

Programación Concurrente y de Tiempo Real

Guión de prácticas 1: Introducción a Java

Natalia Partera Jaime
Alumna colaboradora de la asignatura

Índice

1. Breve introducción a Java	2
1.1. Programas en Java: la plataforma J2SE, compilación y ejecución	2
1.2. Comparación entre Java y C/C++	6
1.3. Breve introducción a la orientación a objetos	8
2. Estructura de un programa en Java	9
2.1. Clases y estructura general de una clase	10
2.2. Métodos y estructura de un método	11
3. Elementos básicos de Java	12
3.1. Identificadores	12
3.2. Tipos, variables y valores	13
3.3. Expresiones y operadores	14
3.4. Instrucciones	17
3.4.1. Instrucción <code>if</code>	17
3.4.2. Instrucción <code>while</code>	18
3.4.3. Instrucción <code>for</code>	19
3.4.4. Instrucción <code>do while</code>	19
3.4.5. Instrucciones <code>break</code> - <code>continue</code>	20
3.4.6. Instrucción <code>switch</code>	21
3.5. Entrada y salida de consola	22
4. Paquetes de biblioteca	25
5. Clases de envoltura	26
6. Ejercicios para profundizar	29
7. Soluciones de los ejercicios	30
7.1. Ejercicio 3	30
7.2. Ejercicio 4	30
7.3. Ejercicio 5	31
7.4. Ejercicio 7	33
7.5. Ejercicio 8	33
7.6. Ejercicio 9	34
7.7. Ejercicio 10	35
7.8. Ejercicio 11	35
7.9. Ejercicio 12	36
7.10. Ejercicio 14	38
7.11. Ejercicio 15	38
7.12. Ejercicio 16	39
7.13. Ejercicio 17	40
7.14. Ejercicio 18	41
7.15. Ejercicio 19	43
7.16. Ejercicio 20	43
7.17. Ejercicio 21	44
7.18. Ejercicio 22	45

1. Breve introducción a Java

Java es un lenguaje de programación independiente de la plataforma. La independencia a la plataforma se debe a que Java se compila sobre su propia máquina virtual. Esto permite que en cualquier plataforma se pueda ejecutar un mismo programa en código Java, independientemente de las características que soporte la plataforma, como la concurrencia, ya que es la máquina virtual de Java quien proporciona o simula estas características.

Cabe mencionar también que Java es un lenguaje orientado a objetos. Puede que usted no haya estudiado aún el paradigma de la orientación a objetos. De momento tan sólo debe conocer la estructura de un programa en Java.

1.1. Programas en Java: la plataforma J2SE, compilación y ejecución

Observe la siguiente figura y localice el compilador y la máquina virtual de Java.

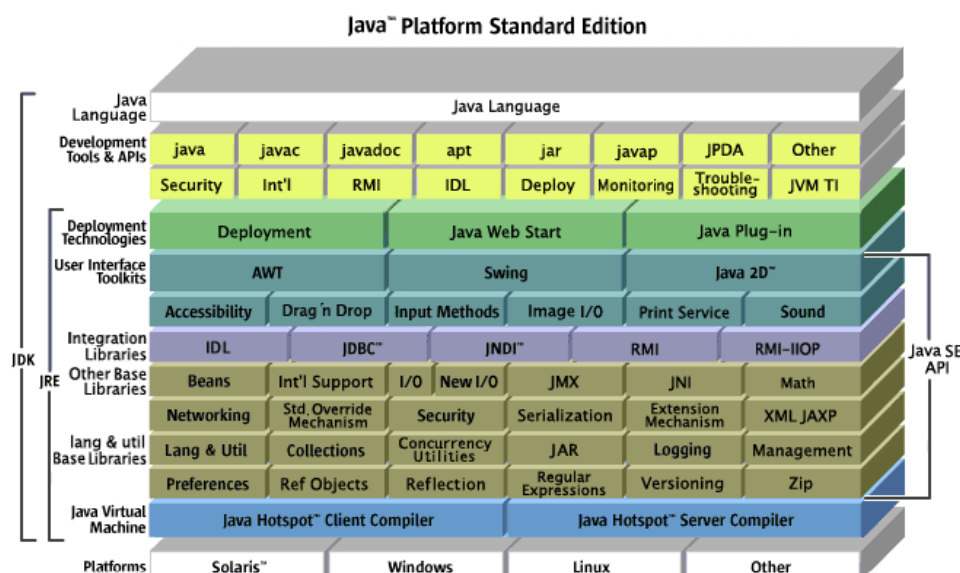


Figura 1: Imagen de la plataforma J2SE.

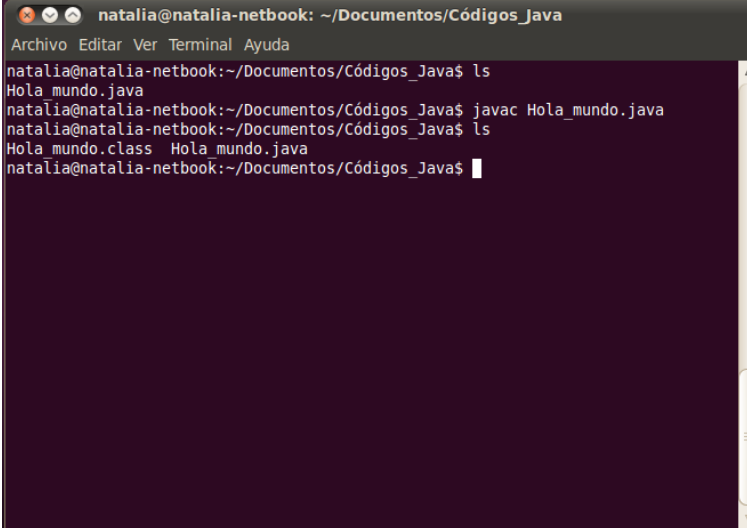
Como puede ver, el compilador de Java se sitúa sobre la máquina virtual. Por esta razón, un programa en Java no tiene que ser compilado en diferentes plataformas, sino que se ejecuta sobre la máquina virtual de Java correspondiente para dicha plataforma.

Hay otros aspectos que necesita saber sobre el desarrollo de programas en Java antes de empezar a programar.

- **Editando:** el nombre de los archivos fuente debe seguir el esquema `nombre_clase.java`. El nombre del archivo (`nombre_clase`) debe coincidir con el nombre de la clase, y la extensión del archivo debe ser `.java`. Preste atención a las letras minúsculas y mayúsculas. No se preocupe si no sabe ahora qué es una clase. Posteriormente encontrará un sencillo ejemplo ilustrativo.

- **Compilando:** tras instalar en su equipo el JDK y configurarlo adecuadamente (vea el documento `guia-jcreator.pdf`), podrá compilar sus programas en Java. Para ello debe ejecutar el comando `javac` seguido del nombre del fichero fuente a compilar (incluyendo la extensión). El comando a ejecutar en su terminal deberá seguir el esquema:

`javac nombre_clase.java`

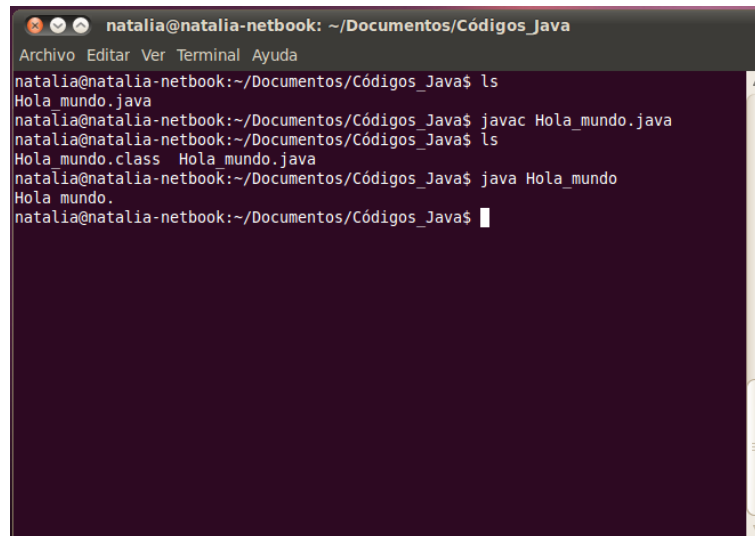


```
natalia@natalia-netbook: ~/Documentos/Códigos_Java
Archivo Editar Ver Terminal Ayuda
natalia@natalia-netbook:~/Documentos/Códigos_Java$ ls
Hola_mundo.java
natalia@natalia-netbook:~/Documentos/Códigos_Java$ javac Hola_mundo.java
natalia@natalia-netbook:~/Documentos/Códigos_Java$ ls
Hola_mundo.class  Hola_mundo.java
natalia@natalia-netbook:~/Documentos/Códigos_Java$
```

Figura 2: Imagen de la compilación.

