UNIDAD DIDÁCTICA 2

LENGUAJE DE PROGRAMACIÓN JAVA

MÓDULO PROFESIONAL: PROGRAMACIÓN

CESUR
Tu Centro Oficial de FP

Índice

RESUMEN INTRODUCTORIO

En esta unidad comenzaremos viendo una introducción al lenguaje de programación Java. Aprenderemos el significado de diferentes conceptos asociados al lenguaje Java, como son JRE, JDK y JVM, para poder preparar el entorno que utilizaremos en el desarrollo y ejecución de cada uno de los ejemplos que irán apareciendo en la unidad.
Para escribir el código utilizaremos un IDE llamado Eclipse, que es uno de los recomendados para empezar en programación, en la unidad se explica dónde se puede descargar y los pasos a seguir para instalarlo.

Una vez que tengamos todo preparado para poder codificar en Java, comenzaremos aprendiendo los elementos de un programa, constantes, variables y cómo requerir al usuario información por teclado y cómo presentarla en pantalla de una forma básica.
Trataremos otra parte fundamental en cualquier lenguaje de programación como es conocer y saber emplear los diferentes tipos de datos. En Java los más importantes son boolean, byte, short, int, long, double, float y char, en próximas unidades se tratarán otros más complejos.

Para terminar, pondremos en práctica la lógica de programación adquirida en la unidad 1 con la utilización de estructuras de control empleadas en la realización de algoritmos en pseudocódigo y diagramas de flujo, pero ahora implementando dichas estructuras de control en Java.

INTRODUCCIÓN

El cuerpo de un programa se compone de un conjunto de sentencias que especifican las acciones que se realizan durante su ejecución. Dentro de cualquier programa se escriben sentencias que definen la secuencia de acciones a ejecutar. Estas sentencias incluyen acciones de cálculo, entrada y salida de datos, almacenamiento de datos, etc. Las sentencias se ejecutan una a una en el orden en el que han sido escritas o se encierran entre determinadas estructuras de control para que se ejecuten en el orden que establezca el programador o desarrollador en base al cumplimiento de ciertas condiciones, como ya se ha visto en la unidad anterior.

Para aprender los conceptos básicos de programación se ha elegido el lenguaje Java, por su sintaxis sencilla y por su gran potencia como lenguaje orientado a objetos.

Para codificar en Java, necesitamos una parte fundamental llamada kit de desarrollo (JDK), imprescindible para la creación de aplicaciones en Java y la Java Virtual Machine (JVM) que se encarga de ejecutar el código.

Por otro lado, también necesitaremos un editor de código o un IDE (Integrated Development Environment-Entorno de desarrollo integrado), que es una herramienta software que nos permite escribir código y en el caso del IDE proporciona un conjunto de funcionalidades integradas que facilitan aún más dicha labor.

CASO INTRODUCTORIO

En la última entrevista que has realizado para un puesto de *software developer* te han escogido y estás muy contento, empiezas en unos días, en una empresa que se dedica al desarrollo de aplicaciones en Java. Sabes algo de HTML, CSS y JavaScript, pero nunca habías realizado nada con Java. Sin perder tiempo decides comenzar tu aprendizaje haciendo un curso de Java básico.

Al finalizar el estudio de la unidad sabrás adaptar el entorno de programación para satisfacer las necesidades que requiere la codificación en lenguaje Java, conocerás los tipos de datos, declaración de variables, constantes, expresiones y operadores que se pueden utilizar en el lenguaje de programación Java y las distintas estructuras de control en Java y para qué se utiliza cada una. Sabrás crear y desarrollar programas escritos en Java aplicando las estructuras de control para mejorar el código de sus programas. Serás capaz de traducir algoritmos realizados en pseudocódigo, diagramas de flujo, etc., a lenguaje Java y sabrás adaptar los conocimientos adquiridos en programación a distintos entornos de desarrollo.

1. INTRODUCCIÓN A JAVA

Parece que Java es un lenguaje que ya lleva tiempo usándose y que tiene una comunidad muy grande detrás, eso te vendrá bien para preguntar dudas la página "stackoverflow". Pero tienes muchas dudas sobre cómo empezar, que editor de código usar o cómo se compila el código.

El lenguaje de programación Java apareció en 1995 y muchas veces se tiende a confundirlo con JavaScript, pero no tienen nada en común prácticamente.

Java es un lenguaje de alto nivel y orientado a objetos cuya arquitectura se basa principalmente en C y C++. El código Java se ejecuta sobre la Java Virtual Machine (JVM), que se encarga de transformar el código fuente a código máquina adecuado a un determinado sistema operativo, esto hace que sea un lenguaje multiplataforma, es decir, el mismo código Java funcionará en cualquier sistema operativo solo necesitamos instalar la máquina virtual apropiada para el sistema en el que vamos a ejecutar código Java.

Aunque Java lleva existiendo hace tiempo hoy día se sigue utilizando y está muy presente en gran parte de dispositivos no solo en ordenadores, por ejemplo, en coches, lavadoras, móviles, equipamiento médico, televisores inteligentes, Internet, etc. Es el lenguaje con el que suele empezar a aprender programación en ciclos formativos, universidades, etc.

Es muy popular a nivel mundial en aplicaciones del lado del servidor (back-end) para construir la lógica empresarial, gestionar la lógica de la base de datos, procesar solicitudes y generar respuestas HTTP, implementar APIs, servicios web, etc. También se puede usar del lado del cliente (front-end) usando principalmente tecnologías como JavaServer Faces (JSF) y JavaFx que nos permiten crear interfaces de usuarios para web y para escritorio.

Hay gran demanda de profesionales con habilidades en programación Java ya que este lenguaje cuenta con una amplia gama de herramientas, bibliotecas de terceros, frameworks y una extensa comunidad que lo apoya y facilita el desarrollo de aplicaciones software.

ENLACE DE INTERÉS

Conoce la página stackoverflow en este enlace:

2. PREPARACIÓN DEL ENTORNO

*Estás investigando para aprender a preparar el entorno para escribir y ejecutar código en Java. Y has descubierto que necesitas una serie de herramientas indispensables como un **kit de desarrollo** y un IDE. Te han recomendado usar el IDE **Eclipse** por su facilidad de manejo y la ayuda que proporciona a la hora de programar.*

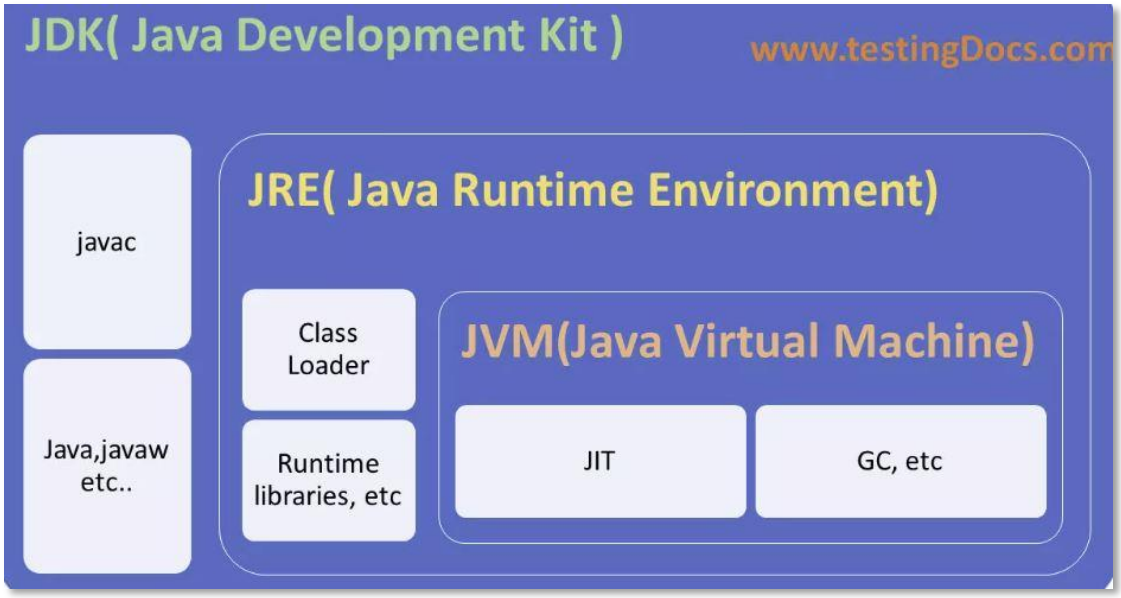
Para empezar a programar en Java se necesita una serie de elementos básicos:

- **JDK** (Java Development kit), que es un conjunto de herramientas y utilidades proporcionadas por Oracle para desarrollar aplicaciones Java.

- **JRE** (Java Runtime Environment) o entorno de ejecución de Java que es un conjunto de software que proporciona el entorno necesario para ejecutar aplicaciones Java en un sistema informático y que se incorpora en el JDK.

- **JVM** (Java Virtual Machine), que forma parte del JRE y se encarga de interpretar y ejecutar el bytecode generado en el proceso de compilación del código fuente por el compilador de Java (javac). Es importante destacar que existen diferentes implementaciones de la JVM proporcionadas por proveedores tales como Oracle, OpenJDK, etc. Aunque pueden variar en características específicas y rendimiento todas siguen los estándares definidos por la especificación de la plataforma de Java.

- **Editor de código** o **IDE** (Entorno de desarrollo integrado), que usaremos para escribir el código fuente, que en este caso será código Java.



Java SE Components.
Fuente: https://www.testingdocs.com/java-se-components/

Primero empezaremos instalando el JDK y, posteriormente, el IDE Eclipse para poder empezar a probar cada uno de los ejemplos prácticos que aparecen en esta unidad y en las siguientes.

