Unidad 4

Elementos de un programa informático

Programación

4

Índice

Introducción

Esta unidad tratará sobre los elementos que forman parte de un programa informático que haya sido desarrollado en Java. Se abordarán de forma general para posteriormente ser cada vez más específicos.

Los lenguajes de programación cuentan con unas reglas de sintaxis, las cuales debemos respetar a la hora de escribir nuestro código, ya que, si las instrucciones que escribimos no son sintácticamente correctas, el ordenador no podrá interpretarlas, por lo que serán rechazadas.

Al finalizar esta unidad

- Analizaremos la estructura y elementos básicos de un programa informático.
- Conoceremos los tipos de datos primitivos.
- Comprenderemos el uso de operaciones y conoceremos su orden de procedencia.
- Aprenderemos a identificar correctamente los diferentes elementos de un programa.

4.1. Soluciones y proyectos

El proyecto es el primer elemento a considerar, un proyecto consiste en un grupo de archivos y carpetas que se organizan de acuerdo a un criterio que dé lugar a un orden lógico. Los proyectos de Java suelen contar con archivos de código intermedio (.class), archivos de código fuente (.java) y otro tipo de archivos que se utilizan desde el programa, como pueden ser imágenes y archivos de texto plano. De manera que, para llevar a cabo el desarrollo de un programa, lo primero que deberemos hacer será crear el proyecto.

Cuando un proyecto o varios que están relacionados entre sí se agrupan, forman una solución, sin embargo, una solución puede contener también archivos sin conexión con los proyectos de esa solución. En la tecnología .NET también encontramos este tipo de conceptos, por ejemplo, en NetBeans, se denominan Project Groups.

Imagen 1. Grupo de proyectos.

4.2. Identificadores

Un identificador sirve para que el programador denomine un elemento del programa, ya sean variables, clases, métodos, etc. Y este consiste en una secuencia de uno o varios caracteres. Los identificadores se forman siguiendo estas reglas:

- Por lo menos debe contener un carácter.
- Solo pueden usarse letras (a-Z), dígitos (0-9), el símbolo dólar ($) o el guion bajo (_).
- No se puede utilizar un dígito como primer carácter.

El lenguaje es bastante flexible con los identificadores, pero aun así, conviene seguir una serie de recomendaciones:

- En paquetes es conveniente usar minúsculas, escrito sin espacios. Ya que un paquete puede estar contenido dentro de otro, utilizaremos le carácter punto "." para determinar la jerarquía.
- Todas las palabras juntas y la primera letra en mayúsculas para clases e interfaces.
- Todas las palabras juntas y la primera letra en minúsculas para métodos y variables.
- Todas las palabras en mayúsculas y separadas por "_" para las constantes.

4.3. Palabras reservadas

Existen ciertas palabras que cuentan con su propio significado, por lo que no se pueden utilizar como identificadores en los programas. A continuación, la lista de palabras reservadas:

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

Imagen 2. Lista de palabras reservadas.

