el ejercicio anterior pero esta vez se debe pintar una pirámide hueca.

Ejercicio 21

Realiza un programa que vaya pidiendo números hasta que se introduzca un numero negativo y nos diga cuantos números se han introducido, la media de los impares y el mayor de los pares. El número negativo sólo se utiliza para indicar el final de la introducción de datos pero no se incluye en el cómputo.

Ejercicio 22

Muestra por pantalla todos los números primos entre 2 y 100, ambos incluidos.

Ejercicio 23

Escribe un programa que permita ir introduciendo una serie indeterminada de números mientras su suma no supere el valor 10000. Cuando esto último ocurra, se debe mostrar el total acumulado, el contador de los números introducidos y la media.

Ejercicio 24

Escribe un programa que lea un número `n` e imprima una pirámide de números con `n` filas como en la siguiente figura:

```
1
121
12321
1234321
```

Ejercicio 25

Realiza un programa que pida un número por teclado y que luego muestre ese número al revés.

Ejercicio 26

Realiza un programa que pida primero un número y a continuación un dígito. El programa nos debe dar la posición (o posiciones) contando de izquierda a derecha que ocupa ese dígito en el número introducido.

Ejercicio 27

Escribe un programa que muestre, cuente y sume los múltiplos de 3 que hay entre 1 y un número leído por teclado.

Ejercicio 28

Escribe un programa que calcule el factorial de un número entero leído por teclado.

Ejercicio 29

Escribe un programa que muestre por pantalla todos los números enteros positivos menores a uno leído por teclado que no sean divisibles entre otro también leído de igual forma.

Ejercicio 30

Realiza una programa que calcule las horas transcurridas entre dos horas de dos días de la semana. No se tendrán en cuenta los minutos ni los segundos. El día de la semana se puede pedir como un número (del 1 al 7) o como una cadena (de “lunes” a “domingo”). Se debe comprobar que el usuario introduce los datos correctamente y que el segundo día es posterior al primero. A continuación se muestra un ejemplo:

```
Por favor, introduzca la primera hora.
Día: lunes
Hora: 18
Por favor, introduzca la segunda hora.
Día: martes
Hora: 20
Entre las 18:00h del lunes y las 20:00h del martes hay 26 hora/s.
```

Ejercicio 31

Realiza un programa que pinte la letra **L** por pantalla hecha con asteriscos. El programa pedirá la altura. El palo horizontal de la **L** tendrá una longitud de la mitad (división entera entre 2) de la altura más uno.

Ejemplo:

```
Introduzca la altura de la L: 5
*
*
*
*
```

```
* * *
```


Ejercicio 32

Escribe un programa que, dado un número entero, diga cuáles son y cuánto suman los dígitos pares. Los dígitos pares se deben mostrar en orden, de izquierda a derecha. Usa `long` en lugar de `int` donde sea necesario para admitir números largos.

Ejemplo 1:

```
Por favor, introduzca un número entero positivo: 94026782
Dígitos pares: 4 0 2 6 8 2
Suma de los dígitos pares: 22
```

Ejemplo 2:

```
Por favor, introduzca un número entero positivo: 31779
Dígitos pares:
Suma de los dígitos pares: 0
```

Ejemplo 3:

```
Por favor, introduzca un número entero positivo: 2404
Dígitos pares: 2 4 0 4
Suma de los dígitos pares: 10
```

Ejercicio 33

Realiza un programa que pinte la letra **U** por pantalla hecha con asteriscos. El programa pedirá la altura. Fíjate que el programa inserta un espacio y pinta dos asteriscos menos en la base para simular la curvatura de las esquinas inferiores.

Ejemplo:

```
Introduzca la altura de la U: 5
 * *
 * *
 * *
 * *
   * * *
```


Ejercicio 34

Escribe un programa que pida dos números por teclado y que luego mezcle en dos números diferentes los dígitos pares y los impares. Se van comprobando los dígitos de la siguiente manera: primer dígito del primer número, primer dígito del segundo número, segundo dígito del primer número, segundo dígito del segundo número, tercer dígito del primer número… Para facilitar el ejercicio, podemos suponer que el usuario introducirá dos números de la misma longitud y que siempre habrá al menos un dígito par y uno impar. Usa `long` en lugar de `int` donde sea necesario para admitir números largos.