2.1 Instalación JDK

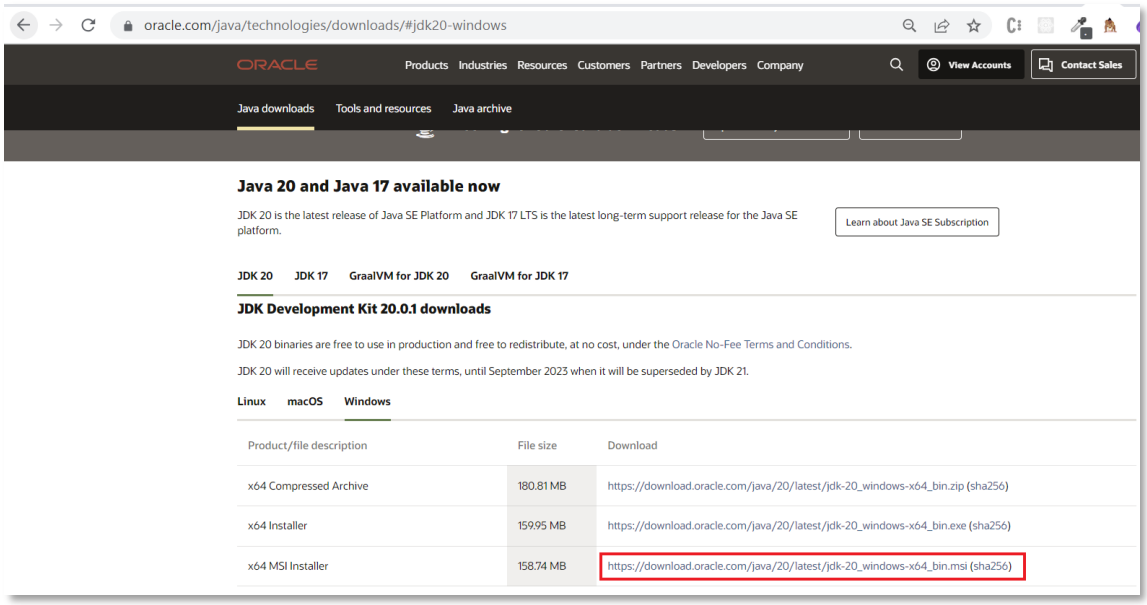
Antes de nada, necesitamos instalar el kit de desarrollo de Java. Para instalarlo accedemos a la página de Java.

Instalación -JDK.
Fuente: Elaboración propia.

ENLACE DE INTERÉS

Accede a la página de Java para poder instalar el kit de desarrollo:

Hacemos un clic en el botón con la marca roja 'Java SE Development kit'.



Instalación -JDK.
Fuente: Elaboración propia.

Escogemos la plataforma Linux, macOS o Windows según sea nuestro sistema operativo. Si utilizamos Windows elegiremos el enlace marcado en rojo en la imagen. A continuación, se descargará un archivo que debemos instalar haciendo doble clic en él. El proceso de instalación es muy sencillo solo hay que seguir las indicaciones.

Una vez tengamos el JDK instalado es recomendable configurar la variable de entorno JAVA_HOME. Esta variable es una variable del sistema muy útil en sistemas como Windows o Linux, donde se pueden tener múltiples versiones del JDK instaladas en un mismo sistema y se utiliza para especificar la ubicación del directorio donde se encuentra el JDK que se está utilizando. Esto permitirá a otras aplicaciones o herramientas del sistema ubicar de manera rápida y sencilla las utilidades de Java.

CITA

"Write once, run anywhere". Frase famosa relacionada con Java.

PARA SABER MÁS

El índice TIOBE se utiliza para clasificar la popularidad de los lenguajes de programación y se actualiza mensualmente. Se utiliza como referencia para evaluar la tendencia y popularidad de un lenguaje, pero no debe ser el único factor en la elección de un lenguaje ya que dependerá del proyecto y las características del mismo.

VÍDEO DE INTERÉS

Visualiza todo el proceso completo y resumido sobre cómo descargar JDK y establecer la variable de entorno JAVA_HOME:

2.2 Instalación software Eclipse

A la hora de comenzar a programar, se dispone de diversas herramientas que permiten realizar un desarrollo del software.

Un IDE o entorno de desarrollo integrado ("Integrated Development Enviroment", en inglés) es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

ENLACE DE INTERÉS

Para obtener más información acerca de los IDE's, consulta este enlace:

ARTÍCULO DE INTERÉS

Conoce las 7 mejores IDE para programar en Java:

ARTÍCULO DE INTERÉS

En este artículo se muestran las mejores opciones de IDE:

Actualmente existen multitud de entornos de desarrollo de software. Se va a detallar cómo realizar la instalación del entorno **Eclipse**.

Para comenzar la instalación del IDE Eclipse, se procede a su descarga a través de la página oficial.

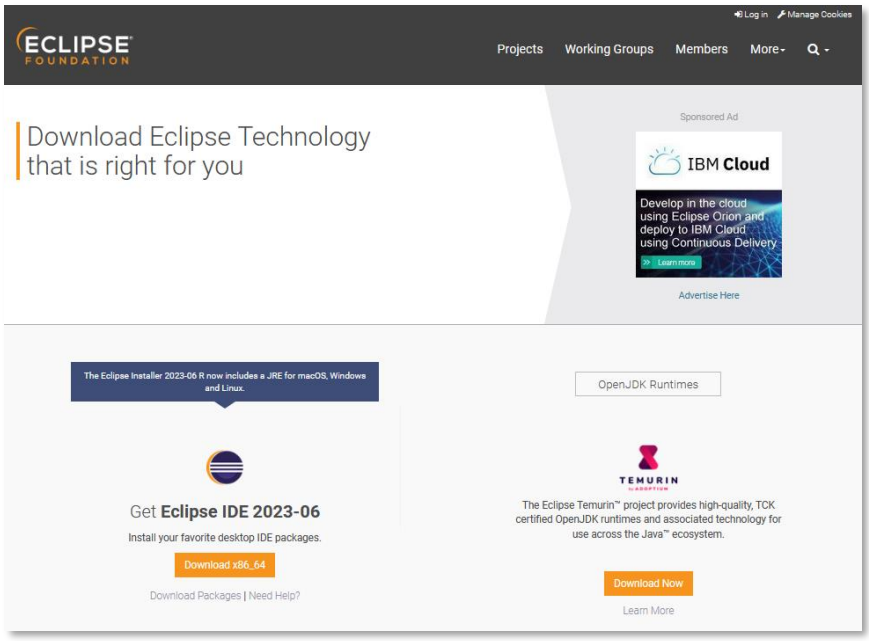
ENLACE DE INTERÉS

Accede a la web Oficial de Eclipse:

Descarga-Eclipse.

Fuente: Elaboración propia.

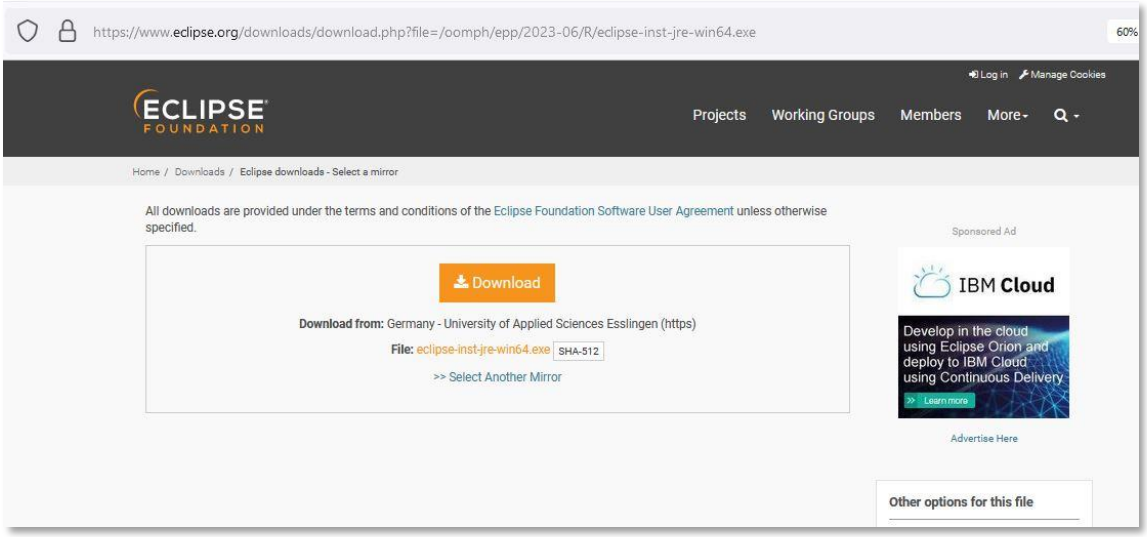
Se pulsa sobre el botón Download que aparece en la esquina superior derecha de la pantalla.



Descarga-Eclipse.

Fuente: Elaboración propia.

A continuación, se pulsa sobre el botón Download 64 bit que aparece en la esquina inferior izquierda para realizar la descarga del archivo de instalación para Windows de 64 bits.



Descarga-Eclipse.
Fuente: Elaboración propia.

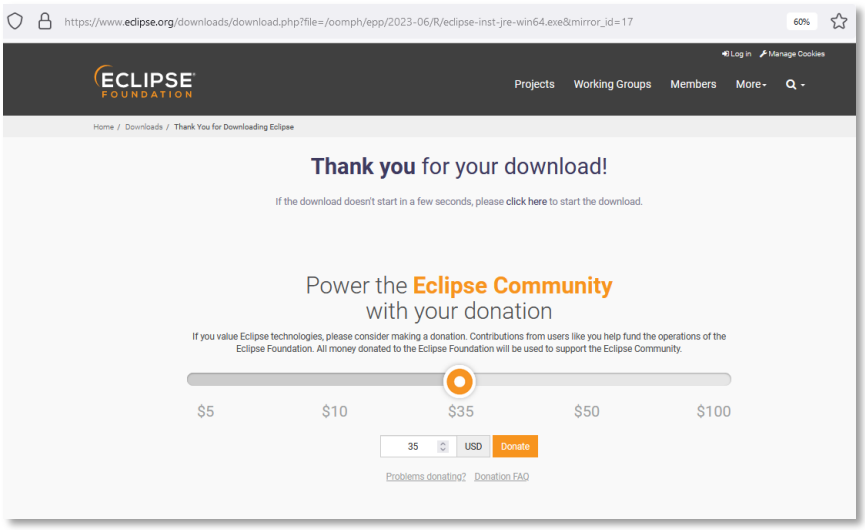
De nuevo se pulsa sobre el botón Download x86_64 para obtener el archivo eclipse-inst-jre-win64.exe de instalación del IDE Eclipse.

