capítulo 3

Elementos básicos de Java

objetivos

En este capítulo aprenderá a:

- Definir los tipos básicos del lenguaje.
- Formar identificadores válidos.
- Escribir expresiones en Java.
- Conocer los operadores aritméticos
- Escribir la sintaxis de expresiones aritméticas que representen fórmulas matemáticas sencillas.
- Editar un programa Java sencillo, así como la compilación y ejecución del mismo.
- Realizar la entrada básica de datos desde el teclado.
- Mostrar por pantalla datos desde un programa.

introducción

Hemos visto cómo crear programas propios, ahora analizaremos los fundamentos de Java. Debido a su gran importancia en el desarrollo de aplicaciones, en este capítulo se repasan los conceptos teóricos y prácticos relativos a la estructura de un programa enunciados en el capítulo anterior, incluyendo los siguientes temas:

- Estructura general de un programa en Java.
- Creación del programa.
- Elementos básicos que lo componen.
- Tipos de datos en Java y cómo se declaran.
- Tipos de datos enumerados (nuevos desde las versiones 5.0 y 6.0 de Java), concepto y declaración de constantes.
- Concepto y declaración de variables.
- Tiempo de vida o duración de variables.
- Operaciones básicas de entrada/salida.

3.1 Estructura general de un programa en Java

Esta sección repasa los elementos que constituyen un programa escrito en Java, fijando y describiendo ideas relativas a su estructura; cada programa se compone de una o más

clases y obligatoriamente `main()` debe ser uno de los métodos de la clase principal; un método en Java es un grupo de instrucciones que realizan una o más acciones. Por otro lado, el programa debe contener una serie de declaraciones `import` que permitan incluir archivos que consten de clases y datos predefinidos. De modo concreto, un programa en Java puede incluir:

- Declaraciones para importar clases de los paquetes.
- Declaraciones de clases.
- El método `main()`.
- Métodos definidos por el usuario dentro de las clases.
- Comentarios del programa (utilizados en su totalidad).

La estructura típica completa de un programa en Java se muestra en la figura 3.1; a continuación, un ejemplo de un programa sencillo en Java.

```
import java.io.*;                          ← Archivo de clases de entrada/salida
public class nombrePrograma                ← Nombre de la clase principal
{
 public static void main(String []ar)      ← Cabecera del método
{                                          ← Nombre del método
...                                        ← Sentencias
}
...
}
```

```
import java.io.*;

//Listado DemoUno.java. Programa de saludo
//Este programa imprime "Bienvenido a la programación en Java

class DemoUno
{
  public static void main(String[] ar)
  {
    System.out.println("Bienvenido a la programación en Java\n");
  }
}
```

La declaración `import` de la primera línea es necesaria para que el programa pueda utilizar las clases de entrada y salida; esta declaración se refiere a un archivo externo, un paquete denominado `java.io` en el que se almacenan clases y objetos relativos a entrada y salida; observe que el asterisco (`*`) se utiliza para indicar la importación de todos los elementos del paquete `java.io`.

La segunda y tercera líneas están conformadas por comentarios identificados por barras dobles inclinadas (`//`), los cuales se incluyen en los programas para proporcionar explicaciones a los usuarios y son ignorados por el compilador; la tercera línea contiene la cabecera de la clase `DemoUno`.

Un programa en Java se puede considerar una colección de clases, en la que al menos una de ellas tenga el mismo nombre que el archivo fuente (`DemoUno.Java`) e incluya de manera obligatoria al método `main()`, también indica el comienzo del programa y requiere la sintaxis:

```
public static void main(String []ar)
```

Nombre del programa:

El nombre del archivo fuente ha de coincidir con el nombre de la clase principal (clase que contiene al método `main()`). Así se puede tener: `nombrePrograma.java`

`import` Declaración para importar clases desde paquetes.
`public static void main()` Método por el que empieza la ejecución; Java exige esta sintaxis.

Método principal *`main`*

```
public static void main(String[] ar)
{
  declaraciones locales
  sentencias
}
```

Definiciones de otros métodos dentro de la clase

```
static tipo func1(...)
{
 ...
}

static tipo func2(...)
{
 ...
}
 ...
```

Figura 3.1 Estructura típica de un programa Java.

El argumento de `main()` es una colección de cadenas que recoge datos de la línea donde se ejecuta el programa; después de `main()` hay una línea que sólo contiene una llave (`{`) que encierra el cuerpo del método y que es necesaria en todos los programas de Java.

Continúa la sentencia:

```
System.out.println("Bienvenido a la programación en Java\n");
```

que ordena al sistema enviar el mensaje `"Bienvenido a la programación en Java\n"` para impresión en el flujo estándar de salida, que normalmente es la pantalla de la computadora. La salida será:

```
Bienvenido a la programación en Java
```

El símbolo `"\n"` señala una línea nueva y al ponerlo al final de la cadena entre comillas ordena al sistema que comience una nueva línea después de imprimir los caracteres precedentes, terminando así, la línea actual.

Es importante notar que el punto y coma (`;`) se coloca al final de esa misma línea porque Java requiere que cada sentencia termine así; es posible poner varias sentencias en la misma línea o que una sentencia se extienda sobre varias líneas.

A TOMAR EN CUENTA

- El programa más corto de Java es el "programa vacío" que no hace nada.
- Puede que no haya argumentos en la línea de órdenes; en cualquier caso es obligatorio especificar `String []`.

Por último, la llave (}) cierra el bloque abierto previamente por ({); en este caso se abrieron un par de bloques: el del método `main()` y el del cuerpo de la clase, por eso hay dos llaves de cierre.

3.1.1 Declaración `import`

En Java, las clases se agrupan en paquetes (*packages*) que definen utilidades o grupos temáticos, y que se encuentran en directorios del disco con su mismo nombre. Para incorporar y utilizar las clases de un paquete en un programa se utiliza la declaración `import`; por ejemplo: para indicar al compilador que agregue la clase `Graphics` del paquete `awt` debe escribir:

```
import java.awt.Graphics;
```

La sintaxis general de la declaración `import` es:

```
import nombrePaquete.nombreClase;
```

`nombreClase` es el identificador de una clase del paquete; pueden incorporarse varias clases con una secuencia de sentencias `import`; por ejemplo, para incorporar las clases `Random`, `Date` y `Math` del paquete `java.util` se debe escribir:

```
import java.util.Random;
import java.util.Date;
import java.util.Math;
```

Se puede especificar que se incorporen todas las clases públicas de un paquete, en ese caso `nombreClase` se sustituye por `*`; así, para que un programa utilice cualquier clase del paquete `io` se debe escribir:

```
import java.io.*;
```

El paquete `java.lang` contiene elementos básicos para construir un programa: objetos definidos para entrada y salida básica, excepciones predefinidas, tipos de datos y, en general, utilidades para su construcción; debido a esto, el compilador siempre incorpora este paquete, haciendo innecesario escribir la declaración `import java.lang.*`.

ADVERTENCIA

Se recomienda incorporar sólo las clases de los paquetes que el programa utiliza pues esto da mayor claridad al programa, además la compilación es más lenta al incorporar más clases aunque no se usen.

El programador puede definir sus paquetes para agrupar las clases creadas; las declaraciones `import` también sirven para incorporar clases de un paquete creado por el programador; por ejemplo: si ha definido un paquete `casa`, con las clases `Climatizador`, `Computador`, `nevera`, `microondas`, se pueden hacer las declaraciones:

```
import casa.Computador;
import casa.Nevera;
```

o bien, para incorporar todas las clases:

```
import casa.*;
```

Se acostumbra escribir las declaraciones `import` en la cabecera del programa; así se puede utilizar a lo largo de todo el fichero donde se encuentre dicho programa.

El paquete `java.io` (Java input/output) proporciona clases que permiten realizar operaciones de entrada y salida; como casi todos los programas que escriba imprimirán información en pantalla y leerán datos del teclado, éstos necesitarán incluir clases propias; lo que implica que cada programa contenga la línea siguiente:

```
import java.io.*;
```

3.1.2 Declaración de clases

Como se ha dicho, el programa debe tener al menos una clase, la principal, que incluya el método `main()` y si es necesario, otros métodos y variables; para declararla es opcional empezar con una palabra clave, generalmente indicando el acceso; seguida por su indicador, la palabra reservada `class`, su nombre y sus miembros: variables y métodos; por ejemplo:

```
class Potencia
{
  int n, p;
  public static void main(String [] ar)
  {
    int r, e;
    int n, p;

    n = 7;
    p = e = 5;
    r = 1;
    for ( ; p > 0; p--)
      r = r*n;
    System.out.println("Potencia de " + n + "^" + e + " = " + r);
  }
}
```

El archivo donde se guarde el programa anterior debe tener como nombre `potencia.java`; el nombre del archivo fuente siempre será el mismo que el de la clase principal, es decir, la que contiene `main()`, y la extensión `java`.

```
class Sumatorio
{
  int n, p;
  public int sumar()
  {
    int k, e;

    n = 71;
    p = e = 5;
    return n+k+e+p;
  }
}
```

Las declaraciones dentro de una clase indican al compilador que los métodos definidos por el usuario o variables son comunes a todos los miembros de su clase. En la clase `Sumatorio` las variables `n, p` se pueden utilizar en cualquier método de la clase; sin embargo, `k` y `e`, situados en el método `sumar()`, sólo se pueden utilizar dentro de ese método.

3.1.3 Método `main()`

Cada programa Java tiene un método `main()` como punto inicial de entrada al programa cuya estructura es:

```
public static void main(String [] ar)
{
  ...              bloque de sentencias
}
```

NOTA

Las sentencias incluidas entre las llaves ({ . . . }) se denominan *bloque*.