- **Ejecutando:** una vez que el programa haya sido compilado correctamente, podrá ejecutarlo en su terminal usando el comando `java` seguido del nombre del programa (mismo nombre que el archivo fuente sin extensión). Su comando en la terminal deberá seguir el siguiente esquema:

`java nombre_clase`

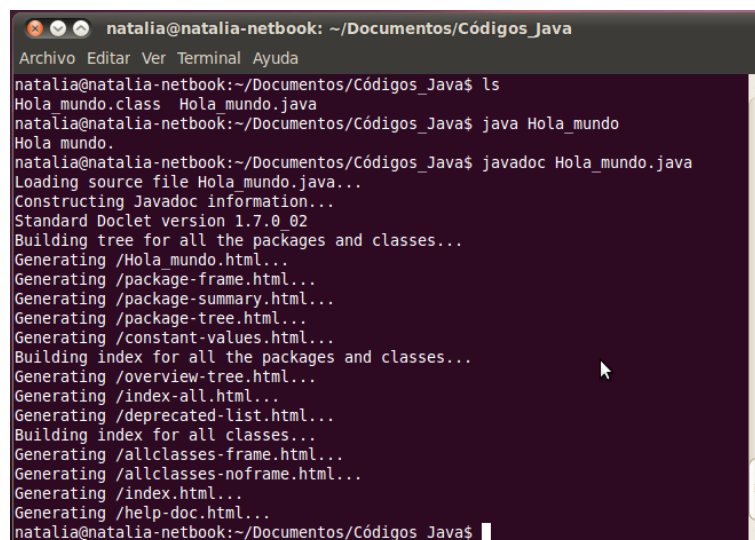
A terminal window titled 'natalia@natalia-netbook: ~/Documentos/Códigos_Java'. The window shows the following commands and output:

```
Archivo Editar Ver Terminal Ayuda
natalia@natalia-netbook:~/Documentos/Códigos_Java$ ls
Hola_mundo.java
natalia@natalia-netbook:~/Documentos/Códigos_Java$ javac Hola_mundo.java
natalia@natalia-netbook:~/Documentos/Códigos_Java$ ls
Hola_mundo.class  Hola_mundo.java
natalia@natalia-netbook:~/Documentos/Códigos_Java$ java Hola_mundo
Hola mundo.
natalia@natalia-netbook:~/Documentos/Códigos_Java$
```

Figura 3: Imagen de la ejecución.

- **Documentando:** en Java la documentación del programa se puede incluir dentro del mismo. La documentación se genera en un archivo `html` usando el comando `javadoc`:

`javadoc nombre_clase.java`

A terminal window titled 'natalia@natalia-netbook: ~/Documentos/Códigos_Java'. The window shows the following commands and output:

```
Archivo Editar Ver Terminal Ayuda
natalia@natalia-netbook:~/Documentos/Códigos_Java$ ls
Hola_mundo.class  Hola_mundo.java
natalia@natalia-netbook:~/Documentos/Códigos_Java$ java Hola_mundo
Hola mundo.
natalia@natalia-netbook:~/Documentos/Códigos_Java$ javadoc Hola_mundo.java
Loading source file Hola_mundo.java...
Constructing Javadoc information...
Standard Doclet version 1.7.0_02
Building tree for all the packages and classes...
Generating /Hola_mundo.html...
Generating /package-frame.html...
Generating /package-summary.html...
Generating /package-tree.html...
Generating /constant-values.html...
Building index for all the packages and classes...
Generating /overview-tree.html...
Generating /index-all.html...
Generating /deprecated-list.html...
Building index for all classes...
Generating /allclasses-frame.html...
Generating /allclasses-noframe.html...
Generating /index.html...
Generating /help-doc.html...
natalia@natalia-netbook:~/Documentos/Códigos_Java$
```

Figura 4: Imagen de la documentación.

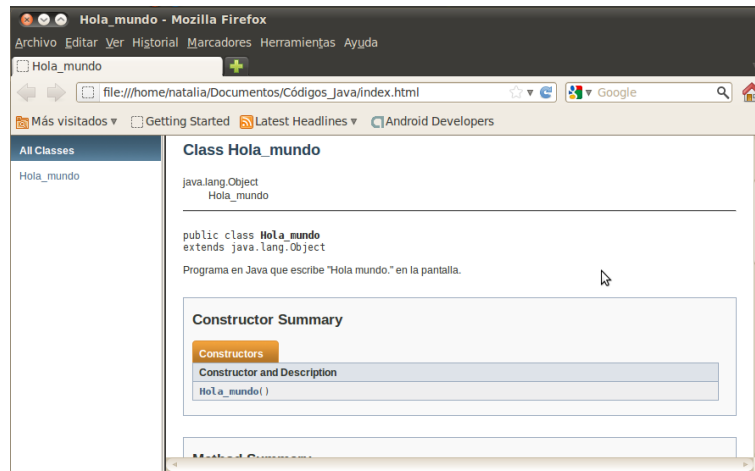


Figura 5: Imagen del resultado de la documentación.

Las etiquetas que más se utilizan al generar documentación son:

Etiqueta	Sirve para
@version	Indica la versión del documento.
@author	Indica el autor del código.
@param	Indica un parámetro de entrada de la función.
@return	Indica el parámetro de devolución de la función.
@throws	Indica las excepciones que puede lanzar la función.
@deprecated	Indica que está obsoleto y ya no se usa.
@see nombre_clase	Genera un enlace a la documentación de la clase indicada.

Cuadro 1: Etiquetas útiles para generar la documentación.

Un ejemplo de un código documentado es el siguiente:

```
/**
 * Programa en Java que pide al usuario el lado de
 * un cuadrado, y muestra su área y su perímetro.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class Cuadrado
{
    /**El método main recibe como argumento
     * un array de cadenas de caracteres.
     * @param args array de cadenas de caracteres
     * @exception no se generan excepciones
     */
}
```

```

public static void main (String[] args)
{
    System.out.println("El lado del cuadrado mide: ");
    Scanner teclado = new Scanner(System.in);
    double lado = teclado.nextDouble();
    System.out.println("El perímetro del cuadrado es " + lado * 4);
    System.out.println("El área del cuadrado es " + lado * lado);
}
}

```

Ejercicio 1 Cree un archivo fuente para el código anterior siguiendo las instrucciones de este apartado. Compílelo, ejecútelo y genere su documentación. Observe el resultado.

1.2. Comparación entre Java y C/C++

En muchos aspectos, Java se parece bastante a los lenguajes de programación C y C++. No obstante, existen algunas diferencias. Si usted está muy familiarizado con C ó C++ es recomendable que lea atentamente este apartado. Si no es el caso, puede leer el resto del documento y volver al final a este apartado para comprobar lo que ha entendido. Si no conoce C++ y la orientación a objetos, no se preocupe si ahora no entiende algunas de las diferencias entre Java y C++. En cualquier caso, aconsejamos que lea también los siguientes apartados donde se explica con detalle el lenguaje Java.

A continuación se indican las características de C ó C++ que Java no soporta o no tiene:

- Punteros y aritmética de punteros.
- Necesidad de ser compilado en diferentes plataformas.
- Macros para el preprocesador, `include`, ni archivos de cabecera.
- Sobrecarga de operadores, en oposición a C++.
- Necesidad de liberar la memoria manualmente (posee recolección automática de basura).
- Herencia múltiple ó clases de base múltiple.
- Campos de bits, `struct`, `union` ó `typedef` (presentes en C y C++).
- `inline`, `register`, `friend` y `template`.
- Plantillas (presentes en C++).
- Variables de referencia.
- Operador `sizeof`.
- Declaraciones `extern`.
- Superclases `protected` ó `private`.
- Conversiones de tipo definidas por el usuario.
- Copia implícita de objetos.
- Notación de asignación explícita.

También hay algunos aspectos de Java que no están presentes en C ni en C++:

- Es interpretado cuando se ejecuta.
- Posee recolección automática de basura (el programador no tiene que encargarse de destruir los objetos no útiles).
- Soporta applets Web.
- Soporta programación de GUI (Graphical User Interface) independiente de la plataforma.
- Soporta programación controlada por eventos.
- Soporta multiprocesamiento y métodos sincronizados.
- Soporta trabajo en red.
- Organización del código por paquetes.
- Creación de objetos mediante `new`.
- Programas y caracteres en Unicode de 16 bits.
- Tipos primitivos `boolean` y `byte`.
- Cadenas y vectores como objetos.
- Variables y parámetros de objeto como referencias.
- Herencia simple (no presente en C, y único tipo de herencia, no como en C++).
- ...

Y hay otras características de Java que trabajan de un modo distinto a como lo hacen en C ó C++:

- Java utiliza `true`-`false` en lugar de cero-no cero para operaciones booleanas.
- Java no tiene tipo `unsigned`. Todos los tipos enteros tienen signo.
- Java soporta los operadores de desplazamiento `>>` y `>>>`.
- Java no tiene tipo de devolución predeterminado para métodos.
- El operador `%` funciona en Java para enteros y punto flotante.
- El tipo `enum` de C, es ahora una clase `Enum` en Java.
- La resolución de una llamada a un método sobrecargado difiere en Java y en C++.
- Las clases de Java utilizan el método `finalize` en lugar de destructores.
- Las clases de Java se organizan en paquetes con un nombre. Cada paquete puede contener uno o más archivos de código fuente.

1.3. Breve introducción a la orientación a objetos

Antes de continuar con la introducción al lenguaje Java, es conveniente que usted entienda el significado y las diferencias entre algunos términos para que no se confunda al leer este documento. No obstante, usted profundizará en el paradigma de la orientación a objetos en otras asignaturas posteriores.

Los términos elementales de la orientación a objetos son los siguientes:

- **Clase.** Una clase es un conjunto de objetos similares. Todos los objetos de una clase tienen los mismos atributos (variables o datos) y métodos (funciones u operaciones). Puede pensar en una clase como un nuevo tipo de datos, y en cada objeto como en una variable de dicho tipo de datos.
- **Objeto.** Un objeto es una entidad que posee atributos (datos) y métodos (operaciones). Los objetos con similares características se agrupan en una misma clase.
- **Atributo.** Un atributo es un dato de un objeto. Aunque todos los objetos de una clase tengan los mismos atributos, el valor de estos atributos es distinto para cada objeto.
- **Método.** Un método es una función de un objeto. Los métodos de una clase funcionan igual para todos sus objetos.