ENLACE DE INTERÉS

En este enlace tienes información sobre los pasos a realizar durante el proceso de instalación:

Durante la instalación se debe escoger el paquete que se necesite. Para desarrollo de aplicaciones en Java hay disponible Eclipse IDE for Java Developers (orientado a desarrollo de aplicaciones Java estándar, incluye herramientas básicas) o bien Eclipse IDE for Java EE Developers (orientado al desarrollo de aplicaciones empresariales basadas en Servlets, JSP, JPA, EJB y otras).

Para practicar los ejemplos de esta unidad y las siguientes descargamos **Eclipse IDE for Java Developers**.



Descarga-Eclipse.

Fuente: Elaboración propia.

ENLACE DE INTERÉS

En caso de que quieras descargar directamente los paquetes y archivos .zip accede a este otro enlace:

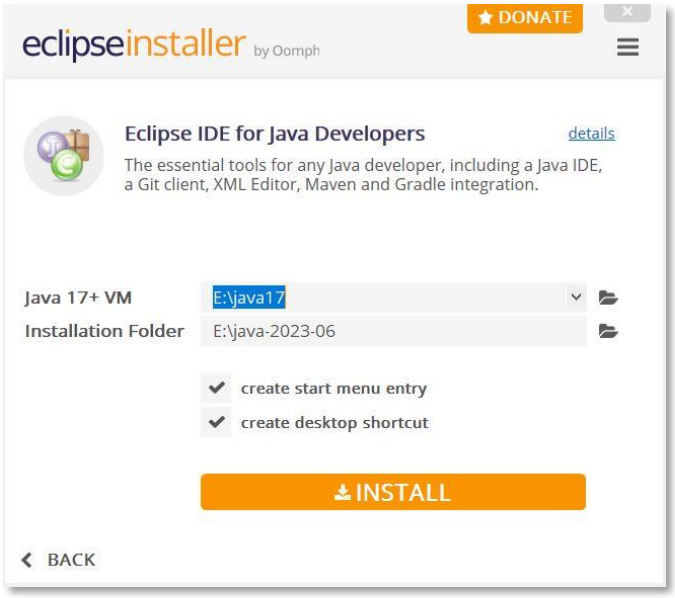
Una vez descargado, ejecutamos el archivo para realizar la instalación.

Diferentes distribuciones-Eclipse.

Fuente: Elaboración propia.

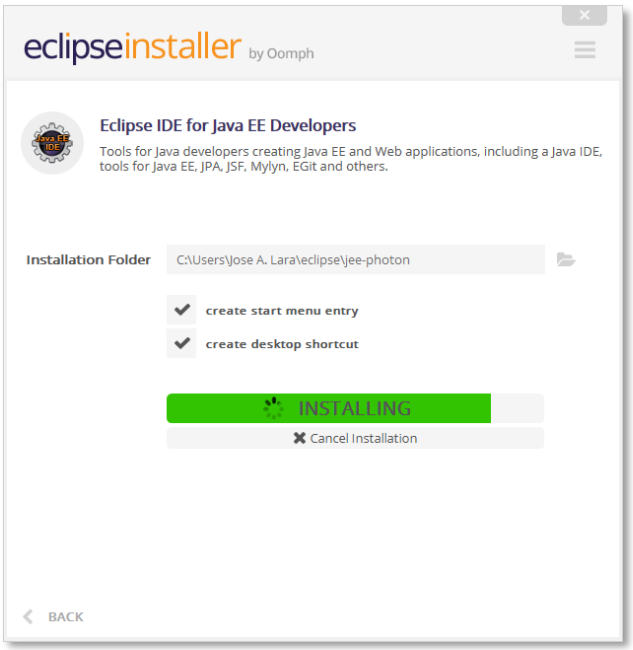
Se elige la versión que se desea instalar. El instalador pregunta por la carpeta en la que se instalará Eclipse, y, finalmente, se pulsa en INSTALL.

En el caso de tener alguna versión de Java ya instalada en el equipo, ésta aparecerá seleccionada por defecto.



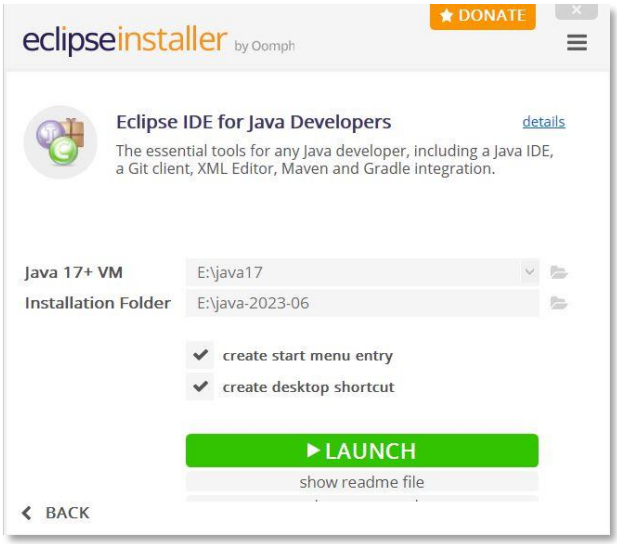
Instalación-Eclipse.
Fuente: Elaboración propia.

Es posible que aparezca una ventana para confirmar la aceptación de algún paquete software.



Instalación-Eclipse.
Fuente: Elaboración propia.

Continuará la instalación y finalmente mostrará la opción de ejecutar Eclipse. Para ello se pulsará en el botón Launch.

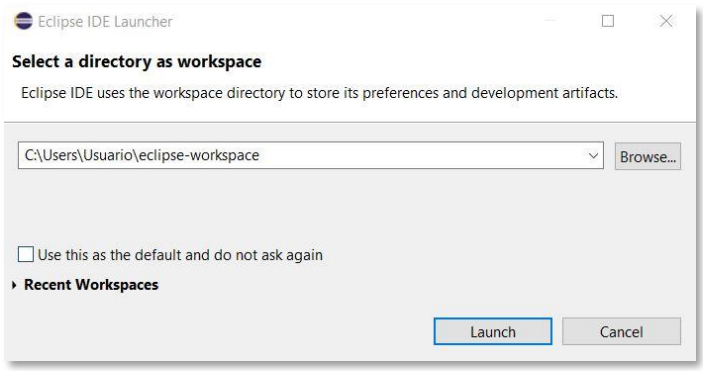


Instalación-Eclipse.
Fuente: Elaboración propia.

A continuación, se mostrará la llamada splash screen que especifica la versión de Eclipse que se va a ejecutar.

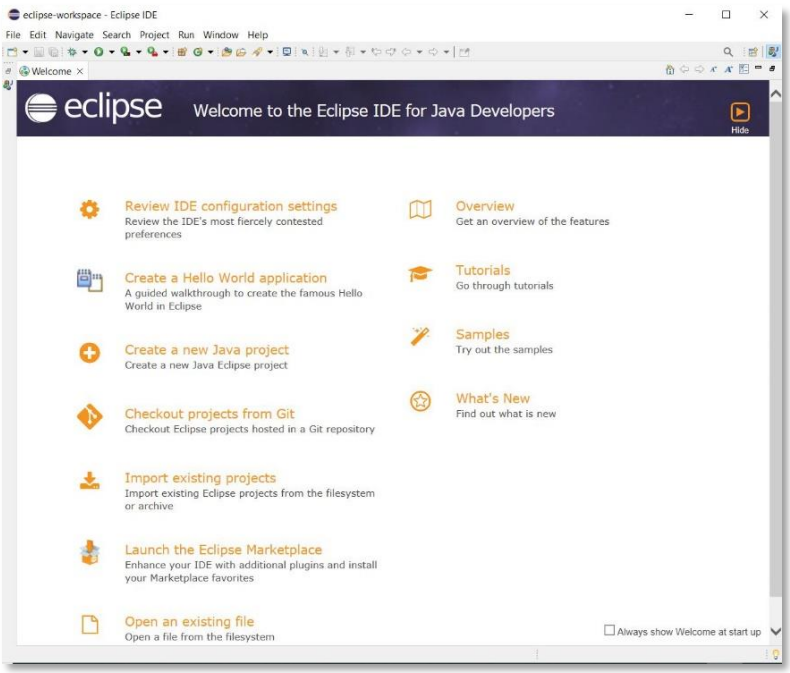
Splah screen-Eclipse.
Fuente: Elaboración propia.

Solicitará que se le indique la carpeta en la cual se van a colocar los proyectos de Eclipse.