Un programa puede tener sólo un método `main()`; hacer dos métodos `main()` produce un error, aunque estén en clases diferentes.

El argumento de `main()` es una colección (*array*) de cadenas que permiten introducir datos sucesivos de caracteres en la línea de ejecución del programa; por ejemplo, suponga que tiene el programa `Nombres`, entonces la ejecución puede ser:

```
java Nombres Luis Gerardo Fernando
```

En esta ejecución `Luis`, `Gerardo` y `Fernando` están asignados a `ar[0]`, `ar[1]` y `ar[2]` y el programa se puede referir a esas cadenas.

Java exige que este método se declare como `public static void`; más adelante se describen las características de `static`, pero conviene reseñar que desde este método sólo se puede llamar a otro método `static` y hacer referencias a variables de la misma clase.

Además de `main()`, un programa puede constar de una colección de clases con tantos métodos como se desee.

En un programa corto se puede incluir todo el programa completo en una clase e incluso tener sólo `main()`; sin embargo, en un programa largo, aunque tenga una sola clase, habrá demasiado código para incluirlo en este método, así que debe incluir llamadas a otros métodos definidos por el usuario, o métodos de clases incorporados con la declaración `import`. El programa siguiente se compone de tres métodos que se invocan sucesivamente: `obtenerdatos()`, `alfabetizar()` y `verpalabras`.

NOTA

Un método en Java es un subprograma que devuelve un único valor, un conjunto de valores, o realiza alguna tarea específica, como E/S, y debe estar en una clase.

```
// declaraciones import

class Programa
{
  public static void main(String [] ar)
  {
    obtenerdatos();

    alfabetizar();

    verpalabras();
  }
  ...

}
```

Las sentencias situadas en el interior del cuerpo de `main()`, u otro método, deben terminar en punto y coma.

3.1.4 Métodos definidos por el usuario

Todos los programas se construyen a partir de una o más clases compuestas por una serie de variables y métodos que se integran para crear una aplicación; todos los métodos contienen una o más sentencias de Java, generalmente creadas para realizar una única tarea, como imprimir en pantalla, escribir un archivo o cambiar el color de la pantalla; es posible declarar un número casi ilimitado de métodos en una clase de Java.

Los métodos definidos por el usuario se invocan, dentro de la clase donde se definieron, por su nombre y los parámetros opcionales que pudieran tener; después de que el método se invoca, el código asociado se ejecuta y, a continuación, se retorna al método llamador. Si la llamada es desde un objeto de la clase, se invoca al método precedido del objeto y el selector punto (`.`). Más adelante se verá a fondo; mientras, a título de ejemplo, se crea un objeto de la clase `Sumatorio` y se invoca al método `sumar()`:

```
Sumatorio sr = new Sumatorio();
sr.sumar();
```

Todos los métodos tienen nombre y una lista de valores atribuidos, llamados parámetros o argumentos, se puede asignar cualquier nombre a un método pero normalmente se procura que dicho nombre describa su propósito.

Los métodos en Java se especifican en la clase a la que pertenecen; su definición es la estructura del mismo.

```
tipo_retorno nombreMetodo (lista_de_parámetros) principio del método
{
  sentencias                  cuerpo del método
  return expresión            valor que devuelve
}                             y fin del método
```

tipo_retorno	Es el tipo de valor, o `void` devuelto por la función
nombre_función	Nombre del método
lista_de_parámetros	Lista de *parámetros*, o *void*, pasados al método; se conoce también como *argumentos* o argumentos formales.

A veces, para referirse a un método, se menciona el *prototipo* que tiene; esto es simplemente su cabecera:

```
tipo_retorno nombreMetodo (lista_de_parámetros);
```

Ejemplo de prototipo:

```
void contarArriba(int valor);
```

La palabra reservada `void` significa que el método no devuelve un valor; el nombre del método es `contarArriba` y tiene un argumento de tipo entero, `int valor`.

Java también proporciona clases con métodos predefinidos denominados *clases de biblioteca*, organizados en paquetes; estos métodos están listos para ser llamados en todo momento, aunque requieren la incorporación del paquete donde se encuentran, o bien sólo la clase del paquete. La invocación de uno de ellos desde la clase a la que pertenecen o desde el objeto; por ejemplo, para llamar al método que calcula la raíz cuadrada y mostrar el resultado debe escribir:

```
double r = 17.8;
System.out.println(Math.sqrt(r));
```

El método `sqrt` se invoca precedido de la clase `Math` y el selector punto (.); lo mismo ocurre con `println()` que es un llamado precedido por objeto `out` definido en la clase `System`.

EJEMPLO 3.1

Éste es un programa formado por una clase que contiene dos métodos además de `main()`:`calcula()` y `mostrar()`; el primero determina el valor de una función para un valor dado de `x`; el segundo muestra el resultado; `main()` llama a cada uno de los métodos auxiliares, especificados como `static` por las restricciones de `main()`; la función se evalúa utilizando el seno de la clase `Math`; ésta se encuentra en el paquete `java.lang`, el cual ya aclaramos que se importa automáticamente.

```
class Evaluar
{
  public static void main(String [] ar)
  {
    double f;
    f = calcula();
    mostrar(f);
  }

  static double calcula()
  {
    double x = 3.14159/4.0;
    return x*Math.sin(x) + 0.5;
  }

  static void mostrar(double r)
  {
    System.out.println("Valor de la función: " + r);
  }
  // termina la declaración de la clase
}
```

3.1.5 Comentarios

Como ya se mencionó, un comentario es información que se añade en las líneas del programa para proporcionar datos que son ignorados por el compilador y no implican la realización de alguna tarea concreta; su uso es totalmente opcional, aunque recomendable.

Generalmente es buena práctica de programación comentar sus programas tanto como sea posible, así usted mismo y otros programadores podrán leer fácilmente el programa; otra buena práctica de programación es comentar su programa en la parte superior de cada archivo fuente; la información que puede incluir es: nombre de archivo, nombre del programador, descripción breve, fecha en que se creó la versión e información de la revisión.

En Java los comentarios de un programa se pueden introducir de dos formas:

- Con los caracteres `/*...*/` para insertar más de una línea.
- Con la secuencia de dos barras (`//`) para incorporar una línea.

Comentarios con /* */

Los comentarios comienzan y terminan con la secuencia /* */; todo el texto situado entre ambas será ignorado por el compilador.

```
/* Saludo.java Primer programa Java */
```

Si se necesitan varias líneas de programa se puede hacer lo siguiente:

```
/*
  Programa       : Saludo.java
  Programador    : Luis Cebo
  Descripción    : Primer programa Java
  Fecha creación : 17 junio 2001
  Revisión       : Ninguna
*/
```

También se pueden situar comentarios de la forma siguiente:

```
System.out.println("Programa Demo");  /* sentencia de salida */
```

Se aconseja utilizar esta forma de construir comentarios cuando éstos ocupen más de una línea.

Comentarios en una línea con //

Todo lo que viene después de la doble barra inclinada (//) es un comentario y el compilador lo ignora; la línea de comentario comienza con dicho símbolo.

```
// Saludo.java     -- Primer programa Java
```

Si se necesitan varias líneas de comentarios, se puede hacer lo siguiente:

```
//
// Programa       : Saludo.java
// Programador    : Luis Ceb
// Descripción    : Primer programa Java
// Fecha creación : 17 junio 1994
// Revisión       : Ninguna
//
```

aunque para comentarios de más de una línea se prefieren los delimitadores /* */.

Como no se pueden anidar comentarios, no es posible escribir uno dentro de otro, y tampoco tiene sentido hacerlo; al tratar de anidar comentarios, el compilador produce errores porque no puede discernir entre ellos.

El comentario puede comenzar en cualquier parte de la línea, incluso después de una sentencia de programa; por ejemplo:

```
// Saludo.java -- Primer programa Java
import java.io.*; //incorpora todas la clases del paquete io
class Principal   //archivo fuente debe ser Principal.java

{
  public static void main(String [] ar)) //método inicial
```

```
  {
    System.out.print("Hola mundo cruel");//visualiza en pantalla
  }
}
```

EJEMPLO 3.2

El siguiente programa crea una cadena de caracteres con un mensaje y lo imprime en la pantalla seguido de un nombre que se escribe en la línea donde se llama al programa para ejecución, el cual sólo consta de la clase `Mensaje` con el método `main()`. Java dispone de la clase `String` para crear cadenas, y por ello se declara una variable del mismo tipo para referenciar al mensaje; el nombre que se escribirá se teclea al ejecutar el programa, en este caso: `java Mensaje Angela`. El argumento `ar` en la posición inicial, `ar[0]`, de `main()` contiene `Angela`; por último, se escriben las cadenas `mensaje` y `ar[0]` con el método `println()`; un miembro de la clase `System`, siempre disponible, es un objeto `out` desde el que se puede invocar a `print()` y a `println()`.

NOTA

Los archivos fuente deben tener el mismo nombre que la clase principal (clase que contiene a `main()`) así como la extensión `.java`.

```
/*
nombre del archivo: Mensaje.java
```

El nombre de la clase principal debe coincidir con el archivo:

```
*/
class Mensaje
{
  public static void main(String [] ar)
  {
    String mensaje = "Tardes del Domingo con ";
    System.out.println(mensaje + ar[0]);
  }
}
```

3.2 Elementos de un programa en Java

Todo programa en Java consta de un archivo donde se encuentran las clases y métodos que escribe el programador y, posiblemente, de otros archivos en los que se encuentran los paquetes con las clases incorporadas; el compilador traduce cada archivo con su programa, además incorpora las clases solicitadas al programa y analiza la secuencia de *tokens* de las que consta el mismo.

