



PROGRAMACIÓN

<https://martinezpenya.es/1DAMProgramacion/>

© 2025 David Martínez licensed under CC BY-NC-SA 4.0

1º Programacion (CFGs Desarrollo de Aplicaciones Multiplataforma)

1. IES Eduardo Primo Marques (Carlet)

David Martinez Peña

© 2025 David Martínez licensed under CC BY-NC-SA 4.0

Indice de contenidos

1. UD00	9
1.1 Información importante	9
Denominación del curso	9
Contenidos	9
Resultados de Aprendizaje (RA)	10
Legislación vigente	17
Evaluación	17
2. UD01	19
2.1 Elementos de un programa informático	19
Piensa como un programador	19
Problemas, algoritmos y programas	22
Java	26
Componentes del lenguaje Java	29
Herramientas útiles para empezar	38
Ejemplo UD01	39
Píldoras informáticas relacionadas	39
2.2 Ejercicios de la UD01	40
Retos	40
Ejercicios	40
Expresiones Lógicas	43
Actividades	44
2.3 Talleres	46
Taller UD01_01: Instalar NoMachine para el control remoto	46
Taller UD01_02: Instalación y uso de entornos de desarrollo	47
Taller UD01_03: Crear cuenta en GitHub	56
Taller UD01_T03: Markdown	63
3. UD02	70
3.1 Utilización de Objetos y Clases	70
Introducción a la POO	70
Características de la POO	70
Objetos y Clases	71
Utilización de Objetos	74
Utilización de Métodos	78
Librerías de Objetos (Paquetes)	81
Cadenas de caracteres. La clase <code>String</code>	83

Ejemplo UD02	85
Ejemplos UD02	87
Píldoras informáticas relacionadas	87
3.2 Ejercicios de la UD02	88
Actividades	88
Ejercicios	92
3.3 Talleres	95
Taller UD02_01: GitHub Classroom	95
4. UD03	109
4.1 Estructuras de control y Excepciones	109
Introducción	109
Sentencias y bloques	110
Estructuras de selección	112
Estructuras de repetición	115
Estructuras de salto	119
Excepciones	121
Aserciones (Assertions)	126
Ejemplos UD03	127
Píldoras informáticas relacionadas	135
4.2 Ejercicios de la UD03	136
Retos	136
Ejercicios	138
Actividades	160
4.3 Talleres	165
Taller UD03_01: GitHub Classroom	165
Taller UD03_02: Acepta el reto	166
5. UD04	167
5.1 Estructuras de datos: Arrays y matrices. Recursividad.	167
Introducción	167
Arrays	167
Problemas de recorrido, búsqueda y ordenación	171
Arrays bidimensionales: matrices	176
Arrays multidimensionales	178
Recursividad	179
Ejemplos UD04	0
Píldoras informáticas relacionadas	0
5.2 Anexo Cheatsheet Strings en Java	0
Introducción	0

Construyendo string	0
Operando con Métodos de la clase string	0
Ejemplo de todos los métodos de string	0
Arrays de String	0
Los string son inmutables	0
String en Argumentos de Línea de Comandos	0
Concatenar cadenas en Java	0
5.3 Ejercicios de la UD04	0
Arrays. Ejercicios de recorrido	0
Arrays. Ejercicios de búsqueda	0
Matrices	0
Recursividad	0
5.4 Talleres	0
Taller UD04_01: GitHub Classroom	0
6. UD05	0
6.1 Desarrollo de clases	0
Introducción	0
Estructura y miembros de una clase	0
Atributos	0
Métodos	0
Encapsulación, control de acceso y visibilidad.	0
Utilización de los métodos y atributos de una clase.	0
Constructores.	0
Clases Anidadas, Clases Internas (<i>Inner Class</i>) [NUEVO]	0
Introducción a la herencia. [NUEVO]	0
Conversión entre objetos (Casting) [NUEVO]	0
Acceso a métodos de la superclase [NUEVO]	0
Empaquetado de clases [NUEVO]	0
Ejercicios resueltos	0
Ejemplos UD05	0
Píldoras informáticas relacionadas	0
6.2 Anexo Wrappers y Fechas	0
Wrappers (Envoltorios)	0
Clase Date	0
Ejemplo Anexo UD05	0
6.3 Ejercicios de la UD05	0
Ejercicios	0
Actividades	0

6.4 Talleres	0
Taller UD05_01: GitHub Classroom	0
7. UD06	0
7.1 Lectura y escritura de información	0
Streams (Flujos)	0
Ficheros	0
Serialización	0
Sockets	0
Manejo de ficheros y carpetas (<code>File</code>)	0
Ejemplos UD06	0
Píldoras informáticas relacionadas	0
7.2 Comparativa CRUD	0
7.3 Ejercicios de la UD06	0
Paquetes Completos	0
Los flujos estándar	0
InputStreamReader	0
Entrada "orientada a líneas".	0
Lectura/escritura en ficheros	0
Uso de buffers	0
Streams para información binaria	0
Streams de objetos. Serialización.	0
Sockets	0
Más ejercicios (Lionel)	0
Aún más ejercicios	0
7.4 Talleres	0
Taller UD06_01: GitHub Classroom	0
Taller UD06_T02: Sockets en la nube (AWS)	0
8. UD07	0
8.1 Colecciones	0
Introducción	0
Estructuras de almacenamiento	0
Clases y métodos genéricos	0
Colecciones	0
Iteradores	0
Comparadores	0
Extras	0
Programación funcional	0
Ejemplos UD07	0

Píldoras informáticas relacionadas	0
8.2 Ejercicios de la UD07	0
Ejercicios	0
Actividades	0
Ejercicios Genericidad	0
Programación funcional. Funciones Lambda.	0
8.3 Talleres	0
Taller UD07_01: GitHub Classroom	0
9. UD08	0
9.1 Composición, Herencia y Polimorfismo	0
Relaciones entre clases.	0
Composición	0
Herencia	0
La clase <code>Object</code> en Java.	0
Clases Abstractas	0
Interfaces	0
Polimorfismo	0
Ejemplos UD08	0
Píldoras informáticas relacionadas	0
9.2 Ejercicios de la UD08	0
Ejercicios Herencia	0
Ejercicios Polimorfismo	0
Actividades	0
Ejercicios Lionel	0
9.3 Talleres	0
Taller UD08_01: GitHub Classroom	0
Taller UD08_02: Librerías Maven vs Jar	0
Taller UD08_T03: Introducción a JSON y YAML	0
10. UD09	0
10.1 Interfaz gráfica	0
Introducción	0
Gráfico de escena	0
Controles de la interfaz de usuario	0
Diseño (Layouts)	0
Estructura de la aplicación	0
Píldoras informáticas relacionadas	0
10.2 Ejercicios de la UD09	0
Cuestiones generales	0

Ejercicios (con SceneBuilder)	0
Actividades	0
Más ejercicios (Sin SceneBuilder)	0
10.3 Talleres	0
Taller UD09_01: GitHub Classroom	0
Taller UD09_02: Scene Builder, ScenicView y FXMLManager	0
Taller UD09_03: Proyecto JavaFX (con Maven)	0
Taller UD09_04: Calculadora en JavaFX (IntelliJ)	0
11. UD10	0
11.1 Acceso a Bases de Datos Relacionales	0
Introducción	0
JDBC	0
Patrones de diseño aplicables	0
Acceso a BBDD	0
Navegabilidad y concurrencia	0
Consultas (Query)	0
Modificación (update)	0
Inserción (insert)	0
Borrado (delete)	0
Sentencias predefinidas	0
Píldoras informáticas relacionadas	0
11.2 Ejercicios de la UD10	0
Actividades	0
Para probar...	0
11.3 Talleres	0
Taller UD10_01: GitHub Classroom	0
Taller UD10_02: Conectores	0
Taller UD10_03: BBDD en la nube (AWS) con IntelliJ	0
Taller UD10_04: Patron DAO (CRUD completo)	0
12. UD11	0
12.1 Bases de Datos Orientadas a Objetos	0
Introducción	0
Los lenguajes oDL y oQL	0
La librería ObjectDB	0
12.2 ObjectDB es un potente sistema de gestión de bases de datos orientado a objetos (oDBMS). Es compacto, fiable de usar y extremadamente rápido. ObjectDB proporciona todos los servicios estándar de administración de bases de datos (almacenamiento y recuperación, transacciones, administración de bloqueos, procesamiento de consultas, etc.), pero de una manera que facilita el desarrollo y acelera las aplicaciones.	0
Características clave de la base de datos ObjectDB	0

ObjectDB , que es la BDD elegida permite tanto JDO como JPA (Java Persistence API). Aunque parece que el mercado evoluciona del lado de JPA y es la modalidad más usada para dotar de persistencia a los datos. Por tanto en este tema usaremos ObjectDB con JPA .	0
Anotaciones para ObjectDB	0
Aquí, por su extensión, solo veremos una parte de todo lo que se puede hacer con ObjectDB y JPA, si te interesa profundizar más, te recomiendo que acudas al manual online que tienen publicado en su web, y en el que se basan estos apuntes: https://house.objectdb.com/java/jpa	0
Comparativa BDR vs BDDO vs ORM	0
Píldoras informáticas relacionadas	0
12.3 Ejercicios de la UD11	0
Actividades	0
Ejercicios	0
Ejercicios propuestos	0
Supuestos prácticos	0
Auto evaluación	0
12.4 Talleres	0
Taller UD11_01: GitHub Classroom	0
Taller UD11_02: Añadir ObjectDB a un proyecto IntelliJ (Maven)	0
Taller UD11_03: CRUD con ObjectDB	0
13.  Sobre mí...	0
13.1  David Martínez Peña	0
13.2  Contacto:	0
14. Fuentes de información	0

1. UD00

2. 1.1 Información importante



PROGRAMACIÓN

<https://martinezpenya.es/1DAMProgramacion/>
 © 2025 David Martínez licensed under CC BY-NC-SA 4.0

2.1. Denominación del curso

Ciclo formativo de Grado Superior en Desarrollo de Aplicaciones Multiplataforma

 Programación (PRG)

B5	UD05: Desarrollo de clases	25
B6	UD06: Lectura y escritura de información	25
B4	UD07: Colecciones y Funciones Lambda	

2.2. Contenidos

Bloque	P R I M E R TRIMESTRE	Horas
B1	UD01: Elementos de un programa informático	19
B2	UD02: Utilización de Objetos	20
PRUEBA UNIDADES 1 Y 2		6
B3	UD03: Estructuras de control y Excepciones	20
B4	UD04: Estructuras de datos Arrays y matrices. Recursividad	18
1^a EVALUACIÓN		6
S E G U N D O TRIMESTRE		77
B5	UD05: Desarrollo de clases	25
B6	UD06: Lectura y escritura de información	25
B4	UD07: Colecciones y Funciones Lambda	21
2^a EVALUACIÓN		6
T E R C E R TRIMESTRE		90
B8	UD08: Composición, Herencia y Polimorfismo	20
B9	UD09: Creación de interfaces gráficas	20
B10	UD10: Acceso a Bases de datos	24

		Horas
B11	UD11: BBDD OO	16
	3^a EVALUACIÓN	6
	CONVOCATÒRIA ORDINÀRIA	4
	T O T A L	256

2.3. Resultados de Aprendizaje (RA)

	Descripció	Pes	UNITAT	Avaluació
RA1	Reconoce la estructura de un programa informático, identificando y relacionando los elementos propios del lenguaje de programación utilizado.	10%		
A	Se han identificado los bloques que componen la estructura de un programa informático.	11%	1	= [[1AVA]]
B	Se han creado proyectos de desarrollo de aplicaciones	11%	2	= [[1AVA]]
C	Se han utilizado entornos integrados de desarrollo.	11%	2	= ([[1AVA]]*0,5) + ([[FEE]]*0,5)
D	Se han identificado los distintos tipos de variables y la utilidad específica de cada uno.	11%	1	= ([[1AVA]]*0,2) + ([[2AVA]]*0,3) + ([[3AVA]]*0,5)
E	Se ha modificado el código de un programa para crear y utilizar variables.	11%	1	= ([[1AVA]]*0,2) + ([[2AVA]]*0,3) + ([[3AVA]]*0,5)
F	Se han creado y utilizado constantes y literales.	11%	1	= ([[1AVA]]*0,2) + ([[2AVA]]*0,3) + ([[3AVA]]*0,5)
G	Se han clasificado, reconocido y utilizado en expresiones los operadores del lenguaje.	11%	1	= ([[1AVA]]*0,2) + ([[2AVA]]*0,3) + ([[3AVA]]*0,5)
H	Se ha comprobado el funcionamiento de las conversiones de tipo explícitas e implícitas.	11%	1	= ([[1AVA]]*0,2) + ([[2AVA]]*0,3) + ([[3AVA]]*0,5)
I	Se han introducido comentarios en el código.	11%	1	= ([[1AVA]]*0,2) + ([[2AVA]]*0,3) + ([[3AVA]]*0,5)
	Descripció	Pes	UNITAT	Avaluació
RA2	Escribe y prueba programas sencillos,	10%		

	Descripció	Pes	UNITAT	Avaluació
	reconociendo y aplicando los fundamentos de la programación orientada a objetos.			
A	Se han identificado los fundamentos de la programación orientada a objetos.	11%	2	=[[1AVA]]
B	Se han escrito programas simples.	11%	2	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
C	Se han instanciado objetos a partir de clases predefinidas.	11%	2	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
D	Se han utilizado métodos y propiedades de los objetos.	11%	2	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
E	Se han escrito llamadas a métodos estáticos.	11%	2	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
F	Se han utilizado parámetros en la llamada a métodos.	11%	2	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
G	Se han incorporado y utilizado librerías de objetos.	11%	2	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
H	Se han utilizado constructores.	11%	2	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
I	Se ha utilizado el entorno integrado de desarrollo en la creación y compilación de programas simples.	11%	2	=([[1AVA]]*0,05)+([[2AVA]]*0,15)+([[3AVA]]*0,30)+([[FEE]]*0,50)
	Descripció	Pes	UNITAT	Avaluació
RA3	Escribe y depura código, analizando y utilizando las estructuras de control del lenguaje.	10%		
A	Se ha escrito y probado código que haga uso de estructuras de selección.	11%	3	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
B	Se han utilizado estructuras de repetición.	11%	3	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
C	Se han reconocido las posibilidades de las sentencias de salto.	11%	3	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
D	Se ha escrito código utilizando control de excepciones.	11%	3	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)

	Descripció	Pes	UNITAT	Avaluació
E	Se han creado programas ejecutables utilizando diferentes estructuras de control.	11%	3	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
F	Se han probado y depurado los programas.	11%	3	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
G	Se ha comentado y documentado el código.	11%	3	=([[1AVA]]*0,05)+([[2AVA]]*0,15)+([[3AVA]]*0,30)+([[FEE]]*0,50)
H	Se han creado excepciones.	11%	3	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
I	Se han utilizado aserciones para la detección y corrección de errores durante la fase de desarrollo.	11%	3	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
	Descripció	Pes	UNITAT	Avaluació
RA4	Desarrolla programas organizados en clases analizando y aplicando los principios de la programación orientada a objetos.	10%		
A	Se ha reconocido la sintaxis, estructura y componentes típicos de una clase.	11%	5	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
B	Se han definido clases.	11%	5	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
C	Se han definido propiedades y métodos.	11%	5	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
D	Se han creado constructores.	11%	5	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
E	Se han desarrollado programas que instancien y utilicen objetos de las clases creadas anteriormente.	11%	5	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
F	Se han utilizado mecanismos para controlar la visibilidad de las clases y de sus miembros.	11%	5	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
G	Se han definido y utilizado clases heredadas.	11%	5	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
H	Se han creado y utilizado métodos estáticos.	11%	5	=([[2AVA]]*0,5)+([[3AVA]]*0,5)

	Descripció	Pes	UNITAT	Avaluació
I	Se han creado y utilizado conjuntos y librerías de clases.	11%	5	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
	Descripció	Pes	UNITAT	Avaluació
RA5	Realiza operaciones de entrada y salida de información, utilizando procedimientos específicos del lenguaje y librerías de clases.	15%		
A	Se ha utilizado la consola para realizar operaciones de entrada y salida de información.	10%	6	=([[1AVA]]*0,05)+([[2AVA]]*0,15)+([[3AVA]]*0,30)+([[FEE]]*0,50)
B	Se han aplicado formatos en la visualización de la información.	10%	6	=([[1AVA]]*0,2)+([[2AVA]]*0,3)+([[3AVA]]*0,5)
C	Se han reconocido las posibilidades de entrada / salida del lenguaje y las librerías asociadas.	10%	6	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
D	Se han utilizado ficheros para almacenar y recuperar información.	10%	6	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
E	Se han creado programas que utilicen diversos métodos de acceso al contenido de los ficheros.	10%	6	=([[2AVA]]*0,5)+([[3AVA]]*0,5)
F	Se han utilizado las herramientas del entorno de desarrollo para crear interfaces gráficos de usuario simples.	20%	9	'=([[3AVA]]*0,25)+([[FEE]]*0,50)+([[UD09_T01]]*0,25)+([[UD09_T02]]*0,25)+([[UD09_T03]]*0,5)*0,25)
G	Se han programado controladores de eventos.	15%	9	'=([[3AVA]]*0,4)+([[UD09_T01]]*0,25)+([[UD09_T02]]*0,25)+([[UD09_T03]]*0,5)*0,6)
H	Se han escrito programas que utilicen interfaces gráficos para la entrada y salida de información.	15%	9	'=([[3AVA]]*0,4)+([[UD09_T01]]*0,25)+([[UD09_T02]]*0,25)+([[UD09_T03]]*0,5)*0,6)
	Descripció	Pes	UNITAT	Avaluació
RA6	Escribe programas que manipulen información seleccionando y utilizando tipos avanzados de datos.	20%		
A		50%	4	

	Descripció	Pes	UNITAT	Avaluació
	Se han escrito programas que utilicen matrices (arrays) .			=([[2AVA]]*0,5)+([[3AVA]]*0,50)
B	Se han reconocido las librerías de clases relacionadas con tipos de datos avanzados.	5%	7	=[[3AVA]]
C	Se han utilizado listas para almacenar y procesar información.	5%	7	=[[3AVA]]
D	Se han utilizado iteradores para recorrer los elementos de las listas.	5%	7	=[[3AVA]]
E	Se han reconocido las características y ventajas de cada una de la colecciones de datos disponibles.	10%	7	=[[3AVA]]
F	Se han creado clases y métodos genéricos.	5%	7	=[[3AVA]]
G	Se han utilizado expresiones regulares en la búsqueda de patrones en cadenas de texto.	5%	7	=[[3AVA]]
H	Se han identificado las clases relacionadas con el tratamiento de documentos escritos en diferentes lenguajes de intercambio de datos.	5%	7	=([[UD08_T01]]*0,5)+([[UD08_T02]]*0,5)
I	Se han realizado programas que realicen manipulaciones sobre documentos escritos en diferentes lenguajes de intercambio de datos.	5%	7	=([[UD08_T01]]*0,5)+([[UD08_T02]]*0,5)
J	Se han utilizado operaciones agregadas para el manejo de información almacenada en colecciones.	5%	7	=[[3AVA]]
	Descripció	Pes	UNITAT	Avaluació
RA7	Desarrolla programas aplicando características avanzadas de los lenguajes orientados a objetos y del	10%		

	Descripció	Pes	UNITAT	Avaluació
	entorno de programación.			
A	Se han identificado los conceptos de herencia, superclase y subclase.	10%	8	=[[3AVA]]
B	Se han utilizado modificadores para bloquear y forzar la herencia de clases y métodos.	10%	8	=[[3AVA]]
C	Se ha reconocido la incidencia de los constructores en la herencia.	10%	8	=[[3AVA]]
D	Se han creado clases heredadas que sobrescriban la implementación de métodos de la superclase.	10%	8	=[[3AVA]]
E	Se han diseñado y aplicado jerarquías de clases.	10%	8	=[[3AVA]]
F	Se han probado y depurado las jerarquías de clases.	10%	8	=[[3AVA]]
G	Se han realizado programas que implementen y utilicen jerarquías de clases.	10%	8	=[[3AVA]]
H	Se ha comentado y documentado el código.	10%	8	=([[3AVA]]*0,50)+([[FEE]]*0,50)
I	Se han identificado y evaluado los escenarios de uso de interfaces.	10%	8	=[[3AVA]]
J	Se han identificado y evaluado los escenarios de utilización de la herencia y la composición.	10%	8	=[[3AVA]]
	Descripció	Pes	UNITAT	Avaluació
RA8	Utiliza bases de datos orientadas a objetos, analizando sus características y aplicando técnicas para mantener la persistencia de la información.	5%		
A	Se han identificado las características de las bases de datos orientadas a objetos.	13%	11	=([[UD11_T1]]*0,3)+([[UD11_T2]]*0,7)

	Descripció	Pes	UNITAT	Avaluació
B	Se ha analizado su aplicación en el desarrollo de aplicaciones mediante lenguajes orientados a objetos.	13%	11	$=([[UD11_T1]]*0,3)+([[UD11_T2]]*0,7)$
C	Se han instalado sistemas gestores de bases de datos orientados a objetos.	13%	11	$=([[UD11_T1]]*0,3)+([[UD11_T2]]*0,7)$
D	Se han clasificado y analizado los distintos métodos soportados por los sistemas gestores para la gestión de la información almacenada.	13%	11	$=([[UD11_T1]]*0,3)+([[UD11_T2]]*0,7)$
E	Se han creado bases de datos y las estructuras necesarias para el almacenamiento de objetos.	13%	11	$=([[UD11_T1]]*0,3)+([[UD11_T2]]*0,7)$
F	Se han programado aplicaciones que almacenen objetos en las bases de datos creadas.	13%	11	$=([[UD11_T1]]*0,3)+([[UD11_T2]]*0,7)$
G	Se han realizado programas para recuperar, actualizar y eliminar objetos de las bases de datos.	13%	11	$=([[UD11_T1]]*0,3)+([[UD11_T2]]*0,7)$
H	Se han realizado programas para almacenar y gestionar tipos de datos estructurados, compuestos y relacionados.	13%	11	$=([[UD11_T1]]*0,3)+([[UD11_T2]]*0,7)$
	Descripció	Pes	UNITAT	Avaluació
RA9	Gestiona información almacenada en bases de datos relacionales manteniendo la integridad y consistencia de los datos.	10%		
A	Se han identificado las características y métodos de acceso a sistemas gestores de bases de datos relacionales.	14%	10	$=([[UD10_T1]]*0,2)+([[UD10_T2]]*0,3)+([[UD10_T3]]*0,5)$
B	Se han programado conexiones con bases de datos.	14%	10	$=([[UD10_T1]]*0,2)+([[UD10_T2]]*0,3)+([[UD10_T3]]*0,5)$
C		14%	10	

	Descripció	Pes	UNITAT	Avaluació
	Se ha escrito código para almacenar información en bases de datos.			=([[UD10_T1]]*0,2)+([[UD10_T2]]*0,3)+([[UD10_T3]]*0,5)
D	Se han creado programas para recuperar y mostrar información almacenada en bases de datos.	14%	10	=([[UD10_T1]]*0,2)+([[UD10_T2]]*0,3)+([[UD10_T3]]*0,5)
E	Se han efectuado borrados y modificaciones sobre la información almacenada.	14%	10	=([[UD10_T1]]*0,2)+([[UD10_T2]]*0,3)+([[UD10_T3]]*0,5)
F	Se han creado aplicaciones que muestren la información almacenada en bases de datos.	14%	10	=([[UD10_T1]]*0,05)+([[UD10_T2]]*0,15)+([[UD10_T3]]*0,3)+([[FEE]]*0,5)
G	Se han creado aplicaciones para gestionar la información presente en bases de datos relacionales.	14%	10	=([[UD10_T1]]*0,05)+([[UD10_T2]]*0,15)+([[UD10_T3]]*0,3)+([[FEE]]*0,5)

2.4. Legislación vigente

-  [RD 450/2010, BOE 20-05-2010](#) (Antigua ley)
-  [RD 405/2023 29-05-2023](#)
-  [RD 500/2024, BOE 21-05-2024](#)
-  [Curriculum C.V.: ORDE 58/2012, de 5 de setembre \(DOGV núm. 6868, 24.09.2012\)](#) (Antiguo)
-  [Propuesta de Decreto del Consell](#)
-  [Horario](#) (Antigua ley)
-  Horario

2.5. Evaluación

-  La evaluación del módulo se realizará con base en los **Resultados de Aprendizaje (RA)** definidos en el currículo del ciclo formativo de Grado Superior en Desarrollo de Aplicaciones Multiplataforma. Cada RA estará asociado a **criterios de evaluación (CE)** que serán los que determinen el grado de adquisición de las competencias previstas para el módulo.
-  La nota final del módulo se obtendrá a partir de la ponderación de los **RA**, como se mencionó anteriormente. Cada **RA** será evaluado de forma independiente, con calificaciones en una escala de 0 a 10.
-  El alumno debe obtener al menos una nota de **5** en cada **RA** para aprobar el módulo.
-  Si un alumno obtiene menos de un **5** en algún **RA**, tendrá que recuperarlo mediante las actividades/exámenes de recuperación diseñadas específicamente para esos resultados de aprendizaje.
-  En programación los primeros **RA's** se distribuyen entre las 3 evaluaciones, así que tener una buena nota en la primera evaluación no quiere decir que has aprobado los **RA** de esa evaluación.
-  **NUEVO SISTEMA DUAL!!** → Busca tu empresa! 120H (aproximadamente en el mes de mayo, también a partir del 2º trimestre por las mañanas)

 **IMPORTANTE:**

- ! Aprobar las distintas evaluaciones no garantiza aprobar el curso.
- ❤ Puedes aprobar (y con muy buena nota) las dos evaluaciones, tener un RA suspendido y por tanto suspender el módulo.

 15 de enero de 2026

2. UD01

3. 2.1 Elementos de un programa informático



3.1. Piensa como un programador

Una de las acepciones que trae el Diccionario de Real Academia de la Lengua Española (RAE) respecto a la palabra Problema es “**Planteamiento de una situación cuya respuesta desconocida debe obtenerse a través de métodos científicos**”. Con miras a lograr esa respuesta, un problema se puede definir como una situación en la cual se trata de alcanzar una meta y para lograrlo se deben hallar y utilizar unos medios y unas estrategias.

La mayoría de problemas tienen algunos elementos en común: un estado inicial; una meta, lo que se pretende lograr; un conjunto de recursos, lo que está permitido hacer y/o utilizar; y un dominio, el estado actual de conocimientos, habilidades y energía de quien va a resolverlo (Moursund, 1999).

Casi todos los problemas requieren, que quien los resuelve, los divida en submetas que, cuando son dominadas (por lo regular en orden), llevan a alcanzar el objetivo. La solución de problemas también requiere que se realicen operaciones durante el estado inicial y las submetas, actividades (conductuales, cognoscitivas) que alteran la naturaleza de tales estados (Schunk, 1997).

Cada disciplina dispone de estrategias específicas para resolver problemas de su ámbito; por ejemplo, resolver problemas matemáticos implica utilizar estrategias propias de las matemáticas. Sin embargo, algunos psicólogos opinan que es posible utilizar con éxito estrategias generales, útiles para resolver problemas en muchas áreas. A través del tiempo, la humanidad ha utilizado diversas estrategias generales para resolver problemas. Schunk (1997), Woolfolk (1999) y otros, destacan los siguientes métodos o estrategias de tipo general:

- **Ensayo y error**: Consiste en actuar hasta que algo funcione. Puede tomar mucho tiempo y no es seguro que se llegue a una solución. Es una estrategia apropiada cuando las soluciones posibles son pocas y se pueden probar todas, empezando por la que ofrece mayor probabilidad de resolver el problema.

Por ejemplo, una bombilla que no prende: revisar la bombilla, verificar la corriente eléctrica, verificar el interruptor.

- **Iluminación**: Implica la súbita conciencia de una solución que sea viable. Es muy utilizado el modelo de cuatro pasos formulado por Wallas (1921): preparación, incubación, iluminación y verificación.

Estos cuatro momentos también se conocen como proceso creativo. **Algunas investigaciones han determinado que cuando en el periodo de incubación se incluye una interrupción en el trabajo sobre un problema se logran mejores resultados desde el punto de vista de la creatividad.** La incubación ayuda a "olvidar" falsas pistas, mientras que no hacer interrupciones o descansos puede hacer que la persona que trata de encontrar una solución creativa se estanque en estrategias inapropiadas.

Ejemplos:

- Dispones de 6 lapices/palillos/cerillas igual de largos, ¿como puedes formar 4 triángulos iguales y equiláteros?
- Mueve 2 cerillas para seguir teniendo una copa pero con la cereza fuera:
cerillas acertijo 1
- **Heurística** : Se basa en la utilización de reglas empíricas para llegar a una solución. El método heurístico conocido como "IDEAL", formulado por Bransford y Stein (1984), incluye cinco pasos: Identificar el problema; definir y presentar el problema; explorar las estrategias viables; avanzar en las estrategias; y lograr la solución y volver para evaluar los efectos de las actividades (Bransford & Stein, 1984).

El matemático Polya (1957) también formuló un método heurístico para resolver problemas que se aproxima mucho al ciclo utilizado para programar computadores. A lo largo de esta Guía se utilizará este método propuesto por Polya.

- **Algoritmos** : Consiste en aplicar adecuadamente una serie de pasos detallados que aseguran una solución correcta. Por lo general, cada algoritmo es específico de un dominio del conocimiento. La programación de computadores se apoya en este método.
- **Modelo de procesamiento de información** : El modelo propuesto por Newell y Simon (1972) se basa en plantear varios momentos para un problema (estado inicial, estado final y vías de solución). Las posibles soluciones avanzan por subtemas y requieren que se realicen operaciones en cada uno de ellos.
- **Análisis de medios y fines** : Se funda en la comparación del estado inicial con la meta que se pretende alcanzar para identificar las diferencias.

Luego se establecen submetas y se aplican las operaciones necesarias para alcanzar cada submeta hasta que se alcance la meta global. Con este método se puede proceder en retrospectiva (desde la meta hacia el estado inicial) o en prospectiva (desde el estado inicial hacia la meta).

- **Razonamiento analógico** : Se apoya en el establecimiento de una analogía entre una situación que resulte familiar y la situación problema. Requiere conocimientos suficientes de ambas situaciones.
- **Lluvia de ideas** : Consiste en formular soluciones viables a un problema. El modelo propuesto por Mayer (1992) plantea: definir el problema; generar muchas soluciones (sin evaluarlas); decidir los criterios para estimar las soluciones generadas; y emplear esos criterios para seleccionar la mejor solución. Requiere que los estudiantes no emitan juicios con respecto a las posibles soluciones hasta que terminen de formularlas.
- **Sistemas de producción** : Se basa en la aplicación de una red de secuencias de condición y acción (Anderson, 1990).
- **Pensamiento lateral** : Se apoya en el pensamiento creativo, formulado por Edwar de Bono (1970), el cual difiere completamente del pensamiento lineal (lógico). El pensamiento lateral requiere que se exploren y consideren la mayor cantidad posible de alternativas para solucionar un problema. Su importancia para la educación radica en permitir que el estudiante: explore (escuche y acepte puntos de vista diferentes, busque alternativas); avive (promueva el uso de la fantasía y del humor); libere (use la discontinuidad y escape de ideas preestablecidas); y contrarreste la rigidez (vea las cosas desde diferentes ángulos y evite dogmatismos). Este es un método adecuado cuando el problema que se desea resolver no requiere información adicional, sino un reordenamiento de la información disponible; cuando hay ausencia del problema y es necesario apercibirse de que hay un problema; o cuando se debe reconocer la posibilidad de perfeccionamiento y redefinir esa posibilidad como un problema (De Bono, 1970).

Ejemplos:

- **El dilema del náufrago.** Un náufrago necesita trasladar a su isla de residencia algunos restos del naufragio de su barco, que afloraron en la orilla de la isla de enfrente. Allí tiene un zorro, un conejo y un racimo de zanahorias, que en su bote puede llevar a razón de uno por viaje. ¿Cómo puede llevarlo todo a su isla, sin que el zorro se coma al conejo, ni éste a las zanahorias?
- **Respuesta:** Deberá llevar primero al conejo y dejar al zorro con las zanahorias. Luego volver y llevarse al zorro, que dejará a solas en su isla, tomar al conejo y llevarlo de vuelta a la de enfrente. Despues llevará las zanahorias, dejando al conejo solo y depositándolas junto al zorro. Finalmente regresará para hacer un último viaje con el conejo.
- **El dilema del ascensor.** Un hombre que vive en el décimo piso de un edificio, toma todos los días el ascensor hasta la planta baja, para ir a trabajar. En la tarde, sin embargo, toma de nuevo el mismo ascensor, pero si no hay nadie con él, baja en el séptimo piso y sube el resto de los pisos por la escalera. ¿Por qué?

Respuesta: El hombre es bajito y no logra presionar el botón del décimo piso.

- **La paradoja del globo.** ¿De qué manera podemos pinchar un globo con una aguja, sin que se fugue el aire y sin que el globo estalle?

Respuesta: Debemos pinchar el globo estando desinflado.

- **El dilema del bar.** Un hombre entra a un bar y le pide al barman un vaso de agua. El barman busca debajo de la barra y de golpe apunta al hombre con un arma. Este último da las gracias y se marcha. ¿Qué acaba de ocurrir?

Respuesta: El barman se percató de que el hombre tenía hipo, y decide curárselo dándole un buen susto.

Como se puede apreciar, hay muchas estrategias para solucionar problemas; sin embargo, esta Guía se enfoca principalmente en dos de estas estrategias: Heurística y Algorítmica.

Según Polya (1957), cuando se resuelven problemas, intervienen cuatro operaciones mentales:

1. Entender el problema;
2. Trazar un plan;
3. Ejecutar el plan (resolver);
4. Revisar;

Es importante notar que estas son flexibles y no una simple lista de pasos como a menudo se plantea en muchos de esos textos (Wilson, Fernández & Hadaway, 1993). Cuando estas etapas se siguen como un modelo lineal, resulta contraproducente para cualquier actividad encaminada a resolver problemas.

Es necesario hacer énfasis en la naturaleza dinámica y cíclica de la solución de problemas. En el intento de trazar un plan, los estudiantes pueden concluir que necesitan entender mejor el problema y deben regresar a la etapa anterior; O cuando han trazado un plan y tratan de ejecutarlo, no encuentran cómo hacerlo; entonces, la actividad siguiente puede ser intentar con un nuevo plan o regresar y desarrollar una nueva comprensión del problema (Wilson, Fernández & Hadaway, 1993; Guzdial, 2000).

La mayoría de los textos escolares de matemáticas abordan la Solución de Problemas bajo el enfoque planteado por Polya. Por ejemplo, en "Recreo Matemático 5" (Díaz, 1993) y en "Dominios 5" (Melo, 2001) se pueden identificar las siguientes sugerencias propuestas a los estudiantes para llegar a la solución de un problema matemático:

1. COMPRENDER EL PROBLEMA.

- Leer el problema varias veces
- Establecer los datos del problema (¿marcarlos de alguna manera?)
- Aclarar lo que se va a resolver (¿Cuál es la pregunta?)
- Precisar el resultado que se desea lograr
- Determinar la incógnita del problema
- Organizar la información
- Agrupar los datos en categorías
- Trazar una figura o diagrama.

2. HACER EL PLAN.

- Escoger y decidir las operaciones a efectuar.
- Eliminar los datos inútiles.
- **Descomponer el problema en otros más pequeños.**

3. EJECUTAR EL PLAN (Resolver).

- Ejecutar en detalle cada operación.
- Simplificar antes de calcular.
- Realizar un dibujo o diagrama

4. ANALIZAR LA SOLUCIÓN (Revisar).

- Dar una respuesta completa
- Hallar el mismo resultado de otra manera.
- Verificar por apreciación que la respuesta es adecuada.

Numerosos autores de libros sobre programación, plantean cuatro fases para elaborar un procedimiento que realice una tarea específica. Estas fases concuerdan con las operaciones mentales descritas por Polya para resolver problemas:

1. Analizar el problema (Entender el problema)

2. Diseñar un algoritmo (Trazar un plan)
3. Traducir el algoritmo a un lenguaje de programación (Ejecutar el plan)
4. Depurar el programa (Revisar)

Como se puede apreciar, hay una similitud entre las metodologías propuestas para solucionar problemas matemáticos (Clements & Meredith, 1992; Díaz, 1993; Melo, 2001; NAP, 2004) y las cuatro fases para solucionar problemas específicos de áreas diversas, mediante la programación de computadores.

Problema de la Jirafa

Primera pregunta: ¿Cómo podríamos meter una jirafa dentro de una nevera? Piensa que es un problema para niños y a ellos no se les pasaría por la cabeza trocear al bello animal para resolver un problema.

Segunda pregunta: Repetimos la jugada con distinto protagonista. ¿Cómo metemos un elefante dentro de la nevera?

Tercera pregunta: Imaginemos que el Rey León está celebrando su cumpleaños y ha invitado a todos los animales del reino. Acuden todos excepto uno. ¿Quién falta?

Cuarta pregunta: Estamos frente a un río que debemos cruzar como sea para continuar nuestro camino. El único problema es que esa zona es el hogar de unos cocodrilos muy agresivos y no disponemos de ningún tipo de embarcación para ir al otro lado. ¿Cómo harías para cruzar el río sin morir en el intento?

3.2. Problemas, algoritmos y programas

3.2.1. Problemas

Podríamos decir que la **programación** es una forma de resolución de **problemas**.

Para que un problema pueda resolverse utilizando un programa informático, éste tiene que poder resolverse de forma mecánica, es decir, mediante una secuencia de instrucciones u operaciones que se puedan llevar a cabo de manera **automática** por un ordenador.

Ejemplos de problemas resolubles mediante un ordenador:

- Determinar el producto de dos números a y b.
- Determinar la raíz cuadrada positiva del número 2.
- Determinar la raíz cuadrada positiva de un número n cualquiera.
- Determinar si el número n, entero mayor que uno, es primo.
- Dada la lista de palabras, determinar las palabras repetidas.
- Determinar si la palabra p es del idioma castellano.
- Ordenar y listar alfabéticamente todas las palabras del castellano.
- Dibujar en pantalla un círculo de radio r.
- Separar las silabas de una palabra p.
- A partir de la fotografía de un vehículo, reconocer y leer su matrícula.
- Traducir un texto de castellano a inglés.
- Detectar posibles tumores a partir de imágenes radiográficas.