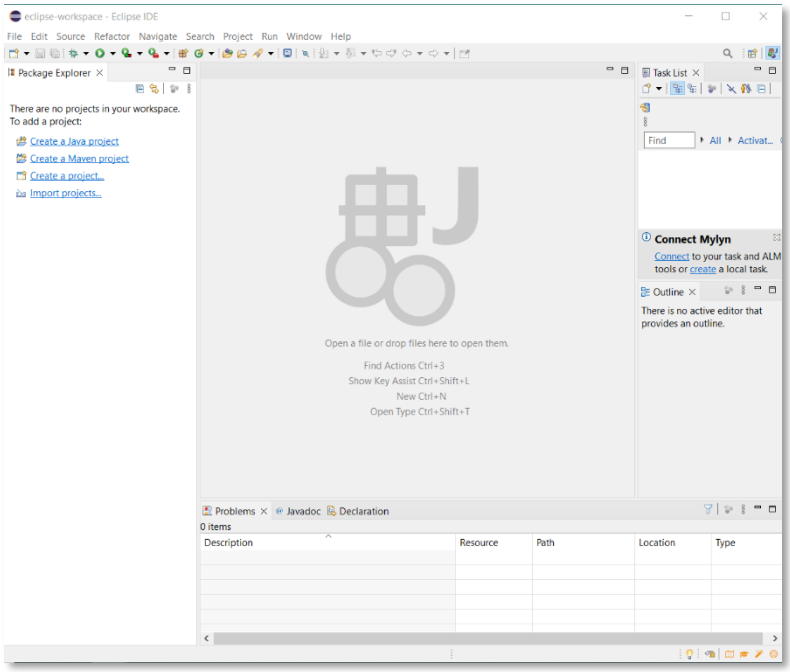
Primeros pasos-Eclipse.
Fuente: Elaboración propia.

La primera vez que se ejecute Eclipse mostrará una ventana de bienvenida, en la cual se ofrecen varias opciones, como revisar la configuración del IDE, crear nuevos proyectos, ver ejemplos y tutoriales, etc.



Primeros pasos-Eclipse.
Fuente: Elaboración propia.

Se cierra la ventana de bienvenida o pulsamos sobre el icono naranja con título **Hide** situado en la esquina superior derecha y ya se puede empezar a utilizar el entorno de desarrollo.



Espacio de trabajo-Eclipse.
Fuente: Elaboración propia.

VÍDEO DE INTERÉS

En este vídeo podrás ver un repaso rápido sobre sintaxis básica en Java para crear una clase:

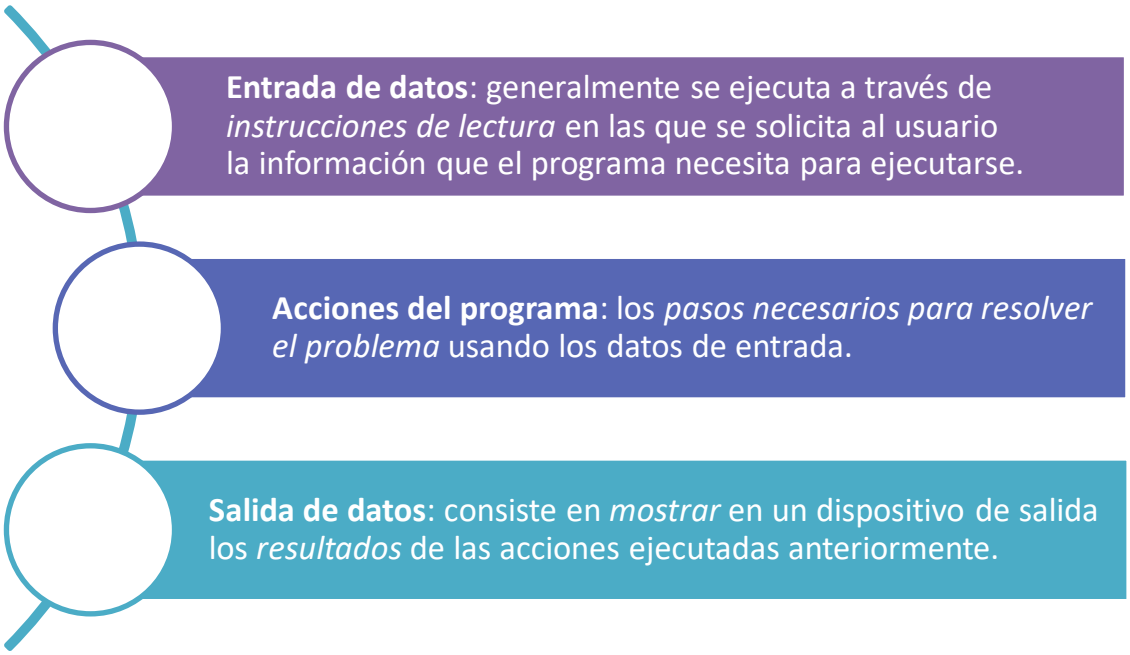
3. CONCEPTO DE PROGRAMA

Te sientes más preparado para empezar a programar, pues has realizado muchos algoritmos que te han despertado la llamada lógica de programación, no obstante, un programa realizado en Java es diferente y un poco más complicado, pues tienes que conocer diferentes elementos de un programa, cómo se tratan, nueva sintaxis, la forma en la que se recogen los datos de entrada por teclado y la manera de visualizar los datos por pantalla.

En la unidad anterior se ha visto el concepto de algoritmo que se compone de una secuencia de pasos lógicos y bien definidos que describen cómo resolver un problema o realizan una tarea específica, pero no está asociado como tal a un lenguaje de programación concreto.

Sin embargo, un programa, aunque también se compone de instrucciones, se trata de una implementación concreta y específica de un algoritmo en un lenguaje de programación y entorno determinado.

Todo programa tiene que estar formado por las siguientes partes:



Para empezar a programar en Java necesitamos un editor de código que es una herramienta software que se utiliza para escribir y editar código fuente (en nuestro caso código en Java). Los editores suelen incluir características y funcionalidades útiles para proporcionar ayuda en el momento de la codificación, por ejemplo, resaltado de sintaxis, autocompletado, numeración de líneas, etc. En una escala superior tenemos los IDE (Entornos de desarrollo integrado) que son editores con funcionalidades ampliadas y que integran otras herramientas y servicios como compiladores depuradores, control de versiones, etc.

Un ejemplo de editor de código muy básico puede ser el Bloc de notas, allí podemos escribir código Java, guardamos con extensión .java y podríamos ejecutarlo desde la línea de comandos. Pero en esta unidad aprenderemos a usar el IDE Eclipse que nos proporciona entre otras herramientas de compilación que hacen este proceso más rápido y sencillo.

3.1 Elementos de un programa

Los elementos básicos que forman un programa o un algoritmo son:

1. **Palabras reservadas**: conjunto de palabras especiales que tienen un significado propio dentro del lenguaje y, por lo tanto, sólo pueden ser utilizadas para ello.

2. **Identificadores**: son todos aquellos nombres que aparecen en el programa dados por el programador. Para crear estos identificadores hay unas normas:

a. Deben estar formados por letras, dígitos y caracteres de subrayado (_).
b. Tienen que comenzar por una letra.
c. No pueden contener espacios en blanco.
d. El nombre asignado debe tener relación con la información que contiene.

3. **Caracteres especiales**: sirven como separadores de instrucciones.

4. **Constantes**: elementos cuyo valor permanece fijo durante toda la ejecución del programa.

5. **Variables**: elementos que sirven para almacenar datos cuyo valor puede sufrir modificaciones durante la ejecución del programa.

6. **Instrucciones**: los pasos que forman parte de un programa y sirven para indicarle la opción que debe recibir.

7. **Bucles**: cualquier fragmento de código cuyas instrucciones se repiten un número finito de veces. Cada vez que se repite un bucle es una iteración.

8. **Contadores**: una variable cuyo valor se incrementa o decrementa en valores constantes en cada iteración de un bucle.

9. **Acumuladores**: variables destinadas a almacenar cantidades variables provenientes de resultados de operaciones matemáticas. Tanto los contadores como los acumuladores es necesario inicializarlos para que la primera vez que lo incremente o decremente tengan un valor con el que operar.

10. **Interruptores**: son variables que sirven como indicador de una determinada información y sólo pueden tomar uno de los dos valores posibles.

3.2 Constantes y variables

En Java, las constantes y las variables son elementos fundamentales utilizados para almacenar y manipular datos dentro de un programa. Sin embargo, tienen propósitos y características distintas:

Constantes

Una constante es un valor que no cambia durante la ejecución del programa. Una vez que se asigna un valor a una constante, este no puede ser modificado. En Java, las

constantes se declaran utilizando la palabra clave final. Algunas características clave de las constantes son:

- Su valor es fijo y no cambia durante la ejecución del programa.
- Deben ser inicializadas en el momento de su declaración o en un bloque de inicialización estático.
- Se nombran utilizando letras mayúsculas y palabras separadas por guiones bajos (convenciones de nomenclatura en Java).

```
final double PI = 3.14159;
final int NUMERO_MAXIMO = 100;
```

Variables

Una variable al igual que una constante, es un espacio de memoria reservado para almacenar datos, pero que pueden cambiar durante la ejecución del programa a diferencia de una constante que no puede cambiar una vez se inicializa. En Java, las variables se declaran con un tipo de dato y un nombre. Pueden ser inicializadas en el momento de su declaración o más tarde durante la ejecución del programa. Algunas características clave de las variables son:

- Su valor puede cambiar a lo largo de la ejecución del programa.
- Deben ser inicializadas antes de usarlas para evitar errores.
- Se nombran utilizando letras minúsculas al comienzo y si es una palabra compuesta la primera letra de cada palabra con mayúscula según notación lowerCamelCase.

```
int edad = 30;
double precio = 19.99;
String nombre = "Juan";
```

3.3 Entrada y salida de información

Los cálculos que realizan los ordenadores requieren, para ser útiles, la entrada de los datos necesarios para ejecutar las operaciones que posteriormente se convertirán en resultados o salidas.

Las operaciones de entrada permiten **leer determinados valores** introducidos por el usuario y **asignarlos** a determinadas variables. Esta entrada se conoce como **operación de lectura o entrada**.

Con los datos de entrada, el ordenador realiza las operaciones oportunas en cada momento y el resultado obtenido aparece en un dispositivo de salida mediante una **operación de salida o escritura**.
Java es un lenguaje de programación **orientado a objetos**, lo que significa que se basa en el concepto de clases y objetos. Para facilitar la entrada y salida de datos Java proporciona varias clases y métodos que podemos utilizar.