Ejemplo 1:

```
Por favor, introduzca un número: 9402
```

```
Introduzca otro número: 6782
El número formado por los dígitos pares es 640822
El número formado por los dígitos impares es 97
```

Ejemplo 2:

```
Por favor, introduzca un número: 137
Introduzca otro número: 909
El número formado por los dígitos pares es 0
El número formado por los dígitos impares es 19379
```

6. Números aleatorios

Los números aleatorios se utilizan con frecuencia en programación para emular el comportamiento de algún fenómeno natural, el resultado de un juego de azar o, en general, para generar cualquier valor impredecible *a priori*.

Por ejemplo, se pueden utilizar números aleatorios para generar tiradas de dados de tal forma que, de antemano, no se puede saber el resultado. Antes de tirar un dado no sabemos si saldrá un 3 o un 5; se tratará pues de un número impredecible; lo que sí sabemos es que saldrá un número entre el 1 y el 6, es decir, podemos acotar el rango de los valores que vamos a obtener de forma aleatoria.

6.1 Generación de números aleatorios con y sin decimales

Para generar valores aleatorios utilizaremos `Math.random()`. Esta función genera un número con decimales (de tipo `double`) en el intervalo [0 - 1), es decir, genera un número mayor o igual que 0 y menor que 1.

El siguiente programa genera diez números aleatorios:

```java
/**
* Generación de números aleatorios.
*
* @author Luis José Sánchez
**/

public class Aleatorio01 {
  public static void main(String[] args) {

    System.out.println("Diez números aleatorios:\n");

    for (int i = 1; i < 11; i++) {
      System.out.println(Math.random());
    }
  }
}
```

La salida del programa puede ser algo como esto:

```
0.30830376099567813
0.8330493363981046
0.2322483682676435
0.3130746053770094
0.34192944433558736
0.9636470440975567
0.3398896383918959
0.79825461825305
0.9868509622870223
0.9967893773101499
```

Te invito a que ejecutes varias veces el programa. Podrás observar que cada vez salen números diferentes, aunque siempre están comprendidos entre 0 y 1 (incluyendo el 0).

Pensarás que no es muy útil generar números aleatorios entre 0 y 1 si lo que queremos es por ejemplo sacar una carta al azar de la baraja española; pero en realidad un número decimal entre 0 y 1 es lo único que nos hace falta para generar cualquier tipo de valor aleatorio siempre y cuando se manipule ese número de la forma adecuada.

Por ejemplo, si queremos generar valores aleatorios entre 0 y 10 (incluyendo el 0 y sin llegar a 10) simplemente tendremos que correr la coma un lugar o, lo que es lo mismo, multiplicar por 10.

```
/**
* Generación de números aleatorios.
*
* @author Luis José Sánchez
**/

public class Aleatorio02 {
  public static void main(String[] args) {

    System.out.println("20 números aleatorios entre 0 y 10");
    System.out.println(" sin llegar a 10 (con decimales):");

    for (int i = 1; i <= 20; i++) {
      System.out.println( Math.random()*10 + " ");
    }
  }
}
```

El programa anterior genera una salida como ésta:


```
0.07637674702636321
```

```
1.025787417143682
1.854993461897163
5.690351111720931
3.82310645589797
5.518007662236258
9.23529380254256
3.9201032643833376
5.836554253122096
6.224559064261578
7.652976185871555
4.9922807025365135
7.498156441347868
8.743251697509109
9.727764845406675
2.929766691797686
0.05801413446517634
2.1575652936687284
```

Si queremos generar números enteros en lugar de números con decimales, basta con hacer un casting para convertir los números de tipo `double` en números de tipo `int`. Recuerda que `(int)x` transforma `x` en una variable de tipo entero; si `x` era de tipo `float` o `double`, perdería todos los decimales.

```
/**
* Generación de números aleatorios.
*
* @author Luis José Sánchez
**/

public class Aleatorio03 {
  public static void main(String[] args) {

    System.out.println("20 números aleatorios entre 0 y 9 (sin decimales):");

    for (int i = 1; i <= 20; i++) {
      System.out.print((int)(Math.random()*10) + " ");
    }

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

La salida del programa debe ser algo muy parecido a esto:


```
20 números aleatorios entre 0 y 9 (sin decimales):
0 8 0 3 8 8 7 3 2 0 8 2 1 2 9 0 6 4 5 4
```

¿Y si en lugar de generar números enteros entre 0 y 9 queremos generar números entre 1 y 10? Como habrás podido adivinar, simplemente habría que sumar 1 al número generado, de esta forma se “desplazan un paso” los valores generados al azar, de tal forma que el mínimo valor que se produce sería el 0 + 1 = 1 y el máximo sería 9 + 1 = 10.