Por otra parte, el científico Alan Turing, demostró que existen problemas irresolubles, de los que ningún ordenador será capaz de obtener nunca su solución.

Los problemas deben definirse de forma general y precisa, **evitando ambigüedades**.

Ejemplo: Raíz cuadrada.

- Determinar la raíz cuadrada de un número n.
- Determinar la raíz cuadrada de un número n, entero no negativo, cualquiera.

Ejemplo: Dividir.

- Calcular la división de dos números de dos números a y b.
- Calcular el cociente entero de la división a/b, donde a y b son números enteros y b es distinto de cero. ($5/2 = 2$).
- Calcular el cociente real de la división a/b, donde a y b son números reales y b es distinto de cero ($5/2 = 2.5$).

3.2.2. Algoritmos

Dado un problema P, un **algoritmo** es un conjunto de reglas o pasos que indican cómo resolver P en un tiempo finito.

Secuencias de reglas básicas que utilizamos para realizar operaciones aritméticas: sumas, restas, productos y divisiones.

Algoritmo para desayunar

```

1 Begin
2   Sentarse
3   Servirse café con leche
4   Servirse azúcar
5   If tengo tiempo
6     While tenga apetito
7       Untar mantequilla en una tostada
8       Añadir mermelada
9       Comer la tostada
10    End While
11  End If
12 Beberse el café con leche
13 Levantarse
14 End

```

Un algoritmo, por tanto, no es más que la secuencia de pasos que se deben seguir para solucionar un problema específico. La descripción o nivel de detalle de la solución de un problema en términos algorítmicos depende de qué o quién debe entenderlo, interpretarlo y resolverlo.

Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan. Un mismo algoritmo puede ser expresado en diferentes lenguajes de programación y podría ser ejecutado en diferentes dispositivos. Piensa en una receta de cocina, ésta puede ser expresada en castellano, inglés o francés, podría ser cocinada en fogón o vitrocerámica, por un cocinero o más, etc. Pero independientemente de todas estas circunstancias, el plato se preparará siguiendo los mismos pasos.

La **diferencia** fundamental entre **algoritmo** y **programa** es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado lenguaje de programación para que puedan ser ejecutados en el ordenador y así obtener la solución.

3.2.2.1. CARACTERÍSTICAS DE LOS ALGORITMOS

Un algoritmo, para que sea válido, tiene que tener ciertas características fundamentales:

- **Generalidad:** han de definirse de forma general, utilizando identificadores o parámetros. Un algoritmo debe resolver toda una clase de problemas y no un problema aislado particular.
- **Finitud:** han de llevarse a cabo en un tiempo finito, es decir, el algoritmo ha de acabar necesariamente tras un número finito de pasos.
- **Definibilidad:** han de estar definidos de forma exacta y precisa, sin ambigüedades.
- **Eficiencia:** han de resolver el problema de forma rápida y eficiente.

Juego de las monedas (Eduardo Sáenz Cabezón)



Desde el comienzo del enlace hasta 7 minutos después.

3.2.2.2. REPRESENTACIÓN DE ALGORITMOS

Los métodos más usuales para representar algoritmos son los diagramas de flujo y el pseudocódigo. Ambos son sistemas de representación independientes de cualquier lenguaje de programación. Hay que tener en cuenta que el diseño de un algoritmo constituye un paso previo a la codificación de un programa en un lenguaje de programación determinado (C, C++, Java, Pascal). La independencia del algoritmo del lenguaje de programación facilita, precisamente, la posterior codificación en el lenguaje elegido.

Un **Diagrama de flujo** (Flowchart) es una de las técnicas de representación de algoritmos más antiguas y más utilizadas, aunque su empleo disminuyó considerablemente con los lenguajes de programación estructurados. Un diagrama de flujo utiliza símbolos estándar que contienen los pasos del algoritmo escritos en esos símbolos, unidos por flechas denominadas líneas de flujo que indican la secuencia en que deben ejecutarse.

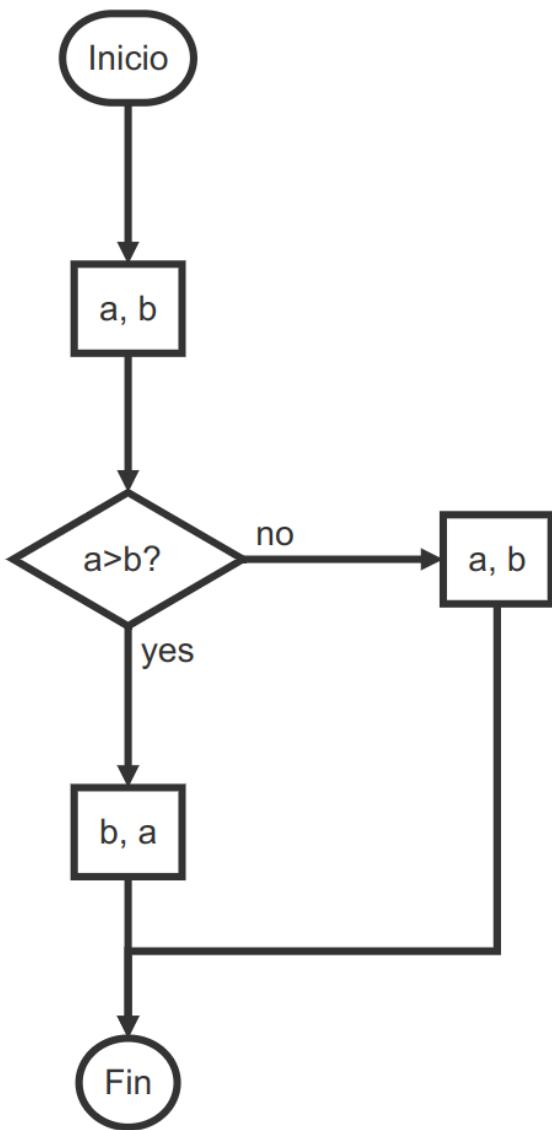
Los símbolos más utilizados son:

Símbolos diagrama de flujo

Ejemplo: Mostrar dos números ordenados de menor a mayor.

```
graph TD
    A[Inicio] --> B[a, b]
    A --> C{a > b ?}
    C -->|Si| D[b, a]
    C -->|No| E[a, b]
    D --> F(Fin)
    E --> F
```

O también en otra representación:



El **pseudocódigo** es un lenguaje de descripción de algoritmos que está muy próximo a la sintaxis que utilizan los lenguajes de programación. Nace como medio para representar las estructuras de control de programación estructurada.

El pseudocódigo no se puede ejecutar nunca en el ordenador, sino que tiene que traducirse a un lenguaje de programación (codificación). La ventaja del pseudocódigo, frente a los diagramas de flujo, es que se puede modificar más fácilmente si detecta un error en la lógica del algoritmo, y puede ser traducido fácilmente a los lenguajes estructurados como Pascal, C, fortran, Java, etc.

El Pseudocódigo utiliza palabras reservadas (en sus orígenes se escribían en inglés) para representar las sucesivas acciones. Para mayor legibilidad utiliza la **indentación** (sangría en el margen izquierdo) de sus líneas.

Ejemplo: Mostrar dos números ordenados de menor a mayor.

```

1 Begin
2   Leer (A, B)
3   If (A>B) then
4     Escribir (B, A)
5   Else
6     Escribir (A, B)
7   End If
8 End
  
```

3.2.3. Programas

La **diferencia** fundamental entre **algoritmo** y **programa** es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado lenguaje de programación para que puedan ser ejecutados en el ordenador y así obtener la solución.

Los lenguajes de programación son sólo un medio para expresar el algoritmo y el ordenador un procesador para ejecutarlo. El diseño de los algoritmos será una tarea que necesitará de la creatividad y conocimientos de las técnicas de programación. Estilos distintos, de distintos programadores a la hora de obtener la solución del problema, darán lugar a programas diferentes, igualmente válidos.

Pero cuando los problemas son complejos, es necesario descomponer éstos en subproblemas más simples y, a su vez, en otros más pequeños. Estas estrategias reciben el nombre de diseño descendente (Metodología de diseño de programas, consistente en la descomposición del problema en problemas más sencillos de resolver) o diseño modular (top-down design) (Metodología de diseño de programas, que consiste en dividir la solución a un problema en módulos más pequeños o subprogramas. Las soluciones de los módulos se unirán para obtener la solución general del problema). Este sistema se basa en el lema **divide y vencerás**.

3.3. Java

3.3.1. ¿Qué y cómo es Java?

Java es un lenguaje sencillo de aprender, con una sintaxis parecida a la de C++, pero en la que se han eliminado elementos complicados y que pueden originar errores. Java es orientado a objetos, con lo que elimina muchas preocupaciones al programador y permite la utilización de gran cantidad de bibliotecas ya definidas, evitando reescribir código que ya existe. Es un lenguaje de programación creado para satisfacer nuevas necesidades que los lenguajes existentes hasta el momento no eran capaces de solventar.

Una de las principales virtudes de Java es su independencia del hardware, ya que el código que se genera es válido para cualquier plataforma. Este código será ejecutado sobre una máquina virtual denominada Maquina Virtual Java (MVJ o JVM - Java Virtual Machine), que interpretará el código convirtiéndolo a código específico de la plataforma que lo soporta. De este modo el programa se escribe una única vez y puede hacerse funcionar en cualquier lugar. Lema del lenguaje: "*Write once, run everywhere*".

Antes de que apareciera Java, el lenguaje C era uno de los más extendidos por su versatilidad. Pero cuando los programas escritos en C aumentaban de volumen, su manejo comenzaba a complicarse. Mediante las técnicas de programación estructurada y programación modular se conseguían reducir estas complicaciones, pero no era suficiente.

Fue entonces cuando la Programación Orientada a Objetos (POO) entra en escena, aproximando notablemente la construcción de programas al pensamiento humano y haciendo más sencillo todo el proceso. Los problemas se dividen en objetos que tienen propiedades e interactúan con otros objetos, de este modo, el programador puede centrarse en cada objeto para programar internamente los elementos y funciones que lo componen.

Las características principales de lenguaje Java se resumen a continuación:

- El código generado por el compilador Java es independiente de la arquitectura.
- Está totalmente orientado a objetos.
- Su sintaxis es similar a C y C++.
- Es distribuido, preparado para aplicaciones TCP/IP.
- Dispone de un amplio conjunto de bibliotecas.
- Es robusto, realizando comprobaciones del código en tiempo de compilación y de ejecución.
- La seguridad está garantizada, ya que las aplicaciones Java no acceden a zonas delicadas de memoria o de sistema. (*ejem, ejem!*)

3.3.2. Breve historia.

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a programar pequeños dispositivos electrónicos. La dificultad de estos dispositivos es que cambian continuamente y para que un programa funcione en el siguiente dispositivo aparecido, hay que reescribir el código. Por eso la empresa Sun quería crear un lenguaje independiente del dispositivo.

Pero no fue hasta 1995 cuando pasó a llamarse Java, dándose a conocer al público como lenguaje de programación para computadores. Java pasa a ser un lenguaje totalmente independiente de la plataforma y a la vez potente y orientado a objetos. Esta filosofía y su facilidad para crear aplicaciones para redes TCP/IP ha hecho que sea uno de los lenguajes más utilizados en la actualidad.

El factor determinante para su expansión fue la incorporación de un intérprete Java en la versión 2.0 del navegador Web Netscape Navigator, lo que supuso una gran revuelo en Internet. A principios de 1997 apareció Java 1.1 que proporcionó sustanciales mejoras al lenguaje. Java 1.2, más tarde rebautizado como Java 2, nació a finales de 1998.

El principal objetivo del lenguaje Java es llegar a ser el nexo universal que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor Web, en una base de datos o en cualquier otro lugar.

Para el desarrollo de programas en lenguaje Java es necesario utilizar un entorno de desarrollo denominado JDK (Java Development Kit), que provee de un compilador y un entorno de ejecución (JRE - Java RunEnvironment) para los bytecodes generados a partir del código fuente. Al igual que las diferentes versiones del lenguaje han incorporado mejoras, el entorno de desarrollo y ejecución también ha sido mejorado sucesivamente.

Java 2 es la tercera versión del lenguaje, pero es algo más que un lenguaje de programación, incluye los siguientes elementos:

- Un lenguaje de programación: Java.
- Un conjunto de bibliotecas estándar que vienen incluidas en la plataforma y que son necesarias en todo entorno Java. Es el Java Core.
- Un conjunto de herramientas para el desarrollo de programas, como es el compilador de bytecodes, el generador de documentación, un depurador, etc.
- Un entorno de ejecución que en definitiva es una máquina virtual que ejecuta los programas traducidos a bytecodes.

3.3.3. Compilar y ejecutar un programa Java . Uso de la consola.

Veamos los pasos para compilar e interpretar nuestro primer programa escrito en lenguaje Java.

3.3.3.1. ESTRUCTURA Y BLOQUES FUNDAMENTALES DE UN PROGRAMA.

Ejemplo Holamundo.java

```

1  public class Holamundo {
2      // programa Hola Mundo
3      public static void main(String[] args) {
4          /* lo único que hace este programa es mostrar
5             la cadena "Hola Mundo!" por pantalla */
6          System.out.println("Hola Mundo!");
7      }
8 }
```

En Java generalmente una clase lleva el identificador `public` y corresponde con un fichero. El nombre de la clase coincide con el del fichero `.java` respetando mayúsculas y minúsculas.

```

1  public class Holamundo {
2      [...]
3 }
```

El código java en las clases se agrupa en funciones o métodos. Cuando java ejecuta el código de una clase busca la función o método `main()` para ejecutarla. Es público (`public`) estático (`static`) para llamarlo sin instanciar la clase. No devuelve ningún valor (`void`) y admite parámetros (`String[] args`) que en este caso no se han utilizado.

```

1  [...]
2  public static void main (String[] args)
3  {
4      [...]
5  }
6  [...]
```

El código de la función `main` se escribe entre las llaves. Por ejemplo:

```

1  [...]
2      System.out.println("Hola Mundo");
3  [...]
```

Muestra por pantalla el mensaje `Hola Mundo`, ya que la clase `System` tiene un atributo `out` con dos métodos: `print()` y `println()`. La diferencia es que `println` muestra mensaje e introduce un retorno de carro.

Todas las instrucciones menos las llaves `{ }` terminan con punto y coma (`;`).

3.3.3.2. SANGRADO O TABULADO

El sangrado (también conocido como tabulado) deberá aplicarse a toda estructura que esté lógicamente contenida dentro de otra. El sangrado será de un tabulador. **Es suficiente entre 2 y 4 espacios.** Para alguien que empieza a programar suele ser preferible unos 4 espacios, ya que se ve todo más claro.

Las líneas no tendrán en ningún caso demasiados caracteres que impidan que se pueda leer en una pantalla. **Un número máximo recomendable suele estar entre unos 70 y 90 caracteres, incluyendo los espacios de sangrado.** Si una línea debe ocupar más caracteres, tiene que dividirse en dos o más líneas, para ello utiliza los siguientes principios para realizar la división:

- Tras una coma.
- Antes de un operador, que pasará a la línea siguiente.
- Una construcción de alto nivel (por ejemplo, una expresión con paréntesis).
- La nueva línea deberá alinearse con un sangrado lógico, respecto al punto de ruptura

Unos pocos ejemplos, para comprender mejor:

Dividir tras una coma:

```
1  funcion(expresionMuuuyLarga1,
2          expresionMuuuyyyyLarga2,
3          expresionMuuuyyyyLarga3);
```

Mantener la expresión entre paréntesis en la misma línea:

```
1  nombreLargo = nombreLargo2*
2      (nombreLargo3 + nombreLArgo4) +
3      4*nombreLargo5;
```

Siempre hay excepciones. Puede resultar que al aplicar estas reglas, en operaciones muy largas, o expresiones lógicas enormes, el sangrado sea ilegible. En estos casos, el convenio se puede relajar.

3.3.3.3. PASO 1: CREACIÓN DEL CÓDIGO FUENTE

Abrimos un editor de texto (da igual cual sea, siempre que sea capaz de almacenar "texto sin formato" en código ASCII). Una vez abierto escribiremos nuestro primer programa, que mostrará un texto "Hola Mundo" en la consola. De momento no te preocupes si no entiendes lo que escribes, más adelante le daremos sentido. Ahora solo queremos ver si podemos ejecutar java en nuestro equipo.

El código de nuestro programa en Java será el siguiente:

```
1  /* Ejemplo Hola Mundo */
2  public class Ejemplo {
3      public static void main(String[ ] args) {
4          System.out.println("Hola Mundo");
5      }
6  }
```

A continuación guardamos nuestro archivo y le ponemos como nombre `Ejemplo.java`. Debemos seguir una norma dictada por Java, hemos de hacer coincidir nombre del archivo y nombre del programa, tanto en mayúsculas como en minúsculas, y la extensión del archivo habrá de ser siempre `.java`.

`Ejemplo.java`

Debemos recordar exactamente la ruta donde guardamos el archivo de ejemplo `Ejemplo.java`.

3.3.3.4. PASO 2: COMPILACIÓN DEL PROGRAMA

Vamos a proceder a compilar e interpretar este pequeño programa Java (no te preocupes si todavía no entiendes el significado de las palabras compilar e interpretar, lo verás en la asignatura de `Entornos de Desarrollo`). Para ello usaremos la consola. Una vez en la consola debemos colocarnos en la ruta donde previamente guardamos el archivo `Ejemplo.java`.

A continuación daremos la instrucción para que se realice **el proceso de compilación del programa**, para lo que escribiremos `javac Ejemplo.java`, donde `javac` es el nombre del compilador (`java compiler`) que transformará el programa que hemos escrito nosotros en lenguaje Java al lenguaje de la máquina virtual Java (`bytecode`), dando como resultado un nuevo archivo `Ejemplo.class` que se creará en este mismo directorio. Comprueba que no aparezca ningún error y que `javac` esté instalado en tu sistema (desde la consola lo puedes comprobar con el comando `javac --version` y debería aparecer el número de versión que tienes instalada). Si aparecen los dos archivos tanto `Ejemplo.java` (código fuente) como `Ejemplo.class` (bytecode creado por el compilador) puedes continuar.

```
1 $ javac Ejemplo.java
```

3.3.3.5. PASO 3: EJECUCIÓN DEL PROGRAMA

Finalmente, vamos a pedirle al intérprete (JVM) que ejecute el programa, es decir, que transforme el código de la máquina virtual Java en código máquina interpretable por nuestro ordenador y lo ejecute. Para ello escribiremos en la ventana consola: `java Ejemplo`.

El resultado será que se nos muestra la cadena `Hola Mundo`. Si logramos visualizar este texto en pantalla, ya hemos desarrollado nuestro primer programa en Java.

```
1 $ java Ejemplo
2 Hola Mundo
```

Por qué no necesito compilar mi archivo `.java` antes de ejecutarlo y funciona directamente si me salto ese paso?

<https://stackoverflow.com/questions/54493058/running-a-java-program-without-compiling>

3.4. Componentes del lenguaje Java

3.4.1. Variables, identificadores, convenciones.

3.4.1.1. VARIABLES

Una **variable** es una zona en la memoria del computador con un valor que puede ser almacenado para ser usado más tarde en el programa. Las variables vienen determinadas por:

- un **nombre**, que permite al programa acceder al valor que contiene en memoria. Debe ser un identificador válido.
- un **tipo de dato**, que especifica qué clase de información guarda la variable en esa zona de memoria
- un **rango de valores** que puede admitir dicha variable.

Las variables declaradas dentro de un bloque `{ }` son accesibles solo dentro de ese bloque. Una variable local no puede ser declarada como `static`. Una variable no puede declararse fuera de la clase.

Visibilidad, ámbito o scope de una variable es la parte de código del programa donde la variable es accesible y utilizable. Las variables de un bloque son visibles y existen dentro de dicho bloque. Las funciones miembro de clase podrán acceder a todas las variables miembro de dicha clase pero no a las variables locales de otra función miembro.

Al nombre que le damos a la variable se le llama identificador. Los identificadores permiten nombrar los elementos que se están manejando en un programa. Vamos a ver con más detalle ciertos aspectos sobre los identificadores que debemos tener en cuenta.

3.4.1.2. IDENTIFICADORES

Un **identificador** en Java es una secuencia ilimitada sin espacios de letras y dígitos Unicode , de forma que el primer símbolo de la secuencia debe ser una letra, un símbolo de subrayado (`_`) o el símbolo dólar (`$`). Por ejemplo, son válidos los siguientes identificadores:

- `x5`
- `ατη`
- `NUM_MAX`
- `numCuenta`

Unicode es un código de caracteres o sistema de codificación, un alfabeto que recoge los caracteres de prácticamente todos los idiomas importantes del mundo. Además, el código Unicode es “compatible” con el código ASCII, ya que para los caracteres del código ASCII, Unicode asigna como código los mismos 8 bits, a los que les añade a la izquierda otros 8 bits todos a cero. La conversión de un carácter ASCII a Unicode es inmediata.

3.4.1.3. CONVENCIONES

Normas de estilo para nombrar variables

A la hora de nombrar un identificador existen una serie de normas de uso generalizado que, no siendo obligatorias, se usan en la mayor parte del código Java. Estas reglas para la nomenclatura de variables son las siguientes:

- Java distingue las mayúsculas de las minúsculas. Por ejemplo, `Alumno` y `alumno` son variables diferentes.
- No se suelen utilizar identificadores que comiencen con `$` o `_`, además el símbolo del dólar, por convenio, no se utiliza nunca.

- No se puede utilizar el valor booleano (`true` o `false`) ni el valor nulo (`null`).
- Los identificadores deben ser lo más descriptivos posibles. Es mejor usar palabras completas en vez de abreviaturas crípticas. Así nuestro código será más fácil de leer y comprender. En muchos casos también hará que nuestro código se auto-documente. Por ejemplo, si tenemos que darle el nombre a una variable que almacena los datos de un cliente sería recomendable que la misma se llamara algo así como `FicheroClientes` o `ManejadorCliente`, y no algo poco descriptivo como `c133`.

Además de estas restricciones, en la siguiente tabla puedes ver otras convenciones, que no siendo obligatorias, sí son recomendables a la hora de crear identificadores en Java.

Identificador	Convención	Ejemplo
nombre de variable	Comienza por letra minúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas. A esto se le llama <i>lowerCamelCase</i> .	numAlumnos, suma
nombre de constante	En letras mayúsculas, separando las palabras con el guión bajo, por convenio el guión bajo no se utiliza en ningún otro sitio	TAM_MAX, PI
nombre de una clase	Comienza por letra mayúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas. A esto se le llama <i>UpperCamelCase</i> .	String, MiTipo
nombre de función	Comienza por letra minúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas. A esto se le llama <i>lowerCamelCase</i> .	modificaValor, obtieneValor

Puedes consultar estas y otras convenciones sobre código Java en este [enlace](#).

Palabras reservadas Las palabras reservadas, a veces también llamadas palabras clave o keywords, son secuencias de caracteres formadas con letras ASCII cuyo uso se reserva al lenguaje y, por tanto, no pueden utilizarse para crear identificadores.

Las palabras reservadas en Java son:

```
1 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.
```

3.4.2. Tipos de datos.

Los tipos de datos se utilizan para declarar variables y el compilador sepa de antemano que tipo de información contendrá la variable.

Java dispone de los siguientes tipos de datos simples:

Tipo de dato	Representación	Tamaño (Bytes)	Rango de Valores	Valor por defecto	Clase Asociada
byte	Numérico Entero con signo	1	-128 a 127	0	Byte
short	Numérico Entero con signo	2	-32768 a 32767	0	Short
int	Numérico Entero con signo	4	-2147483648 a 2147483647	0	Integer
long	Numérico Entero con signo	8	-9223372036854775808 a 9223372036854775807	0	Long
float	Numérico en Coma flotante de precisión simple Norma IEEE 754	4	-3.4x10 ⁻³⁸ a 3.4x10 ³⁸	0.0	Float
double	Numérico en Coma flotante de	8	-1.8x10 ⁻³⁰⁸ a 1.8x10 ³⁰⁸	0.0	Double

Tipo de dato	Representación	Tamaño (Bytes)	Rango de Valores	Valor por defecto	Clase Asociada
	precisión doble Norma IEEE 754				
char	Carácter Unicode	2	\u0000 a \uFFFF	\u0000	Character
boolean	Dato lógico	-	true ó false	false	Boolean
void	-	-	-	-	Void

Sobre valores por defecto y inicialización de variables: <https://stackoverflow.com/questions/19131336/default-values-and-initialization-in-java>

Ejemplo de declaración y asignación de valores a variables:

Tipo de datos	código
byte	byte a;
short	short b, c=3;
int	int d=-30; int e=0xC125; //la 0x significa Hexadecimal
long	long b=46240; long b=5L; // La L en este caso indica Long
char	char car1='c'; char car2=99; //car1 y car2 son iguales, la c equivale al ascii 99 char letra = '\u0061'; //código unicode del carácter "a"
float	float pi=3.1416; float pi=3.1416F; //La F significa float float medio=1/2; //0.5
double	double millon=1e6; // 1x10^6 double medio=1/2D; //0.5, la D significa double double z=.123; //si la parte entera es 0 se puede omitir
boolean	boolean esPrimero; boolean esPar=false;

> Ojo con los tipo float: <https://jvns.ca/blog/2023/01/13/examples-of-floating-point-problems/>

3.4.3. Tipos referenciados

A partir de los ocho tipos datos primitivos, se pueden construir otros tipos de datos. Estos tipos de datos se llaman tipos referenciados o referencias, porque se utilizan para almacenar la dirección de los datos en la memoria del ordenador.

```
1 int[] arrayDeEnteros;
2 Cuenta cuentaCliente;
```

En la primera instrucción declaramos una lista de números del mismo tipo, en este caso, enteros. En la segunda instrucción estamos declarando la variable u objeto `cuentaCliente` como una referencia de tipo `Cuenta`.

Cualquier aplicación de hoy en día necesita no perder de vista una cierta cantidad de datos. Cuando el conjunto de datos utilizado tiene características similares se suelen agrupar en estructuras para facilitar el acceso a los mismos, son los llamados datos estructurados.

Son datos estructurados los `arrays`, `listas`, `árboles`, etc. Pueden estar en la memoria del programa en ejecución, guardados en el disco como ficheros, o almacenados en una base de datos.

Además de los ocho tipos de datos primitivos que ya hemos descrito, Java proporciona un tratamiento especial a los textos o cadenas de caracteres mediante el tipo de dato `String`. Java crea automáticamente un nuevo objeto de tipo `String` cuando se encuentra una cadena de caracteres encerrada entre comillas dobles. En realidad se trata de objetos, y por tanto son tipos referenciados, pero se pueden utilizar de forma sencilla como si fueran variables de tipos primitivos:

```
1 String mensaje;
2 mensaje= "El primer programa";
```

Hemos visto qué son las variables, cómo se declaran y los tipos de datos que pueden adoptar. Anteriormente hemos visto un ejemplo de creación de variables, en esta ocasión vamos a crear más variables, pero de distintos tipos primitivos y los vamos a mostrar por pantalla. Los tipos referenciados los veremos en la siguiente unidad.

Para mostrar por pantalla un mensaje utilizamos `System.out`, conocido como la salida estándar del programa. Este método lo que hace es escribir un conjunto de caracteres a través de la línea de comandos. Podemos utilizar `System.out.print` o `System.out.println`. En el segundo caso lo que hace el método es que justo después de escribir el mensaje, sitúa el cursor al principio de la línea siguiente.

El texto en color gris que aparece entre caracteres // son comentarios que permiten documentar el código, pero no son tenidos en cuenta por el compilador y, por tanto, no afectan a la ejecución del programa.

3.4.4. Tipos enumerados

Los tipos de datos enumerados son una forma de declarar una variable con un conjunto restringido de valores. Por ejemplo, los días de la semana, las estaciones del año, los meses, etc. Es como si definiéramos nuestro propio tipo de datos.

La forma de declararlos es con la palabra reservada `enum`, seguida del nombre de la variable y la lista de valores que puede tomar entre llaves. A los valores que se colocan dentro de las llaves se les considera como constantes, van separados por comas y deben ser valores únicos.

La lista de valores se coloca entre llaves, porque un tipo de datos `enum` no es otra cosa que una especie de clase en Java, y todas las clases llevan su contenido entre llaves.

Al considerar Java este tipo de datos como si de una clase se tratara, no sólo podemos definir los valores de un tipo enumerado, sino que también podemos definir operaciones a realizar con él y otro tipo de elementos, lo que hace que este tipo de dato sea más versátil y potente que en otros lenguajes de programación.

En el siguiente ejemplo puedes comprobar el uso que se hace de los tipos de datos enumerados.

```
1 public class tiposEnumerados {
2     public enum dias {Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo};
3
4     public static void main(String[] args) {
5         dias diaActual = dias.Martes;
6         dias diaSiguiente = dias.Miércoles;
7
8         System.out.print("Hoy es: ");
9         System.out.println(diaActual);
10        System.out.println("Mañana\nes\n"+diaSiguiente);
11    }
12 }
```

El resultado después de la ejecución será:

```
1 Hoy es: Martes
2 Mañana
3 es
4 Miércoles
```

Tenemos una variable `Dias` que almacena los días de la semana. Para acceder a cada elemento del tipo enumerado se utiliza el nombre de la variable seguido de un punto y el valor en la lista. Más tarde veremos que podemos añadir métodos y campos o variables en la declaración del tipo enumerado, ya que como hemos comentado un tipo enumerado en Java tiene el mismo tratamiento que las clases.

En este ejemplo hemos utilizado el método `System.out.print`. Como podrás comprobar si lo ejecutas, la instrucción `print` escribe el texto que tiene entre comillas pero no salta a la siguiente línea, por lo que la instrucción `println` escribe justo a continuación.

Sin embargo, también podemos escribir varias líneas usando una única sentencia. Así lo hacemos en la instrucción `println`, la cual imprime como resultado tres líneas de texto. Para ello hemos utilizado un carácter especial, llamado carácter escape (\n). Este carácter sirve para darle ciertas órdenes al compilador, en lugar de que salga impreso en pantalla. Después del carácter de escape viene otro carácter que indica la orden a realizar, juntos reciben el nombre de secuencia de escape. La secuencia de

escape `\n` recibe el nombre de carácter de nueva línea. Cada vez que el compilador se encuentra en un texto ese carácter, el resultado es que mueve el cursor al principio de la línea siguiente. En el próximo apartado vamos a ver algunas de las secuencias de escape más utilizadas.

3.4.5. Constantes y literales.

Las **constantes** se utilizan para almacenar datos que no varían nunca, asegurándonos que el valor no va a poder ser modificado.

Podemos declarar una constante utilizando:

```
1 final <tipo de datos> <nombre de la constante> = <valor>;
```

El calificador final indica que es constante. A continuación indicaremos el tipo de dato, el nombre de la constante y el valor que se le asigna.

```
1 final double IVA= 0.21;
```

Los **literales** pueden ser de tipo simple, null o string, como por ejemplo 230, null o "Java".

Respecto a los literales existen unos caracteres especiales que se representan utilizando secuencias de escape:

Secuencia de escape	Significado	Secuencia de escape	Significado
<code>\b</code>	Retroceso	<code>\r</code>	Retorno de carro
<code>\t</code>	Tabulador	<code>\"</code>	Carácter comillas dobles
<code>\n</code>	Salto de línea	<code>\'</code>	Carácter comillas simples
<code>\f</code>	Salto de página	<code>\ </code>	Barra diagonal

3.4.6. Operadores y expresiones.

3.4.6.1. OPERADORES ARITMÉTICOS

Los **Operadores Aritméticos** permiten realizar operaciones matemáticas:

Operador	Uso	Operación
<code>+</code>	<code>A + B</code>	Suma
<code>-</code>	<code>A - B</code>	Resta
<code>*</code>	<code>A * B</code>	Multiplicación
<code>/</code>	<code>A / B</code>	División
<code>%</code>	<code>A % B</code>	Módulo o resto de una división entera

Ejemplo:

```
1 double num1, num2, suma, resta, producto, division, resto;
2 num1 =8;
3 num2 =5;
4 suma = num1 + num2;      // 13
5 resta = num1 - num2;     // 3
6 producto = num1 * num2;  // 40
7 division = num1 / num2;  // 1.6
8 resto = num1 % num2;    // 3
```

3.4.6.2. OPERADORES RELACIONALES

Los **Operadores Relacionales** permiten evaluar (la respuesta es un booleano: si o no) la igualdad de los operandos:

Operador	Uso	Operación
<code><</code>	<code>a < b</code>	a menor que b
<code>></code>	<code>a > b</code>	a mayor que b

Operador	Uso	Operación
<code><=</code>	<code>a <= b</code>	a menor o igual que b
<code>>=</code>	<code>a >= b</code>	a mayor o igual que b
<code>!=</code>	<code>a != b</code>	a distinto de b
<code>==</code>	<code>a == b</code>	a igual a b

Por ejemplo:

```

1 int valor1 = 10;
2 int valor2 = 3;
3 boolean compara;
4 compara = valor1 > valor2; // true
5 compara = valor1 < valor2; // false
6 compara = valor1 >= valor2; // true
7 compara = valor1 <= valor2; // false
8 compara = valor1 == valor2; // false
9 compara = valor1 != valor2; // true

```

3.4.6.3. OPERADORES LÓGICOS

Los **Operadores Lógicos** permiten realizar operaciones lógicas:

Operador	Uso	Operación
<code>&& o &</code>	<code>a&&b o a&b</code>	a AND b. El resultado será <i>true</i> si ambos operadores son <i>true</i> y <i>false</i> en caso contrario.
<code> o </code>	<code>a b o a b</code>	a OR b. El resultado será <i>false</i> si ambos operandos son <i>false</i> y <i>true</i> en caso contrario
<code>!</code>	<code>!a</code>	NOT a. Si el operando es <i>true</i> el resultado es <i>false</i> y si el operando es <i>false</i> el resultado es <i>true</i> .
<code>^</code>	<code>a^b</code>	a XOR b. El resultado será <i>true</i> si un operando es <i>true</i> y el otro <i>false</i> , y <i>false</i> en caso contrario.

Ejemplo:

```

1 double sueldo = 1400;
2 int edad = 34;
3 boolean logica;
4 logica = (sueldo>1000 & edad<40); //true
5 logica = (sueldo>1000 && edad >40); //false
6 logica = (sueldo>1000 | edad>40); //true
7 logica = (sueldo<1000 || edad >40); //false
8 logica = !(edad <40); //true
9 logica = (sueldo>1000 ^ edad>40); //true
10 logica = (sueldo<1000 ^ edad>40); //false

```

Para representar resultados de operadores Lógicos también se pueden usar tablas de verdad a las que conviene acostumbrarse:

a	b	<code>a && b</code>	<code>a b</code>	<code>!a</code>	<code>a^b</code>
false	false	false	false	true	false
true	false	false	true	false	true
false	true	false	true	true	true
true	true	true	true	false	false

3.4.6.4. OPERADORES UNARIOS O UNITARIOS

Los **Operadores Unarios o Unitarios** permiten realizar incrementos y decrementos:

Operador	Uso	Operación
<code>++</code>	<code>a++ o ++a</code>	Incremento de a

Operador	Uso	Operación
--	a-- o --a	Decremento de a

Ejemplo:

```
1 int m = 5, n = 3;
2 m++; // 6
3 n--; // 2
```

En el caso de utilizarlo como prefijo el valor de asignación será el valor del operando más el incremento de la unidad. Y si lo utilizamos como sufijo se asignará el valor del operador y luego se incrementará la unidad sobre el operando.

```
1 int a = 1, b;
2 b = ++a; // a vale 2 y b vale 2 //coge lo que vale a, le suma 1 y lo guarda en b
3 b = a++; // a vale 3 y b vale 2 //coge lo que vale a, lo guarda en b, y suma 1 a lo que vale a
```

3.4.6.5. OPERADORES DE ASIGNACIÓN

Los **Operadores de Asignación** permiten asignar valores:

Operador	Uso	Operación
=	a = b	Asignación (como ya hemos visto)
*=	a *= b	Multiplicación y asignación. La operación a*=b equivale a a=a*b
/=	a /= b	División y asignación. La operación a/=b equivale a a=a/b
%=	a %= b	Módulo y asignación. La operación a%=b equivale a a=a%b
+=	a += b	Suma y asignación. La operación a+=b equivale a a=a+b
-=	a -= b	Resta y asignación. La operación a-=b equivale a a=a-b

Ejemplo:

```
1 int dato1 = 10, dato2 = 2, dato;
2 dato=dato1; // dato vale 10
3 dato2*=dato1; // dato2 vale 20
4 dato2/=dato1; // dato2 vale 2
5 dato2+=dato1; // dato2 vale 12
6 dato2-=dato1; // dato2 vale 2
7 dato1%=dato2; // dato1 vale0
```

3.4.6.6. OPERADORES DE DESPLAZAMIENTO

Los **Operadores de desplazamiento** permiten desplazar los bits de los valores:

Operador	Utilización	Resultado
<<	a << b	Desplazamiento de a a la izquierda en b posiciones. Multiplica por 2 el número b de veces.
>>	a >> b	Desplazamiento de a a la derecha en b posiciones, tiene en cuenta el signo. Divide por 2 el número b de veces.
>>>	a >>> b	Desplazamiento de a a la derecha en b posiciones, no tiene en cuenta el signo. (simplemente agrega ceros por la izquierda)
&	a & b	Operación AND a nivel de bits
	a b	Operación OR a nivel de bits
^	a^b	Operación XOR a nivel de bits
~	~a	Complemento de A a nivel de bits

Por ejemplo:

```

1 int j = 33;
2 int k = j << 2;
3 // 0000000000000000000000000000000100001 : j = 33
4 // 000000000000000000000000000000010000100 : k = 33 << 2 ; k = 132
5
6 int o = 132;
7 int p = o >> 2;
8 // 000000000000000000000000000000010000100 : o = 132
9 // 0000000000000000000000000000000100001 : p = 132 >> 2 ; p = 33
10
11 int x = -1;
12 int y = x >>> 2;
13 // 1111111111111111111111111111111111111111111111111111111111111111 : x = -1
14 // 0011111111111111111111111111111111111111111111111111111111111111 : y = x >>> 2; y = 1073741823
15
16 int q = 132; // q: 000000000000000000000000000000010000100
17 int r = 144; // r: 000000000000000000000000000000010010000
18
19 int s = q & r; // s: 000000000000000000000000000000010000000
20 // El resultado da 128
21
22 int t = q | r; // t: 000000000000000000000000000000010010100
23 // El resultado da 148
24
25 int u = q ^ r; // u: 000000000000000000000000000000010100
26 // El resultado da 20
27
28 int v = ~q; // v: 1111111111111111111111111111111101111011
29 // El resultado da -133

```

3.4.6.7. OPERADOR CONDICIONAL O TERNARIO ?:

El **operador condicional** ?: sirve para evaluar una condición y devolver un resultado en función de si es verdadera o falsa dicha condición. Es el único operador ternario de Java, y como tal, necesita tres operandos para formar una expresión.