También debe saber que para acceder a un atributo perteneciente a un objeto o para invocar a un método de un objeto, es necesario declarar y construir un objeto de la clase correspondiente. Durante el ámbito de vida del objeto, puede invocar a sus métodos o atributos desde el objeto:

```
public class NombreClase
{
    ...
    public static void main (String[] args)
    {
        ...
        //Declaración y construcción del objeto:
        NombreClase objeto = new NombreClase();

        //Uso de atributo:
        objeto.atributo;

        //Llamada a método:
        objeto.metodo(parametro);
        ...
    }
    ...
}
```

2. Estructura de un programa en Java

Si usted ya ha aprendido a programar en algún otro lenguaje de programación, seguramente habrá visto con anterioridad el programa "Hola Mundo". A continuación puede observar una versión de dicho programa en Java:

```
/**
 * Programa en Java que escribe "Hola mundo." en la pantalla.
 *
 * @author Natalia Partera
 * @version 1.0
 */
public class Hola_mundo
{
    public static void main (String[] args)
    {
        System.out.println("Hola mundo.");
    }
}
```

Ejercicio 2 ¿Qué espera que haga este programa? Copie este código y guárdelo en el fichero `Hola_mundo.java`. A continuación compílelo y ejecútelo. ¿El resultado obtenido se corresponde con lo esperado por usted?

Analicemos los elementos del programa anterior:

Comentario: Las primeras líneas del código anterior corresponden a un comentario. Los comentarios en Java empiezan con `/**` y acaban en `*/`. Java también acepta los comentarios típicos de C (`/*Comentario*/`) que pueden ocupar varias líneas, y de C++ (`//Comentario`) que ocupa lo que queda de línea.

Definición de la clase: Para definir una clase usamos la palabra reservada `class`, seguida del nombre de la clase y del bloque encerrado entre llaves que le sigue. El contenido de la clase empieza con el carácter `{` y acaba con el carácter `}`.

Definición del método: Dentro de la clase se define el método `main()`. La estructura de este método es similar a como lo es en el lenguaje C, salvo que puede estar precedido por algunas palabras reservadas que le dan unas características nuevas propias de la orientación a objetos. Todos los programas en Java deben tener un método `main()`.

Instrucciones o sentencias: El método `main()` contiene las instrucciones del programa principal. En este caso, el programa tiene una única instrucción, cuyo cometido es mostrar en la salida externa por defecto (en este caso, el monitor) el texto entrecomillado.

En Java, todas las funciones (o métodos) deben pertenecer a alguna clase. Así pues, el método `main()`, que contiene la ejecución del programa, debe ser un método de alguna clase (clase principal).

2.1. Clases y estructura general de una clase

Veamos cómo es la estructura general de una clase en Java:

```
/**
 * Estructura de una clase en Java
 */
public class NombreDeLaClase
{
    //Declaración de los atributos de la clase

    //Declaración de los métodos de la clase

    //Método main, que indica dónde empieza la ejecución
    public static void main (String[] args)
    {
        //Declaracion de las variables del método

        //Sentencias de ejecución del método
    }
}
```

Los elementos que suelen componer una clase cualquiera en Java, son:

- **Atributos de la clase:** Son las variables que definen el estado de los objetos. Cada objeto tiene estas variables, pero el valor de estas variables cambia de un objeto a otro de la misma clase. Son variables accesibles desde todos los métodos del objeto.
- **Declaración de métodos:** Son fragmentos de código que hacen una determinada función. Las funciones que realizan son características de los objetos de la clase.
- **Método principal main():** El método `main()` aparece en la clase principal, y describe la ejecución del programa. Su argumento es una cadena de caracteres que contiene los argumentos introducidos al ejecutar el programa.
- **Declaración de variables:** Son las variables locales del método `main()`.
- **Instrucciones o sentencias:** Son las instrucciones a ejecutar en el método `main()`. Las sentencias se ejecutan siempre en el orden en el que se escriben y, siempre, una detrás de otra, hasta que se acaba el método `main()`.

Una clase puede poseer un atributo, varios atributos, un método, varios métodos, una combinación de las anteriores o estar vacía.

Habrás observado que en la plantilla anterior aparece la palabra reservada `public` delante de la definición de la clase. Este es otro aspecto de la orientación a objetos que estudiará más adelante. De momento necesita saber que esto se debe a que las clases pueden tener distinta visibilidad. Si una clase es pública puede ser utilizada desde cualquier paquete. En caso contrario, si no es pública, dicha clase sólo puede ser utilizada dentro de su propio paquete. Entenderá mejor la importancia de la visibilidad y el uso de los paquetes en el apartado 4.

Del mismo modo, los elementos de las clases también pueden tener distinta visibilidad. Desde cualquier otro programa o cualquier parte del programa se puede acceder a los miembros declarados como `public`. El método `main()` debe ser público siempre para que el intérprete de Java lo pueda invocar.

2.2. Métodos y estructura de un método

Un método es un procedimiento de cálculo definido en una clase. Los métodos contienen instrucciones y variables o referencias. Veamos el esquema de un método:

```
tipo_devolución Nombre (tipo_1 arg_1, tipo_2 arg_2, ...)
{
    declaraciones locales /** Si hay */

    instrucciones
}
```

Al igual que en otros lenguajes, los métodos en Java constan de una cabecera y un cuerpo. En la cabecera del método se especifican los parámetros formales, que son siempre pasados por valor. Si el parámetro formal es una referencia a un objeto, se pasa al método la referencia por valor. En el cuerpo del método se incluye una secuencia de declaraciones e instrucciones entre llaves. En él, se especifican las variables locales si son necesarias.

Los métodos pueden ser de clase o de instancia. Los métodos de clase son los más habituales. Son aquellos que se definen para todos los objetos de una clase, de modo que cada objeto puede llamar a su método de clase y pueden coexistir varios métodos con el mismo nombre pero que han sido llamados desde distintos objetos. En el lado opuesto, los métodos de instancia son los declarados como `static`. Estos métodos tienen la peculiaridad de que sólo puede existir un método con este nombre para todos los objetos de la clase. Además, este método es común a todos los objetos de la clase.

Los métodos pueden pertenecer a clases predefinidas (por ejemplo, los métodos `println()`, `log()` o `read()`) o pueden ser definidos por el programador (como los métodos `concatenar()` y `potencia()` que aparecen a continuación).

Ejemplos de definición y uso de métodos:

```
//Ejemplos de declaración de métodos
String concatenar (String s1, String s2)
{return (s1 + s2);}

void potencia (int b)
{
    String potencia = "El cuadrado de ";
    int cuad = b * b;
    System.out.println(potencia + b + " es " + cuad);
}

//Ejemplos de llamadas a métodos predefinidos:
//Deben ser llamados de la forma Clase.método(argumentos)
//o de la forma objeto_de_la_clase.método(argumentos)
ln = Math.log(1.23);
```

```
scanner1.read();
```

Ejercicio 3 Cree una clase en Java que se llame `MisDatos`. La clase debe incluir los atributos `nombre`, `apellidos`, `ciudad` y `ocupacion` del tipo `String`. Incluirá también los atributos `edad`, `asignaturasAprobadas`, `asignaturasCursadas` del tipo `int`. Todos los atributos estarán inicializados. En la clase existirá un método llamado `asigSuspensas` que reciba dos parámetros del tipo `int` y devuelva otro parámetro del mismo tipo que indique el número de asignaturas que ha suspendido en la carrera. Por último, la clase debe incluir un método `main` que muestre todos estos datos. No olvide crear un objeto de la clase `MisDatos` en el método `main`. Ejecute su programa. Puede ver una posible solución en 7.1.

3. Elementos básicos de Java

Veamos poco a poco cómo construir programas en Java.

3.1. Identificadores

Un identificador es un nombre. Los nombres permiten identificar los elementos que se están manejando en un programa. Un identificador debe empezar con una letra y debe seguir una sucesión de letras y dígitos. Un identificador no debe coincidir con una palabra reservada del lenguaje.

Una letra es cualquier carácter que se considera una letra en Java. Como en Java se utiliza Unicode para los caracteres, un identificador se puede escribir con caracteres hebreos, cirílicos, armenios, katakana, etc. Para formar un identificador también se consideran letras los caracteres subrayado '_' y dólar '\$', aunque el carácter '\$' prácticamente no se suele utilizar, salvo en algún tipo de nombrado automático. Se considera un dígito a cualquier carácter entre los caracteres '0' a '9'. Son válidos los siguientes identificadores:

```
ejemplo, EjemploDeIdentificador, εγγεμπλο, otroEjemplo12, uno2tres4oMás, AñoMás,  
_víó_LôQüëö_Nô_3303459345
```

Es recomendable que cuando elija un identificador:

- Utilice nombres significativos.
 - Los nombres de variables y métodos empiecen con minúscula.
 - Los nombres de clases empiecen con mayúscula.
 - En los nombres compuestos para variables y clases, cada nueva palabra empieza con mayúscula. Evite usar el carácter '_' para separar unas palabras de otras.
 - Los nombres de constantes se escriben en mayúsculas. Si el nombre es compuesto, utilice el carácter subrayado para separar unas palabras de otras.
-

Ejercicio 4 Dados los siguientes identificadores que se van a utilizar en un programa escrito en Java, diga cuáles de ellos son correctos y cuáles no. En el caso de que sean correctos, diga además si es recomendado y corrija si no es recomendado. Justifique sus respuestas.

- a) `mi carta`
- b) `unacarta`
- c) `mis2escritos`
- d) `4cientos`
- e) `es_un_mensaje`
- f) `no_vale nada`
- g) `_____ejemplo_____`
- h) `mi-programa`
- i) `¿cuantos?`
- j) `el%Descontado`
- k) `a150PORHORA`
- l) `TengoMUCHOS$$$`
- m) `LOS400GOLPES`
- n) `quieroUNAsolución`

Recuerde que puede ver la solución en 7.2.

3.2. Tipos, variables y valores

Un programa maneja datos o valores. Para poder manejar los valores en un programa, éstos se guardan en variables. Una variable guarda un único valor. Una variable queda determinada por:

- Un nombre o identificador. Debe ser como se ha indicado en la sección 3.1.
- Un tipo de datos. El tipo de datos indica qué tipos de valores se pueden almacenar en esa variable.
- Un rango de valores. El rango de valores que la variable puede admitir viene determinado por el tipo de la variable.