4.4. Clases

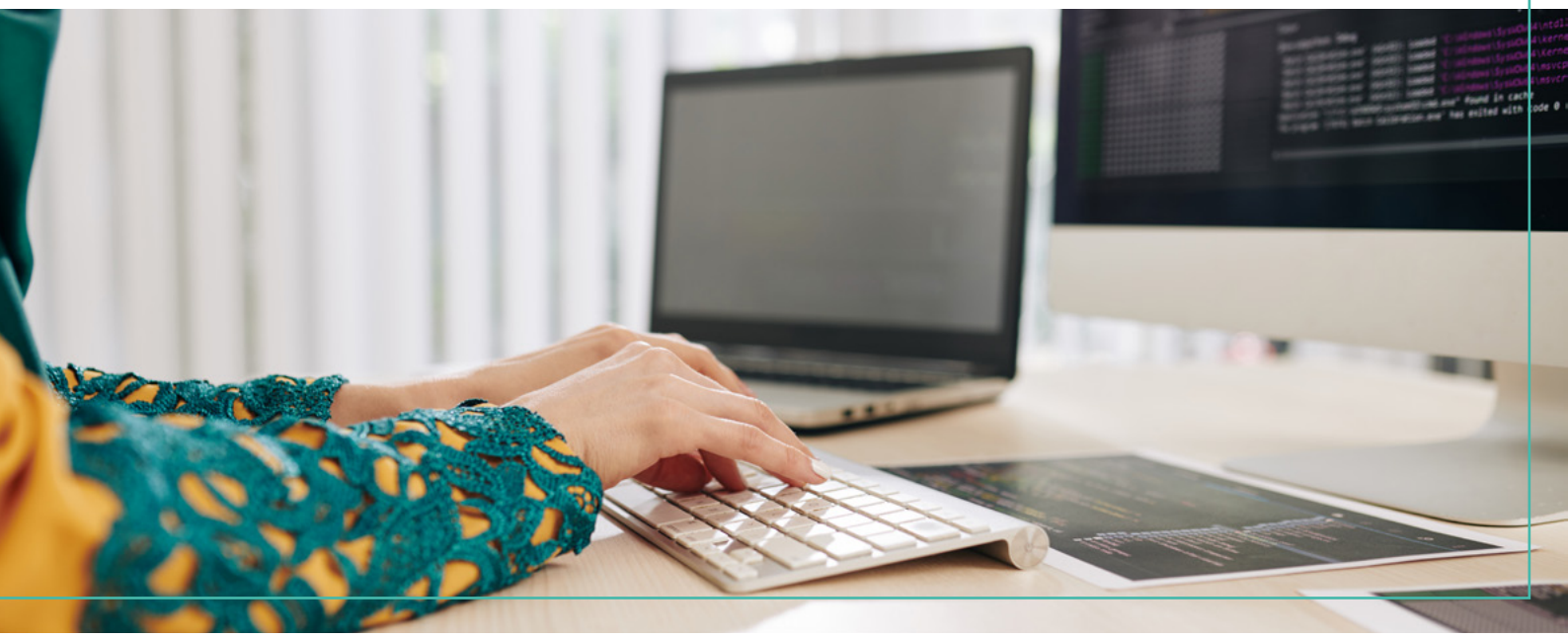
La POO tiene como objetivo la encapsulación y la modularidad. Para ello, Java cuenta, entre otros, con el elemento Clase.

En el apartado siguiente podemos ver un ejemplo simple sobre como declarar una clase, pero no será hasta la unidad 6 que veremos detenidamente su uso en Java.

4.4.1. Método main

Para comenzar su ejecución, todos los programas cuentan con un punto de entrada; en el caso de Java, sería el método main. Debe ser declarado desde una de las clases del programa y debe ser estático, público y conservar las especificaciones de los parámetros de entrada. Así mismo, tanto identificador de clase, como el archivo en el cual haya sido escrito, deben tener el mismo nombre. Sintaxis para crear una clase principal y el método main.

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



4.5. Paquetes

Otro elemento de Java para conseguir esa encapsulación y modularidad es el paquete. Las ventajas de este mecanismo son:

- Este mecanismo identifica las clases cuyas características sean comunes y las agrupa.
- Mantiene independientes las declaraciones que son realizadas en un paquete de las que existen en otros paquetes (esto nos permite usar los mismos identificadores en caso de necesitarlo).
- Podemos controlar la visibilidad de los paquetes, y esto afecta a aquello elementos contenidos en él. Esto nos facilita un mecanismo de control de acceso.

Para declarar un paquete se usa la palabra reservada package, seguidamente se escribe el nombre que le vamos a dar a dicho paquete, todo esto en la primera línea del código del archivo y al final terminaremos con el carácter punto y coma (;). El nombre que elijamos para el paquete puede contener varias palabras, pero estas deben ir seguidas y sin ningún tipo de separador. Es necesario también declarar cada una de las clases que pertenezcan al paquete en la primera línea.

También existe la jerarquía de paquetes, ya que unos paquetes pueden contener a otros. Esta jerarquía se representa separando con puntos los nombres de los paquetes. Podemos acceder al contenido del paquete desde las clases que pertenezcan a dicho paquete, para ellos solo será necesario especificar el nombre de la clase. No obstante, en caso de que la clase pertenezca a un paquete distinto, habrá que especificar el nombre del paquete primero y luego el de la clase, o importar los contenidos del paquete. Antes de llevar a cabo esta importación, se deberá declarar la pertenencia al paquete de la actual clase, y para referirnos a los elementos contenidos en este, utilizaremos el carácter asterisco (*).