El primer operando se sitúa a la izquierda del símbolo de interrogación, y siempre será una expresión booleana, también llamada **condición**. El siguiente operando se sitúa a la derecha del símbolo de interrogación y antes de los dos puntos, y es el **valor** que devolverá el operador condicional **si la condición es verdadera**. El último operando, que aparece después de los dos puntos, es la expresión cuyo **resultado se devolverá si la condición evaluada es falsa**.

```
1 condición ? exp1 : exp2
```

Por ejemplo, en la expresión:

```
1 (x>y)?x:y;
```

Se evalúa la condición de si **x es mayor que y**, en caso **afirmativo** se devuelve el valor de la variable **x**, y **en caso contrario** se devuelve el valor de **y**.

Ejemplo para calcular qué número es mayor:

```

1 int mayor, exp1 = 15, exp2 = 25;
2 mayor=(exp1>exp2)?exp1:exp2;
3 // mayor valdrá 25

```

El operador condicional se puede sustituir por la sentencia `if...then...else` que veremos más adelante.

3.4.6.8. PREVALENCIA DE OPERADORES

Los operadores tienen diferente **Prioridad** por lo que es interesante utilizar paréntesis para controlar las operaciones sin necesidad de depender de la prioridad de los operadores.

Prevalencia de operadores, ordenados de arriba a abajo de más a menos prioridad:

Descripción	Operadores
operadores posfijos	op++ op--
operadores unarios	++op --op +op -op ~ !
multiplicación y división	* / %
suma y resta	+ -
desplazamiento	<< >> >>>

Descripción	Operadores
operadores relacionales	< > <= =>
equivalencia	== !=
operador AND	&
operador XOR	^
operador OR	
AND booleano	&&
OR booleano	
condicional	? :
operadores de asignación	= += -= *= /= %= &= ^= \ = <=>= >>=

Por ejemplo:

```
1 int x, y1 = 6, y2 = 2, y3 =8;
2 x = y1 + y2 * y3; // 22
3 x = (y1 + y2) * y3; // 64
```

"Los paréntesis son como las patatas fritas, cuantas más, mejor!" (Ana de mates)

3.4.7. Conversiones de tipo.

Existen dos tipos de conversiones: **Implícitas** y **Explicitas**. Debemos evitar las conversiones de tipos ya que pueden suponer perdidas de información.

3.4.7.1. CONVERSIONES IMPLÍCITAS

Las **Conversiones Implícitas** se realizan de forma automática y requiere que la variable destino tenga más precisión que la variable origen para poder almacenar el valor.

Ejemplo:

```
1 // Conversión Implicita
2 byte origen = 5;
3 short destino;
4 destino=origen; // 5
```

3.4.7.2. CONVERSIÓN EXPLÍCITA

En la **Conversión Explícita** el programador fuerza la conversión con la operación llamada "**cast**":

Ejemplo:

```
1 // Conversión Explicita
2 short origen2 = 3;
3 byte destino2;
4 destino2=(byte)origen2; // 3
```

3.4.8. Comentarios.

Los comentarios son muy importantes a la hora de describir qué hace un determinado programa. A lo largo de la unidad los hemos utilizado para documentar los ejemplos y mejorar la comprensión del código. Para lograr ese objetivo, es normal que cada programa comience con unas líneas de comentario que indiquen, al menos, una breve descripción del programa, el autor del mismo y la última fecha en que se ha modificado.

Todos los lenguajes de programación disponen de alguna forma de introducir comentarios en el código. En el caso de Java, nos podemos encontrar los siguientes tipos de comentarios:

- Comentarios de **una sola línea**. Utilizaremos el delimitador // para introducir comentarios de sólo una línea.

```
1 // comentario de una sola línea  
2 byte estoEsUnByte=1;
```

- Comentarios de **múltiples líneas**. Para introducir este tipo de comentarios, utilizaremos una barra inclinada y un asterisco (`/*`), al principio del párrafo y un asterisco seguido de una barra inclinada (`*/`) al final del mismo.

```
1 /* Esto es un  
2 comentario  
3 de varias líneas */
```

- Comentarios **Javadoc**. Utilizaremos los delimitadores `/**` y `*/`. Al igual que con los comentarios tradicionales, el texto entre estos delimitadores será ignorado por el compilador. Este tipo de comentarios se emplean para generar documentación automática del programa. A través del programa javadoc, incluido en JavaSE, se recogen todos estos comentarios y se llevan a un documento en formato `.html`.

```
1  /** Comentario de documentación.  
2   Javadoc extrae los comentarios del código y  
3   genera un archivo html a partir de este tipo de comentarios  
4  */
```

3.5. Herramientas útiles para empezar

3.5.1. Generar números aleatorios.

Podemos generar números aleatorios entre 0 y 1 utilizando el método `random` de la clase `Math`.

```
1   Math.random()
```

3.5.2. Introducir un texto desde el teclado.

Este método de leer texto y números desde consola no nos servirá cuando comencemos a usar IDE's.

Podemos introducir texto desde el teclado utilizando `System.console().readLine()`:

Ejemplo 1: Introducción de texto.

```
1 String texto;
2 System.out.print("Introduce un texto: ");
3 texto = System.console().readLine();
4 System.out.println("El texto introducido es: "+ texto);
```

Ejemplo 2: Introducción de un número entero.

```
1 String texto2;
2 int entero2;
3 System.out.print("Introduce un número: ");
4 texto2 = System.console().readLine();
5 entero2 = Integer.parseInt(texto2); //convertimos texto a Integer
6 System.out.println("El número introducido es:"+entero2);
```

Ejemplo 3: Introducción de un número decimal.

```
1 String texto3;
2 double doble3;
3 System.out.print("Introduce un número decimal: ");
4 texto3 = System.console().readLine();
5 doble3 = Double.parseDouble(texto3); // convertimos texto a Double
6 System.out.println("Número decimal introducido es: "+doble3);
```

3.6. Ejemplo UD01

[EjemploUD01.java](#)

3.7. Píldoras informáticas relacionadas

⌚8 de enero de 2026

4. 2.2 Ejercicios de la UD01

4.1. Retos

1. (Reto1) Haga un programa que evalúe una expresión que contenga literales de los cuatro tipos de datos (booleano, entero, real y carácter) y la muestre por pantalla.
2. (Reto2) En su entorno de trabajo, cree el programa siguiente. Obsérvese que pasa exactamente. Entonces, intente arreglar el problema.

```

1 // Un programa que usa un entero muuuuy grande
2 public class TresMilMillones {
3     public static void main (String [] args) {
4         System.out.println (3000000000);
5     }
6 }
```

3. (Reto3) Haga un programa con dos variables que, sin usar ningún literal ninguna parte excepto para inicializar estas variables, vaya estimando e imprimiendo sucesivamente los 5 primeros valores de la tabla de multiplicar del 4. Puede usar operadores aritméticos y de asignación, si desea.
4. (Reto4) Haga dos programas, uno que muestre por pantalla la tabla de multiplicar del 3, y otro, la del 5. Los dos deben ser exactamente iguales, letra por letra, excepto en un único literal dentro de todo el código.
5. (Reto5) Experimente qué pasa si en el siguiente programa inicializa la variable realLargo con un valor con varios decimales. El programa continúa compilando? ¿Qué resultado da? Despues inténtelo asignando un valor superior al rango de los enteros (por ejemplo, 3000000000.0).

```

1 public class ConversionExplicita {
2     public static void main (String[] args) {
3         double realLarg = 300000000.0;
4         // Asignación incorrecta. ¿Un real tiene decimales, no?
5         long enterLlarg = (long) realLlarg;
6         // Asignación incorrecta. ¿Un entero largo tiene un rango mayor que un entero, no?
7         int enter = (int) enterLlarg;
8         System.out.println (enter);
9     }
10 }
```

6. (Reto6) Haga un programa que muestre en pantalla de forma tabulada la tabla de verdad de una expresión de disyunción entre dos variables booleanas.
7. (Reto7) Haga un programa que muestre por pantalla la multiplicación de tres números reales entrados por teclado.

4.2. Ejercicios

Solo se puede usar en esta actividad ya que no se ha explicado en profundidad en este tema y lo pueden confundir con el `System.console().readLine();`

1. (Ejs1) Probar la E/S elemental: Escribe el pequeño programa que aparece a continuación.

```

1 import java.util.*;
2 public class EntradaSalida {
3     public static void main (String arg[]){
4         Scanner tec = new Scanner(System.in);
5         int a, b;
6         System.out.println("Introduce un número entero");
7         a = tec.nextInt();
8         System.out.println("Introduce otro número entero");
9         b = tec.nextInt();
10        System.out.println("Los números introducidos son " + a + " y " + b);
11    }
12 }
```

Ejecútalo para ver como se comporta el programa.

¿Qué ocurre si cuando nos pide un número entero le damos un número real? ¿Y si le damos un carácter no numérico?
¿Qué ocurre si eliminamos la instrucción `import java.util.*;`

2. (Ejs2) Averigua mediante pruebas:

- a. ¿Es posible escribir dos instrucciones en la misma línea de un programa?

- b. ¿Se puede "romper" una instrucción entre varias líneas?
- c. Algunos lenguajes de programación dan un valor por defecto a las variables cuando las declaramos sin inicializarlas. Otros no permiten usar el contenido de una variable que no haya sido previamente inicializada. ¿Cuál es comportamiento de Java?
3. (Esj3) ¿Cuáles de los siguientes identificadores son válidos y cuales no? Pruebalo cuando tengas duda

- a. n
- b. MiProblema
- c. MiJuego
- d. Mi Juego
- e. Int
- f. Jose&Co
- g. A b
- h. 1rApellido
- i.aaaaaaaaaaa
- j. NombreApellidos
- k. Saldo-actual
- l. Universidad Alicante
- m. Juan=Rubio
- n. Edad5
- o. _5Java
- p. true
- q. _false
- r. f_false

4. (Por2) Escribir un programa que lea un entero desde teclado, lo multiplique por 2, y a continuación escriba el resultado en la pantalla:

Ejemplo de ejecución:

```
1 Escribe un número:
2 3
3 El doble de 3 es 6
```

5. (Intercambio) Escribir un programa que ...

- a. Lea desde teclado dos valores de texto. Llama a las variables s1 y s2.
- b. Muestre los valores introducidos por el usuario
- c. Intercambie el valor de s1 y s2 (s1 pasa a valer lo que valía s2 y viceversa)
- d. Muestre de nuevo los valores, ahora con su valor intercambiado

Ejemplo de ejecución:

```
1 Escribe un texto para s1: David
2 Escribe un texto para s2: Maria
3 Antes de intercambiar    s1: David y    s2: Maria
4 Despues de intercambiar  s1: Maria y    s2: David
```

6. (ExpresionesMatematicas) Escribir las siguientes expresiones siguiendo la sintaxis de Java.

- a. $\frac{x}{y} + 1$
- b. $\frac{x+y}{x-y}$
- c. $\left[\frac{b}{c+d} \right]$
- d. $(a+b)^2$
- e. $\frac{x+\frac{y}{2}}{x-\frac{y}{z}}$
- f. $\frac{xy}{1-4zx}$
- g. $\frac{(a+b)c}{d}$
- h. $\frac{xy}{mn}$

7. (Superficie) Escribir un programa que solicite al usuario la longitud y la anchura de una habitación y a continuación muestre su superficie (longitud por anchura).

8. (Medidas) Escribir un programa que convierta una medida dada en pies a sus equivalentes en yardas, pulgadas, centímetros y metros, sabiendo que 1 pie = 12 pulgadas, 1 yarda = 3 pies, 1 pulgada = 2.54 cm, 1 m = 100 cm.
9. (Segundos) Escribir un programa que, dada una cantidad de segundos, introducida por teclado, la desglose en días, horas, minutos y segundos.

Ejemplo de ejecución:

```
1 Introduce cantidad de segundos: 3661
2 3661 segundos son:
3 0 dias
4 1 horas
5 1 minutos
6 1 segundos
```

10. (Fuerza) La fuerza de atracción entre dos masas m_1 y m_2 separadas por una distancia d , está dada por la fórmula: $\frac{G \cdot m_1 \cdot m_2}{d^2}$
donde G es la constante de gravitación universal $G = 6.67430 \cdot 10^{-11}$.
- Escribir un programa que lea la masa de dos cuerpos y la distancia entre ellos y a continuación obtenga su fuerza de atracción.
11. (Círculo) Escribir un programa que calcule la longitud de la circunferencia y el área del círculo para un valor del radio introducido por teclado.
12. (Dados) Escribir un programa que simula el lanzamiento de dos dados.

```
1 Dado 1 : 5
2 Dado 2: 4
3 Puntuación total: 9
```

13. (UltimaCifra) Escribir un programa que muestre la última cifra de un número entero que introduce el usuario por teclado.

Pista: ¿Qué devuelve $a \% 10$?

```
1 Introduce un número entero: 3761
2 La última cifra de 3761 es 1
```

14. (PenultimaCifra) Escribir un programa que muestre la penúltima cifra de un número entero que introduce el usuario por teclado.

```
1 Introduce un número entero: 3761
2 La penúltima cifra de 3761 es 6
```

Una vez hayas comprobado que el programa funciona correctamente, prueba qué ocurre si el usuario introduce un valor de una sola cifra (por ejemplo 4). Explica el resultado mostrado por el programa.

15. (Redondear1) `Math.round(x)` redondea x de manera que este queda sin decimales. (`Math.round(35.5289)` da como resultado 36)
- Trata de escribir un programa en el que el usuario introduzca un número real y a continuación se muestre redondeado a un decimal. *Pista : combinar productos, divisiones y Math.round()*

Ejemplo de ejecución:

```
1 Introduce un número real: 35.5289
2 El número 35.5289, redondeado a un decimal es 35.5
```

16. (ExpresionesAritmeticas) 16. Cuál es el valor resultante de dada una de las siguientes expresiones

- a. $5 * 4 - 3 * 6$
- b. $4 * 5 * 2$
- c. $(24 + 2 * 6) / 4$
- d. $8 / 2 / 2 * 5$
- e. $3 + 4 * (8 * (4 - (9 + 3) / 6))$
- f. $4 * 3 * 5 + 8 * 4 * 2$
- g. $4 - 40 \% 5$
- h. $4 * 3 / 2$
- i. $4 / 2 * 3$
- j. $213 / 100$

17. (Einstein) La famosa ecuación de Einstein para la conversión de una masa m en energía viene dada por la fórmula $E=mc^2$, donde c es la velocidad de la luz que vale $2.997925 \cdot 10^8$ m/s. Escribir un programa que lea el valor de la masa y obtenga la energía correspondiente según la anterior fórmula.

18. (FragmentosCódigo) Indica cuales serán los valores de las variables después de ejecutar cada uno de los siguientes fragmentos de código. Resuelve el ejercicio sin escribir los programas correspondientes y probarlos.

- a. `java int a=3, b = 2; a = b + b; b = a + a;`
- b. `java int a=3,b=0; b = b - 1; a = a + b;`
- c. `java int a, b=5; b++; ++b; a= b+1;`
- d. `java int a = 5,b; b = a++;`
- e. `java int a = 5,b; b = ++a;`
- f. `java int a=2, b=3; b+=a;`
- g. `java int a=2, b=3; b-=a; a=-b;`
- h. `java int a=2, b=3; b%=a;`
- i. `java int a=2,b=3,c=4; a = --b + c++; b+=a;`

4.3. Expresiones Lógicas

1. Sean 4 variables enteras:

```
1 int m, j, p, v ;
```

que contienen respectivamente la edad de Miguel, Julio, Pablo y Vicente.

Expresar las siguientes afirmaciones utilizando operadores lógicos y relacionales

Ejemplo: Miguel es mayor de edad.

Solución: $m \geq 18$

- a. (Logica1) Miguel es menor de edad.
- b. (Logica2) Miguel es mayor que Julio
- c. (Logica3) Miguel es el más viejo.
- d. (Logica4) Miguel es el más joven.
- e. (Logica5) Miguel no es el más joven.
- f. (Logica6) Miguel no es el más viejo.
- g. (Logica7) Alguno de ellos es mayor de edad.
- h. (Logica8) Miguel y Julio son los más jóvenes.
- i. (Logica9) Entre todos tienen más de 100 años.
- j. (Logica10) Entre Miguel y Julio suman más edad que Pablo.
- k. (Logica11) Entre Miguel y Julio suman más edad que Pablo y Vicente juntos.
- l. (Logica12) Si los ordenamos por edades de menor a mayor, Julio es el segundo.
- m. (Logica13) Si los ordenamos por edades de menor a mayor, Julio es el segundo y Pablo el tercero.
- n. (Logica14) Al menos uno de ellos es menor de edad.
- o. (Logica15) Al menos dos de ellos son menores de edad.
- p. (Logica16) Todos son menores de edad.
- q. (Logica17) Solo dos de ellos son menores de edad.
- r. (Logica18) Al menos dos de ellos nacieron el mismo año.
- s. (Logica19) Solo dos de ellos nacieron el mismo año.
- t. (Logica20) Al menos uno de ellos es menor que Julio
- u. (Logica21) Solo uno de ellos es menor que Julio
- v. (Logica22) Miguel es mayor de edad y alguno de los otros es menor de edad.

4.4. Actividades

1. (Actividad1) Realiza un conversor de euros a pesetas. La cantidad de euros que se quiere convertir debe ser introducida por teclado.
2. (Actividad2) Realiza un conversor de pesetas a euros. La cantidad de pesetas que se quiere convertir debe ser introducida por teclado.
3. (Actividad3) Escribe un programa que calcule el área de un rectángulo. (`area = base * altura`)
4. (Actividad4) Escribe un programa que calcule el área de un triángulo. (`area = (base * altura) / 2`)
5. (Actividad5) Escribe un programa que calcule el salario semanal de un empleado en base a las horas trabajadas, a razón de 12 euros la hora.
6. (Actividad6) Realiza un conversor de MiB a KiB. [Ayuda](#)
7. (Actividad7) Realiza un conversor de Kib a Mib. [Ayuda](#)
8. (Actividad8) Realiza un programa en Java que genere letras de forma aleatoria.
9. (Actividad9) Realiza un programa en Java que genere el número premiado del Cupón de la ONCE.
10. (Actividad10) Modificar el siguiente programa para que compile y funcione:

```

1  public class Activ10 {
2      public static void main(String[] args) {
3          int n1 = 50, int n2 = 30,
4              boolean suma = 0;
5          suma = n1 + n2;
6          System.out.println("LA SUMA ES: " + suma);
7      }
8  }
```

11. (Actividad11) Modificar el siguiente programa para que compile y funcione:

```

1  public class Activ11 {
2      public static void main(String[] args) {
3          int numero = 2;
4          cuad = numero * numero;
5          System.out.println("EL CUADRADO DE "+NUMERO+" ES: "+cuad);
6      }
7  }
```

12. (Actividad12) Indicar que valor devolverá la ejecución del siguiente programa:

```

1  public class Activ12 {
2      public static void main(String[] args) {
3          int num = 5;
4          num += num - 1 * 4 + 1;
5          System.out.println(num);
6      }
7  }
```

13. (Actividad13) Indicar que valor devolverá la ejecución del siguiente programa:

```

1  public class Activ13 {
2      public static void main(String[] args) {
3          int num = 4;
4          num %= 7 * num % 3 * 3;
5          System.out.println(num);
6      }
7  }
```

14. (Actividad14) Realizar un programa que muestre por pantalla respetando los saltos de carro el siguiente texto (con un solo `println`):

```

1  Me gusta la programación
2  cada día más
```

15. (Actividad15) Realiza un programa en Java que tenga las variables edad, nivel de estudios e ingresos y almacene en una variable llamada jasp el valor verdadero si la edad es menor o igual a 28 y el nivel de estudios es mayor a 3, o bien la edad es menor de 30 y los ingresos superiores a 28000. En caso contrario almacenar el valor falso.
16. (Actividad16) Realizar un programa que realice el cálculo del precio de un producto teniendo en cuenta que el producto vale 120 €, tiene un descuento del 15% y el IVA que se le aplica es del 21%.

17. (Actividad17) Realiza un programa que calcule la nota que hace falta sacar en el segundo examen de la asignatura Programación para obtener la media deseada. Hay que tener en cuenta que la nota del primer examen cuenta el 40% y la del segundo examen un 60%. Ejemplo 1:

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

Ejemplo 2:

```
1 Introduce la nota del primer examen: 8
2 ¿Qué nota quieras sacar en el trimestre? 7
3 Para tener un 7 en el trimestre necesitas sacar un 6.333333333 en el segundo examen.
```

18. (Actividad18) Realizar un programa que dado un importe en euros nos indique el mínimo número de billetes y la cantidad sobrante de euros. Debes usar el operador condicional ?:

```
1 ¿Cuántos euros tienes?: 232
2 1 billete de 200 €
3 1 billete de 20 €
4 1 billete de 10 €
5 Sobran 2 €
```

⌚5 de octubre de 2025

5. 2.3 Talleres

5.1. Taller UD01_01: Instalar NoMachine para el control remoto

5.1.1. ¿Qué es NoMachine ?

Conéctese a cualquier computadora de forma remota a la velocidad de la luz. Gracias a nuestra tecnología NX, NoMachine es el escritorio remoto más rápido y de mayor calidad que jamás haya probado. Conecta con tu ordenador al otro lado del mundo con solo unos pocos clics. Vé donde esté tu escritorio, podrás acceder a él desde cualquier otro dispositivo y compartirlo con quien quieras. NoMachine es tu servidor personal, privado y seguro. Además, es gratis.

<https://www.nomachine.com/>

5.1.1.1. DESCARGA E INSTALA LA APLICACIÓN

Desde la página de descargas:

<https://downloads.nomachine.com/>

E instala la aplicación en tu PC.

5.1.2. Permisos al profesor

Debemos conceder permisos para que el profesor se pueda conectar a nuestro PC mientras estemos en el instituto sin necesidad de contraseña. Esto solo será posible cuando estemos conectados a la red del instituto, y el profesor no podrá acceder cuando estemos en casa.

Agrega la clave SSH pública (es un fichero `authorized.crt` que te proporcionará el profesor a través de AULES) en tu ordenador

- Debes colocarla en la carpeta `<Inicio del usuario>/.nx/config`.
- Cree este directorio si no existe.
- En Linux y macOS, ejecute en una terminal: `mkdir $HOME/.nx/config`
- En Windows, créelo en (`C:\Users\username\.nx\config`) usando las herramientas del sistema (Explorador de archivos).
- Si la carpeta de configuración ya existe, copia el fichero `authorized.crt` en ella.

Ten en cuenta que los navegadores pueden cambiar las extensiones de los archivos, es conveniente tener las opciones de "ver extensiones de archivos" y "ver archivos ocultos" en nuestro gestor de archivos habitual

5.1.3. Tarea

Debes enviar un archivo `*.pdf` a la plataforma de AULES con una simple captura que demuestre que el profesor se ha podido conectar a tu PC.

Debes mantener `NoMachine` instalado y permitir las conexiones automáticas por parte del profesor para pedir ayuda y consultar dudas en clase, para corregir las tareas diarias, y para realizar los exámenes.

⌚26 de septiembre de 2025

5.2. Taller UD01_02: Instalación y uso de entornos de desarrollo

5.2.1. Java

Cada software y cada entorno de desarrollo tiene unas características y funcionalidades específicas. Esto también se verá reflejado en la instalación y configuración del software. Dependiendo de la plataforma, entorno o sistema operativo en el que se vaya a instalar el software, se utilizará un paquete de instalación u otro, y habrá que tener en cuenta unas opciones u otras en su configuración. A continuación se muestra cómo instalar una herramienta de desarrollo de software integrada, como Eclipse. Pero también podrás observar los procedimientos para instalar otras herramientas necesarias o recomendadas para trabajar con el lenguaje de programación JAVA, como Tomcat o la Máquina Virtual de Java. Debes tener en cuenta los siguientes conceptos:

- La JVM (Java Virtual Machine, máquina virtual de Java) es la encargada de interpretar el bytecode y generar el código máquina del ordenador (o dispositivo) en el que se ejecuta la aplicación. Esto quiere decir que necesitamos una JVM distinta para cada entorno.
- JRE (Java Runtime Environment) es un conjunto de utilidades Java que incluye la JVM, las bibliotecas y el conjunto de software necesario para ejecutar aplicaciones cliente Java, así como el conector para que los navegadores de Internet ejecuten applets.
- JDK (Java Development Kit) es el conjunto de herramientas para desarrolladores; contiene, entre otras cosas, el JRE y el conjunto de herramientas necesarias para compilar el código, empaquetarlo, generar documentación...

```
graph TD
    subgraph JDK
        subgraph JRE
            subgraph JVM
                end
            end
        end
    end
```

El proceso de instalación consta de los siguientes pasos: 1. Descargue, instale y configure el JDK. 2. Descargue e instale un servidor web o de aplicaciones. 3. Descargue, instale y configure el IDE (Netbeans o Eclipse). 4. Configurar JDK con IDE. 5. Configure el servidor web o de aplicaciones con el IDE instalado. 6. Si es necesario, instalación de conectores. 7. Si es necesario, instale un nuevo software.

5.2.1.1. DESCARGUE E INSTALE EL JDK

Podemos diferenciar entre:

- Java SE (Java Standard Edition): es la versión estándar de la plataforma, siendo esta plataforma la base para todos los entornos de desarrollo Java ya sea de aplicaciones cliente, de escritorio o web.
- Java EE (Java Enterprise Edition): esta es la versión más grande de Java y generalmente se utiliza para crear grandes aplicaciones cliente/servidor y para el desarrollo de servicios web.

En este curso se utilizarán las funcionalidades de Java SE. El archivo es diferente según el sistema operativo donde se tenga que instalar. Así:

- Para los sistemas operativos Windows y Mac OS hay un archivo instalable.
- Para los sistemas operativos GNU/Linux que admiten paquetes .rpm o .deb, también están disponibles paquetes de este tipo.
- Para el resto de sistemas operativos GNU/Linux existe un archivo comprimido (terminado en .tar.gz).

En los dos primeros casos, simplemente hay que seguir el procedimiento de instalación habitual del sistema operativo con el que estamos trabajando. En este último caso, sin embargo, hay que descomprimir el archivo y copiarlo en la carpeta donde se desea instalar. Normalmente, todos los usuarios tendrán permisos de lectura y ejecución en esta carpeta.

A partir de la versión 11 de JDK, Oracle distribuye el software con una licencia significativamente más restrictiva que las versiones anteriores. En particular, solo se puede utilizar para "desarrollar, probar, crear prototipos y demostrar sus aplicaciones". Cualquier uso "para fines comerciales, de producción o empresariales internos" distinto del mencionado anteriormente queda explícitamente excluido.

Si lo necesitas para alguno de estos usos no permitidos en la nueva licencia, además de las versiones anteriores del JDK, existen versiones de referencia de estas versiones licenciadas "GNU General Public License version 2, with the Classpath Exception", que permiten la mayoría de los usos habituales. Estas versiones están enlazadas a la misma página de descarga y también a la dirección jdk.java.net.

Una alternativa es utilizar <https://adoptium.net/> antes conocido como adoptOpenJDK, que ahora se ha integrado en la fundación Eclipse. Desde allí podemos descargar los binarios de la versión openJDK para nuestra plataforma sin restricciones. [Noticia completa] (<https://es.wikipedia.org/wiki/OpenJDK>).

En GNU/Linux podemos utilizar los comandos:

- `sudo apt install default-jdk` para instalar el jdk predeterminado.
- `java --version` para ver las versiones disponibles en nuestro sistema.
- `sudo update-alternatives --config java` para elegir cuál de las versiones instaladas queremos usar por defecto o incluso ver la ruta de las diferentes versiones que tenemos instaladas.

5.2.1.2. CONFIGURAR LAS VARIABLES DE ENTORNO "JAVA_HOME" Y "PATH"

Una vez descargado e instalado el JDK, debes configurar algunas variables de entorno:

- La variable `JAVA_HOME`: indica la carpeta donde se ha instalado el JDK. No es obligatorio definirla, pero es muy cómodo hacerlo, ya que muchos programas buscan en ella la ubicación del JDK. Además, resulta muy fácil definir las dos variables siguientes.
- La variable `PATH`. Debe apuntar al directorio que contiene el ejecutable de la máquina virtual. Suele ser la subcarpeta `bin` del directorio donde hemos instalado el JDK.

Variable CLASSPATH Otra variable que tiene en cuenta el JDK es la variable `CLASSPATH`, que apunta a las carpetas donde se encuentran las librerías de la aplicación que se quiere ejecutar con el comando `java`. Es preferible, no obstante, indicar la ubicación de estas carpetas con la opción `-cp` del mismo comando `java`, ya que cada aplicación puede tener diferentes librerías y las variables de entorno afectan a todo el sistema. Establecer la variable `PATH` es esencial para que el sistema operativo encuentre los comandos JDK y pueda ejecutarlos.

5.2.2. Eclipse

Eclipse es una aplicación de código abierto desarrollada actualmente por Eclipse Foundation, una organización independiente, sin fines de lucro, que fomenta una comunidad de código abierto y el uso de un conjunto de productos, servicios, capacidades y complementos para la divulgación del uso de código abierto en el desarrollo de aplicaciones informáticas. Eclipse fue desarrollado originalmente por IBM como sucesor de VisualAge. Como Eclipse está desarrollado en Java, es necesario, para su ejecución, tener un JRE (Java Runtime Environment) previamente instalado en el sistema. Para saber si tienes este JRE instalado, puedes hacer el test en la web oficial de Java, en la sección ¿Tengo Java? Si vamos a desarrollar con Java, como es nuestro caso, deberemos tener instalado el JDK (recordemos que es un superconjunto del JRE).

5.2.2.1. INSTALACIÓN

Las versiones actuales del entorno Eclipse se instalan con un instalador. Este, básicamente, se encarga de descomprimir, solucionar algunas dependencias y crear los accesos directos. Este instalador se puede obtener descargándolo directamente desde la página oficial del Proyecto Eclipse www.eclipse.org. Podrás encontrar las versiones para los diferentes sistemas operativos e instrucciones para su uso. No son nada complejas. En el caso de GNU/Linux y MAC OS, el archivo es un archivo comprimido, por lo que hay que descomprimirla y luego ejecutar el instalador. Se trata del archivo `eclipse-inst`, dentro de la carpeta `eclipse`, que es una subcarpeta del resultado de descomprimir el archivo anterior. Si sólo el usuario actual va a utilizar el IDE, la instalación se puede realizar sin utilizar privilegios de administrador o root y seleccionando para la instalación una carpeta perteneciente a este usuario. Si se desea compartir la instalación entre distintos usuarios, se debe indicar al instalador una carpeta sobre la que todos estos usuarios tengan permisos de lectura y ejecución.

Al iniciar el instalador veremos una pantalla similar a esta:

Splash Screen

El instalador nos preguntará qué versión queremos instalar. La versión que utilizaremos es “Eclipse IDE for Java EE Developers”.

Paso 1

Luego nos pedirá la versión de JDK/JRE que vamos a utilizar (en la captura aparece con letras blancas). También nos pide la carpeta donde la instalaremos. Y dos check boxes para indicar si queremos que nos cree el acceso directo al menú de aplicaciones ya en el escritorio.

Paso 2

Para seleccionar la carpeta correcta hay que tener en cuenta qué usuarios van a utilizar el entorno. Todos ellos deben tener permisos de lectura y ejecución sobre la carpeta en cuestión. Una vez introducida la carpeta podemos pulsar el botón **INSTALAR** para iniciar la instalación.

También se nos pedirá que aceptemos las licencias del software a instalar, como muestra la captura de pantalla:

Paso 3

Durante la instalación veremos una pantalla de progreso como la que se muestra a continuación:

Paso 4

Una vez finalizada la instalación, se nos muestra una pantalla que nos invita a ejecutar directamente el entorno.

Paso 5

Esta primera vez podremos ejecutar el entorno Eclipse pulsando el botón LAUNCH. El resto de las veces será necesario invocarlo desde los accesos directos o lanzadores, si se han creado o, en caso contrario, invocando directamente el ejecutable. Este se llama eclipse y lo encontrarás en una subcarpeta de la carpeta de instalación también llamada eclipse. La ruta exacta puede variar de una versión a otra. Si en el futuro es necesario desinstalarlo, sólo se debe borrar la carpeta donde ha sido instalado ya que la instalación de Eclipse no aparece en el repositorio de GNU/Linux ni en el panel de control en Windows. Cuando ejecutamos el entorno nos aparecerá una pantalla como la siguiente:

Paso 6

Inmediatamente se nos preguntará en qué carpeta se ubicará el workspace. Podemos pedirle que lo recuerde para el resto de ejecuciones activando la opción “*Usar esto como predeterminado y no volver a preguntar*”.

Paso 7

La primera vez que lo ejecutemos se mostrará la pestaña de bienvenida. Podemos pedirle que no nos la muestre más desactivando la opción “Mostrar siempre la bienvenida al iniciar”.

Paso 8

Una vez cerrada esta pestaña, el entorno de trabajo será similar a esto:

Paso 9

Por defecto Eclipse nos ofrece la descarga del instalador más ligero que descargaría de Internet los paquetes necesarios para completar la instalación según nuestras elecciones. Si esta instalación nos da problemas, podemos descargar la versión “package” en la que previamente deberemos elegir el paquete de instalación que queramos, ocupará bastante más, pero descargaría todos los paquetes necesarios. Después solo tendremos que descomprimir el archivo descargado en una carpeta de nuestra elección y ya tendremos eclipse instalado. Tendremos que crear nuestro propio menú de inicio e iconos del escritorio (podéis seguir esta [guía] (<https://www.donovanbrown.com/post/Añadir-Eclipse-al-Launcher-en-Ubuntu-1604>) cambiando la ruta donde habéis descomprimido vuestra versión de eclipse).

5.2.2.2. CONFIGURACIÓN

Versión Java

Por defecto Eclipse intenta utilizar las nuevas características del JDK 16, pero en nuestro caso por ejemplo tenemos la versión 11. Podemos personalizar estas opciones en el apartado Ventana/Preferencias/Java/Compilador y elegir en el campo Nivel de conformidad del compilador la versión correcta, en nuestro caso la 11.

Además, si lo necesitamos, podemos configurar los JDKs que están disponibles, añadirlos o eliminarlos desde la opción Ventana/Preferencias/Java/JRE instalados .

Perspectiva

Eclipse llama a la distribución de los paneles en la ventana Perspectiva, hay unos cuantos predefinidos y podemos configurar los nuestros, a nuestro gusto en la sección Ventana/Perspectiva .

Apariencia

Eclipse nos permite personalizar cualquier aspecto de la apariencia de nuestro entorno, cambiar tanto el tema del IDE como el tamaño de fuente y los colores para el coloreado del código fuente. Todas estas opciones están disponibles en Ventana/Apariencia .

5.2.2.3. MÓDULOS

Las opciones y funcionalidades de Eclipse se pueden ampliar añadiendo módulos desde su "store" de plugins. En Help/Eclipse Marketplace... podemos por ejemplo buscar por texto, o buscar en la pestaña de populares. Eso nos mostrará todos los complementos que contengan la palabra buscada, o los complementos más descargados del marketplace. Podemos instalar, por ejemplo, SonarLint 6.0 que nos ayuda a mantener nuestro código limpio de errores comunes, para ello simplemente tenemos que pulsar el botón INSTALAR que aparece a su lado en el listado, aceptar la licencia de uso y automáticamente nos pedirá que reiniciemos el IDE .

5.2.2.4. USO BÁSICO ("¡HOLA MUNDO!")

Eclipse proporciona información sobre su uso en la sección de `Ayuda`, y podemos aprender a crear nuestro primer proyecto en Java (el típico "¡Hola Mundo!"). Para ello debemos abrir la ventana de `Bienvenido`, que es la que nos aparece cuando abrimos Eclipse por primera vez, o bien podemos abrirla desde `Ayuda/Bienvenido`, desde esta ventana podemos elegir la sección de `Tutoriales`, y dentro de la sección de Desarrollo Java, elegir el primer ítem "Crear una aplicación Hola Mundo", y el propio Eclipse nos irá guiando paso a paso para crear y ejecutar nuestro primer proyecto Java en Eclipse.

5.2.2.5. ACTUALIZACIÓN Y MANTENIMIENTO

En la misma sección `Ayuda` Eclipse nos proporciona las opciones para actualizar el propio Eclipse o los complementos que tengamos instalados `Ayuda/Buscar actualizaciones`.

Podemos personalizar el comportamiento respecto a las actualizaciones en la sección `Ventana/Preferencias/Instalar/Actualizar/Actualizaciones automáticas`.

5.2.3. Netbeans

NetBeans es una herramienta de entorno de desarrollo integrado (IDE) muy potente que se utiliza principalmente para el desarrollo en Java y C/C++. Permite desarrollar fácilmente aplicaciones web, de escritorio y móviles desde su marco modular. Puede agregar soporte para otros lenguajes de programación como PHP, HTML, JavaScript, C, C++, Ajax, JSP, Ruby on Rails, etc. mediante extensiones.