3.2.1 *Tokens* (Elementos léxicos del programa)

Existen cinco clases de *tokens*: identificadores, palabras reservadas, literales, operadores y otros separadores.

3.2.1.1 Identificadores

Un identificador es una secuencia de caracteres, letras, dígitos, subrayados (_) y el símbolo $; el primer carácter puede ser una letra, un subrayado o el símbolo $. Las letras mayúsculas y minúsculas son diferentes; por ejemplo;

```
nombre_clase          Indice                Dia_Mes_Año
elemento_mayor        Cantidad_Total        Fecha_Compra_Casa
a                     Habitacion120         i
Suma$                 Valor_Inicial         LongCuerda
```

El identificador puede ser de cualquier longitud; no hay límite en cuanto al número de caracteres de los identificadores de variables, métodos y demás elementos del lenguaje; como Java es sensible a las mayúsculas, distingue entre los identificadores ALFA y alfa; por eso se recomienda utilizar siempre el mismo estilo al escribirlos. Un consejo que puede servir de regla es escribir:

1. Identificadores de variables en minúsculas.
2. Constantes en mayúsculas.
3. Métodos en minúsculas.
4. Clases con el primer carácter en mayúsculas.

NOTA

Reglas básicas de formación de identificadores

1. Secuencia de letras, dígitos, subrayados o símbolos $ que empiezan con letra _ o $.
2. Son sensibles a las mayúsculas: `minum` es distinto de `MiNum`.
3. Pueden tener cualquier longitud.
4. No pueden ser palabras reservadas, tales como `if`, `switch` o `else`.

3.2.1.2 Palabras reservadas

Una palabra reservada (*keyword* o *reserved word*) tal como **`void`**, es una característica de Java asociada con algún significado especial y no se puede utilizar como nombre de identificador, clase, objeto o método; por ejemplo:

```
// ...
void void()           // error
{
  // ...
  int char;           // error
  // ...
}
```

Los siguientes identificadores están reservados para usarlos como palabras reservadas, y no deben emplearse para otros propósitos.

Java contiene las palabras reservadas `true`, `false` que son literales lógicos y `null` que es un literal nulo que representa una referencia a nada/ninguno. `True`, `false` y `null`, no son palabras reservadas sino literales reservados.

NOTA

La versión Java 5.0 introdujo la palabra clave **`enum`**.

Tabla 3.1 Palabras reservadas de Java.

abstract	**double**	**int**	**super**
assert	**else**	**interface**	**switch**
boolean	**enum**	**long**	**synchronized**
break	**extends**	**native**	**this**
byte	**final**	**new**	**throw**
case	**finally**	**package**	**throws**
catch	**float**	**private**	**transient**
char	**for**	**protected**	**try**
class	**goto**	**public**	**void**
const	**if**	**return**	**volatile**
continue	**implements**	**short**	**while**
default	**import**	**static**	
do	**instanceof**	**strictfp**	

Dos de las palabras de la tabla anterior no se utilizan por Java: `const` y `goto` pues pertenecen al lenguaje C++; sólo se mantuvieron para generar mejores mensajes de error si se utilizasen en Java.

3.2.2 Signos de puntuación y separadores

Otros signos de puntuación son:

```
! % $ & * ( ) - + = { } ~ ^ |
[ ] \ ; ´ _ < > ? , . / "
```

Los separadores son espacios en blanco, tabulaciones, retornos de carro y avances de línea.

3.2.3 Paquetes

Java agrupa las clases e interfaces que tienen cierta relación mediante archivos especiales que contienen las declaraciones de clases con sus métodos: los paquetes; estos últimos también proporcionan una serie de paquetes predefinidos, y los más importantes son: `java.lang`, `java.applet`, `java.awt`, `java.io` y `java.util`.

`java.lang`, que ya fue mencionado antes, contiene las clases nucleares de Java: `System`, `String`, `Integer` y `Math`; consideraremos otras en diferentes capítulos.

`java.io` contiene clases utilizadas para entrada y salida, algunas de ellas son: `BufferedReader`, `InputStreamReader` y `FileReader`.

`java.util` guarda diversas clases de utilidad, por ejemplo: `Date` maneja fechas, `Random` genera números aleatorios y `StringTockenizer` permite descomponer una cadena en subcadenas separadas por un determinado símbolo.

`java.applet` y `java.awt` suministran clases para crear applets e interfaces gráficos. Cabe mencionar que el programador puede crear paquetes propios para almacenar clases creadas y después utilizarlas en las aplicaciones que desee; por ejemplo: se tiene la clase `Libro` y se quiere almacenar en el paquete `biblioteca`:

```
package biblioteca;

public class Libro
{
  double precio;
  public static void leerLibro()
  {
  }
  ...
}
```

A partir del directorio creado, Java genera un subdirectorio con el mismo nombre del paquete (en este caso, `biblioteca`) en el cual guarda el archivo con la clase (`Libro.java`); Java puede utilizar la clase del paquete `biblioteca` anteponiendo el nombre del paquete al de la clase. El compilador buscará el archivo de la clase en el subdirectorio con el nombre del paquete: `biblioteca.Libro.leerLibro();`.

Declaración `import`

Como mencionamos en el apartado 3.1.1, en esta declaración se especifican las clases de los paquetes que se utilizarán en un programa y, además, permite que el programa se refiera a la clase con sólo escribir su nombre. Por ejemplo:

```
import biblioteca.Libro;
```

Con esta declaración, la llamada al método `leerLibro()` es más simple: `Libro.leerLibro();`.

Es importante mencionar que la declaración `import` tiene dos formatos:

```
import nombrePaquete.nombreClase;
import nombrePaquete.*;
```

El primero especifica la clase que se va a utilizar, mientras que el segundo precisa que todas las clases del paquete están disponibles.

```
import java.util.StringTockenizer;
import java.awt.*;
```

Al incorporar las clases de más de un paquete puede ocurrir que haya nombres de clases iguales; para evitar ambigüedad, en estos casos hay que preceder el nombre del paquete al nombre de la clase; por ejemplo: si hubiera alguna colisión con la clase `Random` al crear un objeto se escribirá:

```
java.util.Random g = new java.util.Random();
```

3.3 Tipos de datos en Java

Un tipo de datos es el conjunto de valores que puede tomar una variable; así, el tipo de datos `char` representa la secuencia de caracteres Unicode y una variable de tipo `char` podrá tener uno de esos caracteres; los tipos de datos simples representan valores escalares o individuales, los cuales pueden ser `char` o los enteros. Java no soporta un gran número de tipos de datos predefinidos pero tiene la capacidad para crear sus propios tipos de datos a partir de la construcción `class`; todos los tipos de datos simples o básicos de Java son, esencialmente, números; por ejemplo:

- Enteros.
- Números de coma flotante o reales.
- Caracteres.
- Lógicos o *boolean*.

La tabla 3.2 muestra los principales tipos de datos básicos, sus tamaños en bytes y el rango de valores que pueden almacenar.

Los tipos de datos fundamentales en Java son:

- **Enteros**, números completos y sus negativos, de tipo `int`.
- **Variantes de enteros**, tipos `byte`, `short` y `long`.
- **Reales**, números decimales: tipos `float`, `double`.
- **Caracteres**, letras, dígitos, símbolos y signos de puntuación.
- **Boolean**, `true` o `false`.

Tabla 3.2 Tipos de datos básicos de Java.

Tipo	Ejemplo	Tamaño en bytes	Rango mínimo/máximo
`char`	`'C'`	2	`'\0000' .. '\FFFF'`
`byte`	`-15`	1	`-128..127`
`short`	`1024`	2	`-32768..32767`
`int`	`42325`	4	`-2147483648..2147483647`
`long`	`262144`	8	`-9223372036854775808 .. +9223372036854775807`
`float`	`10.5f`	4	$3.4*(10^{-38})..3.4*(10^{38})$
`double`	`0.00045`	8	$1.7*(10^{-308})..1.7*(10^{308})$
`boolean`	`true`	1bit	`false, true`

`char`, `byte`, `short`, `int`, `long`, `float`, `double` y `boolean` son palabras reservadas, o en concreto, *especificadores de tipos.*

3.3.1 Enteros: `int`, `byte`, `short`, `long`

Probablemente el tipo de dato más familiar es el entero o `int`; adecuado para aplicaciones que trabajan con datos numéricos; en Java hay cuatro tipos de datos enteros: `byte`, `short`, `int` y `long`; enumerados de menor a mayor rango. El tipo más utilizado, por semejanza de nombre, es `int`; sus valores se almacenan internamente en 4 bytes (o 32 bits) de memoria; la tabla 3.3 resume los cuatro tipos enteros básicos, junto con el rango de valores y uso recomendado.

3.3.1.1 Declaración de variables

La forma más simple para declarar variables es poner primero el tipo de dato y a continuación el nombre de la variable; si desea asignar un valor inicial a la variable, el formato de la declaración es:

```
<tipo de dato> <nombre de variable> = <valor inicial>
```

También se pueden declarar múltiples variables en la misma línea:

```
<tipo de dato> <nom_var1>, <nom_var2> ... <nom-varn>
```

Tabla 3.3 Tipos de datos enteros.

Tipo Java	Rango de valores	Uso recomendado
`byte`	`-128.. +127`	Bucles `for`, índices.
`short`	`-32768 .. +32767`	Conteo, aritmética de enteros.
`int`	`-2147483648 .. +2147483647`	Aritmética de enteros en general.
`long`	`9223372036854775808 .. +9223372036854775807`	Cálculos con enteros grandes como factorial, etcétera.