Para las operaciones de entrada o lectura en Java, se utilizará por ahora el método **next()** de la clase **java.util.Scanner**.

java.util es el paquete donde se encuentra la clase Scanner dentro de la biblioteca estándar de Java que viene incluida en la instalación predeterminada del JDK (Java Development Kit).

El objeto sc es un objeto instanciado de la clase Scanner que usaremos para acceder a métodos que se encargan de la entrada estándar de diferentes tipos de datos, en este ejemplo se recoge un dato introducido por el usuario que es de tipo String (método next()).

System.in es un objeto que representa la entrada estándar, generalmente el teclado (también se puede indicar otro tipo de fuente de entrada).

```
Scanner sc = new Scanner(System.in);
String valor = sc.next();
System.out.println("El valor introducido es " + valor);
```

Para las operaciones de salida, se utilizan los métodos **System.out.print**, o **System.out.println** (la diferencia entre ambos es que uno no añade un salto de línea y el otro sí, respectivamente).

```
int result, valor;
valor = 1540;
result = valor * 25 /100;
System.out.println("El resultado de la operación es "
+ result);
```

ENLACE DE INTERÉS

A estas alturas te estarás preguntando para qué sirve la instrucción sc.nextLine(); la cual aparece justo después de la introducción de un número decimal con doble precisión y antes de pedir la introducción de una cadena de texto. En este enlace encontrarás la respuesta:

PARA SABER MÁS

A través del siguiente enlace podrás saber más sobre qué es y cómo usar la clase Scanner:

EJEMPLO PRÁCTICO

Debes aprender las diferencias entre usar un tipo de dato u otro en Java, por lo que comienzas practicando con la clase Scanner, que es una parte de la API de Java que se encuentra en el paquete java.util.

La clase Scanner es utilizada para pedir datos por consola o teclado. Es una clase que ya existe y que proporciona métodos para leer diferentes fuentes de entrada, como el teclado o archivos y se encuentra en el paquete 'java.util' por tanto se ha tenido que importar dicho paquete para poder usarlo (1º línea; uso de import).

Su función principal es permitir la lectura de datos de entrada de manera sencilla. El programa propuesto tiene que solicitar al usuario un número entero, uno booleano, dos con decimales (para diferentes precisiones) y por último una cadena de texto, hay que utilizar diferentes tipos de datos y llamadas a métodos de la clase Scanner según dicho tipo de datos.

Para finalizar, se debe cerrar el objeto llamado sc (se puede llamar de cualquier otra forma) que se ha creado de la clase Scanner.

En el ejemplo siguiente se muestra cómo quedaría al usar el IDE Eclipse.

EJEMPLO PRÁCTICO

Se muestra un ejemplo completo tal y como debería quedar cuando se usa el IDE Eclipse:

```
import java.util.Scanner;

public class EjemploPracticoEntradaDatos {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduce un número entero: ");
        int num = sc.nextInt();

        System.out.print("Introduce un valor booleano: ");
        boolean b = sc.nextBoolean();

        System.out.print("Introduce un valor con decimales(tipo dato
con precisión simple): ");
        float f = sc.nextFloat();

        System.out.print("Introduce un valor con decimales(tipo dato
con precisión doble): ");
        double d = sc.nextDouble();

        sc.nextLine();

        System.out.print("Introduce una cadena de texto: ");
        String str = sc.nextLine();

        System.out.println("El número introducido es: " + num);
        System.out.println("El valor booleano introducido es: " + b);
        System.out.println("El valor con decimales de precisión simple
 es: " + f);
        System.out.println("El valor con decimales de precisión doble
 es: " + d);
        System.out.println("La cadena de texto introducida es: "
+ str);
        sc.close();
    }
}
```

La salida por pantalla es la siguiente:
Introduce un número entero: 5
Introduce un valor booleano: true
Introduce un valor con decimales (tipo dato con precisión simple): 5,66666987
Introduce un valor con decimales (tipo dato con precisión doble): 5,66666987
Introduce una cadena de texto: Hola

El número introducido es: 5
El valor booleano introducido es: true
El valor con decimales de precisión simple es: 5.66667
El valor con decimales de precisión doble es: 5.66666987
La cadena de texto introducida es: Hola

3.4 Instrucciones y tipos

El proceso de diseño de un algoritmo y la posterior codificación del programa consiste en definir las acciones o instrucciones que resolverán el problema. Estas instrucciones se deben escribir y posteriormente almacenar en memoria en el mismo orden en el que se van a ejecutar, teniendo en cuenta que un programa puede ser **lineal** o **no lineal**, dependiendo del orden en que se ejecutan sus instrucciones.

Un programa es **lineal** si el orden de ejecución de sus instrucciones es **el mismo orden** que el de lectura, es decir: las acciones se ejecutan secuencialmente. Un programa **no lineal** es el programa en el que se **interrumpe la secuencia de ejecución** mediante instrucciones de bifurcación, es decir: hay saltos que mandan de unas instrucciones a otras. Toda bifurcación en un programa debe realizarse mediante estructuras de control. Las **instrucciones** disponibles en un lenguaje de programación dependen del tipo de lenguaje, pero la clasificación más usual, independientemente del lenguaje es:

Instrucciones de inicio y de fin: indican el *comienzo* y el *final* de un programa.

Instrucciones de asignación: permiten *almacenar un valor* en una variable previamente definida.

Instrucciones de lectura: encargadas de *recoger un dato* de un dispositivo de entrada.

Instrucciones de escritura o de salida: permiten *mostrar información* en dispositivos de salida.

Instrucciones de bifurcación o salto: permiten *interrumpir la ejecución lineal* de un programa.

Estas bifurcaciones pueden ser hacia adelante o hacia atrás, pero siempre deben ser bifurcaciones condicionales.

4. DATOS Y TIPOS DE DATOS EN JAVA

Con la aplicación PSeInt has usado tipos de datos y tienes claro cómo son los tipos de datos de forma general, unos son de tipo texto, número, reales (con decimales), lógicos, etc., pero en Java es diferente, aunque se parecen no son iguales y además existen otros tipos de datos llamados complejos. Es hora de aprender los tipos de datos primitivos en Java y cómo se representan.

El lenguaje de programación Java utiliza tipos de datos primitivos y complejos. Java nos proporciona 8 tipos de datos primitivos, que se pueden poner en cualquier lugar del código fuente de Java. Los tipos de datos primitivos en Java no son objetos, son valores que se almacenan directamente en memoria.

Tipo de variable	*Bytes que ocupa*	*Rango de valores*	Ejemplo
boolean	1	true, false	boolean b= false; boolean c= true;
byte	1	-128 a 127	byte b= 68;
short	2	-32.768 a 32.767	short s= 68;
int	4	-2.147.483.648 a 2.147.483.649	int i= 68;
long	8	-9 · 10^18 a 9 · 10^18	long l= 68L;
double	8	-1,79 · 10^308 a 1,79 · 10^308	double d= 68.58;
float	4	-3,4 · 10^38 a 3,4 · 10^38	float f= 68f;
char	2	Caracteres (en Unicode)	char c= 68; char c= 'B';

Veamos en detalle para qué se utilizan cada uno de ellos:

- **boolean**: El tipo boolean se utiliza para representar valores de verdad (verdadero o falso). Esencialmente, se utiliza para tomar decisiones en el código basadas en condiciones lógicas.

- **byte**: El tipo byte se utiliza para representar valores enteros de 8 bits (1 byte) con signo. Suele usarse cuando se necesita almacenar datos pequeños que no requieren un rango extenso, como códigos ASCII o valores numéricos en un rango limitado.

- **short**: El tipo short se emplea para almacenar valores enteros de 16 bits (2 bytes) con signo. Aunque el rango es más amplio que el tipo byte, todavía es

relativamente limitado. Puede ser útil en situaciones donde el rango completo del tipo int no es necesario.

- **int**: El tipo int es uno de los tipos de datos más comunes en Java. Se utiliza para representar valores enteros de 32 bits (4 bytes) con signo. Es adecuado para la mayoría de las operaciones numéricas y es ampliamente utilizado en la mayoría de las aplicaciones.

- **long**: El tipo long se utiliza para representar valores enteros de 64 bits (8 bytes) con signo. Debido a su mayor rango, es útil para manejar valores más grandes, como marcas de tiempo en milisegundos, valores monetarios más precisos o identificadores únicos extensos.

- **double**: El tipo double es similar al float, pero utiliza 64 bits (8 bytes) para representar valores de punto flotante. Es más preciso que float y se utiliza comúnmente en cálculos que requieren mayor precisión, como cálculos financieros y científicos.

- **float**: El tipo float se utiliza para representar valores de punto flotante de 32 bits (4 bytes). A menudo se utiliza en cálculos científicos y de ingeniería, donde la precisión no necesita ser extremadamente alta.

- **char**: El tipo char se utiliza para representar un único carácter Unicode de 16 bits (2 bytes). Puede contener letras, números, símbolos y caracteres especiales. Es ampliamente utilizado para representar caracteres en texto.

¿SABÍAS QUE...?

En Java, los nombres de datos primitivos se escriben en minúscula.
Si la primera letra es mayúscula, te estarás refiriendo (probablemente) a una clase envoltoria o wrapper class que sirve para encapsular a datos primitivos y proporcionar funcionalidades adicionales, como métodos y constructores que no están disponibles en los métodos primitivos directamente.

Tipos de datos primitivos = int, double, float, char, long, etc.
Tipos envoltorio o wrapper = Integer, Double, Float, Character, Short, etc.

ENLACE DE INTERÉS

Amplia información sobre los elementos básicos de programación en JAVA visite:

5. ESTRUCTURAS DE CONTROL

Te has puesto en contacto con otros compañeros y compañeras que, como tú, quieren empezar con el desarrollo de software y en el grupo de WhatsApp que habéis creado te dan algunos consejos. Todos coinciden en la importancia de dominar las estructuras de control, ya que proporcionan una base sólida para diseñar algoritmos más complejos y escribir código más eficiente y mantenible. ¡Así que, comienza este emocionante viaje hacia el control del flujo de datos en Java!