Se ha lanzado NetBeans IDE 12 con soporte para Java JDK 11. También incluye las siguientes características:

- Soporte para PHP 7.0 a 7.3, PHPStan y Twig.
- Incluir módulos en el clúster "webcommon". Es decir, todas las funciones de JavaScript en Apache NetBeans GitHub son parte de Apache NetBeans 10.
- Los módulos de clúster "groovy" están incluidos en Apache NetBeans 10.
- OpenJDK puede detectar automáticamente JTReg desde la configuración de OpenJDK y registrar el JDK expandido como una plataforma Java.
- Soporte para JUnit 5.3.1

5.2.3.1. INSTALACIÓN

Podemos instalar NetBeans de tres maneras:

5.2.3.1.1. Instalar desde binarios

Paso 1: Descargue el archivo NetBeans

Descargue el archivo binario de NetBeans 12 `netbeans-12.4-bin.zip`.

Paso 2: Extraer el archivo

Espere a que finalice la descarga y luego extráigala.

```
1 $ unzip netbeans-12.4-bin.zip
```

Confirme el contenido del archivo de directorio creado:

```
1 $ ls netbeans
2 apisupport enterprise groovy javafx netbeans.css profiler
3 bin ergonomics harness LICENSE NOTICE README.html
4 cpplite etc ide licenses php webcommon
5 DEPENDENCIES extide java nb platform websvccommon
```

Step 3: Move the `netbeans` folder to `/opt`

Ahora movamos la carpeta `netbeans/` a `/opt`

```
1 $ sudo mv netbeans/ /opt/
```

Paso 4: Ruta de configuración

El binario ejecutable de Netbeans se encuentra en `/opt/netbeans/bin/netbeans`. Necesitamos agregar su directorio principal a nuestro `$PATH` para poder iniciar el programa sin especificar la ruta absoluta al archivo binario. Abra su archivo `~/.bashrc` o `~/.zshrc`.

```
1 $ nano ~/.bashrc
```

Añade la siguiente línea al final

```
1 export PATH = "$PATH:/opt/netbeans/bin/"
```

Obtenga el archivo para iniciar Netbeans sin reiniciar el shell.

```
1 $ source ~/.bashrc
```

Paso 5: Crear el iniciador de escritorio NetBeans IDE (opcional)

Cree un nuevo archivo en `/usr/share/applications/netbeans.desktop`.

```
1 $ sudo nano /usr/share/applications/netbeans.desktop
```

Añade los siguientes datos.

```
1 [Desktop Entry]
2 Name=Netbeans IDE
3 Comment=Netbeans IDE
4 Type=Application
5 Encoding=UTF-8
6 Exec=/opt/netbeans/bin/netbeans
7 Icon=/opt/netbeans/nb/netbeans.png
8 Categories=GNOME;Application;Development;
9 Terminal=false
10 StartupNotify=true
```

Para desinstalar NetBeans debemos eliminar la carpeta `netbeans/` que está dentro de la carpeta `/opt/`, podemos utilizar el comando:

```
1 $ sudo rm /opt/netbeans -rf
```

Paso 6: Configurar correctamente el JDK (opcional)

En el fichero `/opt/netbeans/etc/netbeans.conf` debemos especificar correctamente la ruta de nuestro JDK en la variable `netbeans_jdkhome`. En GNU/Linux podemos saber los JDK disponibles con el comando `sudo update-alternatives --config java` que nos mostrará un resultado similar a este:

```
1 Hi ha 3 possibilitats per a l'alternativa java (que proveeix /usr/bin/java).
2
3 Selecció Camí Prioritat Estat
4 -----
5 * 0 /usr/lib/jvm/java-14-openjdk-amd64/bin/java 1411 mode automàtic
6   1 /usr/lib/jvm/java-11-openjdk-amd64/bin/java 1111 mode manual
7   2 /usr/lib/jvm/java-14-openjdk-amd64/bin/java 1411 mode manual
8   3 /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java 1081 mode manual
9
10 Premeu retorn per a mantenir l'opció per defecte[*], o introduiu un número de selecció:
```

En la configuración de netbeans no es necesario especificar el final de la ruta `bin/java`

```
1 netbeans_jdkhome="/usr/lib/jvm/java-11-openjdk-amd64/"
```

5.2.3.1.2. Instalar desde script

Paso 1: Descargue el archivo NetBeans

También puede instalar Netbeans 12.4 en GNU/Linux desde un script proporcionado para descargar `Apache-NetBeans-12.4-bin-linux-x64.sh`.

Paso 2: Ejecutar el script

Debes ejecutar el script de instalación

```
1 $ sudo sh ./Apache-NetBeans-12.4-bin-linux-x64.sh
```

Si ejecuta el script como `root` (`sudo`) Netbeans estará disponible para todos los usuarios. Por el contrario, si ejecuta el usuario sin `sudo`, solo estará disponible para su usuario.

Aparecerá una barra de progreso como esta:

Paso 2a

Ahora podemos elegir los componentes que queremos instalar con el IDE de Netbeans, lo dejaremos por defecto y pulsaremos el botón siguiente.

Paso 2b

Paso 3: Aceptar la licencia

Luego debemos aceptar el acuerdo de licencia de uso marcando la casilla y presionando el botón siguiente.

Paso 3

Paso 4: Elija la ruta de instalación y el JDK

Ahora debemos elegir la ruta donde se instalará Netbeans 12.4. Y debemos elegir la ruta donde se encuentra el JDK (por defecto indica `/usr`, pero debemos especificar la ubicación como por ejemplo `/usr/lib/jvm/java-11-openjdk-amd64`).

Paso 4

Paso 5: Actualizaciones automáticas

En este punto se muestra un resumen de la instalación, y podemos elegir si queremos que NetBeans busque e instale actualizaciones desde Internet, y pulsar el botón instalar.

Paso 5

Paso 6: Instalación

Aparecerá una barra de progreso.

Paso 6

Paso 7: Paso final

Al terminar, aparecerá una pantalla con las acciones realizadas por el instalador y ya tendremos los launchers creados en el menú de aplicaciones.

Paso 7

5.2.3.1.3. Instalar mediante snap

Quizás una forma más sencilla de instalar la última versión de Netbeans en nuestro sistema GNU/Linux es a través de `snap`:

```
1 $ sudo snap install netbeans --classic
```

5.2.3.1.4. Primera ejecución

Cuando ejecutamos el entorno nos aparecerá una pantalla como la siguiente:

Paso 8a

La primera vez que lo ejecutemos se mostrará la pestaña de bienvenida. Podemos pedir que no se nos muestre más desactivando la opción "Mostrar al iniciar".

Paso 8

Una vez cerrada esta pestaña, el entorno de trabajo será similar a esto:

Paso 9

NetBeans puede solicitarnos permiso para utilizar nuestra información a nivel estadístico, elegimos el comportamiento deseado y aceptamos.

Paso 8b

Para desinstalar NetBeans en este caso debemos ejecutar el archivo `uninstall.sh` que se encuentra en la carpeta de instalación.

5.2.3.2. CONFIGURACIÓN

Activar módulos

Por defecto Netbeans tiene los módulos desactivados y será la primera vez que los necesitemos cuando pasen a estar activos y disponibles. Por ejemplo, si creamos un nuevo proyecto y elegimos `Java Application` dentro de la categoría `Java with Ant`, veremos en la parte inferior que Netbeans nos avisa de que el módulo necesario no está activo y que debemos pulsar `Next` para que esté disponible. Lo hacemos, y a continuación nos pedirá que activemos el módulo `nb-javac Impl`, dejamos el check marcado y pulsamos el botón `Activate`, y nos aparecerá el asistente para crear nuestro primer proyecto Java.

Versión Java

Dentro del menú `Herramientas/Plataformas Java` podemos cambiar o ver la ubicación de nuestra instalación JDK.

Perspectiva

En Netbeans las perspectivas no son necesarias, el entorno de Netbeans, aunque es personalizable, se adapta automáticamente a las tareas que estés realizando en cada momento.

Apariencia

Netbeans nos permite personalizar cualquier aspecto de la apariencia de nuestro entorno, cambiar el tema del IDE así como el tamaño de fuente y los colores para el coloreado del código fuente. Todas estas opciones están disponibles en `Herramientas/Opciones`, y dentro de esta ventana elegimos la tercera pestaña `Fuente y Colores` y la penúltima pestaña `Apariencia`.

Configuración de exportación/importación

Una opción muy interesante de Netbeans es que nos permite exportar o importar configuraciones y compartirlas con otros compañeros o incluso entre nuestros equipos o diferentes instalaciones. La opción está disponible en `Herramientas/Opciones`, abajo a la izquierda encontramos los botones `Exportar...` e `Importar...`.

5.2.3.3. MÓDULOS

Las opciones y funcionalidades de Netbeans se pueden ampliar añadiendo módulos desde su sección de plugins. En `Tools/Plugins` podemos por ejemplo buscar por texto, o buscar en la pestaña de plugins disponibles. Eso nos mostrará todos los plugins que contienen la palabra buscada, o los plugins disponibles. Podemos instalar por ejemplo `sonarlint4netbeans` que nos ayuda a mantener nuestro código limpio de errores comunes, para ello simplemente tenemos que marcar la casilla delante del nombre del plugin, y pulsar el botón `INSTALAR` que aparece más abajo, pulsar siguiente, aceptar la licencia de uso e instalar. Cuando termine la instalación nos pedirá que reiniciemos el `IDE`.

5.2.3.4. USO BÁSICO ("¡HOLA MUNDO!")

Para crear nuestra primera aplicación en Netbeans, debemos crear una aplicación Java, desde el menú `Archivo/Nuevo Proyecto...` debemos elegir `Aplicación Java` dentro de la categoría `Java con Ant`. A continuación debemos especificar el nombre del proyecto, por ejemplo "App Hello World", y nos aseguramos de dejar marcada la opción `crear clase principal app.hello.world.AppHolaWorld` y nos debería aparecer algo como esto:

Aplicación Hola Mundo 1

En este punto, sólo nos queda incluir la línea de código necesaria para imprimir el mensaje de texto en pantalla. Para ello, nos dirigiremos al final de la línea `// TODO code application logic here` y pulsaremos la tecla `ENTER` para crear una nueva línea.

Una vez situados en el lugar adecuado utilizaremos una de las funcionalidades más interesantes de Netbeans, que son las plantillas de código. Tecleamos la palabra "sout" y luego pulsamos la tecla `TAB` y Netbeans la sustituirá por el código correcto: `System.out.println ("");`.

Ahora debemos escribir entre las dos comillas dobles el mensaje de texto que debe aparecer en pantalla, y debe quedar así:

```
1 System.out.println("Hola Mundo!");
```

Luego podemos presionar el botón superior con un triángulo verde (`Ejecutar proyecto`) o presionar la tecla `F6` del teclado:

Aplicación Hola Mundo 2

Aparecerá una nueva sección en la ventana (en la parte inferior) llamada `Salida` en la que podremos visualizar el resultado de la ejecución de nuestro primer programa.

5.2.3.5. ACTUALIZACIÓN Y MANTENIMIENTO

En la sección `Ayuda`, Netbeans nos proporciona las opciones para actualizar el propio Netbeans con la opción `Ayuda/Buscar actualizaciones`.

5.2.4. IntelliJ (recomendado)

IntelliJ IDEA es un entorno de desarrollo integrado (IDE) escrito en Java para desarrollar software informático escrito en Java, Kotlin, Groovy y otros lenguajes basados en JVM. Está desarrollado por JetBrains (antes conocido como IntelliJ) y está disponible como una edición comunitaria con licencia Apache 2 y en una edición comercial propietaria. Ambas se pueden utilizar para el desarrollo comercial.

Nuestra institución dispone de licencias para nuestros alumnos mientras tengáis correo electrónico @ieseduardoprimo.es.

5.2.4.1. INSTALACIÓN

Descargue desde <https://www.jetbrains.com/idea/> la versión de la herramienta toolbox correspondiente a su sistema operativo.

Siga las instrucciones para su sistema operativo desde <https://www.jetbrains.com/help/idea/installation-guide.html#toolbox>

Una vez instalada la caja de herramientas, puede elegir instalar todos los productos de JetBrains.

Una vez instalada la Idea (IDE) puedes crear una entrada de escritorio desde la pantalla inicial:

Create desktop icon

Y en la opción Administrar licencias debes seguir estas instrucciones: https://www.jetbrains.com/help/license_server/Activating_license.html

La dirección del servidor es: <https://iesepm.flx.jetbrains.com/>

5.2.4.2. AJUSTES

Documentos para configurar su IDE: <https://www.jetbrains.com/help/idea/configuring-project-and-ide-settings.html>

5.2.4.3. MÓDULOS

Puedes agregar complementos siguiendo estas instrucciones:

<https://www.jetbrains.com/help/idea/managing-plugins.html>

5.2.4.4. USO BÁSICO ("¡HOLA MUNDO!")

Los documentos te ayudan con tu primer programa en Java: <https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-application.html>

Mucha más información:

- Si vienes de Eclipse: <https://www.jetbrains.com/help/idea/migrating-from-eclipse-to-intellij-idea.html>
- Si estuvieras en NetBeans: <https://www.jetbrains.com/help/idea/netbeans.html>
- Si quieres aprender por tu cuenta: <https://www.jetbrains.com/help/idea/product-educational-tools.html>

5.2.5. Por qué debería elegir IntelliJ en lugar de VsCode para la codificación en Java

5.2.5.1. IDEA INTELLIJ:

Ventajas:

1. **Entorno integrado completo:** IntelliJ IDEA está diseñado específicamente para el desarrollo de Java y ofrece un conjunto completo de herramientas y características optimizadas para esta tarea.
2. **Análisis estático avanzado:** Proporciona un análisis de código en profundidad que detecta errores y problemas potenciales antes de la compilación.
3. **Depuración avanzada:** ofrece un potente conjunto de herramientas de depuración que ayudan a identificar y resolver problemas en el código.
4. **Refactorización guiada:** Proporciona herramientas para reorganizar y optimizar el código de forma segura, promoviendo buenas prácticas de programación.
5. **Compatibilidad con marcos y tecnologías Java:** Integración nativa con muchos marcos y tecnologías utilizados en el desarrollo Java, lo que facilita la creación de aplicaciones completas.
6. **Generación automática de código:** ayuda a los programadores a generar automáticamente fragmentos de código repetitivos, como captadores y definidores.
7. **Integración con herramientas de compilación:** facilita la integración con herramientas de compilación como Maven y Gradle.

8. **Soporte para pruebas unitarias:** Ofrece integración con marcos de prueba como JUnit para el desarrollo basado en pruebas.
9. **Facilidad de configuración:** Proporciona asistentes guiados para configurar de manera eficiente proyectos Java.

Contras:

1. **Mayor consumo de recursos:** Debido a su naturaleza integral y rica en funciones, IntelliJ IDEA puede consumir más recursos del sistema en comparación con IDE más livianas.
2. **Curva de aprendizaje:** Dado que ofrece una amplia gama de funciones, los principiantes pueden tardar un tiempo en familiarizarse con todas las herramientas disponibles.

5.2.5.2. VISUAL STUDIO CODE (VS CODE):

Ventajas:

1. **Ligero y rápido:** VSCode es un editor de código liviano y rápido, lo que lo hace ideal para proyectos más pequeños o para aquellos que prefieren una experiencia más ágil.
2. **Amplia gama de extensiones:** Tiene una amplia comunidad que desarrolla extensiones para diversas tecnologías y lenguajes, incluido Java.
3. **Versatilidad:** Si bien no está diseñado específicamente para Java, se puede personalizar para que funcione con Java a través de extensiones.
4. **Integración de control de versiones:** ofrece integración nativa con sistemas de control de versiones como Git.
5. **Curva de aprendizaje rápida:** Debido a su enfoque más ligero, puede resultar más sencillo para los principiantes comenzar a trabajar con él.

Contras:

1. **Funcionalidad limitada de Java:** Aunque existen extensiones de Java, VSCode no ofrece el mismo conjunto completo de herramientas optimizadas para Java que IntelliJ IDEA.
2. **Análisis menos profundo:** Las capacidades de análisis estático y corrección de código podrían no ser tan avanzadas como las de IntelliJ IDEA.
3. **Depuración limitada:** si bien ofrece depuración, es posible que no sea tan avanzada o completa como la de IntelliJ IDEA.
4. **Configuración manual del proyecto:** La configuración de proyectos Java puede requerir más pasos y configuración manual en comparación con IntelliJ IDEA.

5.2.6. Tarea

Debes entregar un documento *.pdf explicando que IDE has elegido para empezar a programar (más adelante lo puedes cambiar siquieres), justificando porqué lo has elegido.

Además envía una captura de pantalla en la que se vea el resultado del comando:

```
1  java --version
```

Y por último capturas de pantalla donde se pueda ver que editas el fichero fuente (HolaMundo.java), lo compilas y lo ejecutas dentro del IDE que has elegido (explica los pasos que has seguido)

⌚ 6 de diciembre de 2025

5.3. Taller UD01_03: Crear cuenta en GitHub

5.3.1. Qué es GitHub

Github es una plataforma en la nube basada en Git que permite a los desarrolladores almacenar, gestionar y colaborar en proyectos de código. Es el portafolio universal de los programadores.

Crear una cuenta es esencial para quien aprende o busca trabajar en programación porque: sirve como tu currículum técnico, donde muestras tus proyectos y evolución; te permite colaborar en proyectos open source para ganar experiencia real; y es una herramienta fundamental para el control de versiones y trabajo en equipo, usada por prácticamente todas las empresas tech.

5.3.2. Crea tu cuenta

Accede a la plataforma GitHub: <https://github.com/>

Pulsa sobre el botón [Sign Up] y sigue las instrucciones para crear tu cuenta.

Una vez creada tu cuenta, entra en tu página principal, por ejemplo la mia es esta: <https://github.com/martinezpenya> (`martinezpenya` es mi usuario de github) y realiza una captura de pantalla.

5.3.3. Solicitar corrección de los apuntes

Ahora, para probar nuestra nueva cuenta y colaborar con algún proyecto, no hay nada mejor que ayudar a mejorar los apuntes del profesor de Programación 😊.

Accedemos a la página de los apuntes en la que hemos detectado el error o queremos sugerir un cambio y en la parte superior derecha debe aparecer el icono:

The screenshot shows a GitHub repository page. The sidebar on the left lists the repository's structure: '1º Programacion (CFGs Desarrollo de Aplicaciones Multiplataforma)', 'UD00', 'UD01' (which is expanded to show '1. Introducción a Markdown'), 'Elementos de un programa informático', 'Ejercicios', 'Talleres', 'T01 NoMachine', 'T02 JDK e IDE', 'T03 GitHub', and 'T04 Markdown'. Below this is a section titled 'Introducción a Markdown' with a note: 'Markdown es un maravilloso lenguaje para escribir documentos de una manera sencilla de escribir, y que en todo momento mantenga un diseño legible que contengan elementos como secciones, párrafos, listas, vínculos e imágenes, etc. Pandoc <http://pandoc.org> ha extendido enormemente la sintaxis original de Markdown y ha añadido unas pequeñas nuevas características tales como notas al pie de página, citas y tablas. Lo más importante que hace Pandoc es hacer posible la generación de documentos en una amplia variedad de formatos desde Markdown: HTML, LaTeX/PDF, MSWord v'.

Esto nos llevará a crear un Fork del repositorio (este concepto lo aprenderás más adelante en el módulo de Entornos de Desarrollo):

The screenshot shows a GitHub fork confirmation page. At the top, it says 'martinezpenya / 1DAMProgramacion'. Below that is a search bar with 'Type ⏎ to search'. Underneath are navigation links: 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Security', and 'Insights'. The main content area shows the file path '1DAMProgramacion / docs / UD01 / UD01_T04_Markdown.md' and the status 'in main'. Below this is a large message: 'You need to fork this repository to propose changes.' followed by the text 'Sorry, you're not able to edit this repository directly—you need to fork it and propose your changes from there instead.' At the bottom are two buttons: 'Fork this repository' (highlighted in green) and 'Learn more about forks'.

Ahora debemos pulsar el botón **[Fork this repository]**, y a continuación veremos el código de la página en nuestro fork que es `MarkDown` (Puedes aprender más sobre `MarkDown` en el Taller 4):

You're making changes in a project you don't have write access to. Submitting a change will write it to a new branch in your fork [profeDAMCarlet/1DAMProgramacion](#), so you can send a pull request.

1DAMProgramacion / docs / UD01 / UD01_T04_Markdown.md in [main](#)

Commit changes...

```

1 # Taller UD01_T04: Markdown
2
3 ## Introducción a Markdown
4
5 
6
7 **Markdown** nace como herramienta de **conversión de texto plano a HTML**. Fue creada en 2004 por John Gruber, y se distribuye de manera gratuita bajo una \[licencia BSD\](https://es.wikipedia.org/wiki/Licencia\_BSD).
8
9 Markdown es un maravilloso **lenguaje** para escribir documentos de una manera **sencilla de escribir, y que en todo momento mantenga un diseño legible** que contengan elementos como *secciones*, *párrafos*, *listas*, *vínculos* e *imágenes*, *etc*. Pandoc [http://pandoc.org](http://pandoc.org/) ha extendido enormemente la \[sintaxis original de Markdown\](http://daringfireball.net/projects/markdown/) y ha añadido unas pequeñas nuevas características tales como notas al pie de página, citas y tablas. Lo más importante que hace Pandoc es hacer posible la generación de documentos en una amplia variedad de formatos desde Markdown, HTML, LaTeX/PDF, MSWord y Slides.

```

Ahora debemos buscar el texto a modificar y una vez hayamos cambiado algo del documento se activará el botón [Commit changes...]:

You're making changes in a project you don't have write access to. Submitting a change will write it to a new branch in your fork [profeDAMCarlet/1DAMProgramacion](#), so you can send a pull request.

1DAMProgramacion / docs / UD01 / UD01_T04_Markdown.md in [main](#)

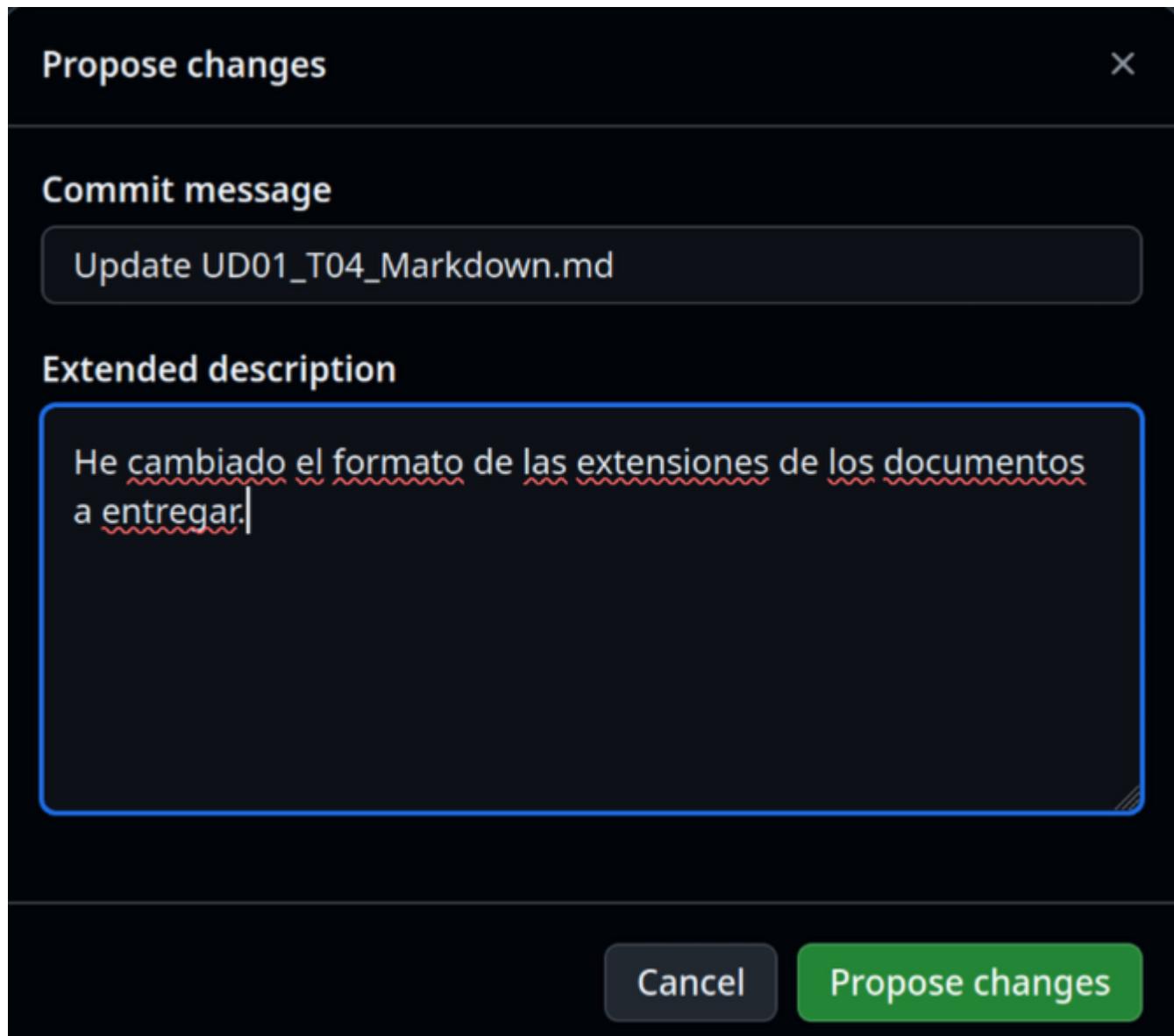
Commit changes...

```

1 # Taller UD01_T04: Markdown
2
3 ## Introducción a Markdown
4
5 
6
7 **Markdown** nace como herramienta de **conversión de texto plano a HTML**. Fue creada en 2004 por John Gruber, y se distribuye de manera gratuita bajo una \[licencia BSD\](https://es.wikipedia.org/wiki/Licencia\_BSD).

```

Ahora debes explicar cual ha sido la modificación que hemos realizado y pulsar el botón [Propose changes]:



Todavía no hemos terminado! ahora hay que comunicar los cambios propuestos en nuestro Fork al propietario del repositorio, para que los visualice y valore si los quiere incluir en la página de documentación. Para ello debemos pulsar el botón [Create pull request]:

The screenshot shows the GitHub interface for comparing changes between two repositories. At the top, it displays the repositories: 'martinezpenya / 1DAMProgramacion' and 'profeDAMCarlet / 1DAMProgramacion'. Below this, there are tabs for Code, Issues, Pull requests, Actions, Projects, Security, and Insights. The 'Code' tab is selected.

The main area is titled 'Comparing changes' and instructs the user to choose two branches to see what's changed or to start a new pull request. It includes links to 'compare across forks' and 'learn more about diff comparisons'.

Below this, there are dropdown menus for 'base repository', 'base: main', 'head repository', 'head: patch-1', and 'compare: patch-1'. A red arrow points from the 'Create pull request' button to the right.

The comparison summary indicates 1 commit, 1 file changed, and 1 contributor. The commit details show a single commit made on Sep 6, 2025, by 'profeDAMCarlet' with a timestamp of 'authored 1 minute ago'. The commit message is 'Update UD01_T04_Markdown.md ...'. The commit is verified and has a hash '5f152ae'. The commit details also mention 'Showing 1 changed file with 1 addition and 1 deletion.'

The code diff view shows a single file 'docs/UD01/UD01_T04_Markdown.md' with 2 changes. The first change is a deletion at line 393, position 4, and the second is an insertion at line 393, position 4, both with a count of 1. The message for the change is 'Como tarea, se propone:'.

Ahora podemos modificar el mensaje (pero no hace falta), directamente pulsamos sobre el botón [Create pull request]:

The screenshot shows the GitHub interface for creating a pull request. At the top, the repository 'martinezpenya / 1DAMProgramacion' is selected. Below it, the navigation bar includes 'Code' (highlighted), 'Issues', 'Pull requests', 'Actions', 'Projects', 'Security', and 'Insights'. The main section is titled 'Open a pull request' with the sub-instruction 'Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. [Learn more about diff comparisons here.](#)' Below this, the pull request configuration shows 'base repository: martinezpenya/1DAMProgra...' and 'base: main' compared to 'head repository: profeDAMCarlet/1DAMProgra...' and 'compare: patch-1'. The 'Add a title' field contains 'Update UD01_T04_Markdown.md'. The 'Add a description' field contains the text 'He cambiado el formato de las extensiones de los documentos a entregar.' A red arrow points to the green 'Create pull request' button at the bottom right of the form.

Ahora si, deberías ver una página similar a la siguiente, de la que también deberás obtener una captura y adjuntarla al .pdf , y además explicar los 4 campos que hay redondeados:

Update UD01_T04_Markdown.md #1

profedAMCarlet wants to merge 1 commit into [martinezpenya:main](#) from [profedAMCarlet:patch-1](#)

Conversation 0 Commits 1 Checks 0 Files changed 1

profedAMCarlet commented now

He cambiado el formato de las extensiones de los documentos a entregar.

No conflicts with base branch
Changes can be cleanly merged.

Add a comment

Write Preview

Add your comment here...

Markdown is supported | Paste, drop, or click to add files

[Close pull request](#) [Comment](#)

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

ProTip! Add `.patch` or `.diff` to the end of URLs for Git's plaintext views.

Reviewers
No reviews
Still in progress? [Convert to draft](#)

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.
None yet

Notifications [Unsubscribe](#)
You're receiving notifications because you authored the thread.

1 participant

Allow edits by maintainers [?](#)

Como resumen:

1. Hemos creado un fork de un repositorio.
2. Hemos modificado un archivo en nuestro fork.
3. Hemos comparado nuestro fork con el original y hemos creado un pull request con las diferencias.

Ahora pueden pasar dos cosas, que el propietario del repositorio original acepte nuestros cambios, y por tanto pasaremos a ser colaboradores del repositorio original.

O bien, que el cambio no sea aceptado.

En cualquiera de los dos casos, si adjuntas las capturas y explicas los campos la actividad estará correcta.

En este caso concreto se ha aceptado la modificación:

Update UD01_T04_Markdown.md #1

Merged martinezpenya merged 1 commit into `martinezpenya:main` from `profeDAMCarlet:patch-1` 1 minute ago

Conversation 1 Commits 1 Checks 0 Files changed 1

profeDAMCarlet commented 10 minutes ago

He cambiado el formato de las extensiones de los documentos a entregar.

martinezpenya commented 1 minute ago

Perfecto, gracias!

martinezpenya closed this 1 minute ago

martinezpenya merged commit `8052475` into `martinezpenya:main` 1 minute ago

5.3.4. Tarea

- Crea un documento `.pdf` donde debes adjuntar la captura de tu perfil de github.
- Añade una **captura** de pantalla donde se vea que has solicitado el **pull request** y que estás esperando a que se integre en el repositorio original.
- Además, **explica** que significan cada uno de los **4 apartados** señalados en la captura.

Adjunta el documento `.pdf` con las capturas y las explicaciones a la tarea de AULES.

29 de septiembre de 2025

5.4. Taller UD01_T03: Markdown

5.4.1. Introducción a Markdown

Markdown nace como herramienta de **conversión de texto plano a HTML**. Fue creada en 2004 por John Gruber, y se distribuye de manera gratuita bajo una [licencia BSD](#).

Markdown es un maravilloso **lenguaje** para escribir documentos de una manera **sencilla de escribir, y que en todo momento mantenga un diseño legible** que contengan elementos como *secciones, párrafos, listas, vínculos e imágenes, etc.* Pandoc <http://pandoc.org> ha extendido enormemente la [sintaxis original de Markdown](#) y ha añadido unas pequeñas nuevas características tales como notas al pie de página, citas y tablas. Lo más importante que hace Pandoc es hacer posible la generación de documentos en una amplia variedad de formatos desde Markdown, HTML, LaTeX/PDF, MSWord y Slides.

Este método te permitirá añadir formatos tales como **negritas, cursivas o enlaces**, utilizando texto plano, lo que permitirá hacer de tu escritura algo más simple y eficiente al evitar distracciones.

Con Markdown **no vas a reemplazar todo**, sino cubrir las funcionalidades más comunes que se requieren para escribir un documento relativamente complicado.

5.4.2. Para qué sirve Markdown

Markdown será perfecto para ti sobre todo **si públicas de manera constante en Internet**, donde el lenguaje HTML está más que presente: WordPress, Squarespace, Jekyll...

Pero no estoy hablando solo de [blogs](#) o páginas web. **Servicios** como Trello o **foros** como Stackoverflow también soportan este lenguaje, y con el paso del tiempo encontrarás aún más lugares que lo utilicen.

Además, Markdown está cada vez más extendido en el **mundo “offline”**. Nada te impedirá utilizar este lenguaje para **tomar notas y apuntes** de tus clases o reuniones en una determinada [aplicación](#) (incluso podrías **escribir un libro con él**, ya que puedes exportar fácilmente el resultado final a un formato ePub).

Gracias a la simplicidad de su sintaxis podrás utilizarlo siempre que necesites escribir y dar formato rápidamente, sobre todo si quieras hacerlo desde dispositivos móviles.

5.4.3. Por qué utilizar Markdown

5.4.3.1. VENTAJAS

- **Markdown para todo.** Para crear apuntes, documentos, notas, sitios web, libros, documentación técnica, etc. de forma off-line.
- **Markdown transportable.** Este tipo de formato siempre será **compatible con todas las plataformas** que utilices, así que utilizar Markdown es una manera de mantener todo tu contenido siempre accesible desde cualquier dispositivo (smartphones, ordenadores de escritorio, tablets...), ya que en cualquiera de ellas siempre encontrarás [las aplicaciones adecuadas](#) para leer y editar este tipo de contenido.
- Ideal para escribir un libro, pues permite la exportación fácil en ePub, PDF...

Si en el futuro Microsoft Word desapareciese perderías acceso a todo el contenido que has creado durante años utilizando dicho procesador. Así que lo más inteligente para evitar eso es **generar tu contenido de la manera más sencilla posible**: utilizando texto plano.

5.4.3.2. DESVENTAJAS

- No tiene muchas funcionalidades (esto es lo que lo hace muy compatible).
- Al no tener todas las opciones de un procesador de textos a veces tendrás que combinar Markdown con HTML para lograr ciertos formatos.

5.4.4. Editores para Markdown

5.4.4.1. OFF-LINE

- **Typora**
- MarkdownPad
- HarooPad
- Markdown Monster

- ...

5.4.4.2. ONLINE

- Dillinger
- GitHub
- ...

5.4.5. Párrafos y saltos de línea

Si queremos generar un nuevo párrafo en Markdown simplemente separa el texto mediante una línea en blanco (**pulsando dos veces intro**).

Al igual que sucede con HTML, **Markdown no soporta dobles líneas en blanco**, así que si intentas generarlas estas se convertirán en una sola al procesarse.

Para realizar un salto de línea y empezar una frase en una línea siguiente dentro del mismo párrafo, tendrás que pulsar **dos veces la barra espaciadora antes de pulsar una vez intro**.

Por ejemplo si quisieras escribir un poema quedaría tal que así:

«*La tierra estaba seca, No había ríos ni fuentes. Y brotó de tus ojos.*

Donde cada verso tiene **dos espacios en blanco al final**.

5.4.6. Encabezados

Las **# almohadillas** son uno de los métodos utilizados en Markdown para crear encabezados. Debes usarlos añadiendo **uno por cada nivel**.

Es decir,

```
1 # Encabezado 1
2 ## Encabezado 2
3 ### Encabezado 3
4 #### Encabezado 4
5 ##### Encabezado 5
6 ##### Encabezado 6
```

Se corresponde con:

También puedes cerrar los encabezados con el mismo número de almohadillas, por ejemplo escribiendo `### Encabezado 3 ###`. Pero la única finalidad de esto es un **motivo estético**.

5.4.7. Texto básico

Un párrafo no requiere sintaxis especial.

Para aplicar **negrita** al texto, se escribe entre dos asteriscos.

Para aplicar *cursiva* al texto, se escribe entre un solo asterisco.

Para tachar el texto, se escribirá dos virgulillas antes y dos después de éste.

```
1 Este texto es en **negrita**.
2 Este texto es en *italica*.
3 Este texto está ~~tachado~~.
4 Este texto es en ambos ***negrita e italica***.
```

Se corresponde a:

Este texto es en ****negrita****.

Este texto es en *italica*.

Este texto está ~~tachado~~.

Este texto es en ambos ***negrita e italica***.

En Markdown no podemos subrayar el texto. Sin embargo, podremos añadir la etiqueta de html underline \u.

```
1 Este texto está <u>subrayado</u>
```

Este texto está **subrayado**

Para **ignorar los caracteres** de formato de Markdown, ponga \ antes del carácter:

5.4.8. Citas

Las citas se generan utilizando el carácter *mayor que* > al comienzo del bloque de texto.

```
1 > No hay que ir para atrás ni para darse impulso. — Lao Tsé.
```

No hay que ir para atrás ni para darse impulso. — Lao Tsé.

Si la cita en cuestión se compone de **varios párrafos**, deberás añadir el mismo símbolo > al comienzo de cada uno de ellos.

5.4.9. Listas

5.4.9.1. LISTAS ORDENADAS

Para crear **listas numeradas**, empieza una línea con 1. or 1).

No debes mezclar los formatos dentro de la misma lista. No es necesario especificar los números. GitHub lo hace por tí.

```
1 1. ítem 1 de la lista.
2 1. Siguiente ítem de la lista.
3 1. Siguiente ítem, el tercero, de la lista.
```

Se corresponde con:

1. Ítem 1 de la lista.
2. Siguiente ítem de la lista.
3. Siguiente ítem, el tercero, de la lista.

5.4.9.2. LISTAS NO ORDENADAS

Para crear listas no numeradas, o de viñetas, empieza una línea con * , - o + , pero no mezcles los formatos dentro de la misma lista. (No mezclar formatos de viñetas, como * y + por ejemplo, dentro del mismo documento).

```
1 * ítem 1 de la lista.
2 * Siguiente ítem de la lista.
3 * Siguiente ítem, el tercero, de la lista.
```

Se corresponde con:

- Ítem 1 de la lista.
- Siguiente ítem de la lista.
- Siguiente ítem, el tercero, de la lista.