Por ejemplo:

```
int valor; int valor = 99
int valor1, valor2; int num_parte = 1141, num_items = 45;
long sim, jila = 999111444222;
```

Java siempre realiza la aritmética de enteros de tipo `int` en 32 bits a menos que en la operación intervenga un entero `long`; por ello conviene utilizar variables de estos tipos cuando se realicen operaciones aritméticas; en el siguiente ejemplo se generan errores por el tipo de variables:

```
short x;
int a = 19, b = 5;
x = a+b;
```

Al devolver `a+b`, no se puede asignar a `x` un valor de tipo `int` porque es de tipo `short`; aunque en este caso es mejor declarar las variables de tipo `int`, el error también se puede resolver forzando una conversión de tipo de datos:

```
x = (short)(a+b);
```

Las constantes enteras como `-1211100`, `2700` o `42250` siempre se consideran de tipo `int` y para que el compilador las considere como `long` se utiliza el sufijo `l` o `L`; por ejemplo: `-2312367L`.

Note que si el resultado de una operación sobrepasa el valor entero máximo, no se genera un error de ejecución sino que se pierden los bits de mayor peso en la representación del resultado.

En aplicaciones generales, las constantes enteras se pueden escribir en decimal o base 10; por ejemplo, `100`, `200` o `450`; en octal o base 8 (cualquier número que comienza con un 0 y contiene dígitos en el rango de 1 a 7; por ejemplo, 0377); o en hexadecimal o base 16 (comienza con 0x y van seguidas de los dígitos 0 a 9 o las letras `A-F` o `a-f`; por ejemplo, `0xFF16`). La tabla 3.4 muestra ejemplos de constantes enteras representadas en notaciones o bases decimal, hexadecimal y octal.

Cuando el rango del tipo `int` no es suficientemente grande para sus necesidades, se consideran tipos enteros largos; por ejemplo:

```
long medidaMmilimetros;
long distanciaMmedia;
long numerosGrandes = 40000L;
```

3.3.2 Tipos de coma flotante (`float/double`)

Los tipos de datos de coma o punto flotante representan números reales que contienen una coma o punto decimal, tal como `3.14159`, o números grandes como `1.85*10`^{15}. La declaración de las variables de coma flotante es igual que la de variables enteras; por ejemplo:

```
float valor;          //declara una variable real
float valor1, valor2; //declara varios valores de coma flotante
float valor = 99.99f; //asigna el valor 99.99 a la variable valor
double prod;
```

Tabla 3.4 Constantes enteras en tres bases diferentes.

Base 10 Decimal	Base 16 Hexadecimal (Hex)	Base 8 Octal
8	0x08	010
10	0x0A	012
16	0x10	020
65536	0x10000	0200000
24	0x18	030
17	0x11	021

Java soporta dos formatos de coma flotante (ver tabla 3.5); `float`, requiere 4 bytes de memoria y `double`, 8 bytes.

Éstos son algunos ejemplos:

```
double d;                          // definición de f
d = 5.65  ;                        // asignación
float x = -1.5F;                   // definición e inicialización
System.out.println("d: " + d);     // visualización
System.out.println("x: " + x);
```

De manera predeterminada, Java considera de tipo `double` las constantes que representan números reales y coma flotante; para que una constante se considere como `float` se añade el sufijo `F` o `f`; para que una constante se estime como `double` se incorpora el sufijo `D` o `d`. Es importante tener en cuenta el tipo predeterminado ya que puede dar lugar a errores; si se define

```
float t = 2.5;
```

se comete un error al inicializar la variable `t`, ya que ésta es de tipo `float` mientras que la constante es `double`; la solución es sencilla: `float t = 2.5F`.

3.3.3 Caracteres (`char`)

Un carácter es cualquier elemento de un conjunto de grafías predefinidas o alfabeto. Java fue diseñado para utilizarse en cualquier país sin importar el tipo de alfabeto que se utilice; para reconocer cualquier tipo de carácter, los elementos de este tipo utilizan 16 bits, 2 bytes, el doble de los que la mayoría de los lenguajes de programación emplean. De esta forma, Java puede representar el estándar `Unicode` que recoge más de treinta mil caracteres distintos procedentes de las distintas lenguas escritas. La mayoría de las computadoras utilizan el conjunto de caracteres `ASCII`, que se almacenan en el byte de

Tabla 3.5 Tipos de datos en coma flotante en Java.

Tipo Java	Rango de valores		Precisión
float	3.4×10^{-38} ...	3.4×10^{38}	7 dígitos
double	1.7×10^{-308} ...	1.7×10^{308}	15 dígitos

menor peso de un `char`; el valor inicial de los caracteres `ASCII` y también `Unicode` es `'\u0000'` y el último carácter `ASCII` es `'\u00FF'`.

Java procesa datos carácter tales como texto utilizando el tipo de dato `char`; y en unión con la clase `String`, que se verá posteriormente, se puede utilizar para almacenar cadenas o grupos de caracteres. Se puede definir una variable carácter escribiendo:

```
char datoCar;
char letra = 'A';
char respuesta = 'S';
```

De manera interna, los caracteres se almacenan como números; por ejemplo: la letra `A` se almacena como el número 65, la `B` como 66, la `C` como 67, etcétera. Puesto que los caracteres se almacenan de esta manera, se pueden realizar operaciones aritméticas con datos tipo `char`; por ejemplo, la letra minúscula *a* se puede convertir en mayúscula al restar 32 del código ASCII, convertir el entero a `char` y realizar la conversión restando 32 del tipo de dato `char`, como sigue:

```
char carUno = 'a';
...
carUno = (char)(carUno - 32);
```

Esto convierte la `a` del código `ASCII` 97 en `A` del código `ASCII` 65; de modo similar, añadiendo 32 convierte el carácter de letra mayúscula a minúscula:

```
carUno = (char)(carUno + 32);
```

Como existen caracteres que tienen un propósito especial y no se pueden describir utilizando el método normal, Java proporciona **secuencias de escape**; por ejemplo: el carácter literal de un apóstrofo se puede escribir así:

```
'\"
```

y el carácter nueva línea se escribe así:

```
'\n'
```

La tabla 3.5 (ver pág. 66) enumera las diferentes secuencias de escape de Java.

3.3.4 Boolean

Java incorpora el tipo de dato `boolean` cuyos valores son verdadero (true) y falso (false); las expresiones lógicas devuelven valores de este tipo. Por ejemplo:

```
boolean bisiesto;
bisiesto = true;
boolean encontrado, bandera;
```

Dadas estas declaraciones, todas las asignaciones siguientes son válidas:

```
encontrado = true;          // encontrado toma el valor verdadero
indicador = false;          // indicador toma el valor falso
encontrado = indicador;     // encontrado toma el valor de indicador
```

Las expresiones lógicas resultan en `true` o `false` y se forman con operandos y operadores relacionales o lógicos; así se pueden hacer estas asignaciones:

```
encontrado = (x>2) && (x<10); //encontrado toma el valor verdadero
                              //si x está comprendido entre 2 y 10
```

Como sucede con el resto de las variables, las de tipo `boolean` se pueden inicializar mientras se especifican; así, se puede definir `boolean sw = true`.

La mayoría de las expresiones lógicas aparecen en estructuras de control que sirven para determinar la secuencia en que se ejecutan las sentencias de Java; si es necesario, se puede visualizar el valor de la variable `boolean` utilizando el método `print()`:

```
int y = 9;
boolean estaEnRango;

estaEnRango = (y<-1) || (y>15);

System.out.print( "El valor de estaEnRango es " + estaEnRango);
```

Visualizará lo siguiente:

```
El valor de estaEnRango es false
```

3.3.5 El tipo de dato `void`

`void` es una palabra reservada que se utiliza como tipo de retorno de un método que ya se ha utilizado en el método `main()`:

```
public static void main(String [] ars);
```

Los métodos devuelven un valor de un tipo de datos determinado, `int`, `double`, etcétera; por ejemplo, se dice que: `int indice(double [] v)` es de tipo entero. Un método puede implementarse de tal manera que no devuelva valor alguno, en este caso se utiliza `void`; dichos métodos no devuelven valores:

```
void mostrar();
void pantalla();
```

En Java, la utilización de `void` es únicamente para el tipo de retorno de un método; no se pueden definir variables de este tipo porque el compilador lo detecta como error; por ejemplo, en la siguiente declaración:

```
void pntel;
```

El compilador genera el error siguiente:

```
Instance variables can´t be void: pntel
```

3.4 Tipo de datos enumeración (`enum`)

A partir de la versión de Java SE 5.0 se pueden definir tipos de datos enumerados (`enum`) o enumeraciones que constan de diferentes elementos, especifican diversos valores por

medio de identificadores y son definidos por el programador mediante la palabra reservada `enum`; también tienen un número finito de elementos y valores nombrados; por ejemplo, la siguiente sentencia:

```
enum Notas { A, B, C, D, E );
```

La sentencia anterior define `Notas` como un tipo enumerado (`enum`); los valores de sus elementos son `A`, `B`, `C`, `D` y `E` cuyos valores se denominan constantes de enumeración o `enum`, se encierran entre llaves ({}) y se separan por comas; las constantes dentro de un tipo `enum` deben ser únicas; por ejemplo, el tipo de datos `DEPORTES`:

```
enum DEPORTES { TENIS, ESQUÍ, FUTBOL, BALONCESTO, GOLF };
```

Define el tipo `DEPORTES` como `enum` y sus valores son las constantes similares: `TENIS`, `ESQUÍ`, `FUTBOL`, `BALONCESTO` y `GOLF`. Cada `enum` es un tipo especial de clase y los valores que le pertenecen son objetos de la clase.