En el mundo de la programación, uno de los conceptos fundamentales y poderosos es el control del flujo de los datos. Las estructuras de control en Java son herramientas esenciales que permiten a los programadores tomar decisiones, repetir tareas y definir la lógica de sus programas.

5.1 Estructuras selectivas o alternativas

Como hemos visto en la unidad anterior, estas estructuras son aquellas que permiten la **ejecución de unas u otras acciones** dependiendo de que se cumpla o no una condición o dependiendo del valor que tome una determinada variable.

ESTRUCTURA IF SIMPLE

La estructura if se denomina estructura de selección única porque ejecuta un bloque de sentencias solamente cuando se cumple la condición del if.

- Si la condición es verdadera se ejecuta el bloque de sentencias.
- Si la condición es falsa, el flujo del programa continúa en la sentencia inmediatamente posterior al if.

La sentencia **if en Java** tiene la siguiente sintaxis:

```
if (condición) {
    bloque-de-sentencias
}
```

La condición es una expresión que evalúa un valor lógico, por lo que el resultado solo puede ser true (verdadero) o false (falso). La condición siempre se escribe entre paréntesis. La selección se produce sobre el bloque de sentencias delimitado por llaves. Toda llave que abre debe tener su correspondiente llave de cierre y dependiendo del lugar de cierre el flujo del programa puede ser diferente dando lugar a resultados no esperados y errores de compilación o ejecución.

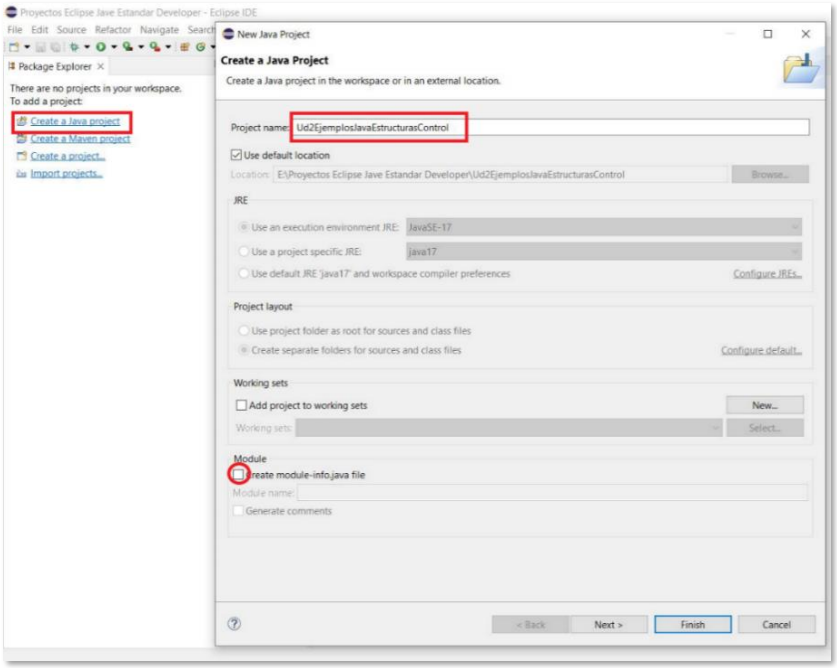
Si el bloque de sentencias solo tiene una sentencia, entonces se puede escribir sin las llaves, tal y como se muestra a continuación:

```
if (condición) sentencia;
```

Para comenzar por primera vez a escribir código en Java, abrimos Eclipse y creamos un nuevo proyecto Java. En nuestro caso al ser un proyecto de uso interno le daremos un nombre que será descriptivo de la funcionalidad o utilidades que contiene.

Si no se cambia el directorio por defecto, el proyecto y su contenido se almacenará en el lugar especificado durante la instalación (opción use default location).

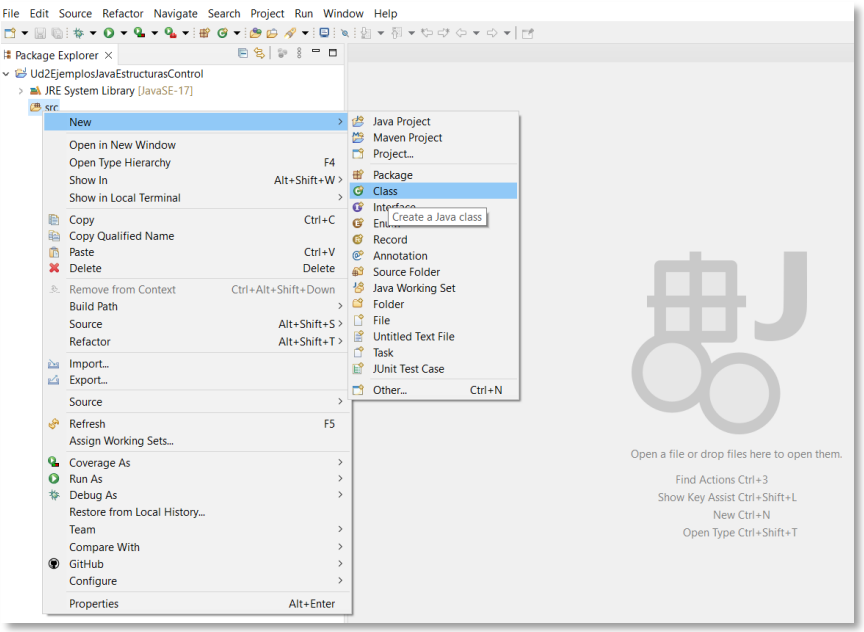
La opción create **module-info.java** quedará **desmarcada** porque con la programación que se verá en esta y próximas unidades no hará falta y puede generar algunos conflictos que dan lugar a errores.



Creación de un proyecto en Java-Eclipse.
Fuente: Elaboración propia.

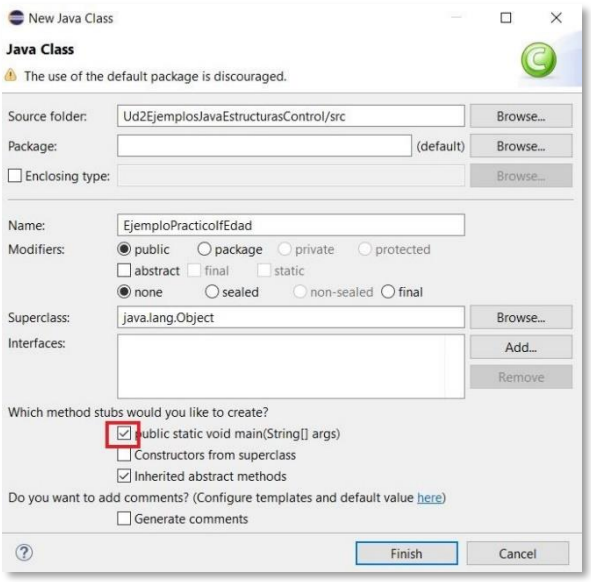
En Java todo se organiza en clases debido a su enfoque en la programación orientada a objetos (POO).

Para crear una clase en Java haremos un clic derecho encima de la carpeta src y en el menú que aparece escogeremos la opción New.



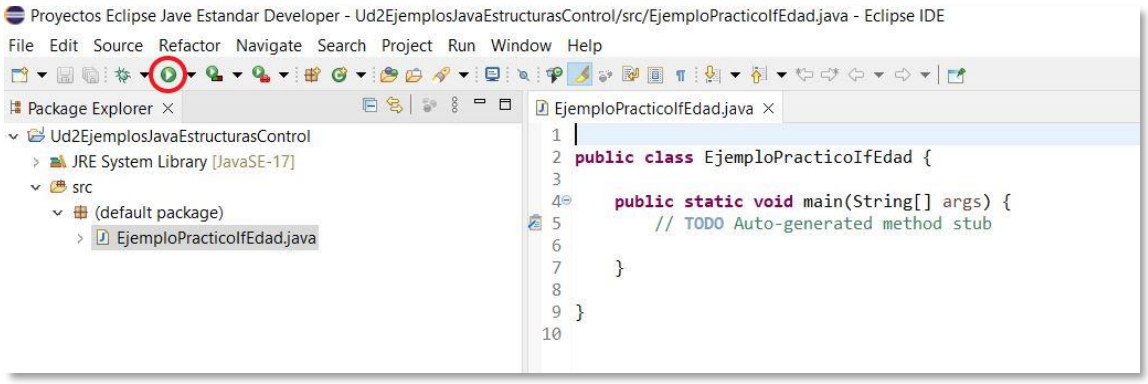
Creación de una clase en Java-Eclipse.
Fuente: Elaboración propia.

De momento solo hace falta rellenar el campo Name, para dar un nombre a la clase (usamos notación UpperCamelCase, ver apartado "*Para Saber más*") y marcamos la opción public **static void main(string[] args)** para que la clase sea una clase main (ejecutable), para finalizar pulsamos Finish.



Creación de una clase con método main en Java-Eclipse.
Fuente: Elaboración propia.

Al ser una clase main será posible ejecutarla de forma independiente. En próximas unidades se verá cómo utilizar dentro de un mismo proyecto clases main y no main.



Ejecución de una clase con método main en Java-Eclipse.

Fuente: Elaboración propia.

Entre las llaves de apertura y cierre de la clase main se escribe código Java, por ejemplo, el código correspondiente a cada uno de los ejercicios prácticos que se irán exponiendo a lo largo de la unidad y que se pueden ejecutar haciendo clic en el botón que aparece rodeado con un círculo rojo en la imagen anterior.