La declaración de variables es igual que en C. La inicialización de variables se puede realizar en su declaración o posteriormente:

```
tipo_variable1 nombre_variable1;  
tipo_variable2 nombre_variable2 = valor_variable2;  
nombre_variable1 = valor_variable1;
```

La declaración de constantes se realiza anteponiendo al tipo la palabra reservada `final`. El valor de una constante no se puede modificar en el programa, por eso hay que darle un valor a la vez que se declara.

```
final tipo_cte NOMBRE_CTE = valor_cte;
```

Los tipos primitivos en Java son:

Tipo	Descripción	Rango
<code>boolean</code>	Verdadero o falso	<code>true</code> o <code>false</code>
<code>char</code>	Carácter Unicode de 16 bits	<code>\u0000</code> a <code>\uFFFF</code>
<code>byte</code>	Entero con signo de 8 bits	-128 a 127
<code>short</code>	Entero con signo de 16 bits	-32768 a 32767
<code>int</code>	Entero con signo de 32 bits	-2147483648 a 2147483647
<code>long</code>	Entero con signo de 64 bits	-922117036854775808 a 922117036854775807
<code>float</code>	Coma flotante de 32 bits	$\pm 3,40282347e + 38$ a $\pm 1,40239846e - 45$
<code>double</code>	Coma flotante de 64 bits	$\pm 1,79169313486231570e + 308$ a $\pm 4,94065645841246544e - 324$

Cuadro 2: Tipos primitivos en Java.

Aunque realmente no es un tipo primitivo, cabe mencionar el tipo `String`. En Java, las variables de este tipo son objetos que se utilizan de forma sencilla como si de variables de un tipo sencillo se trataran.

```
String texto;
texto = "En un lugar de la Mancha de cuyo nombre ...";
```

Para concatenar cadenas, use el operador de concatenación `+` entre ellas. También es posible concatenar con los tipos básicos de Java.

3.3. Expresiones y operadores

En Java hay un conjunto de operadores que se pueden utilizar sobre los tipos de valores para formar expresiones o cálculos. A continuación, un cuadro resumen de los operadores en Java:

Nombre del operador	Tipo de operador	Valor que recibe	Valor de devolución	Ejemplo de operación
Suma unaria	Unario	Entero Real	Entero Real	+44 +2,5
Resta unaria	Unario	Entero Real	Entero Real	-56 -75,63
Producto	Multiplicativo binario	Entero, entero Real, real	Entero Real	3 * 15 2,178 * 5,609
División	Multiplicativo binario	Entero, entero Real, real	Entero Real	12 / 3 345,209 / 29,78
Resto	Multiplicativo binario	Entero, entero Real, real	Entero Real	5 % 2 45,8 % 2,05
Suma binaria	Aditivo binario	Entero, entero Real, real	Entero Real	4 + 5 3,16 + 7,82
Resta binaria	Aditivo binario	Entero, entero Real, real	Entero Real	98 - 52 78,3 - 25,7
Pre-incremento Post-incremento	Aditivo unario	Entero Real	Entero Real	++65 o 987++ ++6,75 o 8,32++
Pre-decremento Post-decremento	Aditivo unario	Entero Real	Entero Real	--61 o 83-- --4,53 o 7,51--
Menor que	Relación binaria	Entero, entero Real, real	Boolean Boolean	35 < 28 45,62 < 9,74
Menor o igual que	Relación binaria	Entero, entero Real, real	Boolean Boolean	29 <= 56 47,0237 <= 32,812
Mayor que	Relación binaria	Entero, entero Real, real	Boolean Boolean	502 > 809 890,53 > 762,13
Mayor o igual que	Relación binaria	Entero, entero Real, real	Boolean Boolean	362 >= 472 72,456 >= 69,014
Igual que	Relación binaria	Entero, entero Real, real	Boolean Boolean	723 == 773 34,62 == 93,26
Distinto de	Relación binaria	Entero, entero Real, real	Boolean Boolean	564 != 998 62,34 != 62,34
Asignación	Asignación simple	Entero, entero Real, real	Nada Nada	x = 45 y = 6,13
Suma y asignación	Aditivo y asignación	Entero, entero Real, real	Nada Nada	x += 3 y += 4,2
Resta y asignación	Aditivo y asignación	Entero, entero Real, real	Nada Nada	x -= 6 y -= 9,1
Producto y asignación	Multiplicativo y asignación	Entero, entero Real, real	Nada Nada	x *= 10 y *= 2,5
División y asignación	Multiplicativo y asignación	Entero, entero Real, real	Entero Real	x /= 2 y /= 1,5
Resto y asignación	Multiplicativo y asignación	Entero, entero Real, real	Nada Nada	x %= 2 y %= 7,71
Negación lógica	Booleano binario	Boolean Expresión arit.-lóg.	Boolean Boolean	!false !(8 - 3 + 2 - 7)
Y lógico O lógico	Booleano binario Booleano binario	Boolean Expresión arit.-lóg. Boolean Expresión arit.-lóg.	Boolean Boolean Boolean Boolean	true && false (3+5) < (5*2) && (3 > 8) true false 0 (8-2)

Cuadro 3: Tabla de los operadores en Java.

Estos operadores tienen un orden de preferencia a la hora de ser evaluados. El orden de prioridad al evaluar los operadores es:

1. Operadores unarios.
2. Operadores multiplicativos, de izquierda a derecha.
3. Operadores aditivos, de izquierda a derecha.
4. Operadores de relación.
5. Operadores de asignación.

Las expresiones aritmético-lógicas son expresiones que devuelven un valor booleano y donde se utilizan operadores aritméticos y operadores relacionales y de igualdad. En una expresión aritmético-lógica se pueden combinar varias expresiones sencillas mediante operadores lógicos.

La precedencia de los operadores booleanos es menor que la de los operadores relacionales, por lo que primero se evalúan las desigualdades y después los operadores booleanos. El orden de prioridad entre los operadores booleanos es: la negación, después el Y lógico y por último el O lógico. La prioridad de los operadores de asignación es la menor de todas.

Ejercicio 5 Dadas las siguientes expresiones aritmético-lógicas calcule cuál es el resultado de evaluarlas. Suponga que las variables `a` y `b` que aparecen son del tipo `int` y `a` tiene el valor 5 y `b` tiene el valor 3.

- a) `!(a > b && 2 * a <= b)`
- b) `++b < 3 || a + b <= 8 && !(a > b)`
- c) `++a <= 6 && (b += 2) < a`
- d) `++a / 2 <= b && (++a / 2 > b || (a * 2 < b * 4))`

Puede comprobar sus soluciones con las que aparecen en el apartado 7.3.

El siguiente programa ilustra el manejo de tipos reales para calcular el interés obtenido en seis meses.

```
import java.util.*;

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

        System.out.println("Cálculo de intereses.");
        System.out.println("Dinero ingresado:");
        //Hay que separar los decimales con coma (,)
        float dinero = teclado.nextFloat();
        System.out.println("Interes anual (%): ");
        float interes = teclado.nextFloat();
        float ganancia = dinero*interes/100/2;
```

```
        System.out.println("Intereses a los seis meses: " + ganancia);
        System.out.println("Dinero disponible en total a los seis meses: " + (dinero +
            ganancia));
    }
}
```

Ejercicio 6 Copie el ejemplo anterior en un archivo y compílelo. Observe el resultado.

Ejercicio 7 Escriba un programa que solicite el radio de la base y la altura de un triángulo y calcule su área. Cuando lo ejecute, puede comprobar su código con el que aparece en 7.4.

3.4. Instrucciones

Sabiendo definir y usar variables de los tipos primitivos, es momento de conocer las instrucciones selectivas y repetitivas presentes en Java para hacer programas más interesantes. Anteriormente se mencionó que el cuerpo de un método puede contener instrucciones. Estas instrucciones de Java pueden ser simples o compuestas.

Las instrucciones simples terminan en un punto y coma (;). En cambio, las instrucciones complejas comprenden a varias instrucciones simples encerradas entre llaves, que pueden emplearse en cualquier lugar donde se emplee una simple. A las instrucciones compuestas también se les llama bloque.

Recuerde:

- Una declaración siempre termina en ;.
- Una instrucción simple siempre termina en ;.
- No hay ; después de una instrucción compuesta.
- No hay ; después de una definición de clase.

3.4.1. Instrucción if

Es una instrucción de bifuración condicional y por tanto permite modificar la dirección de la ejecución de una lista de instrucciones. Su esquema es el siguiente:

```
if (condición)
    instrucción1;
else
    instrucción2;
```

Si la condición se cumple, se ejecuta la **instrucción1**. En caso contrario, se ejecuta la **instrucción2**. Estas instrucciones pueden ser sustituidas por bloques de instrucciones entre llaves, la instrucción **if** puede anidarse y el bloque del **else** puede omitirse si es necesario. Ejemplo:

```
boolean doble (int a, int b)
{
    if(a == 2 * b)
        return (true);
    else if (b == 2 * a)
        return (true);
    else
        return (false);
}
```

Ejercicio 8 Realice un programa que dados tres números cualesquiera diga si son pares o impares. Luego puede compararlo con el programa en 7.5.