```java
/**
* Generación de números aleatorios.
*
* @author Luis José Sánchez
**/

public class Aleatorio04 {

  public static void main(String[] args) {

    System.out.println("20 números aleatorios entre 1 y 10 (sin decimales):");

    for (int i = 1; i <= 20; i++) {
      System.out.print( (int)(Math.random()*10 + 1) + " ");
    }

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

Vamos a ponerlo un poco más difícil. Ahora vamos a generar números enteros entre 50 y 60 ambos incluidos. Primero multiplicamos `Math.random()` por 11, con lo que obtenemos números decimales entre 0 y 10.9999… (sin llegar nunca hasta 11). Luego desplazamos ese intervalo sumando 50 por lo que obtenemos números decimales entre 50 y 60.9999… Por último, quitamos los decimales haciendo casting y *voilà*, ya tenemos números enteros aleatorios entre 50 y 60 ambos incluidos.


```java
/**
* Generación de números aleatorios.
*
* @author Luis José Sánchez
*/

public class Aleatorio05 {
  public static void main(String[] args) {

    System.out.println("20 números aleatorios entre 50 y 60 (sin decimales):");

    for (int i = 1; i <= 20; i++) {
      System.out.print(((int)(Math.random()*11) + 50 ) + " ");
    }

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

6.2 Generación de palabras de forma aleatoria de un conjunto dado

Hemos visto cómo generar números aleatorios con y sin decimales y en diferentes intervalos. Vamos a producir ahora de forma aleatoria una palabra - piedra, papel o tijera - generando primero un número entero entre 0 y 2 y posteriormente haciendo corresponder una palabra a cada número.

```
/**
* Generación de números aleatorios.
*
* @author Luis José Sánchez
*/

public class Aleatorio06 {

  public static void main(String[] args) {

    System.out.println("Genera al azar piedra, papel o tijera:");

    int mano = (int)(Math.random()*3); // genera un número al azar
                                       // entre 0 y 2 ambos incluidos
    switch(mano) {
      case 0:
        System.out.println("piedra");
```



```
        break;
      case 1:
        System.out.println("papel");
        break;
      case 2:
        System.out.println("tijera");
        break;
      default:
    }
  }
}
```

¿Cómo podríamos generar un día de la semana de forma aleatoria? En efecto, primero generamos un número entre 1 y 7 ambos inclusive y luego hacemos corresponder un día de la semana a cada uno de los números.

```
/**
* Generación de números aleatorios.
*
* @author Luis José Sánchez
*/

public class Aleatorio07 {
  public static void main(String[] args) {

    System.out.println("Muestra un día de la semana al azar:");
```

```
    int dia = (int)(Math.random()*7) + 1; // genera un número aleatorio
                                          // entre el 1 y el 7
    switch(dia) {
      case 1:
        System.out.println("lunes");
        break;
      case 2:
        System.out.println("martes");
        break;
      case 3:
        System.out.println("miércoles"); break;
      case 4:
        System.out.println("jueves");
        break;
      case 5:
        System.out.println("viernes");
        break;
      case 6:
```