EJEMPLO PRÁCTICO

En este ejemplo, se comprueba si la edad es mayor o igual a 18 años. Si se cumple la condición (true), imprime por pantalla Es mayor de edad y, además, establece el valor de la variable booleana mayor Edad a true.

```
int edad;
boolean mayorEdad;

edad=30;

if (edad>=18){
System.out.println("Es mayor de edad");
mayorEdad=true;
}
```

EJEMPLO PRÁCTICO

En la siguiente estructura condicional se valida si el número almacenado en la variable llamada número, es positivo. Si se cumple esta condición (true) aparecerá por consola un mensaje que indica que el número es positivo, un mensaje que muestra el resultado de multiplicar dicho número por 2 y un último mensaje que nos dará información sobre el resultado de efectuar el cuadrado de dicho número.

Aquí el signo '+' es usado para concatenar el texto entrecomillado con el resultado producido de las operaciones aritméticas, es decir, mostrar en la misma línea texto y resultado.

```
int numero = 10;

if (numero > 0) {
    System.out.println("El número es positivo");
    System.out.println("El número multiplicado por 2 es: " +
(numero * 2));
    System.out.println("El número al cuadrado es: " +
(numero * numero));
}
```

EJEMPLO PRÁCTICO

El uso de variables de tipo booleano es muy común a la hora de programar. Este tipo de variable se suele denominar interruptor o bandera (flag en inglés) cuando nos ayuda a controlar el flujo de ejecución de un programa para activar/desactivar ciertas funcionalidades o partes de un código. Esta variable se cambiará en determinados puntos del programa para modificar el comportamiento según convenga. Queremos realizar un pequeño programa que nos diga si una variable interruptor ha sido activada.

```
boolean interruptor = false;

if (interruptor) {
    // Código a ejecutar si interruptor es verdadero
    System.out.println("El interruptor ha sido activado");
}
```

RECUERDA

En un proyecto se pueden crear todas las clases que se necesiten para estructurar y organizar el código, dependerá de la complejidad del proyecto.

Es recomendable que si el proyecto es grande y complejo se creen paquetes para almacenar clases que estén relacionadas entre sí. Este tipo de organización favorecerá la legibilidad, modularidad y reutilización de código.

PARA SABER MÁS

Camel Case es una convención muy popular para nombrar identificadores, etc. Podemos usarlo en modo Upper Camel Case (también conocido como PascalCase) o Lower Camel Case, la diferencia es si comenzamos el nombre de la variable con mayúscula o minúscula.
Para los nombres de clase en Java, se usa la notación UpperCamelCase (primera letra mayúscula y si es palabra compuesta la primera letra de cada una de ellas en mayúscula). En este enlace puedes ampliar esta información:

ENLACE DE INTERÉS

En este enlace puedes encontrar una recopilación sobre programación básica en Java:

ESTRUCTURA IF ELSE

La estructura if-else se denomina de selección doble porque selecciona entre dos bloques de sentencias mutuamente excluyentes.

- Si se cumple la condición, se ejecuta el bloque de sentencias asociado al if.
- Si la condición no se cumple, entonces se ejecuta el bloque de sentencias asociado al else.

La sentencia if-else tiene la siguiente sintaxis:

```
if (condición) {
        bloque-de-sentencias-if
} else {
        bloque-de-sentencias-else
}
```

Para añadir comentarios a partes del código y que sirvan de explicación al desarrollador pero no influyan en la ejecución del programa, es decir, que sean ignorados por el compilador se puede hacer de dos formas en Java:

- **Comentarios de una línea**. Comienzan con dos barras diagonales (**//***Texto que será ignorado por el compilador*). Toda la línea hasta el final será ignorada por el compilador.

- **Comentarios de bloque.** (**/***Texto que será ignorado por el compilador***/**). Pueden abarcar varias líneas, se usan para comentarios largos que ocupan varias líneas o para desactivar bloques de código temporalmente.

EJEMPLO PRÁCTICO

Al ejemplo visto anteriormente de uso de if, se le ha añadido un else, para que imprima 'Es menor de edad' y establezca el valor de la variable booleana mayorEdad a false cuando la edad sea menor de 18 años.

Ahora la edad se pide al usuario desde teclado.

```
import java.util.Scanner;
public class EjemploPracticoIfElseEdad {

    public static void main(String[] args) {
        //declaración de variables
        int edad;
        boolean mayorEdad;
        Scanner sc = new Scanner(System.in);

        System.out.print("Introduce la edad: ");

    edad = sc.nextInt();

        if (edad>=18){
            System.out.println("Es mayor de edad");
            mayorEdad=true;
        } else {
            System.out.println("Es menor de edad");
            mayorEdad=false;
        }
        sc.close();
    }

}
```

EJEMPLO PRÁCTICO

Al ejecutar el siguiente código se pedirá al usuario que active o desactive un interruptor. Si el usuario escribe true se activará y en caso contrario si escribe false se desactivará.
Como se ha visto anteriormente este tipo de variable booleana usada dentro de una condición, se suele denominar interruptor o bandera (flag en inglés) porque nos ayuda a controlar el flujo de ejecución de un programa y para activar/desactivar ciertas funcionalidades o partes de un código.
Dicha variable se puede cambiar en determinados puntos del programa para modificar el comportamiento según convenga.

```
import java.util.Scanner;
public class EjemploPracticoIfElseInterruptor {
    public static void main(String[] args) {
        //declaración de variables
        boolean interruptor;
        Scanner sc = new Scanner(System.in);
        System.out.print("Para activar interruptor escribe true,
para desactivar false:");
        interruptor = sc.nextBoolean();
        if (interruptor) {
            /* Código a ejecutar si interruptor es
Verdadero*/
            System.out.println("El interruptor ha sido
activado");
        } else {
            // Código a ejecutar si interruptor es falso
            System.out.println("El interruptor ha sido
desactivado");
        }
        sc.close();
    }
}
```

PARA SABER MÁS

La API de Java (Application Programming Interface) es un conjunto de clases, interfaces, métodos y constantes predefinidos que proporciona el entorno de ejecución de Java para el desarrollo de aplicaciones.
En el paquete java.util se encuentran clases y utilidades de propósito general, es muy utilizado y forma parte de la biblioteca estándar de Java.

VÍDEO DE INTERÉS

Para comprender cómo funciona la API de Java, sus módulos, paquetes y clases, visualiza este vídeo:

ESTRUCTURA IF ELSE IF

La estructura if-else-if se puede aplicar en los mismos casos en que se utiliza un if-else anidado. Esta estructura permite escribir de forma abreviada las condiciones de un if-else anidado.

Una sentencia if-else-if tiene la siguiente sintaxis:

```
if (condición_1) {
// Instrucciones si la condición_1 es verdadera
Bloque de sentencias
} else if (condición_2) {
// Instrucciones si la condición_2 es verdadera
Bloque de sentencias
} else if (condición_3) {
// Instrucciones si la condición_3 es verdadera
Bloque de sentencias
} else {
// Instrucciones si ninguna de las condiciones
anteriores es verdadera
Bloque de sentencias
}
```

EJEMPLO PRÁCTICO

Necesitas conocer la puntuación de una serie de estudiantes en base a una tabla de baremación de 0 a 10. El usuario debe introducir la puntuación en forma de número entero o decimal y la ejecución mostrará en pantalla un mensaje textual que indica la calificación obtenida que aparece entre comillas en el siguiente listado.

- Igual o mayor a 9 le corresponde “Excelente”.
- Igual o mayor a 8 y menor que 9 le corresponde “Muy Bueno”.
- Igual o mayor a 7 y menor que 8 le corresponde “Bueno”.
- Igual o mayor a 5 y menor que 7 le corresponde “Suficiente”.
- Si no está comprendida entre ninguna de las citadas, entonces es porque es menor de 5 y le corresponde “Insuficiente”.

SUFICIENTE		BUENO	MUY BUENO	EXCELENTE	
5	6	7	8	9	10

```
import java.util.Scanner;

public class EjemploPracticoPuntuacionEstudiante {
    public static void main(String[] args) {
        //Declaración de variables
        Scanner sc = new Scanner(System.in);
        double calificacion;

        System.out.print("Introduzca la puntuación del estudiante
 (0-10): ");
        calificacion = sc.nextDouble();

        if (calificacion >= 9) {
            System.out.println("Categoría: Excelente");
        } else if (calificacion >= 8) {
            System.out.println("Categoría: Muy Bueno");
        } else if (calificacion >= 7) {
            System.out.println("Categoría: Bueno");
        } else if (calificacion >= 5) {
            System.out.println("Categoría: Suficiente");
        } else {
            System.out.println("Categoría: Insuficiente");
        }
        sc.close();
    }
}
```

ESTRUCTURA SWITCH

La estructura switch es una estructura de selección múltiple que permite seleccionar un bloque de sentencias entre varios casos. En cierto modo, es parecido a una estructura de if-else anidados. La diferencia está en que la selección del bloque de sentencias depende de la evaluación de una expresión que se compara por igualdad con cada uno de los casos.

La estructura switch consta de una expresión y una serie de etiquetas case y una sola opción default. La sentencia break indica el final de la ejecución del switch.

Una sentencia switch tiene la siguiente sintaxis:

```
switch (expresión) {
case valor1:
            bloque-de-sentencias-1;
            break;
case valor2:
            bloque-de-sentencias-2;
            break;
case valor3:
            bloque-de-sentencias-3;
            break;
case valor4:
            bloque-de-sentencias-4;
            break;
default:
            bloque-de-sentencias-default;
}
```

RECUERDA

Es importante colocar una instrucción break al final de cada case que desees termine la ejecución, una vez se cumpla la condición. De esta forma solo se ejecutará el código del case correspondiente y no se continuará con los case siguientes. En default no es necesario poner break porque suele ser el último caso que se ejecutará cuando no se cumpla ninguna de las condiciones especificadas en los case anteriores, por tanto, llega a su fin y no hay más casos que evaluar.