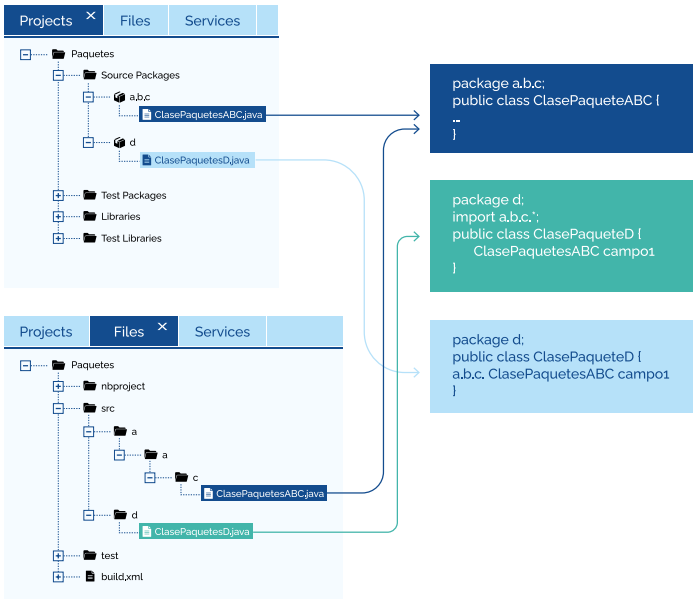


Imagen 3. Estructura física y lógica con paquetes.

Siempre que trabajemos desde el terminal con paquetes, debemos de recordar que, a la hora de compilar un programa, es conveniente que indiquemos el directoria donde serán depositados los ficheros .class (modificador -d), además de que, para crear el jar ejecutable o para la ejecución simple, deberemos colocar primero el nombre de los paquetes y luego el nombre de la clase contenida en ese paquete que implementa el método main.

4.6. Variables

Las variables constituyen elementos muy importantes para la programación, permiten almacenar en la memoria principal valores mientras el programa se está ejecutando. En Java podemos distinguir dos tipos: las referencias a objetos y las de tipo primitivo. Estas variables pueden ser declaradas tanto en la misma línea como en líneas diferentes. De cualquier manera, estas líneas deben acabar en ";". Este sería un ejemplo de cómo declarar variables (aquellas palabras que aparecen entre corchetes son opcionales):

```
[Visibilidad] tipo nombreVariable1;
nombreVariable1 = valor1;
[Visibilidad] tipo nombreVariable2 = valor2;
```

4.6.1. Ámbito de una variable

Dependiendo de cómo declaremos una variable, esta podrá accederse desde una zona del código u otra. A esta característica se la conoce como alcance, ámbito o scope de la variable. Cuando hablamos del ámbito de una variable debemos considerar:

- Cuando un código se encuentra entre llaves "{}", este pertenece al mismo ámbito.
- Podemos acceder a las variables declaradas en un ámbito, ya sea desde ese mismo ámbito o desde subámbitos que se creen dentro de este.
- Una variable será destruida cuando salgamos del ámbito donde fue declarada.

En este ejemplo podemos observar la forma en que el ámbito de declaración de una variable afecta a la hora de acceder a ella:

```
public class Ambitos {
    //ejemplo ámbitos
    public static void main(String[] args) {
        {
            {
                int a = 2;
                System.out.println("a = "+ a);
            }
            // La línea siguiente provocaría error
            // System.out.println("a = "+ a);
            int a = 1;
            int b = 1000;
            System.out.Println("a = "+ a);
        }
        // La línea siguiente provocaría error
        // System.out.Println("b = "+b)
        int a = 0;
        System.out.println("a = "+ ++a);
        {
            System.out.println("a = "+ ++a);
            {
                System.out.println("a = "+ ++a);
            }
        }
        System.out.println("a = "+ ++a)
    }
```

```
run:
a = 2
a = 1
a = 1
a = 2
a = 3
a = 4
```

Imagen 3. Ejemplo de ámbito de declaración.

Como podemos observar, se incluye un elemento en la sintaxis de declaración de una variable para establecer su visibilidad. Este no afecta al ámbito de dicha variable dentro de la clase si se trata de una variable miembro (campo).

4.7. Constantes

Las constantes son elementos que, a lo largo de la vida del programa, mantienen un mismo valor. En Java han de declararse en la misma línea, y se hace como si fuera una simple variable, pero debemos escribir static final antes del tipo. He aquí un ejemplo:

```
[Visibilidad] static final tipo NOMBRE_CONSTANTE = valor;
```

4.8. Tipos de datos primitivos

Dependiendo del tipo que sean los datos, podrán contener un tipo de valores y se podrán realizar ciertas operaciones sobre ellos. Java es un lenguaje de tipo estático, de manera que se deben declarar las variables antes de su uso.