3.4.2. Instrucción while

La instrucción **while** es controlada por una condición que repite el cuerpo de la misma mientras ésta no se evalúe como **false**. El esquema de la instrucción es:

```
while (condición)
{
    bloque de sentencias
}
```

La instrucción **while** prueba el valor de la condición. Si es **true**, ejecuta el cuerpo. Si no, termina el ciclo y pasa el control a la siguiente instrucción. La variable que controla el ciclo es llamada variable de control. Ejemplo:

```
public boolean primo (int numero)
{
    int divisor = 2;
    boolean primo = true;

    while ((divisor * divisor <= numero) && primo)
    {
        if (numero % divisor == 0)
            primo = false;
        divisor++;
    }

    return primo;
}
```

Ejercicio 9 Realice un programa que muestre en pantalla una cuenta atrás desde 5 hasta 0. Posteriormente puede comprobarlo con la versión que aparece en 7.6.

3.4.3. Instrucción for

La instrucción `for` es una forma especializada de `while`, donde el cuerpo del ciclo se ejecuta un número fijo de veces, siempre que la condición de control es `true`. Si la condición es `false` al principio, el cuerpo no se ejecuta nunca. Su forma general es:

```
for (inic; condición_de_control; incremento)
{
    cuerpo del ciclo
}
```

Un ejemplo del uso de la instrucción `for` es:

```
void repetirDiez (String cadena)
{
    int iter = 1;
    for (iter = 1; iter <= 10; iter++) {
        System.out.println(cadena);
    }
    System.out.println("Fin de la repetición.");
}
```

Ejercicio 10 Realice un programa muestre en pantalla los 10 primeros múltiplos de 29. Cuando lo haya ejecutado, puede compararlo con el código que aparece en 7.7.

3.4.4. Instrucción do while

La instrucción `do while` es similar a la instrucción `while`, pero garantiza que el cuerpo del bucle se ejecuta al menos una vez. Su forma general es:

```
do
{
    cuerpo del bucle
} while (condición);
```

Un ejemplo de la utilización de esta instrucción es la siguiente:

```
import java.util.*;

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

```

    int numero;
    String linea;

    Scanner teclado = new Scanner(System.in);

    do {
        System.out.println("Introduzca un número entre 0 y 100: ");
        numero = teclado.nextInt();
    } while (numero < 0 || numero > 100);

    System.out.println("El número introducido es: " + numero);
}
}

```

Ejercicio 11 Escriba un método que reciba dos valores enteros como parámetros y dibuje un rectángulo en el que la base es el mayor de los dos parámetros recibidos y el otro es la altura. Cree una objeto de la clase, pida al usuario los valores de la base y la altura e imprima el rectángulo. Consejo: puede utilizar un símbolo (como el '*') que represente cada celda perteneciente al rectángulo. Puede comprobar su programa con la solución que aparece en 7.8.

3.4.5. Instrucciones break - continue

La instrucción **break** suele utilizarse para salir de una iteración de forma incondicional. Al ejecutarse, el control se transfiere de inmediato a la primera instrucción a continuación del bloque o de la iteración actual. El siguiente código muestra un ejemplo:

```

public int cuentaPalabras (String texto)
{
    int palabras = 0, i = 0;
    String textoAux = texto.trim();
    while(true) {
        if (i == textoAux.length())
            break;
        while (i < textoAux.length() && textoAux.charAt(i) != " ")
            i++;
        palabras++;
        while (i < textoAux.length() && textoAux.charAt(i) == " ")
            i++;
    }
    return palabras;
}

```

La instrucción **continue** aborta la iteración actual e inicia la siguiente iteración. Dentro de un bucle **while** o **do while**, transfiere el control de inmediato a la prueba de la condición.

3.4.6. Instrucción switch

La instrucción `switch` proporciona una manera de elegir una entre un conjunto de opciones predefinidas. Su sintaxis es la siguiente:

```
switch (expresión_de_tipo_int)
{
    case valor_1: instrucciones
    case valor_2: instrucciones
    ...
    case valor_n: instrucciones
    default: instrucciones
}
```

La instrucción `switch` evalúa la expresión de cambio y compara su valor con todas las etiquetas `case`. El control se transfiere a aquella etiqueta cuyo valor coincida con el de la expresión o a `default` si no coincide ninguna. Todas las etiquetas `case` deben ser distintas. A continuación, un ejemplo:

```
public class DiasDelMes
{
    enum Mes {Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio,
        Agosto, Septiembre, Octubre, Noviembre, Diciembre}

    public static void main(String[] args)
    {
        int dias;
        Mes mes = Mes.Noviembre;
        switch (mes) {
            case Abril:
            case Junio:
            case Septiembre:
            case Noviembre:
                dias = 30;
                break;
            case Febrero: //No se conoce el año
                dias = 28;
                break;
            default:
                dias = 31;
                break;
        }
        System.out.println("El mes " + mes + " tiene " + dias + " días.");
    }
}
```

Ejercicio 12 Escriba un método que reciba como parámetros tres valores enteros con el día, mes y año de una fecha y devuelva un valor booleano que indique si se trata de valores válidos para una fecha. Puede comparar su código con la solución indicada en 7.9.

3.5. Entrada y salida de consola

Todo programa en Java tiene asociados tres flujos estándar de E/S: flujo de entrada, flujo de salida y flujo de error. Para poder utilizar estos flujos, hay que importar el paquete `java.io`. Verá qué son los paquetes en el apartado 4. De momento solo necesita saber que para importar un paquete debe poner la palabra reservada `import` seguida de un espacio y el nombre del paquete antes de comenzar a declarar la clase.

Entrada estándar La entrada estándar se representa con el objeto `System.in` y lee el teclado. Una forma de obtener los datos desde la entrada estándar, es haciendo que el objeto `System.in` llame a la función `read()`. Ejemplo:

```
System.in.read()
```

La función `read()` devuelve un byte (entre 0 y 255) como tipo de dato `int` o `-1`.

Otro modo de obtener datos desde la entrada estándar es utilizar la clase `Scanner` que se construye pasándole un objeto de la clase `System.in`. Ejemplo:

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

La clase `Scanner` proporciona los siguientes métodos de lectura para los tipos básicos del lenguaje:

Tipo	Método a invocar
byte	<code>teclado.nextByte();</code>
short	<code>teclado.nextShort();</code>
int	<code>teclado.nextInt();</code>
long	<code>teclado.nextLong();</code>
float	<code>teclado.nextFloat();</code>
double	<code>teclado.nextDouble();</code>
boolean	<code>teclado.nextBoolean();</code>

Cuadro 4: Métodos de la clase `Scanner` para la lectura de distintos tipos de datos.

Salida estándar La salida estándar se representa con el objeto `System.out` y escribe en la pantalla. Una forma para escribir en pantalla es con el método `write()`. Este método recibe un entero y envía al flujo de salida el byte menos significativo de dicho entero.

```
System.out.write(int c)
```

Otro método que se puede llamar con el objeto `System.out` es el método `println()`, que recibe una cadena y envía al flujo de salida dicha cadena y una terminación de línea dependiente de la plataforma.

```
System.out.println(String str)
```

Existe un método equivalente al anterior pero que no añade la terminación de línea:

```
System.out.print(String str)
```

Error estándar El error estándar se representa con el objeto `System.err` y proporciona un flujo de salida donde almacenar los errores o excepciones que se dan durante la ejecución del programa. Los métodos que se pueden utilizar para añadir información al flujo de error estándar son los mismos que se utilizan en el flujo de salida estándar.

El ejemplo a continuación ilustra el uso de los métodos `println()`, `read()` y `write()` y de los objetos `System.out` y `System.in`:

```
import java.io.*;
class Minuscula
{
    public static void main (String[] args)
        throws IOException
    {
        int i;
        int dato;

        //esto fuerza la recolección de basura
        System.gc();
        //muestra el tiempo actual en milisegundos
        System.out.println(System.currentTimeMillis());
        //muestra propiedades del sistema
        System.out.println(System.getProperties());
        System.out.println("Introduzca un carácter...");
        do
        {
            i = System.in.read();
            dato = Character.toLowerCase((char) i);
            System.out.write(dato)
        } while (true); //salir con CTRL-B, en JCreator o CTRL-C desde un shell
    }
}
```

El siguiente ejemplo muestra el uso de la clase `Scanner`:

```
import java.util.Scanner;

public class EjemploScanner {

    public static void main (String[] args) {
        byte by;
        short s;
        int i;
        long l;
        float f;
        double d;
        boolean b;
```



```

//Se envuelve System.in en un objeto de clase Scanner
Scanner Tec = new Scanner(System.in);

System.out.println ("Introduzca: ");
System.out.print ("Un byte ");
by = Tec.nextByte();
System.out.println("Byte: " + by);
System.out.print ("Un short ");
s = Tec.nextShort();
System.out.println("Short: " + s);
System.out.print ("Un int ");
i = Tec.nextInt();
System.out.println("Int: " + i);
System.out.print ("Un long ");
l = Tec.nextLong();
System.out.println("Long: " + l);
System.out.print ("Un float ");
f = Tec.nextFloat();
System.out.println("Float: " + f);
System.out.print ("Un double ");
d = Tec.nextDouble();
System.out.println("Double: " + d);
}
}

```

Ejercicio 13 Copie estos ejemplos, compíelos, ejecútelos y obsérvelos. Haga algunas modificaciones para pedir o mostrar más datos.

Ejercicio 14 Ahora que ya sabe cómo se utiliza la entrada y la salida estándar, realice un programa que pida al usuario dos números y calcule su media. El programa deberá mostrar los números de los que debe realizar la media y el resultado. Cuando lo termine, puede comparar su solución con la ofrecida en 7.10.

4. Paquetes de biblioteca

Los paquetes son agrupaciones de clases que están relacionadas entre sí. Las clases de un mismo paquete son clases amigas entre sí. Si una función es amiga de otra, es que puede ser llamada desde el cuerpo de la otra función. Todas las clases de un fichero son amigas, ya que en Java un fichero fuente es un paquete por sí mismo. Si las clases están definidas en varios ficheros se pueden agrupar en un único paquete también. Esto se hace anteponiendo a la definición de la clase la palabra reservada **package** seguida del nombre del paquete:

```
package nombre_paquete;
```

Las clases que no se declaren dentro de ningún paquete forman un paquete "por defecto" y son amigas entre sí. El espacio de nombres de un paquete es propio a sus clases. El nombre de las clases pasa a incluir el nombre del paquete. Por tanto, para utilizar una clase `clase1` del paquete `paquete1`:

- puede usarse el nombre completo, `paquete1.clase1`;
- puede importarse al código en el que quiere usarse `import paquete1.clase1;;`
- o pueden importarse todas las clases del paquete, `import paquete1.*;`.

Así se definen clases en un paquete y se usan:

```
package Polinomios;
public class Polinomio {
    //Uso de clase amiga
    public Polinomio() {new Monomio();}
}
```

```
package Polinomios;
public class Monomio {
    private int grado;
    private double coeficiente;
    public Monomio() {}
    public Monomio(int gr, double coef) {
        grado = gr;
        coeficiente = coef;
    }
    public int Grado() {return grado;}
    public double Coeficiente() {return coeficiente;}
}
```

```
package Polinomios;
public class Prueba {
    public static void main(String[] args) {
        //Uso de clases amigas.
        Polinomio p = new Polinomio();
        Monomio m = new Monomio(2, 3.75);
        System.out.print("El monomio de grado " + m.Grado() + " tiene coeficiente " +
            m.Coficiente());
    }
}
```

En el caso de clases que no son amigas, hay que importarlas como a continuación:

```
import Polinomios.Monomio;
public class Polinomio2grado {

    public static void main(String[] args) {
        //Para usar las clases de mi paquete podemos importarlas y usarlas...
        Monomio m1 = new Monomio();
        //o bien, utilizar su nombre completo incluyendo el prefijo mipaquete...
```

```
Polinomios.Monomio m2 = new Polinomios.Monomio(2, 5,5);
System.out.println("El monomio m2 es de grado " + m2.Grado() +
    " y de coeficiente " + m2.Coeficiente());
}
}
```

Ejercicio 15 Realice un programa en el que importe el paquete `java.Math` y el paquete `java.util.Scanner`. Use en él la clase `Scanner` para leer números por la entrada y que use dos funciones del paquete `java.math`. Puede ver un ejemplo en 7.11.

5. Clases de envoltura

A veces puede ser necesario que las variables de los tipos básicos en Java funcionen como si fueran objetos. Para conseguir este efecto, existen las clases de envoltura (o *wrappers*, en inglés). Las clases de envoltura disponen de:

- Constructores a partir de un tipo simple.
- Constructores a partir de una cadena.
- Método `toString()` que transforma el valor en una cadena.
- Método que devuelve el tipo básico. El nombre se forma concatenando el tipo básico y `Value`: `charValue()`, `intValue()`, etc.
- Método `equals()` para comparar el valor entre envoltorios.

Los constructores son métodos propios de la orientación a objetos. Son funciones especiales que sirven para definir e inicializar los objetos.

Un ejemplo de cómo se usan es el siguiente:

```
int dato = 3;
...
Integer d = new Integer(dato);
```

Los tipos básicos `byte` y `short` se incluyen en Java por razones de eficiencia, por lo que no tienen envoltorios predefinidos, usándose la clase `Integer`. Para los demás tipos simples, las clases de envoltura son:

Clase envoltorio	Tipo básico
Boolean	boolean
Character	char
Integer	int
Long	long
Float	float
Double	double

Cuadro 5: Clases de envoltura definidas en Java

Un ejemplo algo más complejo donde se usan las clases de envoltura es el siguiente:

```
public class Envoltura
{
    public static void main (String[] args)
    {
        float s = 0;
        s = 10 / s;
        Float infinito = new Float(s);

        //Imprimirá "infinito es Infinity"
        System.out.println("infinito es " + infinito);
        Float noNumero = new Float(Math.sqrt(-1));
        //Imprimirá "noNumero es NaN"
        System.out.println("noNumero es " + noNumero);
    }
}
```

Ejercicio 16 Revise el paquete `java.Math` y realice un programa que pida al usuario varios números y calcule el seno, el coseno, la tangente, el máximo entre dos valores, el mínimo entre dos valores y el logaritmo. Encapsule los datos que reciba en clases de envoltura para asegurarse del correcto funcionamiento de los cálculos, y muéstrellos por pantalla. Puede ver un ejemplo en 7.12.

6. Ejercicios para profundizar

En esta sección se encuentran propuestos algunos ejercicios adicionales que utilizan elementos explicados a lo largo del presente guión. Es aconsejable que los realice para mejorar su dominio de los contenidos del tema.

Ejercicio 17 Escriba un programa para calcular el consumo medio de un automóvil. Para ello el programa debe solicitar información sobre las tres últimas veces que se repostó combustible. De la primera solicitará el precio del litro del combustible, el total pagado en llenar el depósito y el número de kilómetros que marcaba el cuentakilómetros. De la segunda vez sólo solicitará el precio del litro del combustible y el total pagado en llenar el depósito, y de la tercera vez solicitará el valor que indicaba el cuentakilómetros. Con estos datos debe calcular el consumo por cada 100 km y el coste por kilómetro. Cuando lo termine, puede comprobar su solución con el código en el apartado 7.13.

Ejercicio 18 Escriba un programa que escriba en la pantalla el valor de la suma de los n primeros números pares. Puede comprobar su programa con el que aparece en 7.14.

Ejercicio 19 Investigue la clase `String`. Luego, escriba un programa que declare una cadena de caracteres denominada `nombre`, con valor inicial "Juan", `primerApellido`, con valor "García" y `segundoApellido` con valor "López". Después, el programa concatena a `nombre` el primer apellido y luego el segundo. Por último, el programa imprime el nombre completo y su longitud. Cuando termine, puede comparar su código con el programa propuesto en 7.15.

Ejercicio 20 Cuando haya realizado el ejercicio anterior, escriba un método llamado `escribeAlReves` que reciba un `String` y escriba dicho texto en la pantalla al revés. Cree un programa para ejecutarlo y comprobar su método. Puede compararlo con el código ofrecido en 7.16.

Ejercicio 21 Tras haber hecho los ejercicios anteriores, escriba un programa que cuente el número de vocales que aparecen en un texto que se le solicita al usuario. Puede comprobar su solución con la que aparece en 7.17.

Ejercicio 22 Investigue sobre la clase `Math`. Después escriba un método que dado un ángulo en radianes, lo escriba en pantalla en grados, así como los valores del seno, el coseno y la tangente. Haga un programa que imprima los valores de esas tres funciones para los valores 0, $\pi/4$, $\pi/2$, y π en radianes. Puede comparar su programa, cuando termine, con el que aparece en 7.18.

7. Soluciones de los ejercicios

7.1. Ejercicio 3

Esta es una posible solución para el ejercicio 3.

```
/**
 * Programa en Java que muestra mis datos.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class MisDatos
{
    String nombre = "Sara";
    String apellidos = "Sánchez Sánchez";
    String ciudad = "Cádiz";
    String ocupacion = "estudiante";
    int edad = 22;
    int asignaturasAprobadas = 12;
    int asignaturasCursadas = 15;

    public int asigSuspensas (int asigCursadas, int asigAprobadas)
    {
        return (asigCursadas - asigAprobadas);
    }

    public static void main (String[] args)
    {
        MisDatos yo = new MisDatos();