EJEMPLO PRÁCTICO

Aparte de los if-else hay otra estructura muy conocida y usada, se denomina switch. Hay que seguir un cierto patrón a la hora de emplearla. Practica con esta estructura para que, según una variable llamada número cuyo valor es 1 se almacene en otra variable de nombre cadena un texto que indique si es cero, uno, dos, o distinto de todos ellos. Al final se mostrará el valor de la variable cadena.

Te habrás fijado que en la solución que se ha propuesto, siempre se visualizará en pantalla el mismo mensaje. Reflexiona sobre… cómo hacer para que aparezcan distintos mensajes (la variable número deberá entonces almacenar un valor que sea introducido por el usuario).

La estructura switch se corresponde con la estructura Segun…Hacer...FinSegun de pseudocódigo vista en la unidad anterior.

La salida por pantalla será: Es uno

```
public class EjemploPracticoSwitch {
    public static void main(String[] args) {
        int numero=1;
        String cadena;
        switch (numero) {
        case 0: cadena="Es cero";
                break;
        case 1: cadena="Es uno";
                break;
        case 2: cadena="Es dos";
                break;
        default: cadena="Es distinto de 0, 1 ó 2";
        }
        System.out.println(cadena);
    }
}
```

ARTÍCULO DE INTERÉS

Revisa este ejemplo sobre cómo una estructura if puede sustituirse por una estructura case:

PARA SABER MÁS

Visualiza este listado de ejemplos que ayudan a comprender y saber más sobre cómo aplicar la estructura switch en Java:

NOTA DE INTERÉS

Unas de las mejoras en las expresiones switch a partir de la versión 12 de Java ha sido la incorporación de una nueva funcionalidad llamada 'switch expression' que permite usar la declaración switch como una expresión que devuelve un valor, antes solo podía ser usada como una estructura de control que ejecutaba diferentes bloques de código según el valor de una variable.

Cuando comprendas cómo usar esta estructura de la forma convencional puedes probar a incluir y utilizar esta nueva funcionalidad.

5.2 Estructuras repetitivas

Estas estructuras permiten repetir un **número determinado** de instrucciones un **número finito** de veces.

ESTRUCTURA WHILE

La estructura de repetición while repite el bloque de sentencias mientras la condición del while es verdadera. La condición se comprueba justo al principio. Si el resultado es **falso**, entonces **no se ejecuta** el bloque de sentencias. Por eso se dice que se puede ejecutar el bloque de sentencias de 0 (ninguna) a n (muchas) veces.

La sintaxis de while en Java es la siguiente:

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

La condición del while se escribe obligatoriamente entre paréntesis. Cuando el programa ejecuta un while, lo primero que hace es evaluar la condición. Si es verdadera ejecuta el bloque de sentencias, si es falsa finaliza el while.

En cada iteración, cuando finaliza la ejecución del bloque de sentencias, vuelve a evaluar la condición. De nuevo, si es verdadera, ejecuta el bloque de sentencias, si es falsa finaliza el while. Cuando esto último se produce, el flujo del programa continúa en la sentencia inmediatamente posterior al while. La estructura while se corresponde con la estructura Mientras …finMientras vista en pseudocódigo en la unidad anterior.

EJEMPLO PRÁCTICO

Pide un número al usuario para que se muestre por pantalla la tabla de multiplicar de dicho número hasta el número 10.

Utiliza la estructura while en Java.

```
import java.util.Scanner;
public class EjemploPracticoWhileTablaMultiplicar {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int numero, contador, resultado;
        System.out.print("Introduzca un número: ");
        numero = sc.nextInt();
contador = 1;
        while (contador<=10) {
            resultado=contador*numero;
            System.out.println(numero + " * "
+ contador
+ " = "
                    + resultado);
            contador = contador+1;
        }
        sc.close();
    }
}
```

Resultado obtenido al introducir el número 4 por teclado:

```
4 * 1  = 4
4 * 2  = 8
4 * 3  = 12
4 * 4  = 16
4 * 5  = 20
4 * 6  = 24
4 * 7  = 28
4 * 8  = 32
4 * 9  = 36
4 * 10 = 40
```

ESTRUCTURA FOR

La estructura de repetición for repite el bloque de sentencias mientras la condición del for es verdadera. Un for es un caso particular de la estructura while. Sólo se debe utilizar cuando se sabe el número de veces que se debe repetir el bloque de sentencias.
Un for tiene la siguiente sintaxis:

```
for (inicialización; condición; actualización) {
    bloque-de-sentencias;
}
```

EJEMPLO PRÁCTICO

Realiza el ejemplo anterior, pero utilizando una estructura for en Java para realizar la tabla de multiplicar de un número introducido por teclado hasta el 10.

```
import java.util.Scanner;

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

        //Declaración de variables
        int numero, resultado;
        Scanner sc = new Scanner(System.in);

        System.out.print("Introduzca un número: ");
        numero = sc.nextInt();

        for (int contador = 0; contador <= 10; contador++) {
            resultado = contador*numero;
            System.out.println(numero + " * " + contador
+ " = " + resultado);
        }
        sc.close();
    }
}
```

EJEMPLO PRÁCTICO

Anidar estructuras for en Java es muy común cuando se trabaja con arrays bidimensionales, te proponemos un ejemplo sencillo para imprimir los números en un patrón triangular.

Cada fila tiene una secuencia de números que comienza en 1 y aumenta en uno hasta el número de la fila correspondiente. Esto crea un patrón triangular que se va formando a medida que aumenta el número de filas. El número de filas dependerá de la condición de salida del bucle más externo, en este caso será cuando i sea mayor a 5 (mientras i sea menor o igual a 5 el bucle seguirá ejecutándose).

```
public class EjemploPracticoForAnidados {

    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(j + " ");
            }
            System.out.println();
        }
    }
}
```

La salida por pantalla sería la siguiente:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

ESTRUCTURA DO-WHILE

La estructura de repetición do-while ejecuta el bloque de sentencias al menos una vez. Después comprueba la condición y repite el bloque de sentencias mientras la condición es verdadera.

Un do-while tiene la siguiente sintaxis:

```
inicialización;
do {
    bloque-de-sentencias;
    actualización;
} while (condición);
```

Esta es la sintaxis general. La condición se escribe obligatoriamente entre paréntesis.

EJEMPLO PRÁCTICO

Para adquirir soltura con las diferentes estructuras de control te proponemos realizar el ejemplo anterior de la tabla de multiplicar combinando diferentes estructuras como son do-while y while en Java.

Se debe validar que el número introducido sea mayor que cero porque si no lo es se le preguntará de nuevo al usuario hasta que introduzca uno correcto.

```
import java.util.Scanner;
public class EjemploPracticoDoWhile {
    public static void main(String[] args) {
        int numero=0, resultado, contador;
        boolean bandera;
        Scanner sc = new Scanner(System.in);
        /*se establece la variable bandera a true para que por lo
 menos entre una vez en el bucle while
         */
        bandera = true;

        while(bandera) {
            System.out.print("Introduzca un número: ");
            numero = sc.nextInt();
            if(numero>0) {
                /*Si número es mayor que cero se puede
 realizar la tabla entonces
                   se establece la variable
bandera a false para poder
                   salir del bucle while
                 */
                bandera = false;
            }
        }
        //se inicializa contador a 1
        contador = 1;
        do {
            resultado = contador * numero;
            System.out.println(numero +" * " + contador
+ " = " + resultado);
            //se incrementa contador
            contador = contador + 1;

            /*cuando contador sea mayor a 10 ya no se cumplirá
 la condición del do-While y finaliza el bucle
             */
        } while (contador<=10);

        sc.close();
    }
}
```

Resultado obtenido al introducir el número 4 por teclado:

```
4 * 1  = 4
4 * 2  = 8
4 * 3  = 12
4 * 4  = 16
4 * 5  = 20
4 * 6  = 24
4 * 7  = 28
4 * 8  = 32
4 * 9  = 36
4 * 10 = 40
```

RECUERDA

En Java el nombre de una clase debe ser el mismo que el nombre del archivo .java que la contiene. En el último ejemplo práctico de esta unidad la clase que envuelve nuestro código se llama EjemploPracticoDoWhile, por tanto, el archivo donde estará almacenada dicha clase se llamará EjemploPracticoDoWhile.java.

RESUMEN FINAL

En esta unidad se han estudiado las distintas estructuras de control y cuando se utilizan en Java. Además, a través de ejemplos prácticos se ha aprendido a combinar diferentes estructuras para dirigir el flujo de control en función de las necesidades y requerimientos del problema a resolver.

Para empezar a desarrollar aplicaciones en Java primero necesitamos un kit de desarrollo denominado JDK, que es un conjunto de herramientas y recursos proporcionados por Oracle (anteriormente, por Sun Microsystems) para el desarrollo de aplicaciones en el lenguaje de programación Java. El JDK incluye varios componentes esenciales que permiten a los desarrolladores escribir, compilar, depurar y ejecutar programas Java. Entre estos componentes se encuentra la JVM (Java Virtual Machine), que es una parte fundamental del JDK, aunque también se encuentra en el JRE (Java Runtime Environment). La JVM es una máquina virtual que permite la ejecución de programas Java en cualquier plataforma sin necesidad de recompilar el código fuente. Es responsable de interpretar el bytecode y ejecutar el programa en tiempo de ejecución.

También necesitamos un editor o IDE, como por ejemplo Eclipse, que podemos instalar en nuestro ordenador después de preparar todo el entorno para Java.

Los tipos de datos primitivos son boolean, byte, short, int, long, double, float y char, después tenemos los datos primitivos que veremos en otra unidad y que son clases, por tanto, se utilizará la nomenclatura usada para nombrar a una clase (empieza la primera letra con mayúsculas), Ej; String es un tipo compuesto.

Con respecto a las estructuras de control, las más usadas y destacadas son las selectivas como if, if-else, if-else-if y switch y en las repetitivas while, for y do-while.