Después de definir un tipo `enum`, se pueden declarar variables con referencia a tal tipo; por ejemplo:

```
DEPORTES miDeporte;
```

y se le puede asignar un valor a la variable:

```
miDeporte = DEPORTES.TENIS;
```

Una variable de tipo `DEPORTES` sólo puede contener uno de los valores listados en la declaración del tipo de dato o el valor especial `null` indicando que no se establece ningún valor específico a la variable; en el apartado "Enumeraciones" del capítulo 12 se amplía el concepto.

3.5 Conversión de tipos (*cast*)

En el capítulo 4, al tratar operadores y expresiones, analizaremos en detalle las conversiones de tipos de datos implícitas y explícitas que proporciona Java; no obstante y dada su vinculación con los valores que toman los tipos de datos en el momento de su proceso de ejecución, haremos ahora unas breves consideraciones prácticas que influyen durante el desarrollo de un programa.

Cuando se evalúan expresiones aritméticas, se observa que los valores enteros (`int`) se convierten automáticamente en valores de coma flotante (`double`) siempre que sea necesario y con el objetivo de evitar errores de cálculo; estas conversiones numéricas son posibles en Java, pero como la información se puede perder es necesaria una operación de conversión manual realizada por el programador; en otras palabras, si un tipo de dato se trata automáticamente como diferente, se ha producido una conversión implícita de tipo de dato. Para evitar esto, Java permite la conversión explícita de tipos (en inglés, *casting*) mediante un operador de conversión de tipos o moldeado de tipos que tiene la siguiente sintaxis:

```
(nombreTipoDato) expresión
```

En esta instrucción se evalúa la *`expresión`* y su valor se trata como uno del tipo especificado en *`nombreTipoDato`*. Cuando en la práctica se utiliza el operador de conver-

sión para tratar un número de coma flotante o decimal como si fuera entero, se quita la parte decimal y se trunca el número real. Los siguientes ejemplos y los del apartado 4.12 muestran cómo funciona el operador de conversión:

Expresión	Se evalúa a
`(int) (8.5)`	`8`
`(int) (4.3)`	`4`
`(double) (42)`	`42.0`
`double x = 10.78;`	
`int num = (int) x;`	La variable `num` toma el valor 10

3.6 Constantes

En Java pueden distinguirse dos tipos de constantes:

- Literales.
- Declaradas.

Las constantes literales son las más usuales; toman valores tales como `45.32564`, `222` o bien por medio de datos que se introducen directamente en el texto del programa; las constantes declaradas son como las variables: sus valores se almacenan en memoria pero no se pueden modificar.

3.6.1 Constantes literales

Las constantes literales o simplemente constantes, en general, se clasifican en cuatro grupos que pueden ser constantes:

- Enteras
- Caracteres
- De coma flotante
- De cadena

Constantes enteras

La escritura de constantes enteras requiere seguir estas reglas:

- Nunca utilizar comas ni otros signos de puntuación en números enteros o completos; por ejemplo:

  `123456` en lugar de `123.456`

- Para forzar un valor al tipo `long` se debe terminar con la letra `L`; se recomienda mayúscula porque la minúscula puede confundirse con el 1; por ejemplo:

  `1024` es un tipo entero, `1024L` es un tipo largo (`long`)

- Una constante entera que empieza por 0 se considera en base 8 u octal; por ejemplo:

  Formato decimal, `123`; Formato octal `0777`, están precedidas de `0`

- Para escribir una constante entera en hexadecimal, base `16`, se utiliza el prefijo `0x`, o bien `0X`. Las constantes hexadecimales constan de dígitos y de las letras `A`, `B`, `C`, `D`, `E` y `F` tanto en mayúsculas como en minúsculas; por ejemplo:

  Formato hexadecimal `0XFF3A`, están precedidas de "`Ox`" u "`OX`"

Constantes reales

Una constante de coma flotante representa un número real puesto que siempre tiene signo y representa aproximaciones en lugar de valores exactos; por ejemplo:

```
82.347, .63,83. ,47e-4, 1.25E7     o     61.e+4
```

La notación científica se representa con un exponente positivo o negativo como se indica a continuación:

`2.5E4` equivale a `25000`
`5.435E-3` equivale a `0.005435`

Existen dos tipos de constantes: `float` de 4 bytes y `double` de 8 bytes. Java asume las constantes reales como tipo `double`, por ejemplo: `-12.5`, `1E3` se guarda en dicho tipo; para que una constante real se considere de tipo `float`, se añade el sufijo `F` o `f`: `11.5F`, `-1.44e-2F`; `2e8f`.

NaN e infinito

Infinito, tanto positivo como negativo, es una constante real que resulta de ciertas funciones matemáticas, por ejemplo:

```
System.out.println("Log(0) = " + Math.log(0.0));
```

Se imprime:

```
Log(0) = -infinito
```

Hay funciones matemáticas que no están definidas para todos los números reales; de igual forma, ciertas operaciones matemáticas no están definidas para ciertos valores; por ejemplo: la división entre cero. El resultado de estas funciones matemáticas o expresiones es `Not a Number` o `NaN`, para continuar con la función logarítmica:

```
System.out.println("Log(-1.0) = " + Math.log(-1.0));
```

Se imprime:

```
Log(-1.0) = NaN
```

Si una expresión tiene un operando `NaN`, toda la expresión será `NaN`; esta clave indica que el resultado de la expresión o la función matemática evaluada no es un número; se considera mejor obtener `NaN` en lugar de resultados numéricos incorrectos.

Constantes carácter

Una constante carácter (`char`) es una grafía del conjunto universal de caracteres denominado Unicode; existen diversas formas de escribirlos, la más sencilla es encerrarla entre comillas:

```
'A'       'b'       'c'
```

Además de los caracteres `ASCII` estándar, una constante carácter soporta grafías especiales que no se pueden representar con el teclado, como los códigos ASCII altos y las secuencias de escape; por ejemplo, el carácter sigma (Σ) —código ASCII 228, en octal

344— se representa mediante la secuencia de escape '\nnn', siendo éste el número octal del código ASCII, como sigue:

```
char sigma = '\344';
```

Este método se utiliza para almacenar o imprimir cualquier carácter de la tabla ASCII por su número octal; en el ejemplo anterior, la variable sigma no contiene cuatro caracteres sino únicamente el símbolo sigma; con este formato '\nnn' representa todos los caracteres ASCII, los cuales están en el rango de 0 a 377 (0 a 255 en decimal) y, por tanto, ocupan el byte de menor peso de los dos bytes que utiliza las constantes carácter.

Ciertos caracteres especiales se representan utilizando una barra oblicua (\) y un código entre simples apóstrofos llamado secuencia o código de escape. La tabla 3.6 muestra diferentes secuencias de escape y su significado.

```
// Programa: Pruebas códigos de escape

class Codigo
{
  public static void main(String[] ar)
  {
    char nuevaLinea = '\n';  //nueva línea
    char bs = '\\';          //barra inclinada inversa

    System.out.println("Salto: " + nuevaLinea + "Secuencia escape");
    System.out.println("Back Slash: " + bs);
  }
}
```

Otra forma de representar un carácter es con el formato hexadecimal: '\uhhhh', donde hhhh son dígitos hexadecimales; a ese formato se le denomina Unicode porque se puede escribir cualquier carácter del conjunto universal; por ejemplo:

```
char hl = '\u000F';
char bs = '\u1111';
```

Tabla 3.6 Caracteres secuencias (códigos) de escape.

Código de Escape	Significado	Códigos ASCII	
		Dec	Hex
'\n'	nueva línea	13 10	OD OA
'\r'	retorno de carro	13	OD
'\t'	tabulación	9	09
'\b'	retroceso de espacio	8	08
'\f'	avance de página	12	OC
'\\'	barra inclinada inversa	92	5C
'\''	comilla simple	39	27
'\"'	doble comilla	34	22
'\000'	número octal	*Todos*	*Todos*
'\uhhhh'	número hexadecimal	*Todos*	*Todos*

Aritmética con caracteres Java

Dada la correspondencia entre un carácter y su código Unicode es posible realizar operaciones aritméticas sobre datos de caracteres; observe el siguiente segmento de código:

```
char c;
c = 'T' + 5;          // suma 5 al carácter ASCII(Unicode)
```

En caso de compilar esas dos sentencias, la suma daría error; no por sumar un carácter con un entero, sino por asignar un dato entero (32 bits) a una variable carácter (16 bits); el problema se soluciona con una conversión:

```
c = (char)('T' + 5);
```

Lo que realmente sucede es que `Y` se almacena en `c`, el valor `ASCII` de la letra *T* es 84, y al sumar 5 resulta 89, que es el código de la letra `Y`. A la inversa, se pueden almacenar constantes de carácter en variables enteras; así:

```
int j = 'p'
```

Cabe resaltar que no se coloca una letra `p` en `j`, sino que se asigna el valor 80 —código `ASCII` de `p`— a la variable `j`.

Constantes cadena

Una constante cadena (también llamada literal cadena o cadena) es una secuencia de caracteres encerrados entre dobles comillas; éstos son ejemplos:

```
""                    // Cadena vacía
"123"
"12 de octubre 1492"
"esto es una cadena"
```

Se puede concatenar cadenas, escribiendo lo siguiente:

```
"ABC" + "DEF" + "GHI" +
"JKL"
```

De lo cual resulta:

```
"ABCDEFGHIJKL"
```

Entre los caracteres de una cadena se puede incluir cualquier carácter, incluyendo los de la secuencia de escape. La siguiente cadena contiene un tabulador y un fin de línea:

```
"\tEl día de la apertura es:\n\t15 de octubre 2001"
```

Una cadena no puede escribirse en más de una línea, y esto no es problema porque se parte en cadenas más cortas y se concatenan con el operador `+`.