        System.out.println("Me llamo " + yo.nombre + " " + yo.apellidos + ".");
        System.out.println("Vivo en " + yo.ciudad + " y tengo " + yo.edad + " años.");
        System.out.println("Soy " + yo.ocupacion + ". He cursado " +
            yo.asignaturasCursadas + " asignaturas: he aprobado " +
            yo.asignaturasAprobadas + ", y he suspendido " +
            yo.asigSuspensas(yo.asignaturasCursadas, yo.asignaturasAprobadas) + ".");
    }
}
```

7.2. Ejercicio 4

A continuación la solución del ejercicio 4. En los casos en los que el identificador parece que se refiere claramente a un objeto, una clase o una constante y puede mejorarse, sólo se incluye el caso al que se refiere. Puede practicar más añadiendo soluciones para identificar objetos, clases y constantes para cada identificador.

- a) `mi carta`: este no es un identificador porque el espacio no está permitido. Este identificador sería adecuado para un objeto si fuera `miCarta`.

- b) **unacarta**: este identificador es correcto pero no es recomendado. Podría mejorarse para que se refiriera a un objeto siendo **unaCarta**.
- c) **mis2escritos**: este identificador es correcto. Otra forma para que este identificador resultara más claro y se refiriera a un objeto es **misDosEscritos**.
- d) **4cientos**: este no es un identificador porque un identificador no puede empezar por dígito. Un posible identificador para un objeto sería **cuatrocientos**. Para una constante sería mejor **CUATROCIENTOS**.
- e) **es_un_mensaje**: este identificador es correcto, pero no recomendado. Para describir un objeto sería mejor **esUnMensaje**, para describir una clase sería mejor **EsUnMensaje** y para describir una constante sería mejor **ES_UN_MENSAJE**.
- f) **no_vale nada**: este identificador no es correcto porque el espacio no está permitido. Para describir un objeto el adecuado es **noValeNada**, para describir una clase es **NoValeNada** y para describir una constante es **NO_VALE_NADA**.
- g) **_____ejemplo_____**: este identificador es correcto pero no es recomendado. Para describir un objeto el identificador más adecuado es **ejemplo**.
- h) **mi-programa**: este identificador no es válido, ya que el símbolo - lo puede tomar por un operador. Para describir un objeto el identificador adecuado es **miPrograma**, para una clase es **MiPrograma** y para una constante es **MI_PROGRAMA**.
- i) **¿cuantos?**: este identificador no es válido ya que no empieza por una letra y no puede contener ? por considerarse un operador. Para un objeto el identificador válido es **cuantos**.
- j) **el%Descontado**: este identificador no es válido ya que contiene el símbolo % que es un operador. Para un objeto el identificador adecuado es **elPorcentajeDescontado**. Si se tratara de una clase, el mejor sería **ElPorcentajeDescontado**. En cambio, para una constante es mejor **EL_PORCENTAJE_DESCONTADO**.
- k) **a150PORHORA**: este identificador es válido, aunque podría escribirse de una manera más clara. Para una constante, la mejor forma sería **A_150_POR_HORA**.
- l) **TengoMUCHOS\$\$\$**: este identificador es correcto, aunque debe evitarse el uso del carácter \$ siempre que sea posible. Una versión para identificar objetos es **tengoMuchos\$**. Para una constante podría ser **TENGO_MUCHOS_\$**.
- m) **LOS400GOLPES**: este identificador es correcto. Para una constante se puede usar **LOS_400_GOLPES** para mejorar la claridad.
- n) **quieroUNAsolución**: este identificador es correcto, pero puede hacerse más claro. Para un objeto se podría utilizar **quieroUnaSolución**, para una clase **QuieroUnaSolución** y para una constante **QUIERO_UNA_SOLUCIÓN**.

7.3. Ejercicio 5

En este ejercicio hay que sustituir, en cada una de las expresiones, la variable por su valor cuando se va a utilizar. En el cálculo hay que tener en cuenta la precedencia de los operadores y que los resultados de las operaciones serán números enteros. Tenga en cuenta al realizar operaciones, si estás cambian el valor inicial de las variables.

a) `!(a > b && 2 * a <= b)`

En primer lugar se evalúa la parte del paréntesis. Para ello se evalúa primero la parte izquierda del operador `&&`:

$!(a > b \&\& 2 * a <= b) \equiv !(5 > 3 \&\& 2 * a <= b) \equiv !(true \&\& 2 * a <= b)$

Como la parte izquierda del operador `&&` vale **true**, hay que evaluar su parte derecha:

$!(true \&\& 2 * a <= b) \equiv !(true \&\& 2 * 5 <= 3) \equiv !(true \&\& 10 <= 3) \equiv !(true \&\& false)$

De donde evaluando los operadores booleanos:

$!(true \&\& false) \equiv !(false) \equiv true$

b) `++b < 3 || a + b <= 8 && !(a > b)`

Esta expresión se evalúa de izquierda a derecha. Como tiene mayor prioridad el operador `&&` que el operador `||` se evalúa como si estiese escrito $(e1 || (e2 \&\& e3))$, por tanto se evalúa `e1` y si su valor es **false** se evalúa la parte derecha del operador `||`:

$++b < 3 || a + b <= 8 \&\& !(a > b) \equiv ++3 < 3 || a + b <= 8 \&\& !(a > b) \equiv 4 < 3 || a + b <= 8 \&\& !(a > b) \equiv false || a + b <= 8 \&\& !(a > b)$

Al evaluar antes `++b`, `b` se incrementa en 1 pasando a valer 4. Como la parte izquierda de la expresión vale **false** hay que evaluar la parte derecha. Se comienza entonces por la parte izquierda del operador `&&`:

$false || a + b <= 8 \&\& !(a > b) \equiv false || 5 + 4 <= 8 \&\& !(a > b) \equiv false || 9 <= 8 \&\& !(a > b) \equiv false || false \&\& !(a > b)$

Como la parte izquierda del operador `&&` vale **false** ya no es necesario evaluar la parte derecha:

$false || false \&\& !(a > b) \equiv false || false \equiv false$

c) `++a <= 6 && (b += 2) < a`

Esta expresión sólo tiene un operador booleano `&&`. Por tanto, se evalúa en primer lugar su parte izquierda:

$++a <= 6 \&\& (b += 2) < a \equiv ++5 <= 6 \&\& (b += 2) < a \equiv 6 <= 6 \&\& (b += 2) < a \equiv true \&\& (b += 2) < a$

Como puede ver, el valor de la variable `a` se incrementa en 1 a 6. A continuación se evalúa la parte derecha del operador `&&`. La expresión `(b += 2)` añade a `b` el valor 2 y devuelve el valor resultado de la asignación:

$true \&\& (b += 2) < a \equiv true \&\& (5) < a \equiv true \&\& 5 < 6 \equiv true \&\& true \equiv true$

El resultado final es **true** y las variables `a` y `b` han quedado con los valores `a = 6` y `b = 5`.

d) `++a / 2 <= b && (++a / 2 > b || (a * 2 < b * 4))`

Esta expresión consta de un operador booleano (`&&`) entre dos expresiones, por lo que en primer lugar se evalúa su parte izquierda:

$++a / 2 <= b \&\& (++a / 2 > b || (a * 2 < b * 4)) \equiv ++5 / 2 <= b \&\& (++a / 2 > b || (a * 2 < b * 4)) \equiv 6 / 2 <= b \&\& (++a / 2 > b || (a * 2 < b * 4)) \equiv 3 <= 3 \&\& (++a / 2 > b || (a * 2 < b * 4)) \equiv true \&\& (++a / 2 > b || (a * 2 < b * 4))$

Al evaluarse la expresión `++a`, la variable `a` se ha incrementado en 1, valiendo en este momento 6. Como la parte izquierda del operador `&&` vale **true** hay que evaluar su parte derecha. Para ello se comienza evaluando la parte izquierda del operador `||`:

$true \&\& (++a / 2 > b || (a * 2 < b * 4)) \equiv true \&\& (++6 / 2 > b || (a * 2 < b * 4)) \equiv true \&\& (7 / 2 > b || (a * 2 < b * 4)) \equiv true \&\& (3 > 3 || (a * 2 < b * 4)) \equiv true \&\& (false || (a * 2 < b * 4))$

Al evaluar la `++a`, la variable `a = 6` se incrementa en 1 pasando a valer 7. Como la parte izquierda del operador `||` vale **false** hay que evaluar su parte derecha:

$true \&\& (false || (a * 2 < b * 4)) \equiv true \&\& (false || (7 * 2 < 3 * 4)) \equiv true \&\& (false || (14 < 12)) \equiv true \&\& (false || (false)) \equiv true \&\& (false) \equiv false$

Evaluándose, finalmente, la expresión a **false**.

7.4. Ejercicio 7

Esta es una posible solución para el ejercicio 7.

```
/**
 * Programa en Java que pide al usuario la base y la altura
 * de un triángulo y calcula su área.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.Scanner;

public class AreaTriangulo
{
    public static void main (String[] args)
    {
        double base;
        double altura;

        Scanner teclado = new Scanner(System.in);

        System.out.println("Introduzca los datos del triángulo: ");
        System.out.print("Base: ");
        base = teclado.nextDouble();
        System.out.print("Altura: ");
        altura = teclado.nextDouble();
        System.out.print("El área del triángulo es: ");
        System.out.println((base * altura) / 2);
    }
}
```

7.5. Ejercicio 8

Esta es una posible solución para el ejercicio 8.

```
/**
 * Programa en Java que pide al usuario tres números enteros
 * e imprime si son pares o impares.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.Scanner;

public class ParImpar
{
    void calculaPar (int a) {
        if (a % 2 == 0)
```

```

        System.out.println("El número " + a + " es par.");
    else
        System.out.println("El número " + a + " es impar.");
}

public static void main (String[] args)
{
    int num1, num2, num3;
    ParImpar obj = new ParImpar();

    Scanner teclado = new Scanner(System.in);

    System.out.print("Introduzca un número entero: ");
    num1 = teclado.nextInt();
    obj.calculaPar(num1);
    System.out.print("Introduzca otro número entero: ");
    num2 = teclado.nextInt();
    obj.calculaPar(num2);
    System.out.print("Introduzca un último número entero: ");
    num3 = teclado.nextInt();
    obj.calculaPar(num3);
}
}

```

7.6. Ejercicio 9

Esta es una posible solución para el ejercicio 9.

```

/**
 * Programa en Java que muestra una cuenta atrás
 * desde 5 hasta 0.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class CuentaAtras
{
    public static void main (String[] args)
    {
        int n = 5;

        System.out.println("Empieza la cuenta atrás: ");
        while (n >= 0)
        {
            System.out.println(n);
            --n;
        }
        System.out.println("Fin de la cuenta atrás.");
    }
}

```

7.7. Ejercicio 10

Esta es una posible solución para el ejercicio 10.

```
/**
 * Programa en Java que muestra los 10 primeros múltiplos de 29.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class Multiplos
{
    public static void main (String[] args)
    {
        int n = 1, x = 29;

        System.out.print("Los 10 primeros múltiplos de 29 son: ");
        for(; n <= 10; ++n) {
            System.out.print(x * n + " ");
        }
        System.out.println();
    }
}
```

7.8. Ejercicio 11

Esta es una posible solución para el ejercicio 11.

```
/**
 * Programa en Java que pide al usuario las dimensiones de
 * un rectángulo y lo imprime por pantalla.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.Scanner;

public class Rectangulo
{
    void imprimeRectangulo (int base, int alt) {
        int a = 1, b = 1;

        do {
            b = 1;
            do {
                System.out.print("* ");
            }
        }
    }
}
```

```

        ++b;
    } while(b <= base);
    System.out.println();
    ++a;
    } while(a <= alt);
    System.out.println();
}

public static void main (String[] args)
{
    int base, altura;
    Rectangulo rect = new Rectangulo();

    Scanner teclado = new Scanner(System.in);

    System.out.print("Introduzca el valor de la base: ");
    base = teclado.nextInt();
    System.out.print("Introduzca el valor de la altura: ");
    altura = teclado.nextInt();
    rect.imprimeRectangulo(base, altura);
}
}

```

7.9. Ejercicio 12

Esta es una posible solución para el ejercicio 12.