```
        System.out.println("sábado");
        break;
      case7:
        System.out.println("domingo");
        break;
      default:
    }
  }
}
```


6.3 Ejercicios

Ejercicio 1

Escribe un programa que muestre la tirada de tres dados. Se debe mostrar también la suma total (los puntos que suman entre los tres dados).

Ejercicio 2

Realiza un programa que muestre al azar el nombre de una carta de la baraja francesa. Esta baraja está dividida en cuatro palos: picas, corazones, diamantes y tréboles. Cada palo está formado por 13 cartas, de las cuales 9 cartas son numerales y 4 literales: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K y A (que sería el 1). Para convertir un número en una cadena de caracteres podemos usar `String.valueOf(n)`.

Ejercicio 3

Igual que el ejercicio anterior pero con la baraja española. Se utilizará la baraja de 40 cartas: 2, 3, 4, 5, 6, 7, sota, caballo, rey y as.

Ejercicio 4

Muestra 20 números enteros aleatorios entre 0 y 10 (ambos incluidos) sepa rados por espacios.

Ejercicio 5

Muestra 50 números enteros aleatorios entre 100 y 199 (ambos incluidos) separados por espacios. Muestra también el máximo, el mínimo y la media de esos números.

Ejercicio 6

Escribe un programa que piense un número al azar entre 0 y 100. El usuario debe adivinarlo y tiene para ello 5 oportunidades. Después de cada intento fallido, el programa dirá cuántas oportunidades quedan y si el número intro ducido es menor o mayor que el número secreto.

Ejercicio 7

Escribe un programa que muestre tres apuestas de la quiniela en tres colum nas para los 14 partidos y el pleno al quince (15 filas).

Ejercicio 8

Modifica el programa anterior para que la probabilidad de que salga un 1 sea de 1/2, la probabilidad de que salga `x` sea de 1/3 y la probabilidad de que salga 2 sea de 1/6. Pista: 1/2 = 3/6 y 1/3 = 2/6.

Ejercicio 9

Realiza un programa que vaya generando números aleatorios pares entre 0 y 100 y que no termine de generar números hasta que no saque el 24. El programa deberá decir al final cuántos números se han generado.

Ejercicio 10

Realiza un programa que pinte por pantalla diez líneas formadas por carac teres. El carácter con el que se pinta cada línea se elige de forma aleatoria entre uno de los siguientes: *, -, =, ., |, @. Las líneas deben tener una longitud aleatoria entre 1 y 40 caracteres.

Ejercicio 11

Escribe un programa que muestre 20 notas generadas al azar. Las notas deben aparecer de la forma: suspenso, suficiente, bien, notable o sobresaliente. Al final aparecerá el número de suspensos, el número de suficientes, el número de bienes, etc.

Ejercicio 12

Realiza un programa que llene la pantalla de caracteres aleatorios (a lo Matrix) con el código ascii entre el 32 y el 126. Puedes hacer casting con `(char)` para convertir un entero en un carácter.

Ejercicio 13

Escribe un programa que simule la tirada de dos dados. El programa deberá continuar tirando los dados una y otra vez hasta que en alguna tirada los dos dados tengan el mismo valor.

Ejercicio 14

Realiza un programa que haga justo lo contrario a lo que hace el ejercicio 6. El programa intentará adivinar el número que estás pensando - un número entre 0 y 100 - teniendo para ello 5 oportunidades. En cada intento fallido, el programa debe preguntar si el número que estás pensando es mayor o menor que el que te acaba de decir.

Ejercicio 15

Realiza un generador de melodía con las siguientes condiciones:

a) Las notas deben estar generadas al azar. Las 7 notas son **do**, **re**, **mi**, **fa**, **sol**, **la** y **si**.

b) Una melodía está formada por un número aleatorio de notas mayor o igual a 4, menor o igual a 28 y siempre múltiplo de 4 (4, 8, 12…).

c) Cada grupo de 4 notas será un compás y estará separado del siguiente compás mediante la barra vertical "|" (Alt + 1). El final de la melodía se marca con dos barras.

d) La última nota de la melodía debe coincidir con la primera.

Ejemplo 1:

```
do mi fa mi | si do sol fa | fa re si do | sol mi re do ||
```

Ejemplo 2:

```
la re mi sol | fa mi mi si | do la sol fa | fa re si sol | do sol mi re | fa la do la ||
```


Ejercicio 16

Realiza un simulador de máquina tragaperras simplificada que cumpla los siguientes requisitos:

a) El ordenador mostrará una tirada que consiste en mostrar 3 figuras. Hay 5 figuras posibles: **corazón**, **diamante**, **herradura**, **campana** y **limón**.

b) Si las tres figuras son diferentes se debe mostrar el mensaje **"Lo siento, ha perdido"**.

c) Si hay dos figuras iguales y una diferente se debe mostrar el mensaje **"Bien, ha recuperado su moneda"**.

d) Si las tres figuras son iguales se debe mostrar **"Enhorabuena, ha ganado 10 monedas"**.

Ejemplo 1:

```
diamante diamante limón
Bien, ha recuperado su moneda
```

Ejemplo 2:

```
herradura campana diamante
Lo siento, ha perdido
```

Ejemplo 3:

```
corazón corazón corazón
Enhorabuena, ha ganado 10 monedas
```

7. Arrays

7.1 Arrays de una dimensión

Un *array* es un tipo de dato capaz de almacenar múltiples valores. Se utiliza para agrupar datos muy parecidos, por ejemplo, si se necesita almacenar la temperatura media diaria en Málaga durante el último año se pueden utilizar las variables `temp0`, `temp1`, `temp2`, `temp3`, `temp4`, ... y así hasta 365 variables distintas pero sería poco práctico; es mejor utilizar un *array* de nombre `temp` y usar un índice para referanciar la temperatura de un día concreto del año.

En matemáticas, un *array* de una dimensión se llama vector. En este libro no utilizaremos este término para no confundirlo con la clase `Vector` de Java.

Veamos con un ejemplo cómo se crea y se utiliza un *array*.