También podremos combinar ambos tipos de listas. Como por ejemplo:

1. element de llista 2 - element de llista 2.2
 - element de llista 2.2.1
 - element de llista 2.2.2

5.4.9.3. LISTAS DE TAREAS

Para crear listas de tareas basta con que empiece la línea con - [], si queremos que no esté el check marcado, y - [x] , si queremos que esté el check marcado.

```
1 - [x] regar plantas.
2 - [ ] realizar ejercicios de programación.
```

Se corresponde con:

✓egar plantas.

realizar ejercicios de programación.

5.4.10. Tablas

Las tablas no forman parte de la especificación principal de Markdown, pero Adobe, en cierta forma, las admite.

Para generar una tabla utiliza la barra vertical | para generar filas y columnas.

Si insertamos guiones --- dentro de una celda crearemos el encabezado de la tabla.

```
1 | encabezado1 | encabezado2 | encabezado3 |
2 |---|---|---|
3 | celda 1.1 | celda 1.2 | celda 1.3 |
4 | celda 2.1 | celda 2.2 | celda 2.3 |
```

Quedaría:

encabezado1	encabezado2	encabezado3
celda 1.1	celda 1.2	celda 1.3
celda 2.1	celda 2.2	celda 2.3

Si queremos una **celda con más de una línea** de texto podremos insertar \n (o Shift+Intro) al final de ésta.

5.4.11. Enlaces

Para generar un enlace en Markdown se debe poner un código con dos partes:

- [texto del enlace], que es el texto que se va a mostrar,
- Y después (nombrefichero.md), que es la URL o el nombre de archivo al que se va a vincular.

```
1 [link text](file-name.md)
```

Un ejemplo:

[enlace a web del centro](https://iesmre.com)

La visualización del ejemplo anterior:

[enlace a web del centro](https://iesmre.com)

5.4.12. Imágenes

Para insertar una imagen se debe poner un código con dos partes:

- ! [texto alternativo], que es el texto que se va a mostrar si la imagen no pudiera visualizarse,
- Seguido de (nombrefichero.extension), que es el archivo imagen (con su dirección).

```
1 [texto alternativo](file-name.md)
```

Un ejemplo:

[logo markdown](assets/mardown_logo.png)

La visualización de la imagen anterior:

logo markdown

5.4.13. Código de bloque

Uno de los puntos más útiles de Markdown a la hora de crear un documento con texto específico de informática es que admite la colocación de bloques de código tanto en línea como en un bloque "delimitado" independiente entre frases.

Para ello utilizaremos:

- Dos comillas invertidas `` si queremos escribir código dentro de la misma línea de texto del párrafo.
- Si queremos crear un bloque de código multilínea, con un lenguaje específico, pondremos ``` seguido del nombre del lenguaje del bloque .

Unos ejemplos:

- En la misma línea:

...estamos escribiendo un párrafo ``insertar el bloque y seguimos escribiendo...

- Un bloque de código:

```javascript y escribimos el código.

```
1 function holamundo(){
2 console.log ("hola mundo web");
3 }
```

#### 5.4.14. Línea horizontal

Para crear una línea horizontal, de separación de contenido por ejemplo, se añaden tres guiones: ---

Visualización:

---

#### 5.4.15. Insertar emojis

Para insertar emojis basta con utilizar : seguido del nombre del emoji y cerrar con otro : .

Podemos observar, que en algunos editores markdown, al escribir, por ejemplo, :a nos muestra todos los emojis con la inicial a.

Por ejemplo: :star :

Visualización: ★

#### 5.4.16. Crear diagrama de flujo

Cuando queremos crear documentos con elementos gráficos como diagramas de flujo, debemos generar una especie de *código* para construirlos.

- Por eso, comenzaremos introduciendo la línea de inicio: ```flow
- Es conveniente asignar un nombre (por ejemplo: st, op, cond, e...) a cada elemento que conforma el diagrama; así, después podremos unir todos estos.
- Forma de inicio: st=>start: Nombre
- Forma de fin: e=>end: Nombre
- Rectángulo: op=>operation: texto de nombre
- Condición: cond=>condition: texto de la condición (Si o No?)
- Subrutina: sub1=>subroutine: nombre subtarea
- EntradaSalida: io1=>inputoutput: nombre elemento entrada/salida
- Líneas: st->op->cond
- Caminos de condiciones: cond(yes)\*\*->\*\*e y cond(no)->op
- Línea de cierre: ```

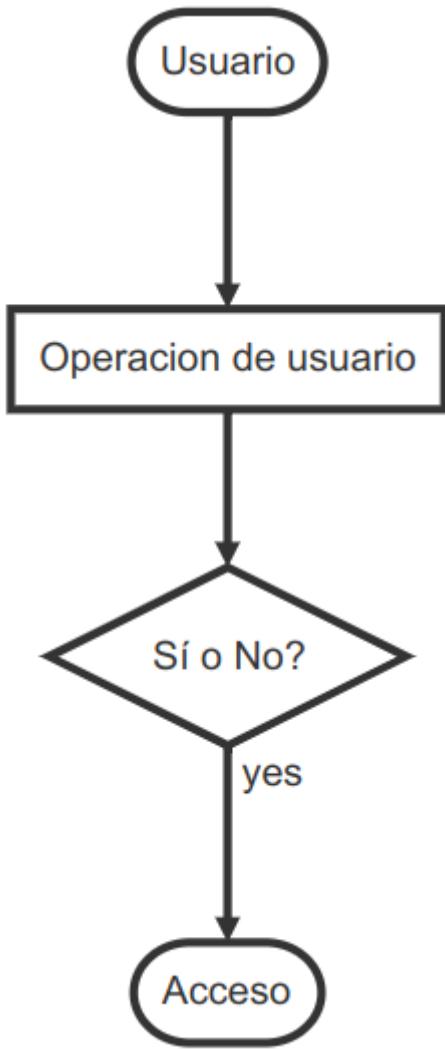
Ejemplo:

```

1 ````flow
2 st=>start: Usuario
3 e==>end: Acceso
4 op=>operation: Operacion de usuario
5 cond=>condition: Sí o No?
6 st->op->cond
7 cond(yes)->e
8 cond(no)->op
9 ```

```

Visualización:



Intenta realizar un diagrama para "programar" un almuerzo. En él, deberás dar los **buenos días**, indicar que **es hora del descanso**, y preguntar si **alguién quiere almorzar**. Si no hay nadie que quiera almorzar contigo, debes **ir a otro grupo de amigos** y volver a indicar que **es hora del descanso**. Si alguien sí quiere almorzar **escribe en la pizarra que os vais a almorzar y sal al patio**.

#### 5.4.17. Crear secuencias

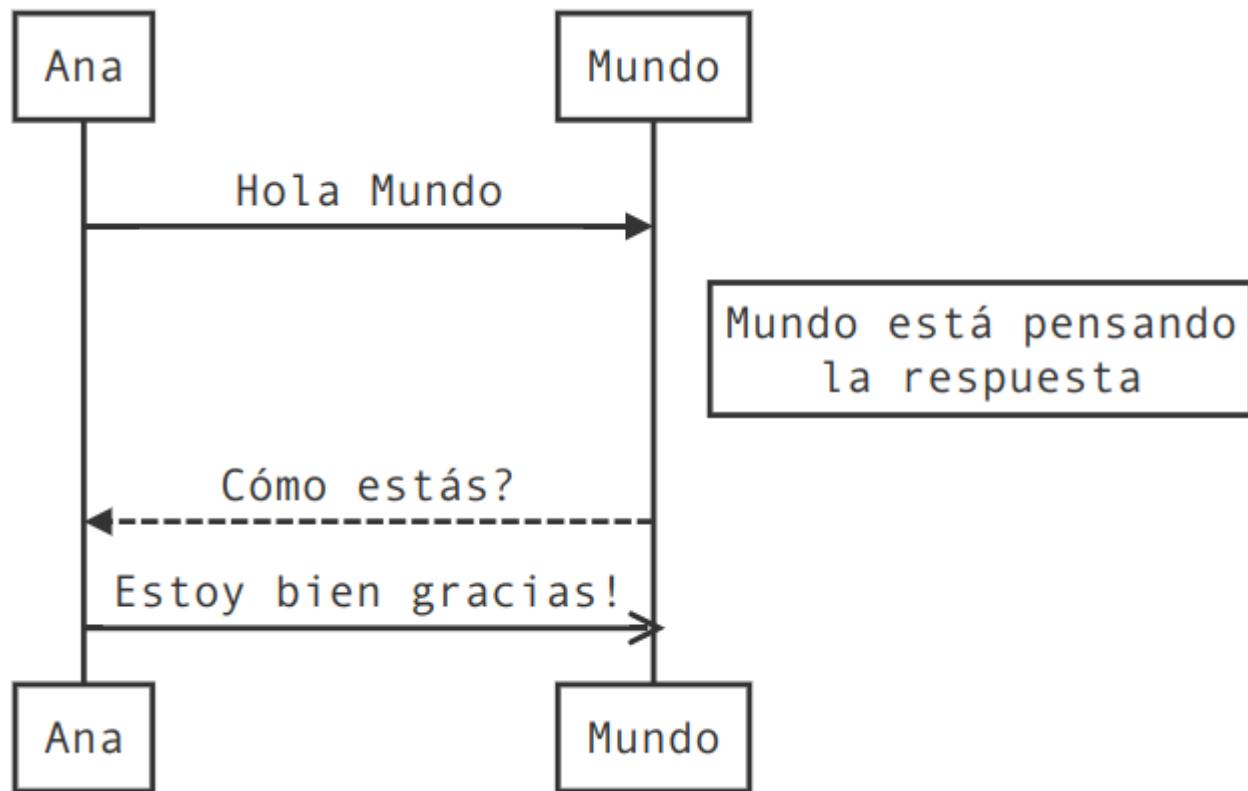
En la secuenciación podemos observar que es bastante parecido a la creación de diagramas; pero la primera línea (crear un bloque de código) no será **flow** sino **sequence**.

```

1 ````sequence
2 Ana->Mundo: Hola Mundo
3 Note right of Mundo: Mundo está pensando\nla respuesta
4 Mundo-->Ana: Cómo estás?
5 Ana->>Mundo: Estoy bien gracias!
6 ```

```

Visualización:



#### 5.4.18. Crear índice

Para crear el índice a partir de los encabezados creados debemos insertar `[toc]`.

#### 5.4.19. Tarea

Como tarea, se propone:

- Crear un documento markdown en tu editor markdown favorito (por ejemplo Typora o VSCode) que documente información acerca de tí mismo.
- En dicho documento crear título, índice.
- Añadir 4 encabezados principales (y otros encabezados secundarios dentro de éstos) en el que hables por ejemplo de: *Tus datos, Currículum, Aficiones y Otros datos de interés*. No hace falta que indiques información personal relevante. (O te la puedes inventar)
- Se valorará la inclusión de distintos elementos como: negrita-cursiva-subrayado, listas ordenadas-desordenadas-tareas, enlaces, imágenes, citas, código, etc.
- Si te atreves con ello, crea un diagrama de flujo en el que indiques los pasos que realizas un sábado por la mañana.
- Exporta el documento a pdf.

**Subir a la plataforma AULES un documento Markdown (.md) y otro documento PDF (.pdf) que sea la exportación del primero.**

23 de septiembre de 2025

## 3. UD02

### 6. 3.1 Utilización de Objetos y Clases



#### 6.1. Introducción a la POO

**Orientado a objetos** hace referencia a una forma diferente de acometer la tarea del desarrollo de software, frente a otros modelos como el de la programación imperativa, la programación funcional o la programación lógica. Supone una reconsideración de los métodos de programación, de la forma de estructurar la información y, ante todo, de la forma de pensar en la resolución de problemas.

La **programación orientada a objetos (POO)** es un modelo para la elaboración de programas que ha impuesto en los últimos años. Este auge se debe, en parte, a que esta forma de programar está fuertemente basada en la representación de la realidad; pero también a que refuerza el uso de buenos criterios aplicables al desarrollo de programas.

#### Acción

La orientación a objetos no es un tipo de lenguaje de programación. Es una metodología de trabajo para crear programas.

En POO, un programa es una colección de objetos que se relacionan entre sí de distintas formas.

#### 6.2. Características de la POO

Cuando hablamos de Programación Orientada a Objetos, existen una serie de características que se deben cumplir. Cualquier lenguaje de programación orientado a objetos las debe contemplar. Las características más importantes del paradigma de la programación orientada a objetos son:

- **Abstracción.** Es el proceso por el cual definimos las características más importantes de un objeto, sin preocuparnos de cómo se escribirán en el código del programa, simplemente lo definimos de forma general. En la Programación Orientada a Objetos la herramienta más importante para soportar la abstracción es la clase. Básicamente, una clase es un tipo de dato que agrupa las características comunes de un conjunto de objetos. Poder ver los objetos del mundo real que deseamos trasladar a nuestros

programas, en términos abstractos, resulta de gran utilidad para un buen diseño del software, ya que nos ayuda a comprender mejor el problema y a tener una visión global de todo el conjunto. Por ejemplo, si pensamos en una clase `Vehículo` que agrupa las características comunes de todos ellos, a partir de dicha clase podríamos crear objetos como `Coche` y `Camión`. Entonces se dice que `Vehículo` es una abstracción de `Coche` y de `Camión`.

- **Modularidad.** Una vez que hemos representado el escenario del problema en nuestra aplicación, tenemos como resultado un conjunto de objetos software a utilizar. Este conjunto de objetos se crean a partir de una o varias clases. Cada clase se encuentra en un archivo diferente, por lo que la modularidad nos permite modificar las características de la clase que define un objeto, sin que esto afecte al resto de clases de la aplicación.
- **Encapsulación.** También llamada "ocultamiento de la información". La encapsulación o encapsulamiento es el mecanismo básico para ocultar la información de las partes internas de un objeto a los demás objetos de la aplicación. Con la encapsulación un objeto puede ocultar la información que contiene al mundo exterior, o bien restringir el acceso a la misma para evitar ser manipulado de forma inadecuada. Por ejemplo, pensemos en un programa con dos objetos, un objeto `Persona` y otro `Coche`. `Persona` se comunica con el objeto `Coche` para llegar a su destino, utilizando para ello las acciones que `Coche` tenga definidas como por ejemplo conducir. Es decir, `Persona` utiliza `Coche` pero no sabe cómo funciona internamente, sólo sabe utilizar sus métodos o acciones.
- **Jerarquía.** Mediante esta propiedad podemos definir relaciones de jerarquías entre clases y objetos. Las dos jerarquías más importantes son la jerarquía "es un" llamada generalización o especialización y la jerarquía "es parte de", llamada agregación. Conviene detallar algunos aspectos:
  - La generalización o especialización, también conocida como herencia, permite crear una clase nueva en términos de una clase ya existente (herencia simple) o de varias clases ya existentes (herencia múltiple). Por ejemplo, podemos crear la clase `CochedeCarreras` a partir de la clase `Coche`, y así sólo tendremos que definir las nuevas características que tenga.
  - La agregación, también conocida como inclusión, permite agrupar objetos relacionados entre sí dentro de una clase. Así, un `Coche` está formado por `Motor`, `Ruedas`, `Frenos` y `Ventanas`. Se dice que `Coche` es una agregación y `Motor`, `Ruedas`, `Frenos` y `Ventanas` son agregados de `Coche`.
- **Polimorfismo.** Esta propiedad indica la capacidad de que varias clases creadas a partir de una antecesora realicen una misma acción de forma diferente. Por ejemplo, pensemos en la clase `Animal` y la acción de expresarse. Nos encontramos que cada tipo de `Animal` puede hacerlo de manera distinta, los `Perros` ladran, los `Gatos` maullan, las `Personas` hablamos, etc. Dicho de otra manera, el polimorfismo indica la posibilidad de tomar un objeto (de tipo `Animal`, por ejemplo), e indicarle que realice la acción de expresarse, esta acción será diferente según el tipo de mamífero del que se trate.

## 6.3. Objetos y Clases

### 6.3.1. Características de los objetos

En este contexto, un objeto de software es una representación de un objeto del mundo real, compuesto de una serie de características y un comportamiento específico. Pero ¿qué es más concretamente un objeto en Programación Orientada a Objetos? Veámoslo.

#### Definición

Un objeto es un conjunto de datos con las operaciones definidas para ellos. Los objetos tienen un estado y un comportamiento.

Por tanto, estudiando los objetos que están presentes en un problema podemos dar con la solución a dicho problema. Los objetos tienen unas características fundamentales que los distinguen:

- **Identidad.** Es la característica que permite diferenciar un objeto de otro. De esta manera, aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí. Puede ser una dirección de memoria, el nombre del objeto o cualquier otro elemento que utilice el lenguaje para distinguirlos. Por ejemplo, dos vehículos que hayan salido de la misma cadena de fabricación y sean iguales aparentemente, son distintos porque tienen un código que los identifica.
- **Estado.** El estado de un objeto viene determinado por una serie de parámetros o atributos que lo describen, y los valores de éstos. Por ejemplo, si tenemos un objeto `Coche`, el estado estaría definido por atributos como `Marca`, `Modelo`, `Color`, `Cilindrada`, etc.
- **Comportamiento.** Son las acciones que se pueden realizar sobre el objeto. En otras palabras, son los métodos o procedimientos que realiza el objeto. Siguiendo con el ejemplo del objeto `Coche`, el comportamiento serían acciones como: `arrancar()`, `parar()`, `acelerar()`, `frenar()`, etc. Definición de clases.

Una clase java se escribe en un fichero con extensión `.java` que tiene el mismo nombre que la clase. Por ejemplo la clase `Vehículo` se escribiría en el fichero `Vehiculo.java`.

Cuando la clase se compila se obtiene un fichero con el mismo nombre que la clase y extensión `.class`. Ej.: `Vehiculo.class`.

### Acción

Los identificadores de clase siguen las mismas reglas que otros identificadores de Java (contienen carácter alfanuméricos y especiales, no pueden comenzar por un dígito, no pueden coincidir con una palabra reservada, etc.). Por convenio los identificadores de las clases comienzan por mayúsculas.

#### 6.3.2. Propiedades y métodos de los objetos

Como acabamos de ver todo objeto tiene un estado y un comportamiento. Concretando un poco más, las partes de un objeto son:

- **Campos, Atributos o Propiedades:** Parte del objeto que almacena los datos. También se les denomina Variables Miembro. Estos datos pueden ser de cualquier tipo primitivo (`boolean`, `char`, `int`, `double`, etc) o ser a su vez otro objeto. Por ejemplo, un objeto de la clase `Coche` puede tener un objeto de la clase `Ruedas` (o más concretamente cuatro).
- **Métodos o Funciones Miembro:** Parte del objeto que lleva a cabo las operaciones sobre los atributos definidos para ese objeto.

La idea principal es que el objeto reúne en una sola entidad los datos y las operaciones, y para acceder a los datos privados del objeto debemos utilizar los métodos que hay definidos para ese objeto.

La única forma de manipular la información del objeto es a través de sus métodos. Es decir, si queremos saber el valor de algún atributo, tenemos que utilizar el método que nos muestre el valor de ese atributo. De esta forma, evitamos que métodos externos puedan alterar los datos del objeto de manera inadecuada. Se dice que los datos y los métodos están encapsulados dentro del objeto.

##### 6.3.2.1. ATRIBUTOS

Los atributos representan la información que almacenan los objetos de la clase. Los atributos son declaraciones de variables dentro de la clase.

Se sigue la siguiente sintaxis (los corchetes indican opcionalidad):

```
1 [ámbito] tipo nombreDelAtributo;
2 [ámbito] tipo nombreDelAtributo1, nombreDelAtributo2, ...;
```

donde ...

- **ámbito** permite indicar desde qué clases es accesible el atributo.
- **tipo** indica el tipo de dato del atributo.
- **nombreDelAtributo** es el identificador del atributo.

##### 6.3.2.2. MÉTODOS

Los métodos determinan qué puede hacer un objeto de la clase, es decir, su comportamiento.

Los métodos realizan algún tipo de acción o tarea y, en ocasiones, devuelven un resultado.

Para realizar su trabajo puede ser necesario que pasemos al método cierta información. Por ejemplo, cuando llamamos al método `round` de la clase `Math`, para redondear un número real, debemos indicar al método cual es el número que queremos redondear. A esa información que pasamos a los métodos se le llama **parámetros o argumentos**.

### Ejemplo

```
1 //Al llamar a Math.round, pasamos al método un parámetro
2 int redondeado1 = Math.round(numero);
3 int redondeado2 = Math.round(125.687);
4 ...
5 //Al llamar a Math.pow, pasamos al método dos parámetros
6 int pot1 = Math.pow(a,b);
7 int pot2 = Math.pow(a,6);
8 ...
```

En la definición de un método se distinguen dos partes

- **La cabecera**, en la cual se indica información relevante sobre el método.
- **El cuerpo**, que contiene las instrucciones mediante las cuales el método realiza su tarea.

Para definirlos, se sigue la siguiente sintaxis (los corchetes indican opcionalidad):

## Ejemplo

```

1 public static void main (String[] args)
2 [ámbito] [static] tipoDevuelto nombreDelMetodo ([parámetros]){
3 //Cuerpo del método (instrucciones)
4 ...
5 ...
6 ...
7 }
```

donde...

- **ámbito** permite indicar desde qué clases es accesible el método.
- **static**, cuando aparece, indica que el método es estático.
- **tipoDevuelto** indica el tipo de dato que devuelve el método. La palabra reservada **void** (que no es ningún tipo de dato), indicaría que el método no devuelve nada.
- **nombreDelMetodo** es el identificador del método
- **parámetros** es una lista, separada por comas, de los parámetros que recibe el método. De cada parámetro se indica el **tipo** y un **identificador**.

### 6.3.3. Interacción entre objetos

Dentro de un programa los objetos se comunican llamando unos a otros a sus métodos. Los métodos están dentro de los objetos y describen el comportamiento de un objeto cuando recibe una llamada a uno de sus métodos. En otras palabras, cuando un objeto, `objeto1`, quiere actuar sobre otro, `objeto2`, tiene que ejecutar uno de sus métodos. Entonces se dice que el `objeto2` recibe un mensaje del `objeto1`.

Un mensaje es la acción que realiza un objeto. Un método es la función o procedimiento al que se llama para actuar sobre un objeto.

Los distintos mensajes que puede recibir un objeto o a los que puede responder reciben el nombre de **protocolo** de ese objeto.

El proceso de interacción entre objetos se suele resumir diciendo que se ha "enviado un mensaje" (hecho una petición) a un objeto, y el objeto determina "qué hacer con el mensaje" (ejecuta el código del método). Cuando se ejecuta un programa se producen las siguientes acciones:

- **Creación** de los objetos a medida que se necesitan.
- **Comunicación** entre los objetos mediante el envío de mensajes unos a otros, o el usuario a los objetos.
- **Eliminación** de los objetos cuando no son necesarios para dejar espacio libre en la memoria del computador.

## Recuerda

Los objetos se pueden comunicar entre ellos invocando a los métodos de los otros objetos.

### 6.3.4. Clases

Hasta ahora hemos visto lo que son los objetos. Un programa informático se compone de muchos objetos, algunos de los cuales comparten la misma estructura y comportamiento. Si tuviéramos que definir su estructura y comportamiento del objeto cada vez que queremos crear un objeto, estaríamos utilizando mucho código redundante. Por ello lo que se hace es crear una clase, que es una descripción de un conjunto de objetos que comparten una estructura y un comportamiento común. Y a partir de la clase, se crean tantas "copias" o "instancias" como necesitemos. Esas copias son los objetos de la clase.

## Recuerda

Las clases constan de datos y métodos que resumen las características comunes de un conjunto de objetos. Un programa informático está compuesto por un conjunto de clases, a partir de las cuales se crean objetos que interactúan entre sí.

En otras palabras, una clase es una plantilla o prototipo donde se especifican:

- Los **atributos** comunes a todos los objetos de la clase.

- Los **métodos** que pueden utilizarse para manejar esos objetos.

Para declarar una clase en Java se utiliza la palabra reservada `class`. La declaración de una clase está compuesta por:

- **Cabecera de la clase.** La cabecera es un poco más compleja que como aquí definimos, pero por ahora sólo nos interesa saber que está compuesta por una serie de modificadores, en este caso hemos puesto `public` que indica que es una clase pública a la que pueden acceder otras clases del programa, la palabra reservada `class` y el nombre de la clase.
- **Cuerpo de la clase.** En él se especifican encerrados entre llaves los atributos y los métodos que va a tener la clase.

### Ejemplo

```

1 //Paquete al que pertenece la clase
2 package NombreDePaquete;
3
4 //Paquetes que importa la clase
5 import ...
6
7 ...
8
9 public class NombreDeLaClase {
10 // Atributos de la clase
11 ...
12 ...
13 ...
14 // Métodos de la clase
15 ...
16 ...
17 ...
18 }
```

### Ejemplo

En la unidad anterior ya hemos utilizado clases, aunque aún no sabíamos su significado exacto. Por ejemplo, en los ejemplos de la unidad o en la tarea, estábamos utilizando clases, todas ellas eran clases principales, no tenían ningún atributo y el único método del que disponían era el método `main()`.

### Ejemplo

También es una clase `Math` y su método era `random()`, el que nos permitía usar números aleatorios.

### Ejemplo

El método `main()` se utiliza para indicar que se trata de una clase principal, a partir de la cual va a empezar la ejecución del programa. Este método no aparece si la clase que estamos creando no va a ser la clase principal del programa.

#### 6.3.4.1. ¿QUÉ SIGNIFICA `public class`?

Significa que la clase que se define es pública. Una clase pública es una clase accesible desde otras clases o, dicho de otra forma, que puede ser utilizada por otras clases. Ya hemos dicho que un programa, de alguna manera, consiste en la creación de objetos de distintas clases, que se relacionan entre sí. Lo más común es que las clases que definimos sean públicas y que en cada fichero de extensión `.java` se defina una única clase.

Sin embargo, en ocasiones se definen clases (**A**) que solo van a ser utilizadas por una clase determinada (**B**). En ese caso, decimos que la clase **A** es una clase privada de la clase **B**. Las clases **A** y **B** se definen en el mismo fichero `.java`.

### Acción

En un fichero pueden definirse varias clases pero solo una de ellas puede ser pública. De esta forma, si en un fichero se definen varias clases, una de ellas sería pública y el resto serían clases privadas de la primera, a las que solo ésta tendría acceso.

## 6.4. Utilización de Objetos

Una vez que hemos creado una clase, podemos crear objetos en nuestro programa a partir de esas clases.

Cuando creamos un objeto a partir de una clase se dice que hemos creado una "instancia de la clase". A efectos prácticos, "objeto" e "instancia de clase" son términos similares. Es decir, nos referimos a objetos como instancias cuando queremos hacer hincapié que son de una clase particular.

Los objetos se crean a partir de las clases, y representan casos individuales de éstas.

### Ejemplo

Para entender mejor el concepto entre un objeto y su clase, piensa en un molde de galletas y las galletas. El molde sería la clase, que define las características del objeto, por ejemplo su forma y tamaño. Las galletas creadas a partir de ese molde son los objetos o instancias.

Otro ejemplo, imagina una clase Persona que reúna las características comunes de las personas (`color de pelo, ojos, peso, altura, etc.`) y las acciones que pueden realizar (`crecer, dormir, comer, etc.`). Posteriormente dentro del programa podremos crear un objeto `Trabajador` que esté basado en esa clase Persona. Entonces se dice que el objeto `Trabajador` es una instancia de la clase `Persona`, o que la clase `Persona` es una abstracción del objeto `Trabajador`.

Cualquier objeto instanciado de una clase contiene una copia de todos los atributos definidos en la clase. En otras palabras, lo que estamos haciendo es reservar un espacio en la memoria del ordenador para guardar sus atributos y métodos. Por tanto, cada objeto tiene una zona de almacenamiento propia donde se guarda toda su información, que será distinta a la de cualquier otro objeto. A las variables miembro instanciadas también se les llama variables instancia. De igual forma, a los métodos que manipulan esas variables se les llama métodos instancia.

En el ejemplo del objeto `Trabajador`, las variables instancia serían `color_de_pelo, peso, altura, etc.` Y los métodos instancia serían `crecer(), dormir(), comer()`, etc.

#### 6.4.1. Ciclo de vida de los objetos

Todo programa en Java parte de una única clase, que como hemos comentado se trata de la clase principal.

Esta clase ejecutará el contenido de su método `main()`, el cual será el que utilice las demás clases del programa, cree objetos y lance mensajes a otros objetos.

Las instancias u objetos tienen un tiempo de vida determinado. Cuando un objeto no se va a utilizar más en el programa, es destruido por el recolector de basura para liberar recursos que pueden ser reutilizados por otros objetos.

A la vista de lo anterior, podemos concluir que los objetos tienen un ciclo de vida, en el cual podemos distinguir las siguientes fases:

- **Creación**, donde se hace la reserva de memoria e inicialización de atributos.
- **Manipulación**, que se lleva a cabo cuando se hace uso de los atributos y métodos del objeto.
- **Destrucción**, eliminación del objeto y liberación de recursos.

#### 6.4.2. Declaración

Para la creación de un objeto hay que seguir los siguientes pasos:

- **Declaración**: Definir el tipo de objeto.
- **Instanciación**: Creación del objeto utilizando el operador `new`. Pero ¿en qué consisten estos pasos a nivel de programación en Java? Veamos primero cómo declarar un objeto. Para la definición del tipo de objeto debemos emplear la siguiente instrucción:

```
1 <tipo> nombre_objeto;
```

Donde:

- **tipo** es la clase a partir de la cual se va a crear el objeto, y
- **nombre\_objeto** es el nombre de la variable referencia con la cual nos referiremos al objeto.

Los tipos referenciados o referencias se utilizan para guardar la dirección de los datos en la memoria del ordenador.

Nada más crear una referencia, ésta se encuentra vacía. Cuando una referencia a un objeto no contiene ninguna instancia se dice que es una referencia nula, es decir, que contiene el valor `null`.

Esto quiere decir que la referencia está creada pero que el objeto no está instanciado todavía, por eso la referencia apunta a un objeto inexistente llamado "nulo".

Para entender mejor la declaración de objetos veamos un ejemplo. Cuando veímos los tipos de datos, decíamos que Java proporciona un tipo de dato especial para los textos o cadenas de caracteres que era el tipo de dato `String`. Veímos que realmente este tipo de dato es un tipo referenciado y creábamos una variable mensaje de ese tipo de dato de la siguiente forma:

```
1 String mensaje;
```

Los nombres de la clase empiezan con mayúscula, como `String`, y los nombres de los objetos con minúscula, como `mensaje`, así sabemos qué tipo de elemento utilizando.

Pues bien, `String` es realmente la clase a partir de la cual creamos nuestro objeto llamado `mensaje` 😊.

Si observas, poco se diferencia esta declaración de las declaraciones de variables que hacíamos para los tipos primitivos. Antes decíamos que `mensaje` era una variable del tipo de dato `String`. Ahora realmente vemos que `mensaje` es un objeto de la clase `String`. Pero `mensaje` aún no contiene el objeto porque no ha sido instanciado, veamos cómo hacerlo.

Por tanto, cuando creamos un objeto estamos haciendo uso de una variable que almacena la dirección de ese objeto en memoria. Esa variable es una referencia o un tipo de datos referenciado, porque no contiene el dato si no la posición del dato en la memoria del ordenador.

```
1 String saludo = new String("Bienvenido a Java");
2 String s; //s vale null
3 s = saludo; //asignación de referencias
```

En las instrucciones anteriores, las variables `s` y `saludo` apuntan al mismo objeto de la clase `String`. Esto implica que cualquier modificación en el objeto `saludo` modifica también el objeto al que hace referencia la variable `s`, ya que realmente son el mismo.

#### 6.4.3. Instanciación

Una vez creada la referencia al objeto, debemos crear la instancia u objeto que se va a guardar en esa referencia. Para ello utilizamos la orden `new` con la siguiente sintaxis:

```
1 [<tipo>] nombre_objeto = new <Constructor_de_la_Clase>([<par1>, <par2>, ..., <parN>]);
```

Donde:

- **nombre\_objeto** es el nombre de la variable referencia con la cual nos referiremos al objeto.
- **new** es el operador para crear el objeto.
- **Constructor\_de\_la\_Clase** es un método especial de la clase, que se llama igual que ella, y se encarga de inicializar el objeto, es decir, de dar unos valores iniciales a sus atributos.
- **par1-parN**, son parámetros que puede o no necesitar el constructor para dar los valores iniciales a los atributos del objeto.

Durante la instanciación del objeto, se reserva memoria suficiente para el objeto. De esta tarea se encarga Java y juega un papel muy importante el `recolector de basura`, que se encarga de eliminar de la memoria los objetos no utilizados para que ésta pueda volver a ser utilizada.

De este modo, para instanciar un objeto `String`, haríamos lo siguiente:

```
1 mensaje = new String;
```

Así estaríamos instanciando el objeto `mensaje`. Para ello utilizaríamos el operador `new` y el constructor de la clase `String` a la que pertenece el objeto según la declaración que hemos hecho en el apartado anterior. A continuación utilizamos el constructor, que se llama igual que la clase, `String`.

En el ejemplo anterior el objeto se crearía con la cadena vacía (`""`), si queremos que tenga un contenido debemos utilizar parámetros en el constructor, así:

```
1 mensaje = new String ("El primer programa");
```

Java permite utilizar la clase `String` como si de un tipo de dato primitivo se tratara, por eso no hace falta utilizar el operador `new` para instanciar un objeto de la clase `String` (pero no es lo habitual en el resto de clases).