Las cadenas en Java son objetos, no de tipo básico como `int`; el tipo de una cadena es `String`, el cual es una clase que se encuentra en el paquete `java.lang`. Como todos los objetos son constantes inmutables que una vez creados no pueden cambiarse; se pue-

den formar tantas cadenas como sean necesarias en los programas, cada una será un nuevo objeto sin nombre.

Recuerde que una constante de caracteres se encierra entre comillas simples, y las constantes de cadena encierran caracteres entre comillas dobles:

```
'Z'       "Z"
```

La primera `'Z'` es una constante carácter simple con una longitud de 1, y la segunda es una constante de cadena de caracteres de la misma longitud; la diferencia es que la constante de cadena es un objeto `String`, y el carácter es un dato simple de tipo `char` que se almena en 2 bytes de memoria; concluyendo que no puede mezclar las constantes caracteres y las cadenas de caracteres en su programa.

3.6.2 Constantes declaradas `final`

El cualificador `final` permite dar nombres simbólicos a constantes que se crean mediante el formato general:

```
final tipo nombre = valor;
```

```
final int MESES = 12; // Meses es constante simbólica valor 12

final char CARACTER = '@'
final int OCTAL = 0233;
final String CADENA = "Curso de Java";
```

RECOMENDACIÓN

Se acostumbra que los identificadores de valores constantes y los identificadores de variables final se escriban en mayúsculas.

El cualificador `final` especifica que el valor de una variable no se puede modificar durante el programa; se puede decir que el valor asociado es el valor definitivo y por tanto, no puede cambiarse; cualquier intento de modificar el valor de la variable definida con `final` producirá un mensaje de error.

```
final int SEMANA = 7
final char CAD[]= {'J','a','v','a',' ','o','r','b','i'};
```

sintaxis de final

```
final tipoDato  nombreConstante = valorConstante;

final long TOPE = 07777777;
final char CH = 'S';
```

3.7 Variables

En Java una variable es una posición con nombre en la memoria donde se almacena un valor con cierto tipo de datos; las hay de tipos de datos básicos, tipos primitivos, o que almacenan datos correspondientes con el tipo; otras son de clases o de objetos con referencias a éstas. Una constante, por el contrario, es una variable cuyo valor no puede modificarse.

Una variable se caracteriza por tener un nombre identificador que describe su propósito; también, al usarse en un programa, debe ser declarada previamente; dicha declaración puede situarse en cualquier parte del programa. La declaración de una variable de tipo primitivo, como `int`, `double`, etcétera consiste en escribir el tipo de dato, el identificador o nombre de la variable y, en ocasiones, el valor inicial que tomará; por ejemplo:

```
char respuesta;
```

significa que se reserva espacio en la memoria para `respuesta`, en este caso, un carácter que ocupa dos bytes. El nombre de una variable debe ser un identificador válido; en la actualidad, y con el objetivo de brindar mayor legibilidad y una correspondencia mayor con el elemento del mundo real que representa, es frecuente utilizar subrayados o el carácter `$` en los nombres, ya sea al principio o en su interior, por ejemplo:

```
salario         dias_de_semana            edad_alumno     _fax  $doblon
```

3.7.1 Declaración

Una declaración de una variable es una sentencia que proporciona información de ésta al compilador; su sintaxis es:

Tipo variable

Tipo es el nombre de un tipo de dato conocido por el Java.
Variable es un identificador (nombre) válido en Java.

Por ejemplo:

```
long dNumero;
double HorasAcumuladas,
    HorasPorSemana,
    NotaMedia;
short  DiaSemana;
```

Es preciso declarar las variables antes de utilizarlas; esto se puede hacer en tres lugares dentro de una clase o un programa:

- En una clase, como miembro de ésta.
- Al principio de un método o bloque de código.
- En el punto de utilización.

3.7.1.1 En una clase, como miembro de la clase

La variable se declara como un miembro de la clase, al mismo nivel que los métodos de la clase; y están disponibles por todos estos últimos.

```
class Suerte
{
  int miNumero;
  void entrada()
  {
    miNumero = 29;
  }
```

```
  void salida()
  {
    System.out.println("Mi número de la suerte es " + miNumero);
  }
}
```

En esta clase, la variable `miNumero` puede utilizarse en cualquier método, aquí se hizo en `entrada()` y `salida()`.

3.7.1.2 Al principio de un bloque de código

Es el medio para declarar una variable en un método o en un bloque de código dentro de un método:

```
long factor(int n)
{
  int k;
  long f;
  f = 1L;
  for (k = 1; k < n; k++)
  {
    long aux=10;
    ...
  }
  return f;
}
```

Aquí, las variables `k` y `f` están definidas en el método `factor` y son locales porque se pueden utilizar en el ámbito del método; la variable `aux` se define en el bloque del bucle `for`, y por tanto, éste es su ámbito.

3.7.1.3 En el punto de utilización

Java proporciona gran flexibilidad en la declaración de variables, ya que es posible llevar a cabo esta tarea en el punto donde se utilizará; esta propiedad se emplea mucho en el diseño de bucles.

```
for(int j = 0; j < 10; j++)
{
  // ...
}
```

Por ejemplo:

```
// Distancia a la luna en kilómetros

class LaLuna
{
  public static void main(Sting [] ar)
  {
    final int LUNA = 238857; //distancia en millas
    System.out.println("Distancia a la Luna" + LUNA + " millas");
    int lunaKilo;
    lunaKilo = LUNA*1.609;  //una milla = 1.609 kilómetros
    System.out.println("En kilómetros es " + lunaKilo +" km");
  }
}
```

EJEMPLO 3.3

Aquí se muestra cómo una variable puede declararse en cualquier parte de un programa Java.

```
class Declaracion
{
  public static void main(String [] ar)
  {
    int x, y1;         // declara las variables x e y1
    x = 75;
    y1 = 89;
    int y2 = 50;       // declara la variable y2 inicializándola a 50
    System.out.println(x + "," + y1 + "," + y2);
  }
}
```

3.7.2 Inicialización de variables

Al declarar una variable se puede proporcionar un valor inicial; cuyo formato general es:

NOTA

Esta sentencia declara y proporciona un valor inicial a una variable.

```
tipo nombre_variable = expresión
```

expresión es cualquier declaración válida cuyo valor es del mismo modelo que *tipo*.

A continuación se presentan algunos ejemplos de declaración e inicialización:

```
char respuesta = 'S';
int contador = 1;
float peso = 156.45F;
int anyo = 1992;
```

Estas acciones crean las variables `respuesta`, `contador`, `peso` y `anyo`, que almacenan en memoria los valores respectivos situados a su derecha; una variable, inicializada o no en la declaración, puede cambiar de valor utilizando sentencias de asignación, como en el siguiente caso en el que la variable no se inicializa y después se le asigna un valor:

```
char barra;
barra = '/';
```

3.8 Duración de una variable

Dependiendo del lugar donde se definan las variables de Java, éstas se pueden utilizar en la totalidad del programa, dentro de un método o pueden existir de forma temporal en el bloque de un método; la zona de un programa en la que una variable activa se denomina, normalmente, ámbito o alcance (*scope*); en general, éste se extiende desde la sentencia que la define hasta los límites del bloque que la contiene o la liberación del objeto al que pertenece; de acuerdo con esto, los tipos básicos de variables en Java son:

- Locales.
- De clases.

3.8.1 Variables locales

Son aquellas definidas en el interior de un método, visibles sólo en ese método específico; las reglas por las que se rigen son:

1. En el interior de un método no se puede modificar por ninguna sentencia externa a aquél.
2. Sus nombres no son únicos; por ejemplo: dos, tres o más métodos pueden definir variables con nombre `interruptor`; cada una es distinta y pertenece al método en el que se declara.
3. No existen en memoria hasta que se ejecuta el método; esta propiedad permite ahorrar memoria porque deja que varios métodos compartan la misma memoria para sus variables locales, aunque no de manera simultánea.

Por esta última razón, las variables locales también se llaman automáticas o auto ya que se crean de manera predeterminada a la entrada del método y se liberan de la misma manera cuando se termina la ejecución del método.

```
void sumar()
{
  int a, b, c;                    //variables locales
  int numero;                     //variable local
  a = (int)Math.random()*999;
  b = (int)Math.random()*999;
  c = (int)Math.random()*999;
  numero = a+b+c;
  System.out.println("Suma de tres números aleatorios " + numero);
}
```

3.8.2 Variables de clases

Los miembros de una clase son métodos o variables; éstas se declaran fuera de aquéllos, son visibles desde cualquier método y se referencian simplemente con su nombre.

```
class Panel
{

  int a, b, c;         //declaración de variables miembro de la clase
  double betas()
  {
    double x;          // declaración local
    ...
    a = 21;            // esta variable es visible por ser de la clase
  }

  int valor;           //declaración de variable de clase
  double alfas ()
  {
    float x;           // declaración local
    ...
    b = 19;            // esta variable es visible por ser de la clase
    valor = 0;         // desde un método de una clase se puede acceder
                       // a cualquier variable
    valor = valor+a+b;
  }
  //...
}
```

A una variable declarada en una clase se le asigna memoria cuando se crea un objeto de dicha clase; la memoria asignada permanece durante la ejecución del programa hasta que el objeto se libera automáticamente, o bien, hasta que termina la ejecución del programa.

NOTA

Todas las variables locales desaparecen cuando termina su bloque pero una variable de clase es visible en todos sus métodos y en todas las clases declaradas en el mismo programa; en ese sentido se puede decir que una variable de clase es global, visible desde el punto en que se define hasta el final del programa o archivo fuente.