```
/**
 * Ejemplo de uso de arrays
 *
 * @author Luis José Sánchez
 */

public class Array01 {
  public static void main(String[] args) {

    int[] n; // se define n como un array de enteros
    n = new int[4]; // se reserva espacio para 4 enteros

    n[0] = 26;
    n[1] = -30;
    n[2] = 0;
    n[3] = 100;

    System.out.print("Los valores del array n son los siguientes: ");
    System.out.print(n[0] + ", " + n[1] + ", " + n[2] + ", " + n[3]);

    int suma = n[0] + n[3];
    System.out.println("\nEl primer elemento del array más el último suman " +
  suma); }
}
```

Observa que el índice de cada elemento de un *array* se indica entre corchetes de tal forma que `n[3]` es el cuarto elemento ya que el primer índice es el 0.

Array n

Índice	Valor
0	26
1	-30
2	0
3	11

Fíjate que la definición del *array* y la reserva de memoria para los cuatro elementos que la componen se ha realizado en dos líneas diferentes. Se pueden hacer ambas cosas en una sola línea como veremos en el siguiente ejemplo.

```
/**
 * Ejemplo de uso de arrays
 *
 * @author Luis José Sánchez
 */

public class Array02 {
  public static void main(String[] args) {

    // definición del array y reserva de memoria en la misma línea
    int[] x = new int[5];

    x[0] = 8;
    x[1] = 33;
    x[2] = 200;
    x[3] = 150;
    x[4] = 11;

    System.out.println("El array x tiene 5 elementos ¿cuál de ellos quiere ver?");
    System.out.print("Introduzca un número del 0 al 4: ");
    int indice = Integer.parseInt(System.console().readLine());
    System.out.print("El elemento que se encuentra en la posición " + indice);
    System.out.println(" es el " + x[indice]);
  }
}
```

El *array* `x` del ejemplo anterior se ha ido rellenando elemento a elemento. Si se conocen previamente todos los valores iniciales del array, se puede crear e inicializar en una sola línea de la siguiente forma

```
int[] x = {8, 33, 200, 150, 11};
```

Cada elemento del *array* se puede utilizar exactamente igual que cualquier otra variable, es decir, se le puede asignar un valor o se puede usar dentro de una

expresión. En el siguiente ejemplo se muestran varias operaciones en las que los operandos son elementos del *array* `num`.

Para recorrer todos los elementos de un *array* se suele utilizar un bucle for junto con un índice que va desde 0 hasta el tamaño del *array* menos 1.

```
/**
 * Ejemplo de uso de arrays
 *
 * @author Luis José Sánchez
 */

public class Array03 {
  public static void main(String[] args) {

    int[] num = new int[10];

    num[0] = 8;
    num[1] = 33;
    num[2] = 200;
    num[3] = 150;
    num[4] = 11;
    num[5] = 88;
    num[6] = num[2] * 10;
    num[7] = num[2] / 10;
    num[8] = num[0] + num[1] + num[2];
    num[9] = num[8];

    System.out.println("El array num contiene los siguientes elementos:");

    for (int i = 0; i < 10; i++) {
      System.out.println(num[i]);
    }
  }
}
```

En Java, a diferencia de otros lenguajes como Ruby o PHP, todos los elementos de un *array* deben ser del mismo tipo; por ejemplo, no puede haber un entero en la posición 2 y una cadena de caracteres en la posición 7 del mismo *array*. En el siguiente ejemplo se muestra un *array* de caracteres.


```
/**
 * Ejemplo de uso de arrays
 *
 * @author Luis José Sánchez
 */

public class Array04 {
  public static void main(String[] args) {
```

```
    char[] caracter = new char[6];

    caracter[0] = 'R';
    caracter[1] = '%';
    caracter[2] = '&';
    caracter[3] = '+';
    caracter[4] = 'A';
    caracter[5] = '2';

    System.out.println("El array caracter contiene los siguientes elementos:");

    for (int i = 0; i < 6; i++) {
      System.out.println(caracter[i]);
    }
  }
}
```

En el siguiente ejemplo se muestra un *array* de números de tipo `double` que almacena notas de alumnos.

```
/**
 * Ejemplo de uso de arrays
 *
 * @author Luis José Sánchez
 */

public class Array05 {

  public static void main(String[] args) {

    double[] nota = new double[4];

    System.out.println("Para calcular la nota media necesito saber la ");
    System.out.println("nota de cada uno de tus exámenes.");

    for (int i = 0; i < 4; i++) {
```