```
1 mensaje = "El primer programa";
```

La declaración e instanciación de un objeto puede realizarse en la misma instrucción, así:

```
1 String mensaje = new String ("El primer programa");
```

o para la clase `String`:

```
1 String mensaje = "El primer programa";
```

#### 6.4.4. Manipulación

Una vez creado e instanciado el objeto ¿cómo accedemos a su contenido? Para acceder a los atributos y métodos del objeto utilizaremos el nombre del objeto seguido del operador punto ( . ) y el nombre del **atributo** o **método** que queremos utilizar. Cuando utilizamos el operador punto se dice que estamos enviando un mensaje al objeto. La forma general de enviar un mensaje a un objeto es:

```
1 nombre_objeto.mensaje
```

Por ejemplo, para acceder a las variables instancia o atributos se utiliza la siguiente sintaxis:

```
1 nombre_objeto.atributo
```

Y para acceder a los métodos o funciones miembro del objeto se utiliza la sintaxis es:

```
1 nombre_objeto.método([par1, par2, ..., parN])
```

En la sentencia anterior `par1`, `par2`, etc. son los parámetros que utiliza el método. (*Aparece entre corchetes para indicar son opcionales*).

Para entender mejor cómo se manipulan objetos vamos a utilizar un ejemplo. Para ello necesitamos la Biblioteca de Clases Java o API (Application Programming Interface - Interfaz de programación de aplicaciones). Uno de los paquetes de librerías o bibliotecas es `java.awt`. Este paquete contiene clases destinadas a la creación de objetos gráficos e imágenes. Vemos por ejemplo cómo crear un rectángulo.

En primer lugar, instanciamos el objeto utilizando el método constructor, que se llama igual que el objeto, e indicando los parámetros correspondientes a la posición y a las dimensiones del rectángulo:

```
1 Rectangle rect = new Rectangle(50, 50, 150, 150);
```

Una vez instanciado el objeto rectángulo si queremos cambiar el valor de los atributos utilizamos el operador punto. Por ejemplo, para cambiar la dimensión del rectángulo:

```
1 rect.height=100;
2 rect.width=100;
```

O bien podemos utilizar un método para hacer lo anterior:

```
1 rect.setSize(200, 200);
```

A continuación puedes acceder al código del ejemplo:

## Ejemplo

```

1 /*
2 * Muestra como se manipulan objetos en Java
3 */
4 import java.awt.Rectangle;
5 public class Manipular {
6 public static void main(String[] args) {
7 // Instanciamos el objeto rect indicando posicion y dimensiones
8 Rectangle rect = new Rectangle(50, 50, 150, 150);
9 //Consultamos las coordenadas x e y del rectangulo
10 System.out.println("----- Coordenadas esquina superior izqda. -----");
11 System.out.println("x=" + rect.x + "\ny=" + rect.y);
12 // Consultamos las dimensiones (altura y anchura) del rectangulo
13 System.out.println("----- Dimensiones -----");
14 System.out.println("Altura = " + rect.height);
15 System.out.println("Ancho = " + rect.width);
16 //Cambiar coordenadas del rectangulo
17 rect.height=100;
18 rect.width=100;
19 rect.setSize(200, 200);
20 System.out.println("----- Nuevos valores de los atributos --");
21 System.out.println("x=" + rect.x + "\ny=" + rect.y);
22 System.out.println("Altura = " + rect.height);
23 System.out.println("Ancho = " + rect.width);
24 }
25 }
```

### 6.4.5. Destrucción de objetos y liberación de memoria

Cuando un objeto deja de ser utilizado, es necesario liberar el espacio de memoria y otros recursos que poseía para poder ser reutilizados por el programa. A esta acción se le denomina destrucción del objeto.

En Java la destrucción de objetos corre a cargo del **recolector de basura (garbage collector)**. Es un sistema de destrucción automática de objetos que ya no son utilizados. Lo que se hace es liberar una zona de memoria que había sido reservada previamente mediante el operador `new`. Esto evita que los programadores tengan que preocuparse de realizar la liberación de memoria.

El recolector de basura se ejecuta en modo segundo plano y de manera muy eficiente para no afectar a la velocidad del programa que se está ejecutando. Lo que hace es que periódicamente va buscando objetos que ya no son referenciados, y cuando encuentra alguno lo marca para ser eliminado.

Después los elimina en el momento que considera oportuno.

Justo antes de que un objeto sea eliminado por el recolector de basura, se ejecuta su método `finalize()`. Si queremos forzar que se ejecute el proceso de finalización de todos los objetos del programa podemos utilizar el método `runFinalization()` de la clase `System`. La clase `System` forma parte de la Biblioteca de Clases de Java y contiene diversas clases para la entrada y salida de información, acceso a variables de entorno del programa y otros métodos de diversa utilidad. Para forzar el proceso de finalización ejecutaríamos:

```
1 System.runFinalization();
```

## 6.5. Utilización de Métodos

Los métodos, junto con los atributos, forman parte de la estructura interna de un objeto. Los métodos contienen la declaración de variables locales y las operaciones que se pueden realizar para el objeto, y que son ejecutadas cuando el método es invocado. Se definen en el cuerpo de la clase y posteriormente son instanciados para convertirse en métodos instancia de un objeto.

Para utilizar los métodos adecuadamente es conveniente conocer la estructura básica de que disponen.

Al igual que las clases, los métodos están compuestos por una cabecera y un cuerpo. La cabecera también tiene modificadores, en este caso hemos utilizado `public` para indicar que el método es público, lo cual quiere decir que le pueden enviar mensajes no sólo los métodos del objeto sino los métodos de cualquier otro objeto externo.

Dentro de un método nos encontramos el cuerpo del método que contiene el código de la acción a realizar. Las acciones que un método puede realizar son:

- **Iniciar** los atributos del objeto
- **Consultar** los valores de los atributos
- **Modificar** los valores de los atributos
- **Llamar a otros métodos**, del mismo del objeto o de objetos externos

### 6.5.1. Parámetros y valores devueltos

Los métodos se pueden utilizar tanto para consultar información sobre el objeto como para modificar su estado. La información consultada del objeto se devuelve a través de lo que se conoce como valor de retorno, y la modificación del estado del objeto, o sea, de sus atributos, se hace mediante la lista de parámetros. En general, la lista de parámetros de un método se puede declarar de dos formas diferentes:

- **Por valor.** El valor de los parámetros no se devuelve al finalizar el método, es decir, cualquier modificación que se haga en los parámetros no tendrá efecto una vez se salga del método. Esto es así porque cuando se llama al método desde cualquier parte del programa, dicho método recibe una copia de los argumentos, por tanto cualquier modificación que haga será sobre la copia, no sobre las variables originales.
- **Por referencia.** La modificación en los valores de los parámetros sí tienen efecto tras la finalización del método. Cuando pasamos una variable a un método por referencia lo que estamos haciendo es pasar la dirección del dato en memoria, por tanto cualquier cambio en el dato seguirá modificado una vez salgamos del método.

#### Acción

En el lenguaje Java, todas las variables se pasan por valor, excepto los objetos que se pasan por referencia.

En Java, la declaración de un método tiene dos restricciones:

- **Un método siempre tiene que devolver un valor (no hay valor por defecto).** Este valor de retorno es el valor que devuelve el método cuando termina de ejecutarse, al método o programa que lo llamó. Puede ser un tipo primitivo, un tipo referenciado o bien el tipo `void`, que indica que el método no devuelve ningún valor.
- **Un método tiene un número fijo de argumentos.** Los argumentos son variables a través de las cuales se pasa información al método desde el lugar del que se llame, para que éste pueda utilizar dichos valores durante su ejecución. Los argumentos reciben el nombre de parámetros cuando aparecen en la declaración del método.

#### Recuerda

El valor de retorno es la información que devuelve un método tras su ejecución.

Según hemos visto en el apartado anterior, la cabecera de un método se declara como sigue:

```
1 public tipo_de_dato_devuelto nombreMetodo (lista_de_parametros);
```

Como vemos, el tipo de dato devuelto aparece después del modificador `public` y se corresponde con el valor de retorno.

La lista de parámetros aparece al final de la cabecera del método, justo después del nombre, encerrados entre signos de paréntesis y separados por comas. Se debe indicar el tipo de dato de cada parámetro así:

```
1 (tipo_parametro1 nombre_parametro1, ..., tipo_parametroN nombre_parametroN)
```

#### Acción

Cuando se llame al método, se deberá utilizar el nombre del método, seguido de los argumentos que deben coincidir con la lista de parámetros.

La lista de argumentos en la llamada a un método debe coincidir en número, tipo y orden con los parámetros del método, ya que de lo contrario se produciría un error de sintaxis.

### 6.5.2. Constructores

¿Recuerdas cuando hablábamos de la creación e instanciación de un objeto? Decíamos que utilizábamos el operador `new` seguido del nombre de la clase y una pareja de abrir-cerrar paréntesis.

Además, el nombre de la clase era realmente el constructor de la misma, y lo definíamos como un método especial que sirve para inicializar valores. En este apartado vamos a ver un poco más sobre los constructores.

Un constructor es un método especial con el mismo nombre de la clase y que no devuelve ningún valor tras su ejecución.

Cuando creamos un objeto debemos instanciarlo utilizando el constructor de la clase. Veamos la clase `Date` proporcionada por la Biblioteca de Clases de Java. Si queremos instanciar un objeto a partir de la clase `Date` tan sólo tendremos que utilizar el constructor seguido de una pareja de abrir-cerrar paréntesis:

```
1 Date fecha = new Date();
```

Con la anterior instrucción estamos creando un objeto fecha de tipo `Date`, que contendrá la fecha y hora actual del sistema.

La estructura de los constructores es similar a la de cualquier método, salvo que no tiene tipo de dato devuelto porque no devuelve ningún valor. Está formada por una cabecera y un cuerpo, que contiene la inicialización de atributos y resto de instrucciones del constructor.

El método constructor tiene las siguientes particularidades:

- **El constructor es invocado** automáticamente **en la creación** de un objeto, **y sólo esa vez**.
- **Los constructores no empiezan con minúscula**, como el resto de los métodos, ya que se llaman igual que la clase y las clases empiezan con letra mayúscula.
- **Puede haber varios** constructores para una clase.
- Como cualquier método, **el constructor puede tener parámetros** para definir qué valores dar a los atributos del objeto.
- **El constructor por defecto es aquél que no tiene argumentos o parámetros**. Cuando creamos un objeto llamando al nombre de la clase sin argumentos, estamos utilizando el constructor por defecto.
- **Es necesario que toda clase tenga al menos un constructor**. Si no definimos constructores para una clase, y sólo en ese caso, el compilador crea un constructor por defecto vacío, que inicializa los atributos a sus valores por defecto, según del tipo que sean: `0` para los tipos numéricos, `false` para los `boolean` y `null` para los tipo carácter y las referencias. Dicho constructor lo que hace es llamar al constructor sin argumentos de la superclase (clase de la cual hereda); si la superclase no tiene constructor sin argumentos se produce un error de compilación.

### Atención

Cuando definimos constructores personalizados, el constructor por defecto deja de existir, y si no definimos nosotros un constructor sin argumentos cuando intentemos utilizar el constructor por defecto nos dará un error de compilación.

#### 6.5.3. El operador `this`

Los constructores y métodos de un objeto suelen utilizar el operador `this`. Este operador sirve para referirse a los atributos de un objeto cuando estamos dentro de él. Sobre todo se utiliza cuando existe ambigüedad entre el nombre de un parámetro y el nombre de un atributo, entonces en lugar del nombre del atributo solamente escribiremos `this.nombre_atributo`, y así no habrá duda de a qué elemento nos estamos refiriendo.

#### 6.5.4. Métodos estáticos

Cuando trabajábamos con cadenas de caracteres utilizando la clase `String`, veíamos las operaciones que podíamos hacer con ellas: obtener longitud, comparar dos cadenas de caracteres, cambiar a mayúsculas o minúsculas, etc. Pues bien, sin saberlo estábamos utilizando métodos estáticos definidos por Java para la clase `String`. Pero ¿qué son los métodos estáticos? Veámoslo.

Los métodos estáticos son aquellos métodos definidos para una clase que se pueden usar directamente, sin necesidad de crear un objeto de dicha clase. También se llaman métodos de clase.

Para llamar a un método estático utilizaremos:

- **El nombre del método**, si lo llamamos desde la misma clase en la que se encuentra definido.
- **El nombre de la clase**, seguido por el operador punto (`.`) más el nombre del método estático, si lo llamamos desde una clase distinta a la que se encuentra definido:

```
1 Nombre_clase.nombre_metodo_estatico
```

- **El nombre del objeto**, seguido por el operador punto (`.`) más el nombre del método estático. Utilizaremos esta forma cuando tengamos un objeto instanciado de la clase en la que se encuentra definido el método estático, y no podamos utilizar la anterior:

```
1 nombre_objeto.nombre_metodo_no_estatico
```

Los métodos estáticos no afectan al estado de los objetos instanciados de la clase (variables instancia), y suelen utilizarse para realizar operaciones comunes a todos los objetos de la clase. Por ejemplo, si necesitamos contar el número de objetos instanciados de una clase, podríamos utilizar un método estático que fuera incrementando el valor de una variable entera de la clase conforme se van creando los objetos.

En la Biblioteca de Clases de Java existen muchas clases que contienen métodos estáticos. Pensemos en las clases que ya hemos utilizado en unidades anteriores, como hemos comentado la clase `String` con todas las operaciones que podíamos hacer con ella y con los objetos instanciados a partir de ella. O bien la clase `Math` para la conversión de tipos de datos. Todos ellos son métodos estáticos que la API de Java define para esas clases. Lo importante es que tengamos en cuenta que al tratarse de métodos estáticos, para utilizarlos no necesitamos crear un objeto de dichas clases.

Fijémonos en esta secuencia de instrucciones

```
1 //Creamos dos círculos de radio 100 en distintas posiciones
2 Circulo c1 = new Circulo(50,50,100);
3 Circulo c2 = new Circulo(80,80,100);
4 ...
5 //Aumentamos el radio del primer círculo a 200
6 c1.setRadio(200);
```

y en esta otra

```
1 System.out.println(Math.sqrt(4));
```

En el primer ejemplo, `.setRadio(200)` va precedido por un objeto. La variable `c1` es un objeto de la clase Círculo, por tanto, la instrucción está modificando el radio de un círculo concreto, el que se encuentra en la posición (50,50). El método `setRadio` es un método no estático. Los métodos no estáticos actúan siempre sobre algún objeto (el que figura a la izquierda del punto).

En el segundo ejemplo, en cambio, a la izquierda de `.sqrt(4)` no se ha puesto el nombre de un objeto, sino el de una clase, la clase `Math`. El método `sqrt` no está actuando sobre un objeto concreto: no tiene sentido hacerlo, solo pretendemos calcular la raíz cuadrada de 4. `Sqrt` es un método estático. Los métodos estáticos se usan poniendo delante del punto el nombre de la clase en que se encuentran definidos.

## 6.6. Librerías de Objetos (Paquetes)

Conforme nuestros programas se van haciendo más grandes, el número de clases va creciendo. Meter todas las clases en único directorio no ayuda a que estén bien organizadas, lo mejor es hacer grupos de clases, de forma que todas las clases que estén relacionadas o traten sobre un mismo tema estén en el mismo grupo.

Un **paquete** de clases es una agrupación de clases que consideramos que están relacionadas entre sí o tratan de un tema común.

### Recuerda

Las clases de un mismo paquete tienen un acceso privilegiado a los atributos y métodos de otras clases de dicho paquete. Es por ello por lo que se considera que los paquetes son también, en cierto modo, unidades de encapsulación y ocultación de información.

Java nos ayuda a organizar las clases en paquetes. En cada fichero `.java` que hagamos, al principio, podemos indicar a qué paquete pertenece la clase que hagamos en ese fichero.

Los paquetes se declaran utilizando la palabra clave `package` seguida del nombre del paquete.

Para establecer el paquete al que pertenece una clase hay que poner una sentencia de declaración como la siguiente al principio de la clase:

```
1 package nombre_de_Paquete;
```

Por ejemplo, si decidimos agrupar en un paquete `ejemplos` un programa llamado `Bienvenida`, pondríamos en nuestro fichero `Bienvenida.java` lo siguiente:

```
1 package ejemplos;
2
3 public class Bienvenida {
4 [...]
5 }
```

El código es exactamente igual que como hemos venido haciendo hasta ahora, solamente hemos añadido la línea `package ejemplos;` al principio.

### 6.6.1. Sentencia `import`

Cuando queremos utilizar una clase que está en un paquete distinto a la clase que estamos utilizando, se suele utilizar la sentencia `import`. Por ejemplo, si queremos utilizar la clase `Scanner` que está en el paquete `java.util` de la Biblioteca de Clases de Java, tendremos que utilizar esta sentencia:

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

Se pueden importar todas las clases de un paquete, así:

```
1 import java.awt.*;
```

Esta sentencia debe aparecer al principio de la clase, justo después de la sentencia `package`, si ésta existiese.

También podemos utilizar la clase sin sentencia `import`, en cuyo caso cada vez que queramos usarla debemos indicar su ruta completa:

```
1 java.util.Scanner teclado = new java.util.Scanner (System.in);
```

Hasta aquí todo correcto. Sin embargo, al trabajar con paquetes, Java nos obliga a organizar los directorios, compilar y ejecutar de cierta forma para que todo funcione adecuadamente.

### 6.6.2. Librerías Java

Cuando descargamos el entorno de compilación y ejecución de Java, obtenemos la API de Java. Como ya sabemos, se trata de un conjunto de bibliotecas que nos proporciona paquetes de clases útiles para nuestros programas. Utilizar las clases y métodos de la Biblioteca de Java nos va ayudar a reducir el tiempo de desarrollo considerablemente, por lo que es importante que aprendamos a consultarla y conozcamos las clases más utilizadas.

#### Ejemplo

```
1 import java.lang.System; // Se importa la clase System.
2 import java.awt.*; // Se importa todas las clases del paquete awt;
```

Los paquetes más importantes que ofrece el lenguaje Java son:

| Paquete o librería | Descripción                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>java.io</b>     | Contiene las clases que gestionan la entrada y salida, ya sea para manipular ficheros, leer o escribir en pantalla, en memoria, etc. Este paquete contiene por ejemplo la clase <code>BufferedReader</code> que se utiliza para la entrada por teclado.                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>java.lang</b>   | Contiene las clases básicas del lenguaje. Este paquete no es necesario importarlo, ya que es importado automáticamente por el entorno de ejecución. En este paquete se encuentra la clase <code>Object</code> , que sirve como raíz para la jerarquía de clases de Java, o la clase <code>System</code> que ya hemos utilizado en algunos ejemplos y que representa al sistema en el que se está ejecutando la aplicación. También podemos encontrar en este paquete las clases que "envuelven" los tipos primitivos de datos. Lo que proporciona una serie de métodos para cada tipo de dato de utilidad, como por ejemplo las conversiones de datos. |
| <b>java.util</b>   | Biblioteca de clases de utilidad general para el programador. Este paquete contiene por ejemplo la clase <code>Scanner</code> utilizada para la entrada por teclado de diferentes tipos de datos, la clase <code>Date</code> , para el tratamiento de fechas, etc.                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>java.math</b>   | Contiene herramientas para manipulaciones matemáticas.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>java.awt</b>    | Incluye las clases relacionadas con la construcción de interfaces de usuario, es decir, las que nos permiten construir ventanas, cajas de texto, botones, etc. Algunas de las clases que podemos encontrar en este paquete son <code>Button</code> , <code>TextField</code> , <code>Frame</code> , <code>Label</code> , etc.                                                                                                                                                                                                                                                                                                                           |

| Paquete o<br>librería | Descripción                                                                                                                                                                                                                                                                |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>java.swing</b>     | Contiene otro conjunto de clases para la construcción de interfaces avanzadas de usuario. Los componentes que se engloban dentro de este paquete se denominan componentes Swing, y suponen una alternativa mucho más potente que AWT para construir interfaces de usuario. |
| <b>java.net</b>       | Conjunto de clases para la programación en la red local e Internet.                                                                                                                                                                                                        |
| <b>java.sql</b>       | Contiene las clases necesarias para programar en Java el acceso a las bases de datos.                                                                                                                                                                                      |
| <b>java.security</b>  | Biblioteca de clases para implementar mecanismos de seguridad.                                                                                                                                                                                                             |

Como se puede comprobar Java ofrece una completa jerarquía de clases organizadas a través de paquetes.

## 6.7. Cadenas de caracteres. La clase String

### 6.7.1. Cadenas de caracteres

Hasta ahora hemos utilizado literales de cadenas de caracteres que, como sabemos, se ponen entre comillas dobles, como en la siguiente expresión

```
1 System.out.println("Hola");
```

Para almacenar cadenas de caracteres en variables se utiliza la clase `String`. `String` se encuentra definida en el paquete `java.lang`. *Recordemos que no es necesario importar este paquete para utilizar sus clases.*

La forma de `String` es la siguiente:

```
1 String variable = new String("texto");
```

Ejemplo:

```
1 String nombre = new String("Javier");
2 System.out.println("Mi nombre es " + nombre);
```

**Sin embargo**, debido a que es una clase que se utiliza ampliamente en los programas, Java permite una forma abreviada de crear objetos `String`:

```
1 String nombreVariable = "texto";
```

Ejemplo:

```
1 String nombre = "Javier";
2 System.out.println("Mi nombre es " + nombre);
```

### 6.7.2. Leer cadenas desde teclado

#### 6.7.2.1. CLASE Scanner

Para leer cadenas de caracteres desde teclado podemos utilizar la clase `Scanner`. Ésta dispone de dos métodos para leer cadenas:

- `next()` : Lee desde la entrada estándar (teclado) una secuencia de caracteres hasta encontrar un delimitador (un espacio). Devuelve un `String`.
- `nextLine()` : Lee desde la entrada estándar (teclado) una secuencia de caracteres hasta encontrar un salto de línea. Devuelve un `String`.

Ejemplo:

```
1 Scanner tec = new Scanner(System.in);
2 //De lo que introduce el usuario, lee la 1º palabra.
3 String nombre = tec.next();
4 //Lee lo que introduce el usuario hasta que pulsa intro.
5 String nombreCompleto = tec.nextLine();
```

#### 6.7.2.2. EJEMPLOS DE LA UD01 PERO UTILIZANDO `Scanner` (COMPATIBLE CON LOS IDE'S)

A continuación vamos a ver los mismos ejemplos de la UD01, pero utilizando la clase `Scanner` que si es compatible con los IDE's. Para poder usar la clase `Scanner` necesitamos importar el paquete: `java.util.Scanner`.

```

1 import java.util.Scanner;
2
3 public class EjemploUD02 {
4
5 public static void main(String[] args) {
6
7 Scanner teclado = new Scanner(System.in);
8
9 //Introducir texto desde teclado
10 String texto;
11 System.out.print("Introduce un texto: ");
12 texto = teclado.nextLine();
13 System.out.println("El texto introducido es: "+ texto);
14
15 //Introducir un número entero desde teclado
16 String texto2;
17 int entero2;
18 System.out.print("Introduce un número: ");
19 texto2 = teclado.nextLine();
20 entero2 = Integer.parseInt(texto2);
21 System.out.println("El número introducido es:"+entero2);
22
23 //Introducir un número decimal desde teclado
24 String texto3;
25 double doble3;
26 System.out.print("Introduce un número decimal: ");
27 texto3 = teclado.nextLine();
28 doble3 = Double.parseDouble(texto3); // convertimos texto a doble
29 System.out.println("Número decimal introducido es: "+doble3);
30 }
31 }
```

#### 6.7.3. La clase `String`

Además de permitir almacenar cadenas de caracteres, `String` tiene métodos para realizar cálculos u operaciones con ellas.

Así por ejemplo, la clase tiene un método `toUpperCase()` que devuelve el `String` convertido a mayúsculas. El siguiente ejemplo ilustra su uso:

```

1 String nombre = "Javier";
2 System.out.println(nombre.toUpperCase()); // Se muestra JAVIER por pantalla
```

Accede a la documentación en línea de Java y estudia los siguientes métodos de la clase:

- `charAt`
- `indexOf`
- `substring`
- `toLowerCase`
- `trim`

#### 6.7.4. `printf` o `format`

El método `printf()` o `format()` (son sinónimos) utilizan unos códigos de conversión para indicar si el contenido a mostrar de qué tipo es. Estos códigos se caracterizan porque llevan delante el símbolo %, algunos de ellos son:

- `%c` : Escribe un carácter.
- `%s` : Escribe una cadena de texto.
- `%d` : Escribe un entero.
- `%f` : Escribe un número en punto flotante.
- `%e` : Escribe un número en punto flotante en notación científica.

Por ejemplo, si queremos escribir el número float `12345.1684` con el punto de los miles y sólo dos cifras decimales la orden sería:

```
1 System.out.printf("%,.2f\n", 12345.1684);
```

Esta orden mostraría el número `12.345,17` por pantalla.

Otro ejemplo seria:

```
1 System.out.format("El valor de la variable float es" +
2 "%f, mientras que el valor del entero es %d" +
3 "y el string contiene %s", variableFloat, variableInt, variableString);
```

Puedes investigar más sobre `printf` o `format` en este [enlace](#)

#### 6.7.5. Salida de error

La salida de error está representada por el objeto `System.err`. No parece muy útil utilizar `out` y `err` si su destino es la misma pantalla, o al menos en el caso de la consola del sistema donde las dos salidas son representadas con el mismo color y no notamos diferencia alguna. En cambio en la consola de varios entornos integrados de desarrollo como NetBeans o Eclipse la salida de `err` se ve en un color diferente. Teniendo el siguiente código:

```
1 System.out.println("Salida estándar por pantalla");
2 System.err.println("Salida de error por pantalla");
```

Tanto NetBeans, Eclipse como IntelliJ Idea mostraran el mensaje `err` en color rojo.

### 6.8. Ejemplo UD02

#### 6.8.1. Clase Pajaro

Vamos a ilustrar mediante un ejemplo la utilización de objetos y métodos, así como el uso de parámetros y el operador `this`. Aunque la creación de clases la veremos en las siguientes unidades, en este ejercicio creamos una pequeña clase para que podamos instanciar el objeto con el que vamos a trabajar.

Las clases se suelen representar como un rectángulo, y dentro de él se sitúan los atributos y los métodos de dicha clase.

En la imagen, la clase `Pajaro` está compuesta por tres atributos, uno de ellos el `nombre` y otros dos que indican la posición del ave, `posX` y `posY`. Tiene tres métodos constructores y un método `volar()`. Como sabemos, los métodos constructores reciben el mismo nombre de la clase, y puede haber varios para una misma clase, dentro de ella se diferencian unos de otros por los parámetros que utilizan.

#### Ejunciado:

Dada una clase principal llamada `Pajaro`, se definen los atributos y métodos que aparecen en la imagen. Los métodos realizan las siguientes acciones:

```
classDiagram
 class Pajaro{
 -String nombre
 -int posX
 -int posY
 +Pajaro()
 +Pajaro(String nombre)
 +Pajaro(String nombre, int posX, int posY)
 +double volar(int posX, int posY)
 }
```

- `Pajaro()` . Constructor por defecto. En este caso, el constructor por defecto no contiene ninguna instrucción, ya que Java inicializa de forma automática las variables miembro, si no le damos ningún valor.
- `Pajaro(String nombre)` . Constructor que recibe como argumentos una cadena de texto (el nombre del pájaro).
- `Pajaro(String nombre, int posX, int posY)` . Constructor que recibe como argumentos una cadena de texto y dos enteros para inicializar el valor de los atributos.
- `double volar(int posX, int posY)` . Método que recibe como argumentos dos enteros: `posX` y `posY`, y devuelve un valor de tipo `double` como resultado, usando la palabra clave `return`. El valor devuelto es el resultado de aplicar un desplazamiento de acuerdo con la siguiente fórmula:

$$\text{desplazamiento} = \sqrt{\text{posX}^2 + \text{posY}^2}$$

Diseña un programa que utilice la clase `Pajaro`, cree una instancia de dicha clase y ejecute sus métodos.

Lo primero que debemos hacer es crear la clase `Pajaro`, con sus métodos y atributos. De acuerdo con los datos que tenemos, el código de la clase sería el siguiente:

```

1 public class Pajaro {
2 //atributos/variables
3 private String nombre;
4 private int posX;
5 private int posY;
6 //constructores
7 public Pajaro() {
8 }
9 public Pajaro(String nombre) {
10 this.nombre = nombre;
11 }
12 public Pajaro(String nombre, int posX, int posY) {
13 this.nombre = nombre;
14 this.posX = posX;
15 this.posY = posY;
16 }
17
18 //metodos
19 public double volar(int posX, int posY) {
20 double desplazamiento = Math.sqrt((posX * posX) + (posY * posY));
21 //desplazamiento=Math.sqrt(Math.pow(posX,2)+Math.pow(posY,2));
22 this.posX = posX;
23 this.posY = posY;
24 return desplazamiento;
25 }
26 //método main()
27 [...]
28 }
```

Debemos tener en cuenta que se trata de una clase principal, lo cual quiere decir que debe contener un método `main()` dentro de ella. En el método `main()` vamos a situar el código de nuestro programa. El ejercicio dice que tenemos que crear una instancia de la clase y ejecutar sus métodos, entre los que están el constructor y el método `volar()`.

También es conveniente imprimir el resultado de ejecutar el método `volar()`. Por tanto, lo que haría el programa sería:

- Crear un objeto de la clase e inicializarlo.
- Invocar al método volar.
- Imprimir por pantalla la distancia recorrida.

Para inicializar el objeto utilizaremos el constructor con parámetros, después ejecutaremos el método `volar()` del objeto creado y finalmente imprimiremos el valor que nos devuelve el método.

Luego crearemos otro `pajaro2` usando el constructor por defecto (sin parámetros). Le asignaremos el nombre y la posición manualmente, y calcularemos su desplazamiento llamando al método, pero usando los atributos del objeto (`pajaro2.posX` y `pajaro2.posY`) en lugar de constantes. El código del método `main()` quedaría como sigue:

```

1 public static void main(String[] args) {
2 //creamos el objeto con parámetros
3 Pajaro pajaro1 = new Pajaro("WoodPecker", 50, 50);
4 double d1 = pajaro1.volar(50, 50);
5 System.out.println("El desplazamiento de " + pajaro1.nombre + " ha sido " + d1);
6
7 Pajaro pajaro2 = new Pajaro();
8 //damos nombre y cambiamos la posición de "Piolin" a mano
9 pajaro2.nombre="Piolin";
10 pajaro2.posX=30;
11 pajaro2.posY=30;
12 double d2 = pajaro2.volar(pajaro2.posX, pajaro2.posY);
13 System.out.println("El desplazamiento de " + pajaro2.nombre + " ha sido " + d2);
14 }
```

Si ejecutamos nuestro programa el resultado sería el siguiente:

```

1 El desplazamiento de WoodPecker ha sido 70.71067811865476
2 El desplazamiento de Piolin ha sido 42.42640687119285
```

### 6.8.2. Clase String

#### Ejemplo:

```

1 package UD02;
2
3 import java.util.Scanner;
4
5 public class EjemploUD02 {
6
7 public static void main(String[] args) {
8
9 Scanner teclado = new Scanner(System.in);
10
11 //Introducir texto desde teclado
12 String texto;
13 System.out.print("Introduce un texto: ");
14 texto = teclado.nextLine();
15 System.out.println("El texto introducido es: " + texto);
16
17 //Introducir un número entero desde teclado
18 String texto2;
19 int entero2;
20 System.out.print("Introduce un número: ");
21 texto2 = teclado.nextLine();
22 entero2 = Integer.parseInt(texto2);
23 System.out.println("El número introducido es:" + entero2);
24
25 //Introducir un número decimal desde teclado
26 String texto3;
27 double doble3;
28 System.out.print("Introduce un número decimal: ");
29 texto3 = teclado.nextLine();
30 doble3 = Double.parseDouble(texto3); // convertimos texto a doble
31 System.out.println("Número decimal introducido es: " + doble3);
32
33 System.out.println("La clase String");
34 String nombre = "Javier"; //Observa que hay un espacio final
35 System.out.println(nombre.toUpperCase()); //JAVIER
36 System.out.println(nombre.charAt(4)); //e
37 System.out.println(nombre.indexOf("i")); //3
38 System.out.println(nombre.substring(0, 3)); //Javi
39 System.out.println(nombre.toLowerCase()); //javier
40 System.out.println(nombre.trim()); //Javier sin espacios finales
41 System.out.printf("%.2f\n", 12345.1684);
42 nombre.toUpperCase().substring(0,3).indexOf("I"); //3
43 System.out.format("El valor de la variable float es %f"
44 + ", mientras que el valor del entero es %d"
45 + " y el string contiene %s", doble3, entero2, texto);
46
47 System.err.println("Salida de error por pantalla");
48 }
49 }
```

## 6.9. Ejemplos UD02

[EjemploUD02.java](#)

[Pajaro.java](#)

## 6.10. Píldoras informáticas relacionadas

⌚8 de enero de 2026

## 7. 3.2 Ejercicios de la UD02

---

### 7.1. Actividades

1. (Temperatura) Crear una clase llamada Temperatura con dos métodos:

- `celsiusToFahrenheit`. Convierte grados *Celsius* a *Fahrenheit*.  

$$F = (1,8 * C) + 32$$
- `fahrenheitToCelsius`. Convierte grados *Fahrenheit* a *Celsius*.  

$$C = \frac{F - 32}{1,8}$$

2. (Moto) A partir de la siguiente clase:

```

1 public class Moto {
2
3 private int velocidad;
4
5 public Moto() {
6 velocidad=0;
7 }
8 }
```

Añade los siguientes métodos:

- `int getVelocidad()`. Devuelve la velocidad del objeto moto.
- `void acelera(int mas)`. Permite aumentar la velocidad del objeto moto.
- `void frena(int menos)`. Permite reducir la velocidad del objeto moto.

3. (Rebajas) Crea una clase `Rebajas` con un método `descubrePorcentaje()` que descubra el descuento aplicado en un producto. El método recibe el precio original del producto y el rebajado y devuelve el porcentaje aplicado. Podemos calcular el descuento realizando la operación:  $\frac{\text{precioOriginal} - \text{precioRebajado}}{\text{precioOriginal}} \times 100$

4. (Finanzas) Realiza una clase `Finanzas` que convierta dólares a euros y viceversa. Codifica los métodos `dolaresToEuros` y `eurosToDolares`. Prueba que dicha clase funciona correctamente haciendo conversiones entre euros y dólares. La clase tiene que tener:

- Un constructor `Finanzas()` por defecto el cual establece el cambio Dólar-Euro en 1.36.
- Un constructor `Finanzas(double cambio)`, el cual permitirá configurar el cambio Dólar-euro a una cantidad personalizada.

5. (MiNumero) Realiza una clase `MiNumero` que proporcione el doble, triple y cuádruple de un número proporcionado en su constructor (realiza un método para `doble`, otro para `triple` y otro para `cuadruple`).

(Opcional, no hay puntos) Haz que la clase tenga un método `main` y comprueba los distintos métodos.

6. (Numero) Realiza una clase `Numero` que almacene un número entero y tenga las siguientes características:

- Constructor por defecto que inicializa a 0 el número interno.
- Constructor que inicializa el número interno.
- Método `anade` que permite sumarle un número al valor interno.
- Método `resta` que resta un número al valor interno.
- Método `getValor`. Devuelve el valor interno.
- Método `getDoble`. Devuelve el doble del valor interno.
- Método `getTriple`. Devuelve el triple del valor interno.
- Método `setNumero`. Inicializa de nuevo el valor interno.

7. (Peso) Crea la clase `Peso`, la cual tendrá las siguientes características:

- Deberá tener un atributo donde se almacene el peso de un objeto en kilogramos. En el constructor se le pasará el peso y la medida en la que se ha tomado ("Lb" para libras, "Li" para lingotes, "Oz" para onzas, "P" para peniques, "K" para kilos, "G" para gramos y "Q" para quintales).
- Deberá de tener los siguientes métodos:
  - `getLibras`. Devuelve el peso en libras.
  - `getLingotes`. Devuelve el peso en lingotes.
  - `getPeso`. Devuelve el peso en la medida que se pase como parámetro ("Lb" para libras, "Li" para lingotes, "Oz" para onzas, "P" para peniques, "K" para kilos, "G" para gramos y "Q" para quintales).
- Para la realización del ejercicio toma como referencia los siguientes datos:

- 1 Libra = 16 onzas = 453 gramos.
- 1 Lingote = 32,17 libras = 14,59 kg.
- 1 Onza = 0,0625 libras = 28,35 gramos.
- 1 Penique = 0,05 onzas = 1,55 gramos.
- 1 Quintal = 100 libras = 43,3 kg.

(Opcional, no hay puntos) Crea además un método `main` para testear y verificar los métodos de esta clase.

8. (Millas) Crea una clase con un método `millasAMetros()` que toma como parámetro de entrada un valor en millas marinas y las convierte a metros. Una vez tengas este método escribe otro `millasAKilometros()` que realice la misma conversión, pero esta vez exprese el resultado en kilómetros.

*Nota: 1 milla marina equivale a 1852 metros.*

9. (Coche) Crea la clase `Coche` con dos constructores. Uno no toma parámetros y el otro sí. Los dos constructores inicializarán los atributos `marca` y `modelo` de la clase. El constructor por defecto (sin parámetros) crea el coche "Ford" modelo "C-MAX".

(Opcional, no hay puntos) Crea dos objetos (cada objeto llama a un constructor distinto) y verifica que todo funciona correctamente.

10. (Consumo) Implementa una clase `Consumo`, la cual forma parte del "ordenador de a bordo" de un coche y tiene las siguientes características:

- Atributos:
  - `kilometros`.
  - `litros`. Litros de combustible consumido.
  - `vmed`. Velocidad media.
  - `pgas`. Precio de la gasolina.
- Métodos:
  - `getTiempo`. Indicará el tiempo empleado en realizar el viaje.
  - `consumoMedio`. Consumo medio del vehículo (en litros cada 100 kilómetros).
  - `consumoEuros`. Consumo medio del vehículo (en euros cada 100 kilómetros).

No olvides crear un constructor para la clase que establezca el valor de los atributos. Elige el tipo de datos más apropiado para cada atributo.

11. (ConsumoModificadores) Para la clase anterior implementa los siguientes métodos, los cuales podrán modificar los valores de los atributos de la clase:

- `setKilometros`
- `setLitros`
- `setVmed`
- `setPgas`

12. (Restaurante) Un restaurante cuya especialidad son las patatas con carne nos pide diseñar un método (`calcularClientes`) con el que se pueda saber cuántos clientes pueden atender con la materia prima que tienen en el almacén. El método recibe la cantidad de patatas y carne en kilos y devuelve el número de clientes que puede atender el restaurante teniendo en cuenta que por cada tres personas, utilizan un dos kilos de patatas y un kilo de carne.

13. (RestauranteClase) Modifica el programa anterior creando una clase que permita almacenar los kilos de patatas y carne del restaurante. Implementa los siguientes métodos:

- `public void addCarne(int x)`. Añade `x` kilos de carne a los ya existentes.
- `public void addPatatas(int x)`. Añade `x` kilos de patatas a los ya existentes.
- `public int getComensales()`. Devuelve el número de clientes que puede atender el restaurante (este es el método del ejercicio anterior).
- `public double getCarne()`. Devuelve los kilos de carne que hay en el almacén.
- `public double getPatatas()`. Devuelve los kilos de patatas que hay en el almacén.

14. (Proveedor) Crear un clase llamada `Proveedor` con las siguientes propiedades:

- `CIF`
- `nombreEmpresa`
- `descripcion`
- `sector`
- `direccion`

- `telefono`
- `poblacion`
- `codPostal`
- `correo`

Crear para la clase `Proveedor` los métodos:

- Constructor que permite crear una instancia con los datos de un proveedor.
- Métodos get (*getters*).
- Métodos set (*setters*).
- Método `verificaCorreo` que devuelve true si la dirección de correo contiene @ .
- Método que muestre todos los datos del proveedor.

Crear un método principal `main` ejecutable que:

- Cree una instancia del objeto `Proveedor` llamado `proveedor`.
- Cambie el sector del `proveedor`.
- Muestre el sector del `proveedor`.
- Verifique si el correo es válido.
- Muestre todos los datos del `proveedor`.

15. (Producto) Crear una clase llamada `Producto` con las siguientes propiedades:

- `codProducto`
- `nombreProducto`
- `descripcion`
- `categoria`
- `peso`
- `precio`
- `stock`

Crear para la clase `Producto` los siguiente métodos:

- `Producto` : Permite crear una instancia con los datos de un producto.
- Getters y Setters para todas las propiedades.
- `aumentaStock` : Permite aumentar el stock de unidades del producto. Se le pasa el dato de unidades que aumentamos.
- `disminuyeStock` : Permite disminuir el stock de unidades del producto. Se le pasa el dato de unidades que disminuimos.
- `ivaProducto` : Permite calcular el IVA aplicado al precio del producto. Se le pasa el dato del porcentaje de IVA.
- `mostrarDatos` : Muestra los datos del producto. (No tiene test)

Crear un método principal `main` ejecutable que:

- Crear dos instancias de la clase `Producto` llamadas `productoHardware` y `productoSoftware`.
- Mostrar los datos de los dos objetos `Producto` que hemos creado.
- Aumenta el stock de unidades del `productoHardware` en 12 unidades.
- Disminuir el stock de unidades del `productoSoftware` en 5 unidades.
- Calcula el IVA de los dos objetos `Producto` que hemos creado.
- Mostrar los datos de los dos objetos `Producto`, así como sus importes de IVA y los precios finales de cada una de las instancias.

16. (Cuenta) Crea una clase llamada `Cuenta` que tendrá los siguientes atributos: `titular` y `cantidad` (puede tener decimales).

Al crear una instancia del objeto `Cuenta`, el titular será obligatorio y la cantidad es opcional. Crea dos constructores que cumplan lo anterior; es decir debemos crear dos métodos constructores con el mismo nombre que será el nombre del objeto.

Crea sus métodos get, set y el método `mostrarDatos` que muestre los datos de la cuenta. Tendrá dos métodos especiales:

- `ingresar(double cantidad)` : se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.
- `retirar(double cantidad)` : se retira una cantidad a la cuenta, si restando la cantidad actual a la que nos pasan es negativa, la cantidad de la cuenta pasa a ser 0 retirando el importe máximo en función de la cantidad disponible en el objeto.

Crear una método principal `main` ejecutable:

- Crear una instancia del objeto `Cuenta` llamada `cuentaParticular1` con el nombre del titular.
- Crear una instancia del objeto `Cuenta` llamada `cuentaEmpresa1` con el nombre del titular y una cantidad inicial de dinero.
- Mostrar el titular de la instancia `cuentaParticular1`.
- Mostrar el saldo de la instancia `cuentaEmpresa1`.

- Ingresar 1000 € en la instancia `cuentaParticular1`.
- Retirar 500 € en la instancia `cuentaEmpresa1`.
- Mostrar los datos de las dos instancias del objeto `Cuenta`.

17. (Libro) Crea una clase llamada `Libro` que guarde la información de cada uno de los libros de una biblioteca. La clase debe guardar las siguientes propiedades:

- `título`
- `autor`
- `editorial`
- `número de ejemplares totales`
- `número de prestados`

La clase contendrá los siguientes métodos:

- Constructor por defecto.
- Constructor con parámetros.
- Métodos Setters/getters.
- Método `prestamo` que incremente el atributo correspondiente cada vez que se realice un préstamo del libro. No se podrán prestar libros de los que no queden ejemplares disponibles para prestar. Devuelve `true` si se ha podido realizar la operación y `false` en caso contrario.
- Método `devolucion` que decremente el atributo correspondiente cuando se produzca la devolución de un libro. No se podrán devolver libros que no se hayan prestado. Devuelve `true` si se ha podido realizar la operación y `false` en caso contrario.
- Método `perdido` que decremente el atributo número de ejemplares por perdida de ejemplar. No se podrán perder libros que no tengan ejemplares o no se hayan prestado. Devuelve `true` si se ha podido realizar la operación y `false` en caso contrario.
- Método `mostrarDatos` para mostrar los datos de los libros.

Crear un método principal `main` ejecutable:

- Crear una instancia del objeto libro `libroInformatica1` con los datos de un libro.
- Consultar el título de la instancia `libroInformatica1`.
- Cambiar la editorial de la instancia `libroInformatica1` por Anaya.
- Realiza el préstamo de la instancia `libroInformatica1`.
- Realiza otro préstamo de la instancia `libroInformatica1`.
- Muestra los préstamos de la instancia `libroInformatica1`.
- Realiza la devolución de la instancia `libroInformatica1`.
- Muestra los préstamos de la instancia `libroInformatica1`.
- Gestiona la pérdida de un ejemplar de la instancia `libroInformatica1`.
- Muestra los ejemplares de la instancia `libroInformatica1`.
- Muestra todos los datos de la instancia `libroInformatica1`.

18. (Hospital) Crear una clase llamada `Hospital` con las siguientes propiedades y métodos:

- Propiedades:
  - `codHospital`
  - `nombreHospital`
  - `direccion`
  - `telefono`
  - `poblacion`
  - `codPostal`
  - `habitacionesTotales`
  - `habitacionesOcupadas`
- Métodos:
  - `Hospital`: Permite crear una instancia con los datos de un hospital.
  - Métodos get.
  - Métodos set.
  - Método `ingreso` que incremente las habitaciones ocupadas. No puede realizarse el ingreso si las habitaciones ocupadas son iguales a las habitaciones totales del hospital. Devuelve `true` si se ha podido realizar el ingreso.

- Método `alta` que decrementa las habitaciones ocupadas. No puede realizarse el alta las habitaciones ocupadas son 0. Devuelve `true` si se ha podido realizar el alta.
- Método que muestre todos los datos del hospital.
- Crear un método principal `main` ejecutable que:
- Cree una instancia de la clase `Hospital` llamada `hospitalRibera`.
- Cambie el número de habitaciones de la instancia `hospitalRibera`.
- Muestre el número de habitaciones de la instancia `hospitalRibera`.
- Realiza un ingreso de la instancia `hospitalRibera`.
- Muestra las habitaciones ocupadas de la instancia `hospitalRibera`.
- Realiza un alta de la instancia `hospitalRibera`.
- Muestra las habitaciones ocupadas de la instancia `hospitalRibera`.
- Muestre todos los datos de la instancia `hospitalRibera`.

19. (Medico) Crear un clase llamada `Medico` con las siguientes propiedades y métodos:

- Propiedades:
  - `codMedico`
  - `nombre`
  - `apellidos`
  - `dni`
  - `direccion`
  - `telefono`
  - `poblacion`
  - `codPostal`
  - `fechaNacimiento`
  - `especialidad`
  - `sueldo`
- Métodos:
  - `Medico`: Permite crear una instancia con los datos de un médico.
  - Métodos `get`. Recuperan datos de la instancia del objeto.
  - Métodos `set`. Asignan datos a la instancia del objeto.
  - `retencionMedico` : Permite calcular la retención aplicada al sueldo del médico. Se le pasa el dato del porcentaje de retención.
  - `mostrarDatos` : Muestra los datos del médico.
  - Crear un método principal `main` ejecutable que:
  - Crear dos instancias de la clase `Medico` llamados `medicoDigestivo` y `medicoTraumatologo`.
  - Cambia el sueldo del `medicoTraumatologo`.
  - Muestra el sueldo del `medicoTraumatologo`.
  - Cambia el dni del `medicoDigestivo`.
  - Muestra el dni del `medicoDigestivo`.
  - Calcula la retención de las dos instancias de la clase `Medico` que hemos creado.
  - Mostrar los datos de las dos instancias de la clase `Medico` que hemos creado, así como las retenciones y los sueldos finales de cada una.

## 7.2. Ejercicios

---

Estos ejercicios utilizan la interfaz gráfica a la que dedicaremos más tiempo hacia finales de curso. De momento con entender algunos conceptos muy básicos de como dibujar elementos gráficos en una ventana podemos intentar resolverlos usando los conceptos de objetos, clases, herencia, etcétera que hemos visto en teoría.

El primero está resuelto y comentado para que te ayude a resolver el resto por tu cuenta o con la ayuda del docente.

1. (LlenarConCirculo) Crear una pizarra cuadrada y dibujar en ella un círculo que la ocupe por completo.