3.8.3 Acceso a variables de clase fuera de la clase

También se puede acceder a las variables de clase fuera del ámbito de la clase en que se declaran, dependiendo del modificador especificado y que puede ser cualquiera de éstos: private, protected o public; por ejemplo:

```
class Calor
{
  private int x, g, t;
  protected double gr;
  float nt;
  double calculo()
  {
    double x;          // declaración local
    ...
    gr = g* t + x;     // variables gr,g,t visibles por ser de la clase
  }

  //...
}
```

En la clase `Calor` las variables `x`, `g` y `t` tienen acceso privado, sólo se pueden usar en los métodos de la clase; `gr` tiene acceso protegido, además de ser visible en la clase también lo es en las clases derivadas de ésta y en cualquiera que esté dentro del mismo paquete o dentro del mismo archivo fuente; `nt` no tiene modificador de accesibilidad, la visibilidad predeterminada es sólo en cualquier clase del archivo o paquete en que se encuentra.

La siguiente clase tiene métodos que acceden a las variables del objeto `Calor` y se supone que se encuentran en el mismo paquete:

```
class Frigo
{
  protected double frigorias;

  public double frigoNria(Calor p)
  {
    double x;
    ...
    x = p.gr + frigorias - p.nt;

    return x;
  }
}
```

A TOMAR EN CUENTA

Visibilidad de las variables de una clase de menor a mayor accesibilidad:

1. `private`, sólo dentro de la clase.
2. Por omisión (sin modificador), desde cualquier clase del paquete.
3. `protected`, en todo el paquete y en clases derivadas de otros paquetes.
4. `public`, desde cualquier clase.

El método `frigoNria(Calor)` tiene como argumento una referencia al objeto `Calor`, con el selector. Se accede a los miembros `gr` y `nt`; esto es posible por estar declarados como `protected`. Se puede

ingresar a los miembros de una clase, tanto métodos como variables declarados `public`, desde cualquier otra clase o clase derivada, ya sea que estén en el mismo paquete o en otro distinto.

En la siguiente clase aparecen variables de clase y locales a un método de la clase o locales a un bloque; `q` es una variable de clase accesible desde todas las sentencias de los métodos mediante simplemente escribir su nombre; sin embargo, las definidas dentro de `marcas()`, como `a`, son locales a `marcas()`, por consiguiente sólo las sentencias interiores a `marcas()` pueden utilizar `a`.

NOTA

Violar las normas de acceso resulta en error al compilar el programa.

```
class Ambito                    Alcance o ámbito de clase
{                               q, variable de clase
  int q;
  void marcas()

  {                             Local a marcas
    int a;                      a, variable local
    a = 124;
    q = 1;

    {                           Primer subnivel en marcas
      int b;                    b, variable local
      b = 124;
      a = 457;
      q = 2;

      {                         Subnivel más interno de marcas
        int c;                  c, variable local
        c = 124;
        b = 457;
        a = 788;
        q = 3;
      }

    }

  }
}
```

3.9 Entradas y salidas

En Java la entrada y salida se lee y escribe en flujos (*streams*); la fuente básica de entrada de datos es el teclado, mientras que la de salida es la pantalla. La clase `System` define dos referencias a objetos `static` para la gestión de entrada y salida por consola:

`System.in`	para entrada por teclado.
`System.out`	para salida por pantalla.

La primera es una referencia a un objeto de la clase `BufferedInputStream` en la cual hay diversos métodos para captar caracteres tecleados; la segunda es una referencia a un objeto de la clase `PrintStream` con métodos como `print()` para salida por pantalla.

3.9.1 Salida (`System.out`)

El objeto `out` definido en la clase `System` se asocia con el flujo de salida, que dirige los datos a consola y permite visualizarlos en la pantalla de su equipo; por ejemplo:

```
System.out.println ("Esto es una cadena");
```

Entonces se visualiza: `Esto es una cadena`

`System.out` es una referencia a un objeto de la clase `PrintStream`, sus siguientes métodos se utilizan con mucha frecuencia:

`print()`	transfiere una cadena de caracteres al buffer de la pantalla.
`println()`	transfiere una cadena de caracteres y el carácter de fin de línea al buffer de la pantalla.
`flush()`	el buffer con las cadenas almacenadas se imprime en la pantalla.

Con estos métodos se puede escribir cualquier cadena o dato de los tipos básicos:

```
System.out.println("Viaje relampago a " + " la comarca de las " + "
Hurdes");
```

Con el operador + se concatenan ambas cadenas, la formada se envía al buffer de pantalla para visualizarla cuando se termine el programa o se fuerce con el método `flush()`.

Como argumento de `print()` o `println()` no sólo se ponen cadenas, sino también constantes o variables de los tipos básicos, `int`, `double`, `char`, etcétera; el método se encarga de convertir a cadena esos datos; por ejemplo:

```
int x = 500;
System.out.print(x);
```

De igual modo, se pueden concatenar cadenas con caracteres, enteros, etcétera; mediante el operador + que internamente realiza la conversión:

```
double r = 2.0;
double area = Math.PI*r*r;
System.out.println("Radio = " + r + ',' + "area: " + area);
```

Java utiliza secuencias de escape para visualizar caracteres no representados por símbolos tradicionales, tales como `\n`, `\t`, entre otros; también proporcionan flexibilidad en las aplicaciones mediante efectos especiales; en la tabla 3.6 se muestran las secuencias de escape.

```
System.out.print("\n Error - Pulsar una tecla para continuar \n");

System.out.print(" Yo estoy preocupado\n" +
                 " no por el funcionamiento \n" +
                 " sino por la claridad .\n");
```

La última sentencia se visualiza como sigue debido a que la secuencia de escape `'\n'` significa nueva línea o salto de línea:

```
Yo estoy preocupado
no por el funcionamiento
sino por la claridad.
```

3.9.2 Salida con formato: `printf`

En la versión 5.0, Java incluyó un método inspirado en la función clásica de la biblioteca C denominado `printf` que se puede utilizar para producir la salida en un formato específico; la sentencia `System.out.printf` tiene un primer argumento cadena, conocido como especificador de formato; el segundo argumento es el valor de salida en el formato establecido; por ejemplo:

```
System.out.printf("%8.3f", x);
```

Por ejemplo:

```
double precio = 25.4;
System.out.printf("$");
System.out.printf("%6.2f",precio);
System.out.printf(" unidad");
```

Al ejecutar este código se visualiza `$ 25.40`. El formato especifica el tipo (f de `float`), el ancho total (6 posiciones) y los dígitos de precisión (2).

```
double x = 10000.0/3.0;
System.out.printf("%9.3f",x);
```

Al ejecutar este código, se visualiza x con un ancho de nueve posiciones y una precisión de tres dígitos.

```
3333.333
```

Cada uno de los especificadores de formato que comienza con un carácter `%` se reemplaza con el argumento correspondiente; el carácter de conversión con el que terminan indica el tipo de valor a dar formato: `f` es un número en coma flotante, `s` una cadena y `d` un entero decimal. En la página web del libro puede consultar todos los especificadores de formato.

3.9.3 Entrada (`system.in`)

La clase `System` define un objeto de la clase `BufferedInputStream` cuya referencia resulta en `in`. El objeto se asocia al flujo estándar de entrada, que por defecto es el teclado; los elementos básicos de este flujo son caracteres individuales y no cadenas como ocurre con el objeto `out`; entre los métodos de la clase se encuentra `read()` que devuelve el carácter actual en el buffer de entrada; por ejemplo:

```
char c;
c = System.in.read();
```

No resulta práctico el captar la entrada carácter a carácter, es preferible hacerlo línea a línea; para esto, se utiliza primero la clase `InputStreamReader`, de la cual se crea un objeto inicializado con `System.in`:

```
InputStreamReader en = new InputStreamReader(System.
in);
```

¡ATENCIÓN!

`System.out` y `System.in` son referencias a objetos asociados a flujos de datos de salida por pantalla y de entrada por teclado respectivamente.

Este objeto creado se utiliza, a su vez, como argumento para inicializar otro objeto de la clase `BufferedReader` que permite captar líneas de caracteres del teclado con el método `readLine()`:

```
String cd;
BufferedReader entrada = new BufferedReader(en);
System.out.print("Introduzca una línea por teclado: ");
cd = entrada.readLine();
System.out.println("Línea de entrada: " + cd);
```

¡ATENCIÓN!

`readline()` devuelve una referencia a una cadena de tipo `String` con la última línea tecleada; además:

- asigna a una variable `String` y
- devuelve `null` si no se ha terminado el texto, fin de fichero.

El método `readLine()` crea un objeto cadena tipo `String` con la línea ingresada; la referencia a ese objeto se asigna, en el fragmento de código anterior, a `cd` porque es una variable de tipo `String`; el objeto de la clase `BufferedReader` se puede crear con una sola sentencia, como la forma habitual en que aparece en los programas.

```
BufferedReader entrada = new BufferedReader(new
        InputStreamReader(System.in));
```

EJEMPLO 3.4

¿Cuál es la salida del siguiente programa si se introducen las letras **LJ** por medio del teclado?

```
class Letras
{
  public static void main(String ar[])
  {
    char primero, ultimo;
    System.out.printf("Introduzca su primera y última inicial: ");
    primero = System.in.read();
    ultimo = System.in.read();
    System.out.println("Hola," + primero + "." + ultimo + ".!\n");
  }
}
```

Introduzca su primera y última inicial: `LJ`

```
Hola, L.J.¡
```

3.9.4 Entrada con la clase `Scanner`

En la versión 5.0, Java incluyó una clase para simplificar la entrada de datos por el teclado llamada `Scanner`, que se conecta a `System.in`; para leer la entrada a la consola se debe construir primero un objeto de `Scanner` pasando el objeto `System.in` al constructor `Scanner`. Más adelante, se explicarán los constructores y el operador `new` con detalle.

```
Scanner entrada = new Scanner(System.in);
```

Una vez creado el objeto `Scanner`, se pueden utilizar diferentes métodos de su clase para leer la entrada: `nextInt` o `nextDouble` leen enteros o de coma flotante.

```
System.out.print("Introduzca cantidad: ");
int cantidad;
cantidad = entrada.nextInt();
System.out.print("Introduzca precio: ");
double precio = entrada.nextDouble();
```

Cuando se llama a uno de los métodos anteriores, el programa espera hasta que el usuario teclee un número y pulsa Enter.