Los datos primitivos son aquellos que ya están definidos en el lenguaje por defecto, por lo que, para referenciarlos, hay que utilizar una palabra reservada.

Nombre	Tipo de dato	Ocupa	Rango	Valor defecto
byte	Entero (signo)	1 byte	-128 a 127	0
short	Entero (signo)	2 bytes	-32768 a 32767	0
int	Entero (signo)	4 bytes	-231 a 231-1	0
	Entero		0 a 232-1	
long	Entero (signo)	8 bytes	-263 a 263-1	0L
	Entero		0 a 264-1	
float	Decimal simple	4 bytes	Punto flotante 32-bit IEEE 754	0.0f
double	Decimal doble	8 bytes	Punto flotante 64-bit IEEE 754	0.0d
char	Carácter simple	2 bytes	'\u0000' a '\ uffff' (65,535)	'\ u0000'
boolean	Booleano	1 byte	true o false	false

Imagen 4 Tipos de datos primitivos.

4.8.1. Conversiones de tipo (casting)

El moldeado, casting, cast, tipado o conversión nos permite transferir la información desde una variable declarada de un tipo a otro diferente, y viceversa.

Al realizar una conversión pueden haber pérdidas de información, por lo que, normalmente, las conversiones que no suelen dar problemas son las que almacenan una cantidad de bytes menor a la del tipo de destino.

Tipo de dato origen	Tipo de dato destino
byte	double, float, long, int, char, short
char	double, float, long, int
short	
int	double, float, long
long	double, float
float	double

Imagen 5. Conversiones seguras.

Cuando realizamos conversiones desde tipos enteros a tipos decimales, pueden ocurrir errores si trabajamos con valores muy altos, pero el sistema nos dejará llevarlas a cabo sin informarnos de errores.

Podemos llevar a cabo sin riesgo algunas conversiones que en principio no son seguras, dependiendo de los valores que manejemos, como, por ejemplo:

```
Byte e = 's' ;
```
← e almacenará el código que representa a 's'

Implícitas

Suceden cuando se lleva a cabo una operación de asignación, pero no hemos declarado el tipo de conversión a realizar. Ejemplos:

```
byte b = 1;
short s = b;
int i = s;
long l = i;
float f = l;
double d = f;
```

Explícitas

Se llevan a cabo colocando primero el tipo de variable destino y después el dato origen de la conversión. Hay algunas conversiones seguras que de forma implícita no pueden hacerse, pero sí con este método. Además de otras conversiones inseguras que no se pueden hacer como hemos explicado anteriormente. Ejemplos de conversiones explícitas:

```
int i = 127;
byte b = (byte) i;
```
← Tanto i como b almacenan 127

```
int i = 128;
byte b = (byte) i;
```
← i almacena 128 y b -128

4.9. String

Manejar y almacenar caracteres de tipo char no es muy útil normalmente, ya que los programas suelen trabajar con cadenas de caracteres. En Java se utilizan objetos de clase String. Con esta clase de objetos podemos trabajar cómodamente con las cadenas y no necesitamos de la implementación de métodos propios.

Podemos crear un String de varias maneras, pero una de las más fáciles es utilizando una asignación de un texto entre comillas dobles. Se pueden utilizar los mismos caracteres que con el tipo primitivo char, el cuál veremos detalladamente más adelante. Ejemplo de cómo declarar un String:

```
String a = “prueba de string”;
String b = a;
```

4.10. Clases envoltorio

O también conocida como Wrapper Class, es un tipo de clase cuya finalidad es la creación de objetos que encierren un dato de tipo primitivo, así como la provisión de métodos para facilitar su manejo. Hay métodos que solo se pueden utilizar en una única clase y otros que están disponibles en cualquiera.

Primitivo	boolean	byte	char	double	float	int	long	short
Wrapper	Boolean	Byte	Character	Double	Float	Integer	Long	Short

Imagen 6. Correspondencia tipo primitivo - Wrapper Class.

toString()	Devuelve un String con valor literal igual al del dato primitivo.
x.compareTo(y)	Devuelve 0 en caso de que los datos contenidos en x e y sean iguales. Devuelve 1 en caso de que el valor de x sea mayor que el de y. Devuelve -1 en caso de que el valor de x sea menor que el de y.
x.equals(y)	Devuelve false en caso de que los datos contenidos en x e y sean diferentes, y true sin son iguales.