```

/**
 * Programa en Java que pide al usuario el día, mes y año
 * de una fecha e indica si es una fecha válida.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.Scanner;

public class FechaOK
{
    public boolean esFechaOK (int dia, int mes, int año) {
        int diasMes;

        //Se comprueba que el mes es correcto.
        if (mes < 1 || mes > 12)
            return false;
        if (año < 1600 || año > 3000)
            return false;

        //Es un mes correcto, se calcula los días que tiene.
        switch (mes) {

```

```

        case 4:
        case 6:
        case 9:
        case 11:
            diasMes = 30;
            break;
        case 2:
            if ((año % 4 == 0) && (año % 100 != 0) || (año % 400 == 0)) {
                diasMes = 29;
            }
            else {
                diasMes = 28;
            }
            break;
        default:
            diasMes = 31;
            break;
    }

    //Se comprueba que el día es correcto.
    return dia >= 1 && dia <= diasMes;
}

public static void main (String[] args)
{
    int dia, mes, año;
    FechaOK f = new FechaOK();

    Scanner teclado = new Scanner(System.in);

    System.out.print("Introduzca el día: ");
    dia = teclado.nextInt();
    System.out.print("Introduzca el mes: ");
    mes = teclado.nextInt();
    System.out.print("Introduzca el año: ");
    año = teclado.nextInt();
    if (f.esFechaOK(dia, mes, año))
        System.out.println("La fecha es correcta.");
    else
        System.out.println("La fecha es incorrecta.");
}
}

```

7.10. Ejercicio 14

Esta es una posible solución para el ejercicio 14.

```

/**
 * Programa en Java que pide al usuario dos números
 * y calcula su media.

```

```

*
* @author Natalia Partera
* @version 1.0
*/

import java.util.Scanner;

public class Media {
    public static void main (String[] args)
    {
        double med, x, y;

        Scanner teclado = new Scanner(System.in);

        med = 0;
        System.out.print("Introduzca un número: ");
        x = teclado.nextDouble();
        System.out.print("Introduzca el otro número: ");
        y = teclado.nextDouble();
        med = (x + y) / 2;
        System.out.println("La media de " + x + " y " + y + " es " + med);
    }
}

```

7.11. Ejercicio 15

Esta es una posible solución para el ejercicio 15.

```

/**
 * Programa en Java que pide al usuario números
 * y realiza cálculos con ellos.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.Scanner;

public class Paquetes
{
    public static void main (String[] args)
    {
        double base, potencia;
        int exponente;
        double radicando, raiz;

        Scanner teclado = new Scanner(System.in);

        System.out.println("Introduzca los siguientes datos: ");
        System.out.print("Base: ");
    }
}

```

```

        base = teclado.nextDouble();
        System.out.print("Exponente: ");
        exponente = teclado.nextInt();
        potencia = Math.pow(base, exponente);
        System.out.println(base + " elevado a " + exponente + " = " + potencia);
        System.out.print("Radicando: ");
        radicando = teclado.nextDouble();
        raiz = Math.sqrt(radicando);
        System.out.println("La raíz cuadrada de " + radicando + " es " + raiz);
    }
}

```

7.12. Ejercicio 16

Esta es una posible solución para el ejercicio 16.

```

/**
 * Programa en Java que pide al usuario números
 * y realiza cálculos con ellos.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.Scanner;
import static java.lang.Math.*;

public class MathEnvoltura
{
    public static void main (String[] args)
    {
        double x, y, z;

        Scanner teclado = new Scanner(System.in);

        System.out.print("Introduzca el primer número: ");
        x = teclado.nextDouble();
        Float xf = new Float(x);
        System.out.print("Introduzca el segundo número: ");
        y = teclado.nextDouble();
        Float yf = new Float(y);
        System.out.print("Introduzca el tercer número: ");
        z = teclado.nextDouble();
        Float zf = new Float(z);

        Float seno = new Float(sin(xf));
        System.out.println("El seno de " + xf + " es " + seno);
        Float coseno = new Float(cos(yf));
        System.out.println("El coseno de " + yf + " es " + coseno);
        Float tangente = new Float(tan(zf));
    }
}

```

```

        System.out.println("La tangente de " + zf + " es " + tangente);

        System.out.println("El máximo entre " + xf + " y " + yf + " es " + max(xf, yf));
        System.out.println("El mínimo entre " + yf + " y " + zf + " es " + min(yf, zf));

        Float logaritmo = new Float(log(xf));
        System.out.println("El logaritmo neperiano de " + xf + " es " + logaritmo);
    }
}

```

7.13. Ejercicio 17

Esta es una posible solución para el ejercicio 17.

```

/**
 * Programa en Java que calcula el consumo medio de
 * un automóvil según 3 repostajes consecutivos.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.Scanner;

public class ConsumoAuto {
    public static void main(String[] args) {
        double precioLitro, litros = 0;
        double pagado, coste = 0;
        int kmInicial, kmFinal, km;

        Scanner teclado = new Scanner(System.in);

        System.out.print("Introduzca el precio por litro del primer repostaje: ");
        precioLitro = teclado.nextDouble();
        System.out.print("Introduzca el coste total del primer repostaje: ");
        pagado = teclado.nextDouble();
        System.out.print("Introduzca el valor del cuentakilómetros en el primer
            repostaje: ");
        kmInicial = teclado.nextInt();
        litros = pagado / precioLitro;
        coste = pagado;

        System.out.println();
        System.out.print("Introduzca el precio por litro del segundo repostaje: ");
        precioLitro = teclado.nextDouble();
        System.out.print("Introduzca el coste total del segundo repostaje: ");
        pagado = teclado.nextDouble();
        litros += pagado / precioLitro;
        coste += pagado;
    }
}

```



```

        System.out.println();
        System.out.print("Introduzca el valor del cuentakilómetros en el tercer
        repostaje: ");
        kmFinal = teclado.nextInt();
        km = kmFinal - kmInicial;

        System.out.println("El consumo medio del automóvil es de " + (litros / km * 100)
        + " litros por cada 100 Km.");
        System.out.println("El gasto medio es de " + coste / km + " por kilómetro.");
    }
}

```

7.14. Ejercicio 18

Esta es una posible solución para el ejercicio 18.

```

/**
 * Programa en Java que calcula la suma de los n
 * primeros números pares.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.Scanner;

public class SumaPares {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        int numero, suma = 0;

        System.out.print("Introduzca el número n: ");
        numero = teclado.nextInt();

        for(int i = 0; i < 2 * numero ; i = i + 2) {
            suma = suma + i;
        }

        System.out.println("La suma de los primeros " + numero + " numeros pares es: " +
        suma);
    }
}

```

7.15. Ejercicio 19

Esta es una posible solución para el ejercicio 19.

```
/**
 * Programa en Java que trabaja con cadenas
 * de la clase String.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class Cadenas {
    public static void main(String[] args) {
        String nombre = "Juan";
        String primerApellido = "Pérez";
        String segundoApellido = "López";

        nombre += primerApellido;
        nombre += segundoApellido;

        System.out.println(nombre + " tiene " + nombre.length() + " letras");
    }
}
```

7.16. Ejercicio 20

Esta es una posible solución para el ejercicio 20.

```
/**
 * Programa en Java que trabaja con cadenas
 * de la clase String y las invierte.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class InvertirCadena {
    public void escribeAlReves(String texto) {
        for(int i = texto.length()-1; i >= 0; i--) {
            System.out.print(texto.charAt(i));
        }
    }

    public static void main(String[] args) {
        String palindromo = "Dabale arroz a la zorra el abad";
        String quijote = "En un lugar de la Mancha ...";

        InvertirCadena ic = new InvertirCadena();

        System.out.println("Sin invertir:");
    }
}
```

```

        System.out.println(palindromo);
        System.out.println(quirote);

        System.out.println("Al revés:");
        ic.escribeAlReves(palindromo);
        System.out.println();
        ic.escribeAlReves(quirote);
        System.out.println();
    }
}

```

7.17. Ejercicio 21

Esta es una posible solución para el ejercicio 21.

```

/**
 * Programa en Java que cuenta el número de
 * vocales de un texto dado.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.Scanner;

public class CuentaVocales {
    public static void main(String[] args) {
        int vocalA, vocalE, vocalI, vocalO, vocalU;
        vocalA = vocalE = vocalI = vocalO = vocalU = 0;

        Scanner teclado = new Scanner(System.in);

        System.out.print("Introduzca un texto: ");
        String texto = teclado.nextLine();
        for(int i = 0; i < texto.length(); ++i) {
            switch(Character.toUpperCase(texto.charAt(i))) {
                case 'A':
                    vocalA++;
                    break;
                case 'E':
                    vocalE++;
                    break;
                case 'I':
                    vocalI++;
                    break;
                case 'O':
                    vocalO++;
                    break;
                case 'U':
                    vocalU++;

```

```

        break;
    default:
        break;
    }
}

System.out.println("La vocal A aparece " + vocalA + " veces.");
System.out.println("La vocal E aparece " + vocalE + " veces.");
System.out.println("La vocal I aparece " + vocalI + " veces.");
System.out.println("La vocal O aparece " + vocalO + " veces.");
System.out.println("La vocal U aparece " + vocalU + " veces.");
}
}

```

7.18. Ejercicio 22

Esta es una posible solución para el ejercicio 22.

```

/**
 * Programa en Java que realiza cálculos trigonométricos.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import static java.lang.Math.*;

public class Trigonometria
{
    void funciones(double alfa) {
        System.out.println(toDegrees(alfa) + " " + sin(alfa) + " " + cos(alfa) + " " +
            tan(alfa));
    }

    public static void main (String[] args)
    {
        Trigonometria trig = new Trigonometria();

        trig.funciones(0);
        trig.funciones(PI / 4);
        trig.funciones(PI / 2);
        trig.funciones(PI);
    }
}

```