El método `nextLine` lee una línea de entrada:

```
System.out.print("¿ Cual es su nombre?");
String nombre;
nombre = entrada.nextLine();
```

El método `next` se emplea cuando se desea leer una palabra sin espacios:

```
String apellido = entrada.next();
```

La clase `Scanner` se define en el paquete `java.util` y siempre que se utiliza una clase no definida en el paquete básico `java.lang` se necesita utilizar una directiva `import`. La primera línea cerca del principio del archivo indica a Java dónde encontrar la definición de la clase `Scanner`:

```
import java.util.Scanner
```

Esta línea significa que la clase `Scanner` está en el paquete `java.util`; `util` es la abreviatura de *utility* (utilidad o utilería), la cual siempre se utiliza en código Java. Como se ha mencionado, un paquete es una biblioteca de clases y la sentencia `import` hace disponible la clase dentro del programa.

```
import java.util.Scanner;

/**
Este programa muestra la entrada por consola
y ha sido creado el 24 de mayo de 2008
*/
public class EntradaTest
{
  public static void main)String [] args)
  {
    Scanner entrada = new Scanner(System.in);
    // obtener la primera entrada
    System.out.print("¿ Cual es su nombre?");
    String nombre = entrada.nextLine();
    // leer la segunda entrada
    System.out.print("¿ Cual es su edad?");
    int edad = entrada.nextInt();
    // visualizar salida
```

```
        System.out.println("Buenos días " + nombre +
                           "; años " + edad);
    }
}
```

3.9.4.1 Sintaxis de entrada de teclado utilizando `Scanner`

- Hacer disponible la clase `Scanner` para utilizarla en su código; incluir la siguiente línea al comienzo del archivo que contiene su programa:

```
import java.util.Scanner;
```

- Antes de introducir algo por medio del teclado, se debe crear un objeto de la clase `Scanner`.

```
Scanner nombreObjeto = new Scanner(System.in);
nombreObjeto es cualquier identificador Java.
Scanner teclado = new Scanner(System.in);
```

- Los métodos `nextInt`, `nextDouble` y `next` leen respectivamente un valor de tipo `int`, un valor de tipo `double` y una palabra.


```
variable_int = nombreObjeto.nextInt();
variable_double = nombreObjeto.nextDouble();
variable_cadena = nombreObjeto.next();
variable_cadena = nombreObjeto.nextLine();
```

Ejemplo

```
int edad;
edad = teclado.nextInt();
double precio;
precio = teclado.nextDouble();
String rio;
rio = teclado.next();
```

3.10 Tipos de datos primitivos (clases envoltorio)

En ocasiones se necesita convertir algún tipo de dato primitivo como `int` en un objeto; en Java, todos los tipos de datos primitivos soportan clases para cada uno de los tipos que se conocen como envoltorios (*wrappers*); por ejemplo, la clase `Integer` corresponde al tipo primitivo `int`. Las clases envoltorio tienen nombres similares a los tipos: `Integer`, `Long`, `Float`, `Double`, `Short`, `Byte`, `Character` y `Boolean`; los seis primeros son herencia de la superclase común `Number`; todas son inmutables, es decir, no se puede cambiar un valor *envuelto* después de construir el envoltorio; también son `final`, de modo que no se pueden extender subclases de las mismas.

NOTA

Java proporciona una clase envoltorio correspondiente a cada tipo de dato primitivo; por ejemplo: la clase envoltorio correspondiente al tipo `int` es `Integer`.

NOTA

En el taller práctico del capítulo 3 de la web podrá ampliar y practicar esos conceptos.

La clase envoltorio `Integer` se utiliza para envolver valores `int` en objetos `Integer` para que dichos valores se consideren objetos; de modo similar, `Long` se utiliza para envolver valores `long` en objetos `Long` y así sucesivamente.

Las clases envoltorio contienen métodos que permiten convertir cadenas numéricas en valores del mismo tipo; `parseInt`, de la clase `Integer`; convierte una cadena numérica de enteros en un valor de tipo `int`. De modo similar, `parseFloat`, de la clase `Float`, se utiliza para convertir una cadena numérica de decimales en un valor equivalente del tipo `float` y `parseDouble` de la clase `Double` se emplea para convertir una cadena numérica decimal en un valor equivalente del tipo `double`; por ejemplo:

```
1. Integer.parseInt ("63")                                  = 63
2. Integer.parseDouble ("525.35")                           = 525.35
3. Integer.parseInt ("63") + Integer.parseInt ("25")  = 63 + 25 = 88
```

resumen

Este capítulo introdujo los componentes básicos de un programa en Java; en capítulos posteriores se analizarán con detalle cada uno de ellos; también se analizó el modo de utilizar las características mejoradas en Java que permiten escribir programas orientados a objetos. En este capítulo aprendió la estructura general de un programa en Java y que:

- La mayoría de los programas en Java tienen una o más declaraciones `import` al principio del programa fuente; las cuales proporcionan información de los paquetes donde se encuentran las clases utilizadas para crear el programa; de esta forma se accede a ellas sin cualificarlas con el nombre del paquete.
- Cada programa debe incluir una clase con el método `public static void main(String[] ar)`, aunque la mayoría tendrán más clases y métodos; el programa siempre inicia la ejecución por el método `main()`.
- En Java, la clase `System` define dos objetos: `System.out` como flujo de datos que se dirigen a pantalla y `System.in` para flujo de caracteres de entrada por teclado.
- Para visualizar salida a la pantalla, se utilizan los métodos `print()`, `println()` y `printf()` llamados desde el objeto `System.out`.
- Los nombres de los identificadores son sensibles a las mayúsculas y minúsculas, deben comenzar con un carácter alfabético, de subrayado (_) o de dólar ($) seguido por caracteres similares.
- Los tipos de datos básicos de Java son: `boolean`, entero (`int`), entero largo (`long`), entero corto (`short`), `byte`, carácter (`char`), coma o punto flotante (`float`) y `double`.
- Los tipos de datos carácter utilizan 2 bytes de memoria para representar el conjunto universal de caracteres Unicode; el tipo entero utiliza 4; el tipo entero largo utiliza 8 y los tipos de coma flotante utilizan 4 y 8 bytes de almacenamiento.
- Se utilizan conversiones forzosas tipo (*cast*) para conversiones entre tipos; el compilador realiza automáticamente muchas de ellas; por ejemplo: si se asigna un entero a una variable float, el compilador convierte automáticamente el valor entero al tipo de la variable.
- Se puede seleccionar explícitamente una conversión de tipos precediendo la variable o expresión con (*tipo*), siendo éste un modelo de dato válido.

- No todas las conversiones son posibles de forma automática; por ejemplo: un dato `int` no se puede asignar a una variable `long`.
- Para cada tipo básico de datos Java se declara una clase, éstas se denominan envoltorio; con esto se tiene la clase `Integer`, `Long`, `Float`, `Double`, `Character`, etcétera.
- Cada variable tienen un ámbito visible, es decir, la parte del programa que se puede utilizar; se puede distinguir entre ámbito de un método o bloque de código y ámbito de clase.

conceptos clave

- `Char`.
- Clase `Character`.
- Clase `Double`.
- Clase `Float`.
- Clase `Integer`.
- Clase principal.
- Código ejecutable.
- Código fuente.
- Código objeto.
- Comentarios.
- Constantes.
- `float/double`.
- Flujos.
- Incluir archivos `import`.
- `Int`.
- Método `main()`.
- Paquetes.
- `Print`.
- `readLine`.
- Variables.

ejercicios

3.1 ¿Cuál es la salida del siguiente programa?

```
class Primero
{
  public static void main()
  {
    System.out.println("Hola mundo!\n" + "Salimos al aire");
  }
}
```

3.2 Identificar el error del siguiente programa

```
// Este programa imprime "¡Hola mundo!":
class Saludo
{
  main()
  {
    System.out.println( "Hola mundo!\n");
  }
}
```

3.3 Escribir y ejecutar un programa que imprima su nombre y dirección.

3.4 Redactar y ejecutar un programa que imprima una página de texto con no más de 40 caracteres por línea.

3.5 Depurar el siguiente programa:

```
// un programa Java sin errores
```

```
class Primo

  void main()
  {
    System.out.println("El lenguaje de programación Java";
  }
}
```

3.6 Escribir un programa que imprima una letra B con asteriscos, de la siguiente manera:

```
*****
*    *
*    *
*    *
*****
*    *
*    *
*    *
*****
```

3.7 ¿Cuál es la salida del siguiente programa si se introducen las letras J y M?

```
class Letras
{
  public static void main(String [] a)
  {
    char primero, ultimo;
    System.out.println("Introduzca sus iniciales:");
    System.out.print("\t Primer apellido:");
    primero = System.in.read();
    System.out.println( "\t Segundo apellido:");
    ultimo = System.in.read();
    System.out.println("Hola, " + primero + "." + ultimo + ".!\n");
  }
}
```