Imagen 7. Método de cualquier Wrapper Class.

4.10.1. Autoboxing – Unboxing

Las clases envoltorio y los tipos primitivos tienen correspondencia directa, por eso, desde el JDK 5 es posible la conversión automática entre ellos:

- **Autoboxing**: Conversión de tipo primitivo a envoltorio.
- **Unboxing**: Conversión de tipo envoltorio a primitivo.

EJEMPLO

```
Integer a = 11, b = 20;
// autoboxing

int c = b – a;
// unboxing
```

4.11. Secuencias de escape

Java permite utilizar secuencias de escape que pueden usarse tanto en variables del tipo primitivo char como en String. Las secuencias String hacen que se trate como si no tuviera significado alguno el carácter que esté contenido en la secuencia, o, por el contrario, permiten reemplazarlo por un contenido diferente.

\b	Retrocede un espacio	\r	Retorno de carro
\t	Tabulación	\"	Comilla doble
\n	Nueva línea	\'	Comilla simple
\f	Salto de página	\\	Barra invertida

Imagen 8. Secuencias especiales de escape.

4.12. Comentarios

Se trata de textos que pueden estar contenidos en el código fuente para preparar la creación de la documentación del código o explicar la función de un código, pero que no son compilados ni afectan al tamaño del archivo ejecutable.

Podemos distinguir tres tipos en Java:

- **De una línea:** son aquellos que se utilizan para añadir algún tipo de explicación al código. Empiezan con la secuencia `//`.
- **De varias líneas:** son aquellos utilizados también para añadir algún tipo de explicación al código. En este caso empiezan con la secuencia `/*` y acaban con `*/`.
- **De documentación:** son utilizados por el programa javadoc para generar de forma automática la documentación del código. Cuentan con varias líneas y empiezan con la secuencia `/**` y terminan con `*/`.

4.13. Operadores y expresiones

Los operadores, como podemos intuir por su nombre, sirven para realizar operaciones con las variables explicadas anteriormente. Podemos realizar operaciones sobre uno, dos o tres operandos. A continuación, las categorías en que se dividen los operadores.

Binarios					
Aritméticos		**Relaciones**		**Lógicos**	
+	Suma en número y concatenación en Strings	==	Igual	&&	y lógico (conjunción)
-	Resta	<	Menor	||	o lógico (disyunción)
*	Multiplicación	≤	Menor o igual	**Especiales**	
/	División real	>	Mayor	=	Asignación
%	Resto o módulo	≥	Mayor o igual	instanceof	Objeto es de tipo
		!=	Distinto		
Unarios					
+	Indica valor positivo	++	Incrementa en 1	!	Negación lógica
-	Niega la expresión	--	Decrementa en 1		
Bit					
A<<B	Desplaza A a la izquierda B posiciones	A&B	Operación AND a nivel de bits	A^B	Operación XOR a nivel de bits
A>>B	Desplaza A a la derecha B posiciones (tiene en cuenta el signo)	A|B	Operación OR a nivel de bits	~A	Complemento de A a nivel de bits
A>>>B	Desplaza A a la derecha B posiciones (no tiene en cuenta el signo)				

Imagen 9. Resumen de operadores.

4.13.1. Precedencia de operadores

Debido que a que los operadores cuentan con un orden de precedencia, aquellos que tengan un nivel más alto se evaluarán antes que los que tengan un nivel más bajo. En caso de que en una expresión aparezcan operadores con igual nivel de precedencia, estos serán leídos según el orden en que están escritos, es decir, de izquierda a derecha. A excepción del operador de asignación, el cual se evalúa de derecha a izquierda, el resto de los operadores binarios se evalúan de izquierda a derecha. A continuación, la tabla con el orden de precedencia de los distintos operadores, de mayor a menor nivel.

Precedencia
expr++ expr--
++expr --expr +expr -expr ~ !
* / %
+ -
<< >> >>>
< > <= >= instanceof
== !=
&
^
|
&&
||
= += -= *= /= %= &= ^= |= <<= >>= >>>=

Imagen 10. Precedencia de operadores.

www.universae.com