```

1 //importaciones necesarias para los ejercicios, no necesitas más.
2 import javax.swing.JFrame;
3 import javax.swing.JPanel;
4 import java.awt.Color;
5 import java.awt.Graphics;
6
7 /*
8 Necesitamos que nuestra clase LlenarConCirculo herede de JPanel para poder
9 pintar en su interior.
10 */
11 public class LlenarConCirculo extends JPanel {
12
13 @Override
14 public void paint(Graphics g) {
15 //Fijamos el color que tendrá la figura
16 g.setColor(Color.RED);
17
18 /* Dibujamos un ovalo relleno fijando las 4 esquinas que lo delimitan:
19 - x1, y1, x2, y2
20 En nuestro caso además hacemos uso de la función reflexiva
21 this.getWidth() y this.getHeight() para conocer la anchura y altura
22 (respectivamente) de nuestra ventana.
23 */
24 g.fillOval(0, 0, this.getWidth(), this.getHeight());
25
26 Otras funciones disponibles para dibujar son:
27 - fill3DRect(int x, int y, int width, int height, boolean raised)
28 Paints a 3-D highlighted rectangle filled with the current color.
29 - fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
30 Fills a circular or elliptical arc covering the specified rectangle.
31 - fillOval(int x, int y, int width, int height)
32 Fills an oval bounded by the specified rectangle with the current color.
33 - fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
34 Fills a closed polygon defined by arrays of x and y coordinates.
35 - fillPolygon(Polygon p)
36 Fills the polygon defined by the specified Polygon object with the graphics
37 context's current color.
38 - fillRect(int x, int y, int width, int height)
39 Fills the specified rectangle.
40 - fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)
41 Fills the specified rounded corner rectangle with the current color.
42 - fill3DRect(int x, int y, int width, int height, boolean raised)
43 Paints a 3-D highlighted rectangle filled with the current color.
44 - fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
45 Fills a circular or elliptical arc covering the specified rectangle.
46 - fillOval(int x, int y, int width, int height)
47 Fills an oval bounded by the specified rectangle with the current color.
48 - fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
49 Fills a closed polygon defined by arrays of x and y coordinates.
50 - fillPolygon(Polygon p)
51 Fills the polygon defined by the specified Polygon object with the graphics
52 context's current color.
53 - fillRect(int x, int y, int width, int height)
54 Fills the specified rectangle.
55 - fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)
56 Fills the specified rounded corner rectangle with the current color.
57 */
58 }
59
60 public static void main(String[] args) {
61 //Creamos una nueva ventana
62 JFrame MainFrame = new JFrame();
63 //Fijamos su tamaño en 300px de ancho por 300px de alto
64 MainFrame.setSize(300, 300);
65 //Creamos el objeto que vamos a dibujar con el método paint()
66 LlenarConCirculo circlePanel = new LlenarConCirculo();
67 //Añadimos el objeto recién creado a la ventana
68 MainFrame.add(circlePanel);
69 //Hacemos visible la ventana (con el dibujo)
70 MainFrame.setVisible(true);
71 }
72 }
```

Este es el esquema básico que necesitas para resolver todos los ejercicios planteados:

```

1 //importaciones necesarias para los ejercicios, no necesitas más.
2 import javax.swing.JFrame;
3 import javax.swing.JPanel;
4 import java.awt.Color;
5 import java.awt.Graphics;
6
7 /*
8 Necesitamos que nuestra clase herede de JPanel para poder
9 pintar en su interior.
10 */
11 public class TuClaseEjercicio extends JPanel {
12
13 @Override
14 public void paint(Graphics g) {
15 // INSERTA TU CÓDIGO AQUÍ!!! <-->
16 //Fijamos el color que tendrá la figura
17 //Dibuja la/s figura/s que te pide el ejercicio
18 }
19
20 public static void main(String[] args) {
21 //Creamos una nueva ventana
22 JFrame MainFrame = new JFrame();
23 //Fijamos su tamaño en 300px de ancho por 300px de alto
24 MainFrame.setSize(300, 300);
25 //Creamos el objeto que vamos a dibujar con el método paint()
26 LlenarConCirculo tuDibujo = new LlenarConCirculo();
27 //Añadimos el objeto recién creado a la ventana
28 MainFrame.add(tuDibujo);
29 //Hacemos visible la ventana (con el dibujo)
30 MainFrame.setVisible(true);
31 }
32 }
```

2. (LlenarConRectangulo) Crear una pizarra de tamaño aleatorio y dibujar en ella un rectángulo que la ocupe por completo.
3. (MitadYMitad) Crear una pizarra de tamaño aleatorio y dibujar un rectángulo ROJO que ocupe la mitad izquierda y uno VERDE que ocupe la mitad derecha.
4. (Dos partes) Crear una pizarra de tamaño aleatorio y dibujar un rectángulo ROJO que ocupe la parte superior (25% de la altura) y uno VERDE que ocupe la parte inferior (75% restante).
5. (CentrarFiguras) Crear una pizarra de tamaño aleatorio. Dibujar en el centro un cuadrado de lado 100 y un círculo de radio 25.
6. (RadioAleatorioCentrado) Crear una pizarra de tamaño aleatorio. Dibujar en centro de la pizarra un círculo de radio aleatorio (entre 50 y 200 pixels de radio)
7. (RadioAleatorio) Crear una pizarra de tamaño aleatorio. Dibujar en la esquina superior izquierda un círculo de radio aleatorio (entre 50 y 200)

 7 de noviembre de 2025

## 8. 3.3 Talleres

### 8.1. Taller UD02\_01: GitHub Classroom

#### 8.1.1. Requisitos previos

Necesitamos:

- Una cuenta de GitHub
- Tener el IDE IntelliJ instalado en nuestro ordenador

#### 8.1.2. Unirnos a GitHub Classroom

Aceptamos el *Assignment* (la tarea/ejercicio) a partir del link del profesor, en este caso: <https://classroom.github.com/a/LTEIJf5H>



1DAM\_Programacion\_25\_26

## Accept the assignment — UD02

Once you accept this assignment, you will be granted access to the `ud02-martinezpenya` repository in the [IES-Eduardo-Primo-Marques](#) organization on GitHub.

[Accept this assignment](#)



Nos mostrará la siguiente pantalla:



## You're ready to go!

You accepted the assignment, **UD02**.

Your assignment repository has been created:

 <https://github.com/IES-Eduardo-Primo-Marques/ud02-martinezpenya>

We've configured the repository associated with this assignment.



### Join the GitHub Student Developer Pack

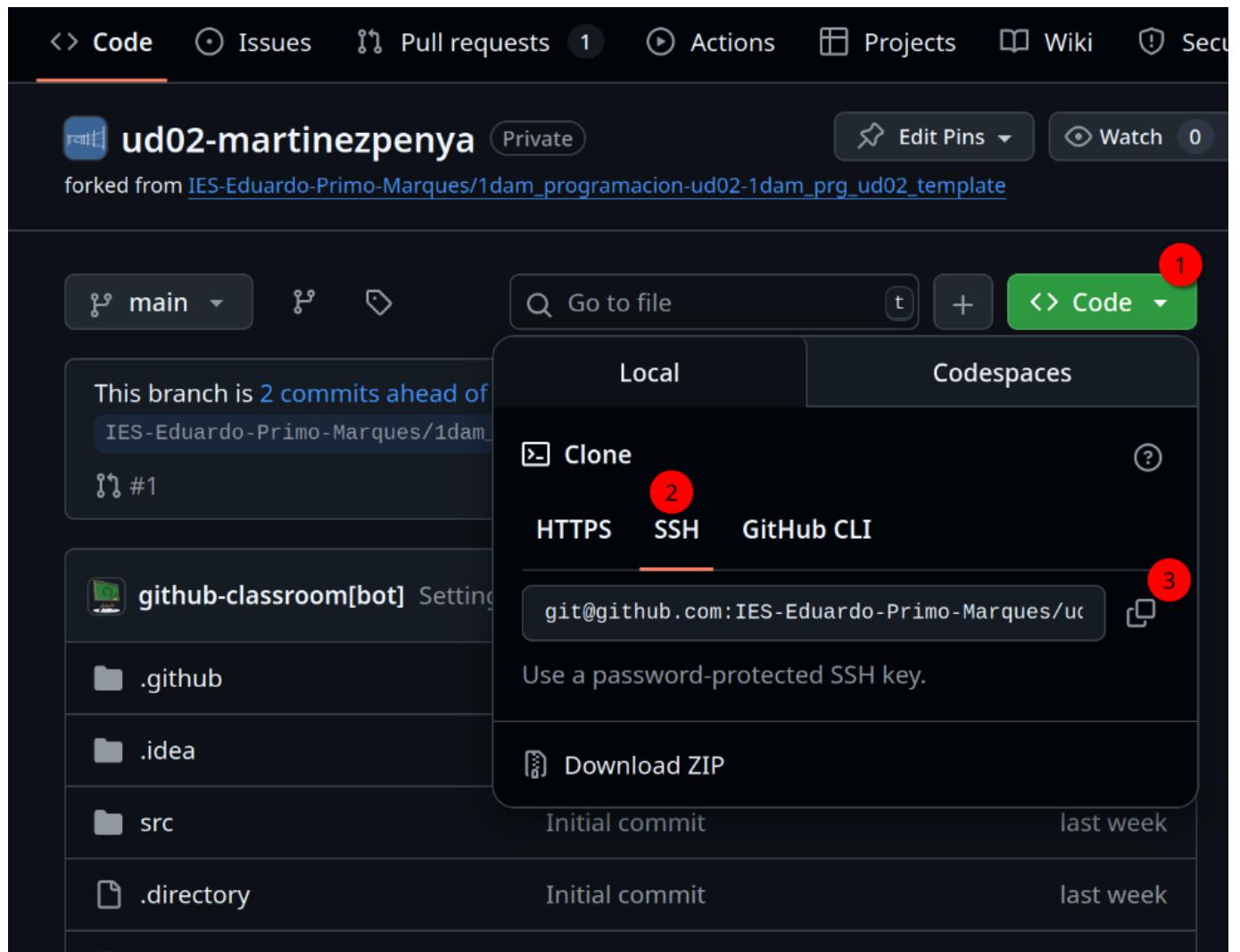
Verified students receive free GitHub Pro plus thousands of dollars worth of the best real-world tools and training from GitHub Education partners — for free. For more information, visit [GitHub Student Developer Pack](#).

[Apply](#)

Note: You may receive an email invitation to join [IES-Eduardo-Primo-Marques](#) on your behalf.  
No further action is necessary.

Abrimos el enlace que aparece con fondo azul: <https://github.com/IES-Eduardo-Primo-Marques/ud02-martinezpenya> (Vuestro enlace será diferente, este es el mio)

Esto nos lleva al repositorio en GitHub, y desde allí, copiamos la URL del repositorio, pero la de `ssh` en lugar de `https`:



#### 8.1.3. Preparar nuestra autenticación a GitHub mediante clave privada/pública (necesaria para usar ssh)

En agosto 2021 GitHub eliminó la autenticación por contraseña para operaciones Git en la línea de comandos. Desde el 13 de agosto de 2021, es OBLIGATORIO usar:

- Tokens de acceso personal (Personal Access Tokens - PAT) para HTTPS
- Claves SSH para conexiones SSH
- GitHub CLI con su propio sistema de autenticación

##### 8.1.3.1. PASO 1: GENERAR UNA NUEVA CLAVE SSH

1. Abre la consola o PowerShell

2. Genera la clave SSH:

```
1 ssh-keygen -t ed25519 -C "tu_email@ejemplo.com"
```

Reemplaza "tu\_email@ejemplo.com" con tu email de GitHub

Para sistemas más antiguos, usa: ssh-keygen -t rsa -b 4096 -C "tu\_email@ejemplo.com"

3. Sigue las instrucciones:

```
1 Enter file in which to save the key (C:\Users\tunombre/.ssh/id_ed25519):
```

Presiona **Enter** para aceptar la ubicación por defecto

## 1. Establece una contraseña segura (opcional pero recomendado):

```
1 Enter passphrase (empty for no passphrase):
2 Enter same passphrase again:
```

## 2. Introduce una contraseña segura y confírmala.

### 8.1.3.2. PASO 2: LOCALIZAR Y COPIAR LA CLAVE PÚBLICA

1. Navega al directorio `.ssh`: Deberías ver `id_ed25519` (clave privada) y `id_ed25519.pub` (clave pública)
2. Copia la clave pública `id_ed25519.pub`:

Abre el archivo en el Bloc de notas y copia el contenido del fichero

### 8.1.3.3. PASO 3: INICIAR EL AGENTE Y AGREGAR LA CLAVE PRIVADA (OPCIONAL)

1. Abre PowerShell (esta vez necesitarás hacerlo como Administrador)

## 2. Verifica si el servicio SSH está instalado:

```
1 Get-WindowsCapability -Online | Where-Object Name -like 'OpenSSH*'
```

## 1. Si falta algún componente, instálalo:

```
1 #Instalar cliente SSH
2 Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
3 #Instalar servidor SSH
4 Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
```

## 1. Inicia el agente SSH:

```
1 # Iniciar servicio
2 Start-Service ssh-agent
3
4 # Configurar para inicio automático
5 Set-Service -Name ssh-agent -StartupType Automatic
```

## 1. Agrega tu clave SSH:

```
1 ssh-add $env:USERPROFILE\.ssh\id_ed25519
```

### 8.1.3.4. PASO 4: CONFIGURAR LA CLAVE SSH EN GITHUB

1. **Inicia sesión en tu cuenta de GitHub** - Ve a [github.com](https://github.com) y accede a tu cuenta
2. **Accede a la configuración de SSH**: - Haz clic en tu **foto de perfil** (esquina superior derecha) - Selecciona **Settings** - En el menú lateral, haz clic en **SSH and GPG keys**
3. **Agrega una nueva clave SSH**: - Haz clic en el botón **New SSH key** o **Add SSH key**
4. **Completa los campos**: - **Title**: Un nombre descriptivo (ej: "Mi PC Windows") - **Key type**: Dejar como "Authentication Key" - **Key**: Pega la clave pública que copiaste anteriormente
5. **Guarda la clave**: - Haz clic en **Add SSH key** - Confirma tu contraseña de GitHub si es necesario

### 8.1.3.5. PASO 5: VERIFICAR LA CONEXIÓN SSH

## 1. Prueba la conexión en la terminal o PowerShell:

Si ejecutas este comando:

```
1 ssh -T git@github.com
```

Deberías ver algo parecido a esto (después de contestar `yes` a la pregunta, fíjate que aparece tu nombre de GitHub, en mi caso `martinezpenya`):

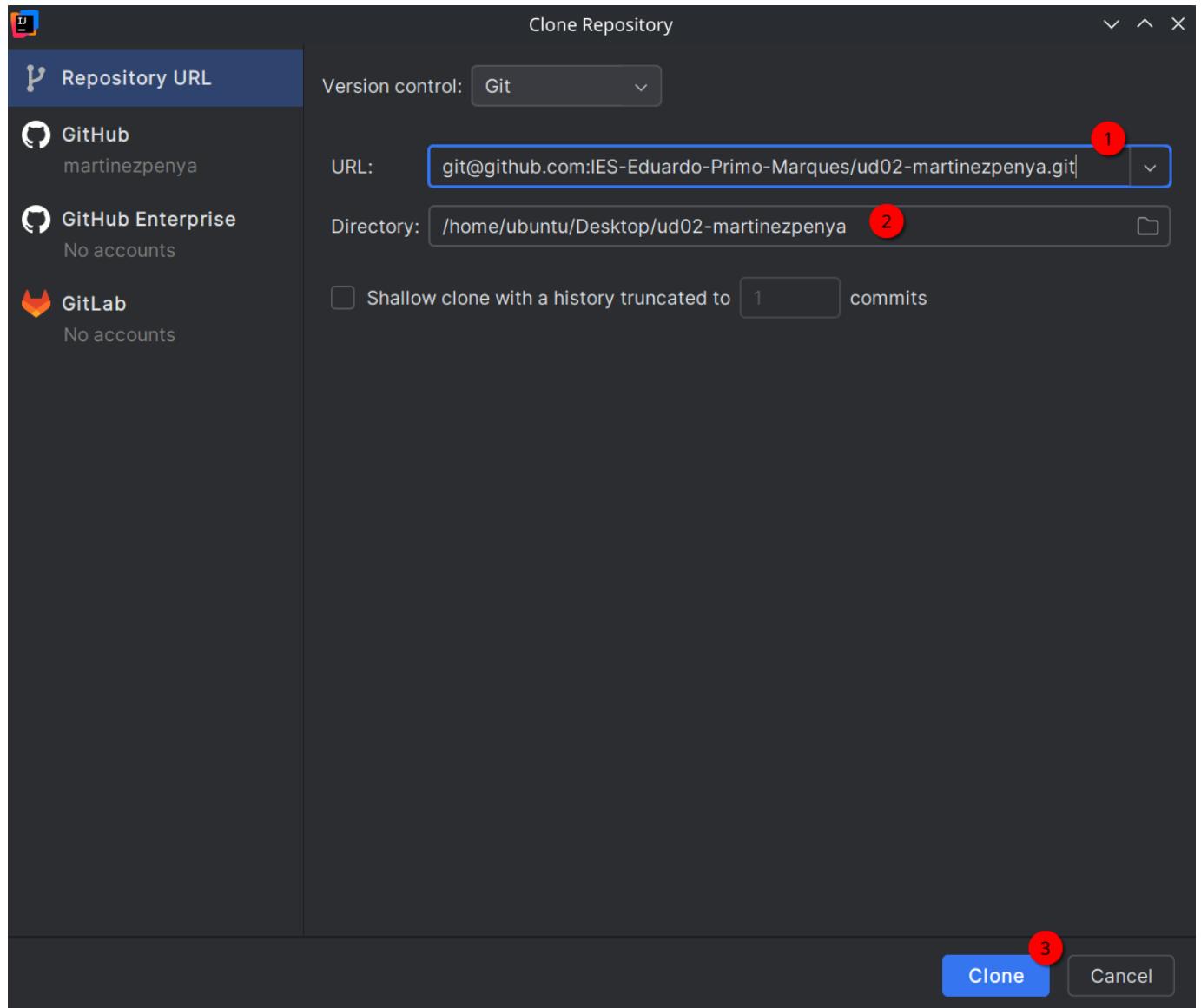
```

1 The authenticity of host 'github.com (140.82.121.3)' can't be established.
2 ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvC0qU.
3 This key is not known by any other names.
4 Are you sure you want to continue connecting (yes/no/[fingerprint])?yes
5 Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
6 Hi martinezpenya! You've successfully authenticated, but GitHub does not provide shell access.

```

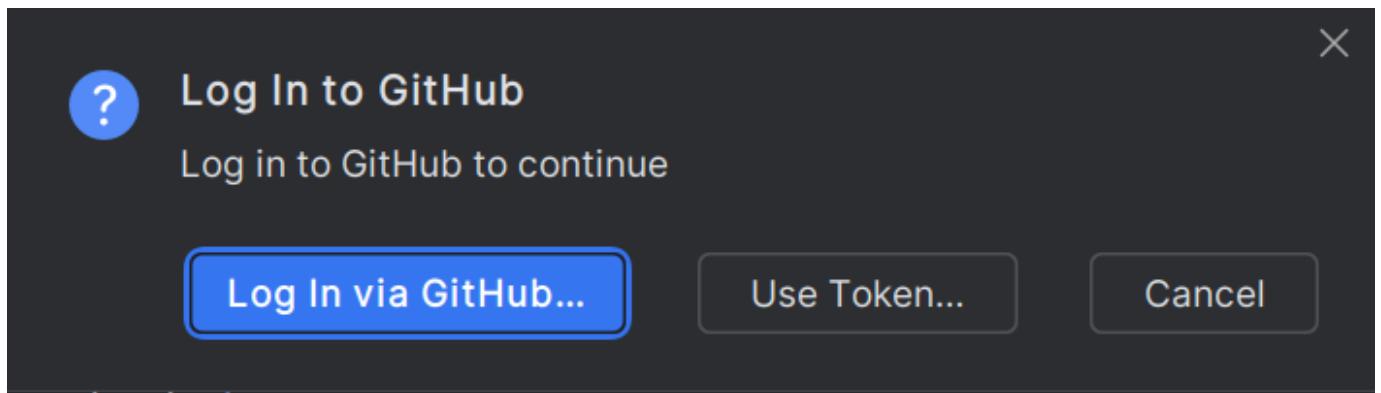
#### 8.1.4. Crear el proyecto en IntelliJ

Ahora abriremos IntelliJ y crearemos un nuevo proyecto (New Project from Version Control, o Project/Clone Repository) a partir de la url que hemos copiado en el punto anterior:



1. Pegamos la URL del paso anterior
2. Elegimos la ubicación de nuestro PC donde guardaremos el proyecto
3. Pulsamos el botón clonar

A continuación nos pedirá hacer Login con nuestra cuenta de GitHub:



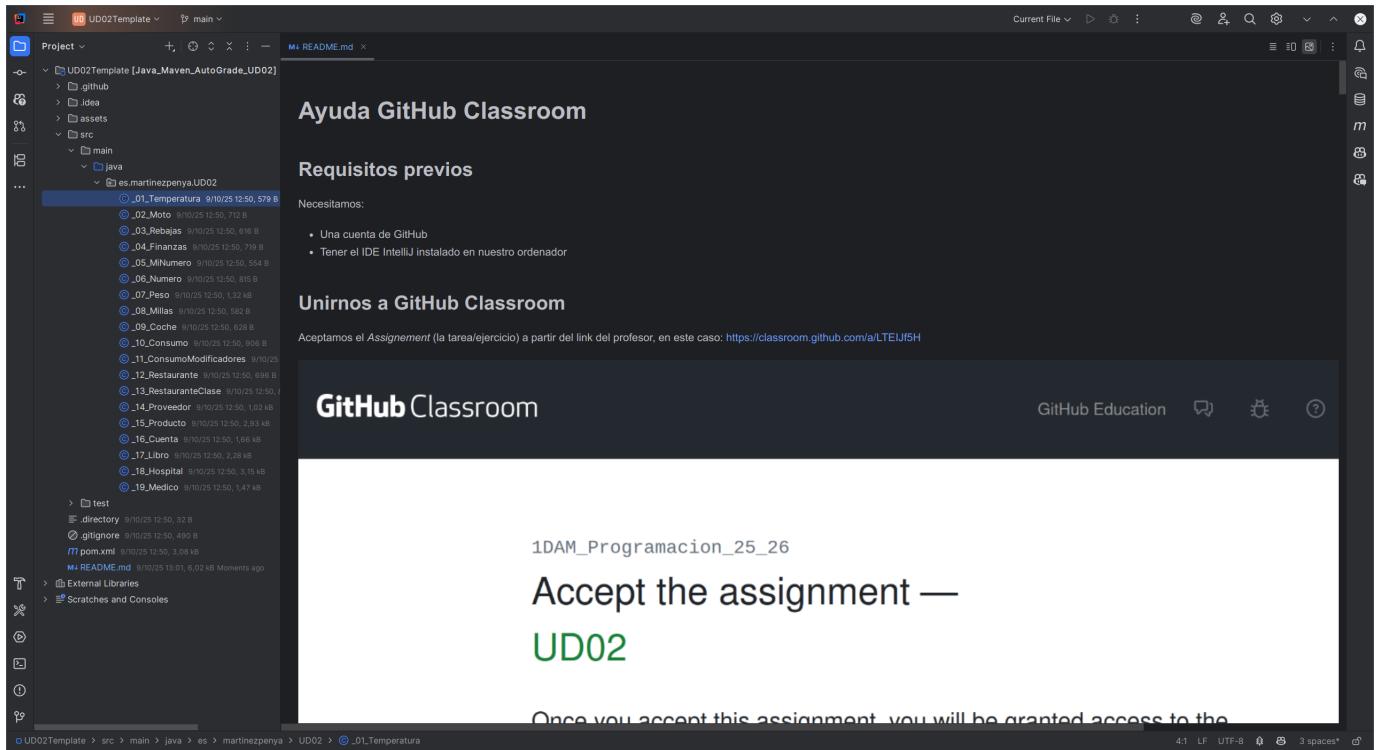
Y en nuestro navegador debemos Autorizar la vinculación de GitHub en el IDE de JetBrains:



Y cuando todo esté correcto aparecerá:



Y ahora en nuestro IDE IntelliJ tenemos:



Ahora en la carpeta `src` debemos buscar el enunciado del ejercicio, en este caso `src/main/java/es/martinezpenya/UD02/_01_Temperatura.java`

```

1 package es.martinezpenya.UD02;
2
3 import java.util.Scanner;
4
5 /**
6 *
7 * @author David Martínez (https://www.martinezpenya.es | https://martinezpenya.es/1DAMProgramacion/)
8 */
9
10 /*
11 ENUNCIADO (Puntos: 2)
12 (Temperatura) Crear una clase llamada Temperatura con dos métodos:
13
14 - `celsiusToFahrenheit`. Convierte grados *Celsius* a *Farenheit*.
15 $$ F=(1,8*C)+32
16 $$ C=\frac{F-32}{1,8}
17
18 - `fahrenheitToCelsius`. Convierte grados *Farenheit* a *Celsius*.
19 $$ C=\frac{F-32}{1,8}
20
21 */
22
23
24 */
25
26 public class _01_Temperatura {
27 /* TODO: Tu solución aquí */
28 }
```

Debemos leer el enunciado y escribir nuestro código donde pone:

```

1 ...
2 /* TODO: Tu solución aquí */
3 ...
```

Una vez comprobamos que el código funciona correctamente, vamos a comprobar si pasará los test impuestos por el profesor.

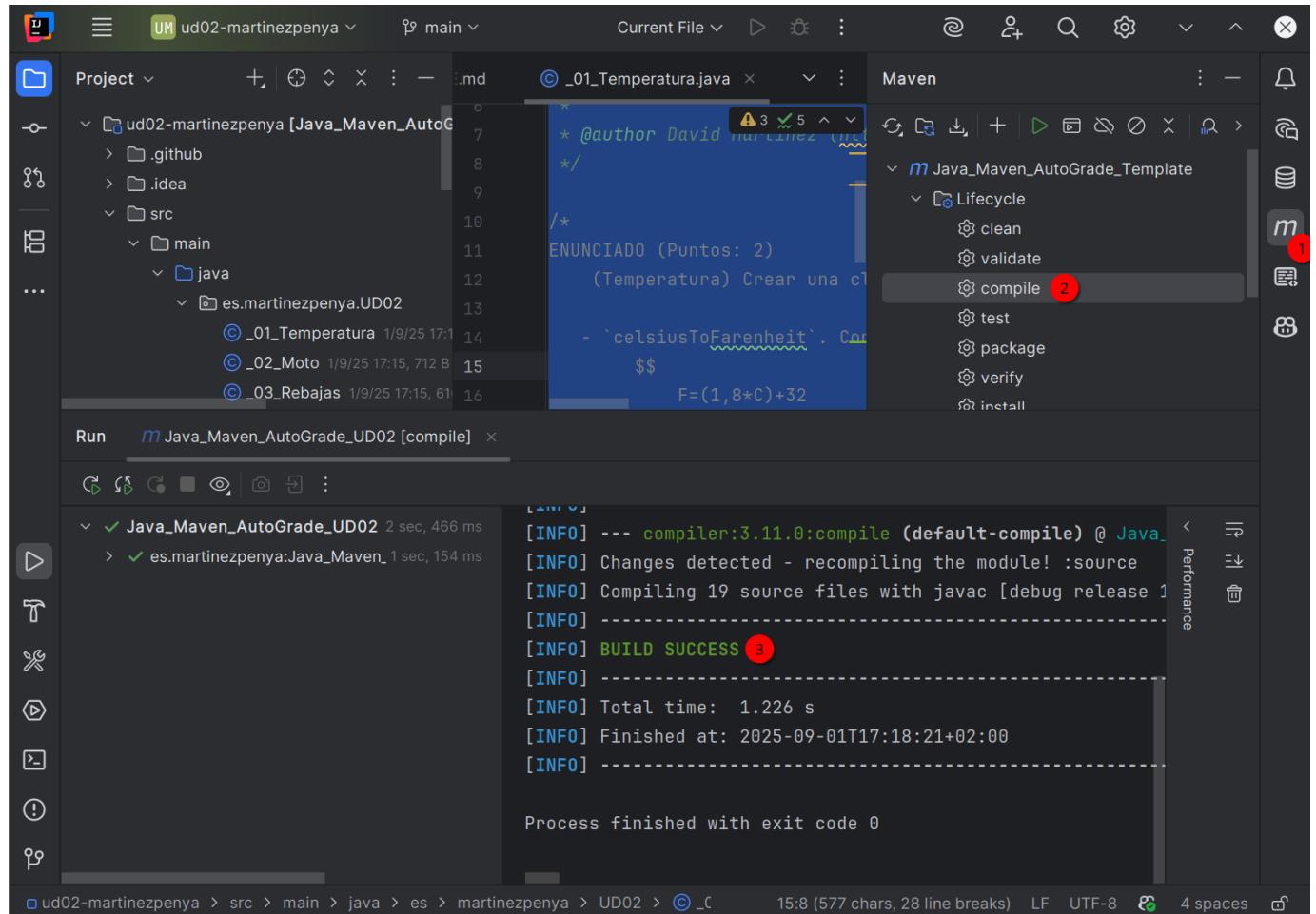
### ⚠️ Importante

En las siguientes capturas se asume que hemos resuelto correctamente el primer ejercicio, por tanto pasamos 2 de los 69 tests que contiene la UD02.

### 8.1.5. Comprobar los tests (Maven)

Debemos realizar dos acciones:

1. Primero compilar el proyecto completo con Maven:



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under 'ud02-martinezpenya' with modules like '.github', '.idea', and 'src/main/java/es/martinezpenya/UD02' containing files '\_01\_Temperatura.java', '\_02\_Moto.java', and '\_03\_Rebajas.java'.
- Maven Tool Window:** Shows the 'Lifecycle' section with goals: clean, validate, compile (highlighted with a red circle), test, package, verify, and install.
- Code Editor:** Displays '\_01\_Temperatura.java' with code related to temperature conversion and some annotations.
- Run Tab:** Shows a successful build for 'Java\_Maven\_AutoGrade\_UD02 [compile]'.
- Output Tab:** Shows Maven build logs with '[INFO] BUILD SUCCESS' highlighted with a red circle.
- Status Bar:** Shows the current file path 'ud02-martinezpenya > src > main > java > es > martinezpenya > UD02 > \_01\_Temperatura.java', character count '15:8 (577 chars, 28 line breaks)', encoding 'LF', and line width '4 spaces'.

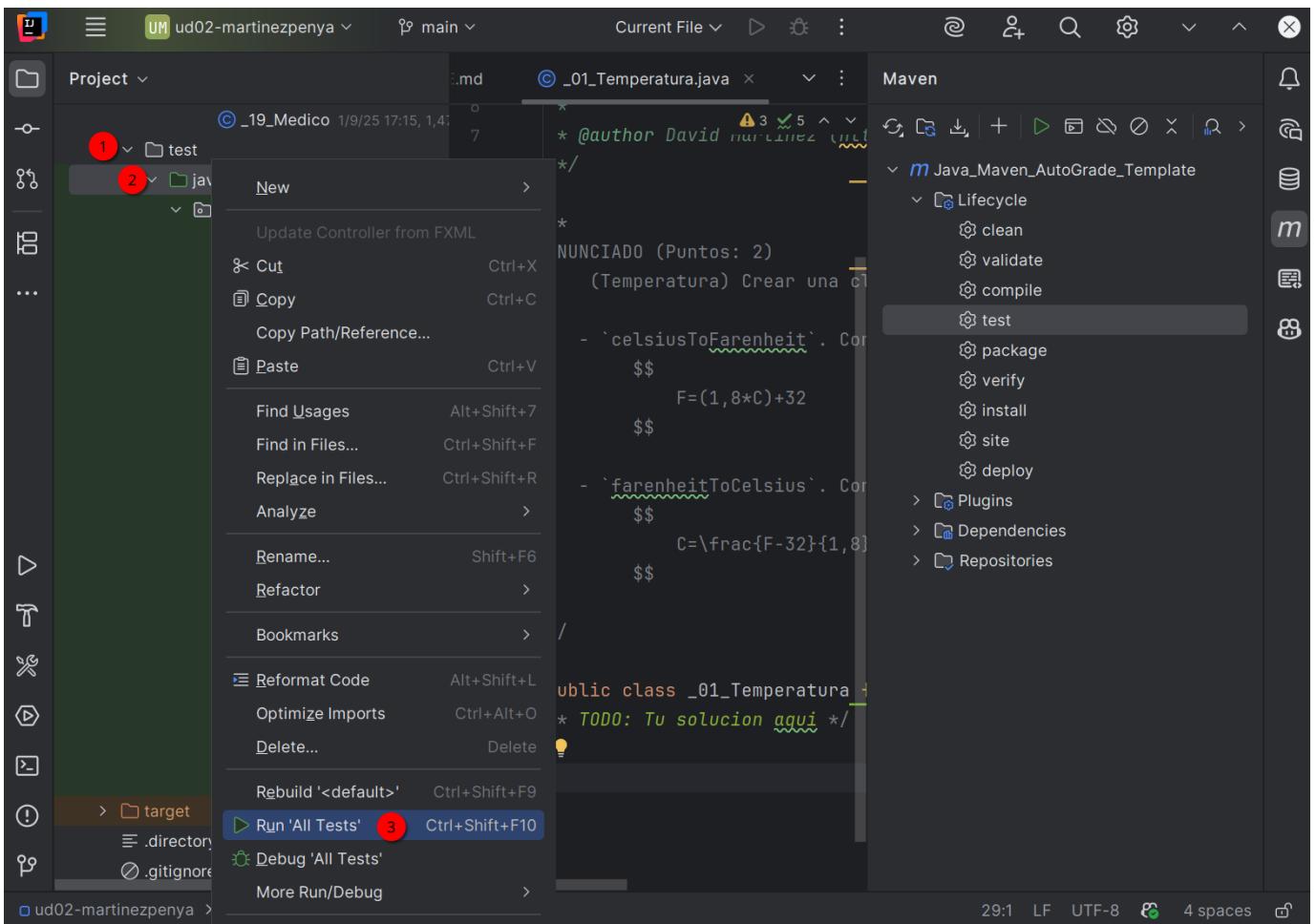
Una vez comprobado que aparece **BUIL SUCCESS** (seguimos).

1. Lanzar la tarea test y comprobar que sea correcta:

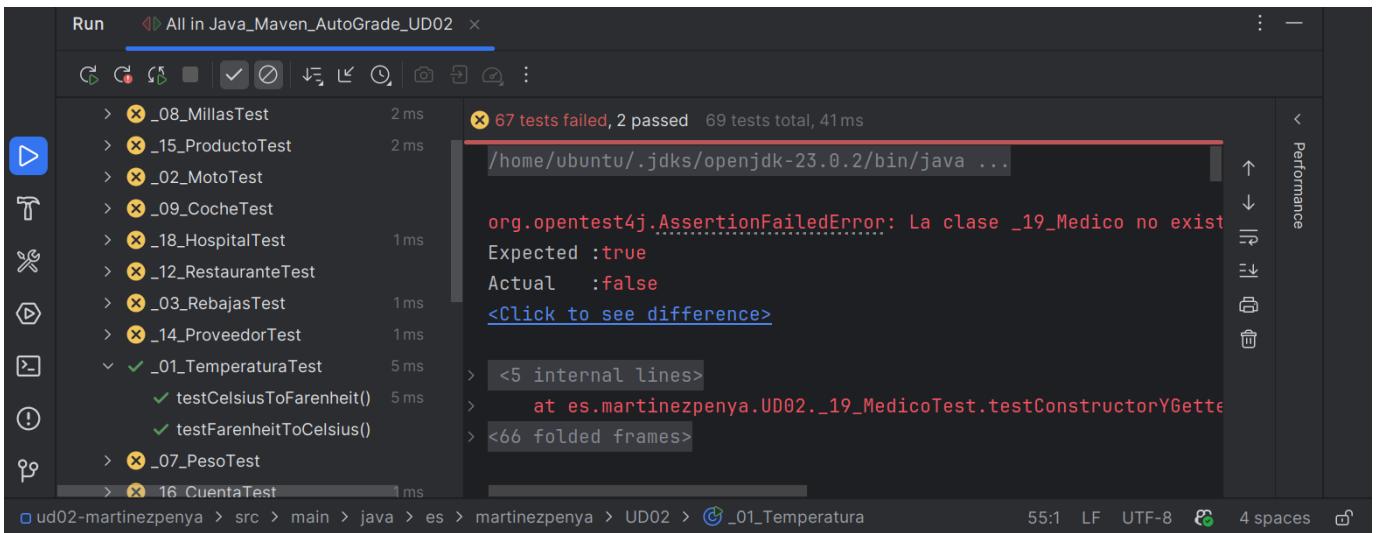
The screenshot shows the IntelliJ IDEA interface. The top navigation bar includes tabs for 'Project', 'Current File', and 'Maven'. The left sidebar has sections for 'Project', 'File Structure', 'Tool Windows', and 'Run/Debug'. The main area displays a Java file named '\_01\_Temperatura.java' with code related to temperature conversion. The 'Maven' tool window on the right shows the 'Lifecycle' section with various goals: 'clean', 'validate', 'compile', 'test' (which is highlighted with a red circle containing the number '2'), 'package', 'verify', and 'install'. The 'Run' tool window at the bottom shows the execution of 'Java\_Maven\_AutoGrade\_UD02 [test]'. The terminal output shows several test cases running, with a green box highlighting the final message: '[ERROR] Tests run: 69, Failures: 69, Errors: 0, Skipped: 0' and '[INFO] BUILD FAILURE'. The status bar at the bottom indicates the current file path as 'martinezpenya > src > main > java > es > martinezpenya > UD02 > \_01\_Temperatura.java', the line count as '15:8 (577 chars, 28 line breaks)', and encoding as 'LF - UTF-8'.

O mejor todavía, podemos ejecutar todos los test con información más visual:

1. Buscamos la carpeta `test` en el proyecto
2. Pulsamos el botón derecho del ratón sobre la carpeta `java`
3. Elegimos la opción "Run 'All Tests'"



Y deberíamos ver un apartado similar en la parte inferior izquierda de nuestro IDE:

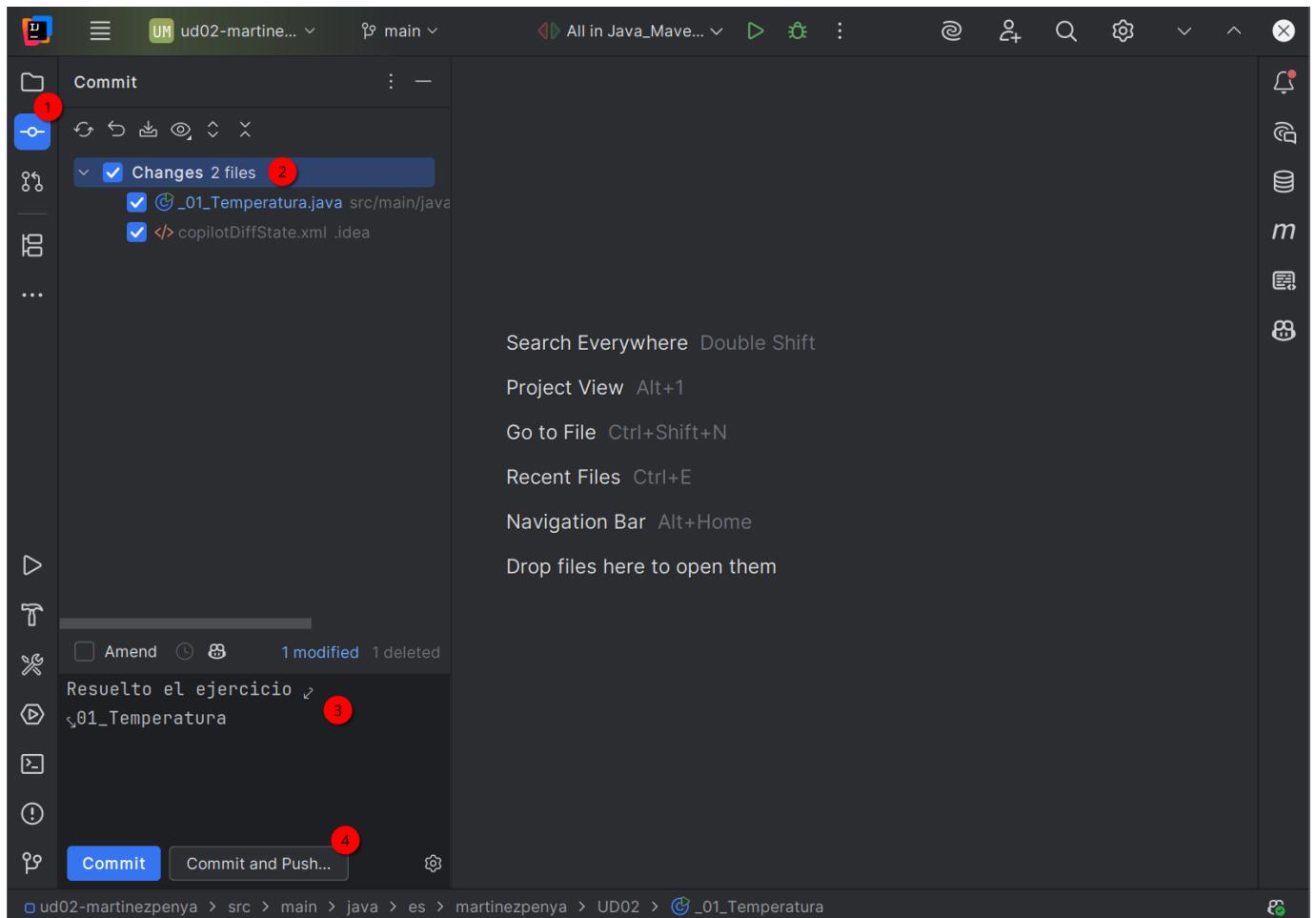


Debes tener en cuenta que:

1. Los tests no se ordenan alfabéticamente.
2. Si aparece un circulo amarillo con una X dentro, el test no pasa (aunque alguno de los tests internos funcionen).
3. Si el tick aparece en verde es que el test ha pasado.
4. Solo se conceden puntos en GitHub Classroom si todos los test de la clase funcionan. En este caso solo la clase `_01_Temperatura` nos otorgará puntos.

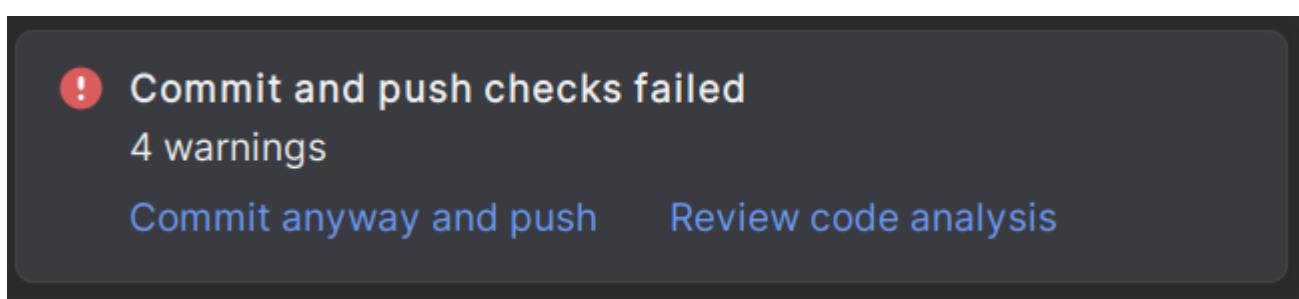
### 8.1.6. Subir nuestra solución a GitHub Classroom

Una vez comprobado que pasamos los tests lo que queda es subir nuestra solución a GitHub Classroom para que sea evaluada y valorada por el docente (Podemos repetir el proceso tantas veces como queramos, así que lo podemos ir haciendo según vayamos realizando los ejercicios).



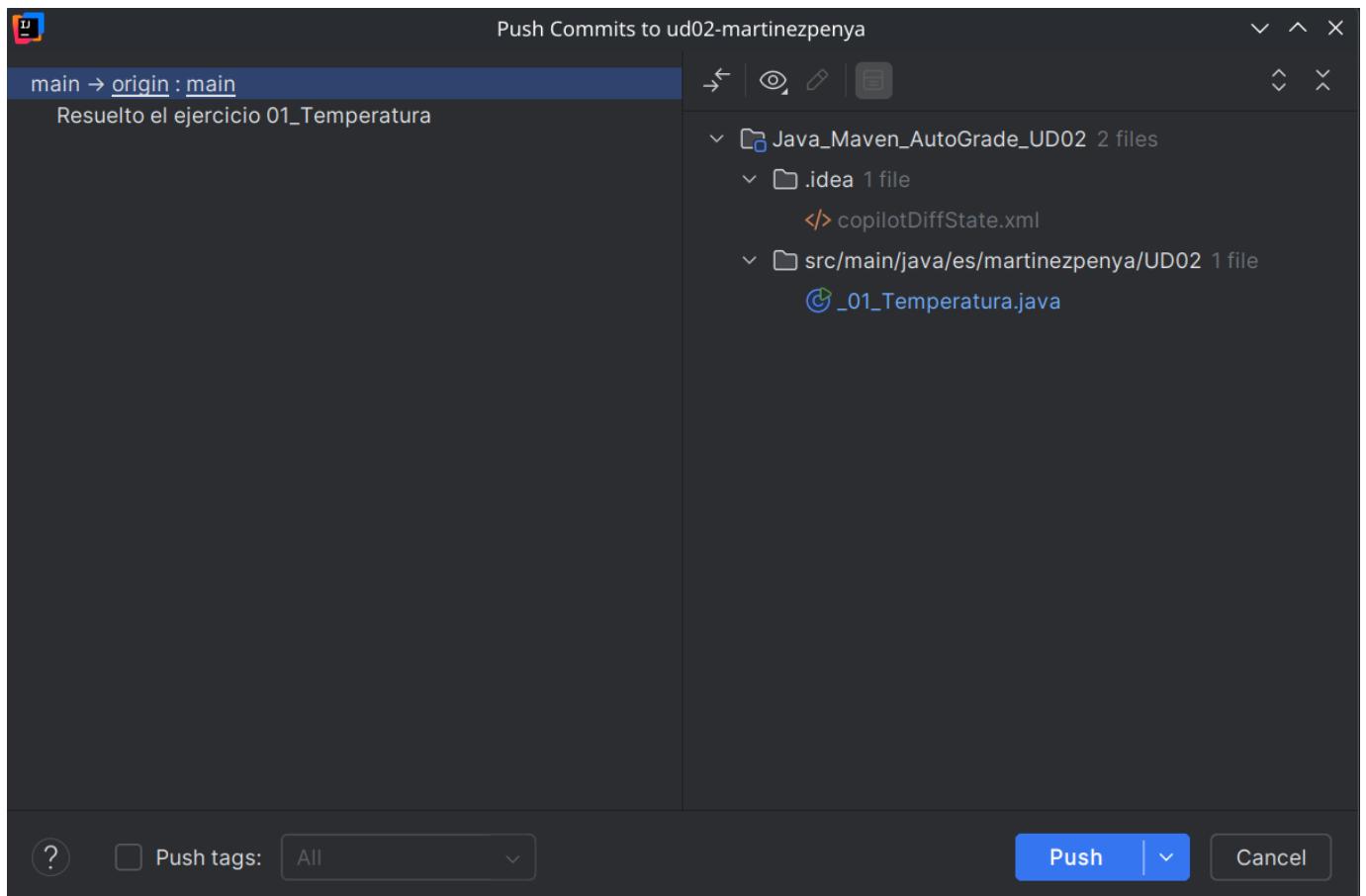
1. Pasamos al apartado de control de versiones
2. Elegimos los archivos que hemos cambiado y que queremos subir al repositorio para ser evaluados
3. Indicamos el mensaje del commit (recomiendo que los puedas identificar de alguna manera, indicando lo que has cambiado, un número de versión, etc.)
4. Finalmente pulsamos el botón Commit adn Push

Si aparece este error:



Podemos elegir la opción `Commit anyway and push` si hemos pasado alguno de los test correctamente, o bien `Review code analysis` para ver porque IntelliJ ha detectado algún problema.

Por último aparece esta pantalla, y pulsamos directamente el botón `Push`:



También podemos ver que test hemos pasado satisfactoriamente en nuestro repositorio de Github:

The screenshot shows the GitHub Classroom Actions page for a repository. The main title is "Resuelto el ejercicio 01\_Temperatura #4". The "Actions" tab is selected, showing one job named "Preparación y ejecución de t..." which failed. The status bar indicates a total duration of 2m 21s and 1 artifact. The "classroom.yml" file is shown with a push event. A detailed view of the failed job shows a summary table and a test results table.

|                        | Tests  | Passed   | Skipped   | Failed    |
|------------------------|--------|----------|-----------|-----------|
| Informe de Autograding | 69 ran | 2 passed | 0 skipped | 67 failed |

| Test                                                    | Result  |
|---------------------------------------------------------|---------|
| <code>es.martinezpenya.UD02_01_TemperaturaTest</code>   | passed  |
| <code>_01_TemperaturaTest.testCelsiusToFarenheit</code> | passed  |
| <code>_01_TemperaturaTest.testFarenheitToCelsius</code> | passed  |
| <code>es.martinezpenya.UD02_02_MotoTest</code>          |         |
| <code>_02_MotoTest.testAcelera</code>                   | failure |
| <code>_02_MotoTest.testGetVelocidad</code>              | failure |
| <code>_02_MotoTest.testFrena</code>                     | failure |
| <code>es.martinezpenya.UD02_02_BebidasTest</code>       |         |

Por último, cuando el docente evalúe nuestra solución y si todo ha ido bien acabaremos viendo en GitHub Classroom que nuestra solución ha pasado los tests y nos ha asignado una puntuación:

Primero aparecerá como enviado (submitted):

The screenshot shows a student submission card for David Martínez (@martinezpenya). It displays his profile picture, name, submission status ("Submitted"), latest commit details ("Latest commit 2 minutes ago", "1 commit"), and a progress bar indicating 100% completion.

Y cuando esté evaluado tendrá puntuación, en este caso 1 punto de un total de 1:

**David Martínez**

Submitted

@martinezpenya

Latest commit 5 minutes ago

-o- 1 commit

2/55

**8.1.7. OJO: No está permitido modificar archivos del proyecto que no estén en la carpeta src/java**

Aunque aparentemente todo te funcionará el docente será notificado e invalidará los resultados de los test:



Submitted

Protected file(s) modified

Latest commit last week

-o- 2 commits

7/55

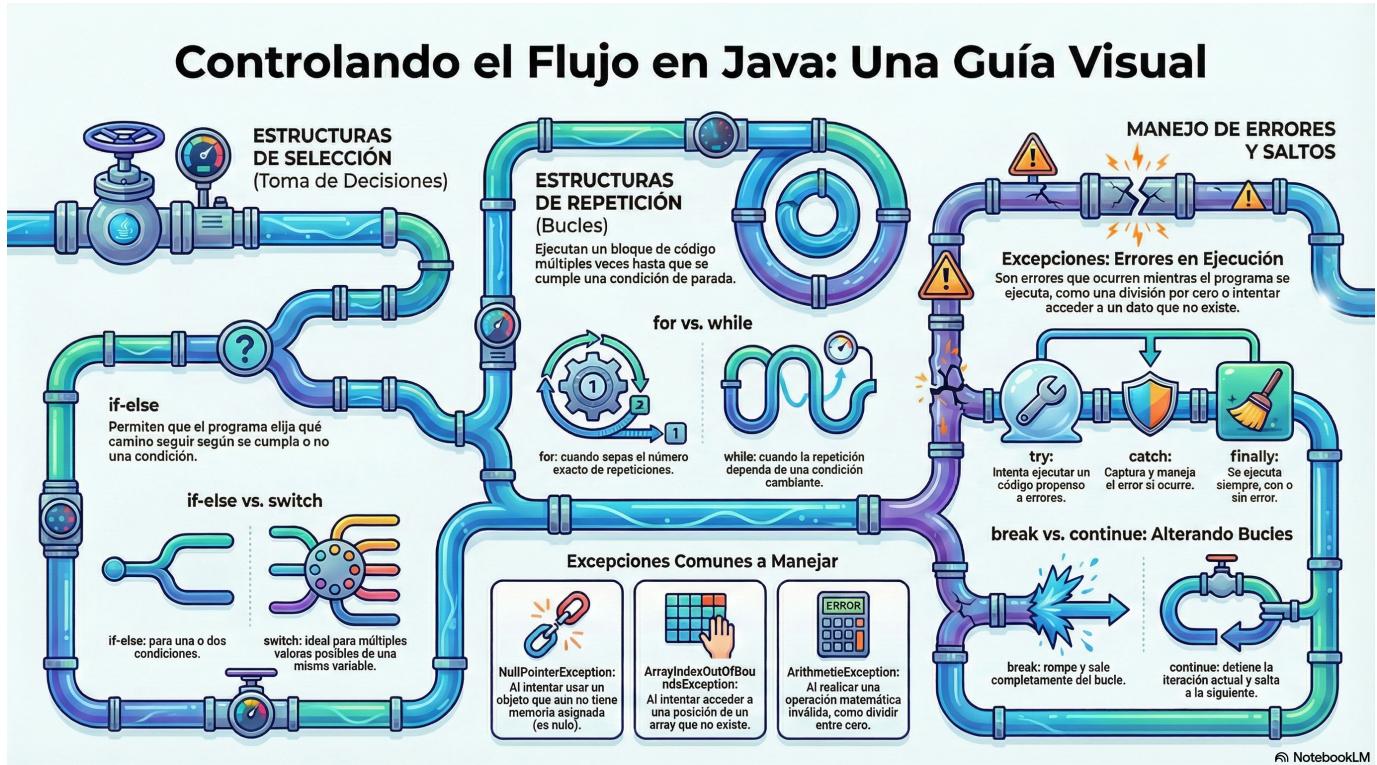
**8.1.8. Tarea**

Debes enviar tus soluciones a GitHub Classroom y superar al menos la mitad de los tests, cuantos más tests superados, mejor nota tendrás en la tarea.

⌚18 de octubre de 2025

## 4. UD03

### 9. 4.1 Estructuras de control y Excepciones



#### 9.1. Introducción

En unidades anteriores has podido aprender cuestiones básicas sobre el lenguaje JAVA: definición de variables, tipos de datos, asignación de valores, uso de literales, diferentes operadores que se pueden aplicar, conversiones de tipos, inserción de comentarios, etc. Posteriormente, nos sumergimos de lleno en el mundo de los objetos. Primero hemos conocido su filosofía, para más tarde ir recorriendo los conceptos y técnicas más importantes relacionadas con ellos: Propiedades, métodos, clases, declaración y uso de objetos, librerías, etc.

Vale, parece ser que tenemos los elementos suficientes para comenzar a generar programas escritos en JAVA, ¿Seguro?

Como habrás deducido, con lo que sabemos hasta ahora no es suficiente. Existen múltiples situaciones que nuestros programas deben representar y que requieren tomar ciertas decisiones, ofrecer diferentes alternativas o llevar a cabo determinadas operaciones repetitivamente para conseguir sus objetivos.

Si has programado alguna vez o tienes ciertos conocimientos básicos sobre lenguajes de programación, sabes que la gran mayoría de lenguajes poseen estructuras que permiten a los programadores controlar el flujo de la información de sus programas. Esto realmente es una ventaja para la persona que está aprendiendo un nuevo lenguaje, o tiene previsto aprender más de uno, ya que estas estructuras suelen ser comunes a todos (con algunos cambios de sintaxis o conjunto de reglas que definen las secuencias correctas de los elementos de un lenguaje de programación.). Es decir, si conocías sentencias de control de flujo en otros lenguajes, lo que vamos a ver a lo largo de esta unidad te va a sonar bastante.

Para alguien que no ha programado nunca, un ejemplo sencillo le va a permitir entender qué es eso de las sentencias de control de flujo. Piensa en un fontanero (programador), principalmente trabaja con agua (datos) y se encarga de hacer que ésta fluya por donde él quiere (programa) a través de un conjunto de tuberías, codos, latiguillos, llaves de paso, etc. (sentencias de control de flujo).

Pues esas estructuras de control de flujo son las que estudiaremos, conoceremos su estructura, funcionamiento, cómo utilizarlas y dónde. A través de ellas, al construir nuestros programas podremos hacer que los datos (agua) fluyan por los caminos adecuados para representar la realidad del problema y obtener un resultado adecuado.

Los tipos de estructuras de programación que se emplean para el control del flujo de los datos son los siguientes:

- **Secuencia:** compuestas por \(\{0\}\), \(\{1\}\) o \(\{N\}\) sentencias que se ejecutan en el orden en que han sido escritas. Es la estructura más sencilla y sobre la que se construirán el resto de estructuras.
- **Selección:** es un tipo de sentencia especial de decisión y de un conjunto de secuencias de instrucciones asociadas a ella. Según la evaluación de la sentencia de decisión se generará un resultado (que suele ser verdadero o falso) y en función de éste, se ejecutarán una secuencia de instrucciones u otra. Las estructuras de selección podrán ser simples, compuestas y múltiples.
- **Iteración:** es un tipo de sentencia especial de decisión y una secuencia de instrucciones que pueden ser repetidas según el resultado de la evaluación de la sentencia de decisión. Es decir, la secuencia de instrucciones se ejecutará repetidamente si la sentencia de decisión arroja un valor correcto, en otro la estructura de repetición se detendrá.

Además de las sentencias típicas de control de flujo, en esta unidad haremos una revisión de las sentencias de salto, que aunque no son demasiado recomendables, es necesario conocerlas. Como nuestros programas podrán generar errores y situaciones especiales, echaremos un vistazo al manejo de excepciones en JAVA. Posteriormente, analizaremos la mejor manera de llevar a cabo las pruebas de nuestros programas y la depuración de los mismos. Y finalmente, aprenderemos a valorar y utilizar las herramientas de documentación de programas.

## 9.2. Sentencias y bloques

Este epígrafe lo utilizaremos para reafirmar cuestiones que son obvias y que en el transcurso de anteriores unidades se han dado por sabidas. Aunque, a veces, es conveniente recordar. Lo haremos como un conjunto de FAQ's:

- **¿Cómo se escribe un programa sencillo?** Si queremos que un programa sencillo realice instrucciones o sentencias para obtener un determinado resultado, es necesario colocar éstas una detrás de la otra, exactamente en el orden en que deben ejecutarse.
- **¿Podrían colocarse todas las sentencias una detrás de otra, separadas por puntos y comas en una misma línea?**, claro que sí, pero no es muy recomendable. Cada sentencia debe estar escrita en una línea, de esta manera tu código será mucho más legible y la localización de errores en tus programas será más sencilla y rápida. De hecho, cuando se utilizan herramientas de programación, los errores suelen asociarse a un número o números de línea.
- **¿Puede una misma sentencia ocupar varias líneas en el programa?**, sí. Existen sentencias que, por su tamaño, pueden generar varias líneas. Pero siempre finalizarán con un punto y coma.
- **¿En Java todas las sentencias se terminan con punto y coma?**, Efectivamente. Si detrás de una sentencia ha de venir otra, pondremos un punto y coma. Escribiendo la siguiente sentencia en una nueva línea. Pero en algunas ocasiones, sobre todo cuando utilizamos estructuras de control de flujo, detrás de la cabecera de una estructura de este tipo no debe colocarse punto y coma. No te preocupes, lo entenderás cuando analicemos cada una de ellas.
- **¿Qué es la sentencia nula en Java?** La sentencia nula es una línea que no contiene ninguna instrucción y en la que sólo existe un punto y coma. Como su nombre indica, esta sentencia no hace nada.
- **¿Qué es un bloque de sentencias?** Es un conjunto de sentencias que se encierra entre llaves y que se ejecutaría como si fuera una única orden. Sirve para agrupar sentencias y para clarificar el código. Los bloques de sentencias son utilizados en Java en la práctica totalidad de estructuras de control de flujo, clases, métodos, etc. La siguiente tabla muestra dos formas de construir un bloque de sentencias.

| Bloque de sentencias 1                     | Bloque de sentencias 2                                          |
|--------------------------------------------|-----------------------------------------------------------------|
| {sentencia1; sentencia2; ...; sentenciaN;} | {<br>sentencia1;<br>sentencia2;<br>...<br>sentenciaN;         } |

- **¿En un bloque de sentencias, éstas deben estar colocadas con un orden exacto?** En ciertos casos sí, aunque si al final de su ejecución se obtiene el mismo resultado, podrían ocupar diferentes posiciones en nuestro programa.

**DEBES CONOCER** Observa los tres archivos que te ofrecemos a continuación y compara su código fuente. Verás que los tres obtienen el mismo resultado, pero la organización de las sentencias que los componen es diferente entre ellos.

**Ejemplo 1:**

```

1 package organizacion_sentencias1;
2 /**
3 *
4 * Organización de sentencias secuencial
5 */
6 public class Organizacion_sentencias_1 {
7 public static void main(String[] args) {
8 System.out.println ("Organización secuencial de sentencias");
9 int dia=12;
10 System.out.println ("El dia es: " + dia);
11 int mes=11;
12 System.out.println ("El mes es: " + mes);
13 int anio=2011;
14 System.out.println ("El año es: " + anio);
15 }
16 }
```

En este primer archivo, las sentencias están colocadas en orden secuencial.

**Ejemplo 2:**

```

1 package organizacion_sentencias2;
2 /**
3 *
4 * Organización de sentencias con declaración previa de variables
5 */
6 public class Organizacion_sentencias_2 {
7 public static void main(String[] args) {
8 // Zona de declaración de variables
9 int dia=10;
10 int mes=11;
11 int anio=2011;
12 System.out.println ("Organización con declaración previa de variables");
13 System.out.println ("El día es: " + dia);
14 System.out.println ("El mes es: " + mes);
15 System.out.println ("El año es: " + anio);
16 }
17 }
```

En este segundo archivo, se declaran al principio las variables necesarias. En Java no es imprescindible hacerlo así, pero sí que antes de utilizar cualquier variable ésta debe estar previamente declarada. Aunque la declaración de dicha variable puede hacerse en cualquier lugar de nuestro programa.

**Ejemplo 3:**

```

1 package organizacion_sentencias3;
2 /**
3 *
4 * Organización de sentencias en zonas diferenciadas
5 * según las operaciones que se realicen en el código
6 */
7 public class Organizacion_sentencias_3 {
8 public static void main(String[] args) {
9 // Zona de declaración de variables
10 int dia;
11 int mes;
12 int anio;
13 String fecha;
14 //Zona de inicialización o entrada de datos
15 dia=10;
16 mes=11;
17 anio=2011;
18 fecha="";
19 //Zona de procesamiento
20 fecha=dia+"/"+mes+"/"+anio;
21 //Zona de salida
22 System.out.println ("Organización con zonas diferenciadas en el código");
23 System.out.println ("La fecha es: " + fecha);
24 }
25 }
```

En este tercer archivo, podrás apreciar que se ha organizado el código en las siguientes partes: declaración de variables, petición de datos de entrada, procesamiento de dichos datos y obtención de la salida. Este tipo de organización está más estandarizada y hace que nuestros programas ganen en legibilidad.

### Acción

Construyas de una forma o de otra tus programas, debes tener en cuenta siempre en Java las siguientes premisas: - **Declara** cada variable antes de utilizarla. - **Inicializa** con un valor cada variable la primera vez que la utilices. No es recomendable usar variables no inicializadas en nuestros programas, pueden provocar errores o resultados imprevistos.

## 9.3. Estructuras de selección

¿Cómo conseguimos que nuestros programas puedan tomar decisiones? Para comenzar, lo haremos a través de las estructuras de selección. Estas estructuras constan de una sentencia especial de decisión y de un conjunto de secuencias de instrucciones.

El funcionamiento es sencillo, la sentencia de decisión será evaluada y ésta devolverá un valor (verdadero o falso), en función del valor devuelto se ejecutará una secuencia de instrucciones u otra.

Por ejemplo, si el valor de una variable es mayor o igual que 5 se imprime por pantalla la palabra APROBADO y si es menor, se imprime SUSPENSO. Para este ejemplo, la comprobación del valor de la variable será la sentencia especial de decisión. La impresión de la palabra APROBADO será una secuencia de instrucciones y la impresión de la palabra SUSPENSO será otra. Cada secuencia estará asociada a cada uno de los resultados que puede arrojar la evaluación de la sentencia especial de decisión. Las estructuras de selección se dividen en:

1. Estructuras de selección simples o estructura if.
2. Estructuras de selección compuesta o estructura ifelse.
3. Estructuras de selección basadas en el operador condicional.
4. Estructuras de selección múltiples o estructura switch.

A continuación, detallaremos las características y funcionamiento de cada una de ellas. Es importante que a través de los ejemplos que vamos a ver, puedas determinar en qué circunstancias utilizar cada una de estas estructuras. Aunque un mismo problema puede ser resuelto con diferentes estructuras e incluso, con diferentes combinaciones de éstas.

### 9.3.1. Estructura if, if else, if else if

La estructura `if` es una estructura de selección o estructura condicional, en la que se evalúa una expresión lógica o sentencia de decisión y en función del resultado, se ejecuta una sentencia o un bloque de éstas. La estructura `if` puede presentarse de las siguientes formas:

#### Estructura if simple:

```
1 if (expresión-lógica)
2 sentencia1;
```

```
1 if (expresión-lógica){
2 sentencia1;
3 sentencia2;
4 ...
5 sentenciaN;
6 }
```

Si la evaluación de la expresión-lógica ofrece un resultado verdadero, se ejecuta la sentencia1 o bien el bloque de sentencias asociado. Si el resultado de dicha evaluación es falso, no se ejecutará ninguna instrucción asociada a la estructura condicional.

### Estructura `if` de doble alternativa.

```

1 if (expresión-lógica)
2 sentencia1;
3 else
4 sentencia2;
5 sentencia3;

```

```

1 if (expresión-lógica){
2 sentencia1;
3 ...
4 sentenciaN;
5 } else {
6 sentencia1;
7 ...
8 sentenciaN;
9 }

```

Si la evaluación de la expresión-lógica ofrece un resultado verdadero, se ejecutará la primera sentencia o el primer bloque de sentencias. Si, por el contrario, la evaluación de la expresión-lógica ofrece un resultado falso, no se ejecutará la primera sentencia o el primer bloque y sí se ejecutará la segunda sentencia o el segundo bloque.

#### Ejemplo

Haciendo una interpretación cercana al pseudocódigo tendríamos que si se cumple la condición (expresión lógica), se ejecutará un conjunto de instrucciones y si no se cumple, se ejecutará otro conjunto de instrucciones.

Hay que tener en cuenta que la cláusula `else` de la sentencia `if` no es obligatoria. En algunos casos no necesitaremos utilizarla, pero sí se recomienda cuando es necesario llevar a cabo alguna acción en el caso de que la expresión lógica no se cumpla.

En aquellos casos en los que no existe cláusula `else`, si la expresión lógica es falsa, simplemente se continuarán ejecutando las siguientes sentencias que aparezcan bajo la estructura condicional `if`.

Los condicionales `if` e `if-else` pueden anidarse, de tal forma que dentro de un bloque de sentencias puede incluirse otro `if` o `if-else`. El nivel de anidamiento queda a criterio del programador, pero si éste es demasiado profundo podría provocar problemas de eficiencia y legibilidad en el código. En otras ocasiones, un nivel de anidamiento excesivo puede denotar la necesidad de utilización de otras estructuras de selección más adecuadas.

Cuando se utiliza anidamiento de este tipo de estructuras, es necesario poner especial atención en saber a qué `if` está asociada una cláusula `else`. Normalmente, un `else` estará asociado con el `if` inmediatamente superior o más cercano que exista dentro del mismo bloque y que no se encuentre ya asociado a otro `else`.

### Estructura `if else if`.

Esta estructura es una alternativa a la anidación de sentencias `if else` funciona de modo que si se cumple una condición ejecuta unas sentencias y el caso contrario comprueba otra condición ejecutando unas sentencias si se cumple y así sucesivamente.

Veamos un ejemplo con `if` anidados:

```

1 if (condicion1) {
2 sentencias1;
3 } else {
4 if (condicion2) {
5 sentencias2;
6 } else {
7 if (condicion3) {
8 sentencias3;
9 } else {
10 sentencias4;
11 }
12 }
13 }

```

### El mismo ejemplo usando `if` `else if` quedaría de este modo:

```

1 if (condicion1) {
2 sentencias1;
3 } else if (condicion2) {
4 sentencias2;
5 } else if (condicion3) {
6 sentencias3;
7 } else {
8 sentencias4;
9 }

```

#### 9.3.2. Estructura `switch`

¿Qué podemos hacer cuando nuestro programa debe elegir entre más de dos alternativas?, una posible solución podría ser emplear estructuras if anidadas, aunque no siempre esta solución es la más eficiente. Cuando estamos ante estas situaciones podemos utilizar la estructura de selección múltiple switch. En la siguiente tabla se muestra tanto la sintaxis, como el funcionamiento de esta estructura.

### Sintaxis `switch`:

```

1 switch (expresion) {
2 case valor1:
3 sentencia1_1;
4 sentencia1_2;
5 ...
6 break;
7 case valor2:
8 ...
9 case valorN:
10 sentenciaN_1;
11 sentenciaN_2;
12 ...
13 break;
14 default:
15 sentencias-default;
16 }

```

#### Condiciones:

- Donde expresión debe ser del tipo `char`, `byte`, `short` o `int`, y las constantes de cada `case` deben ser de este tipo o de un tipo compatible.
- La `expresión` debe ir entre paréntesis.
- Cada `case` llevará asociado un `valor` y se finalizará con dos puntos (`:`).
- El bloque de sentencias asociado a la cláusula `default` puede finalizar con una sentencia de ruptura `break` o no.

#### Funcionamiento:

- Las diferentes alternativas de esta estructura estarán precedidas de la cláusula `case` que se ejecutará cuando el valor asociado al `case` coincida con el valor obtenido al evaluar la expresión del `switch`.
- En las cláusulas `case`, no pueden indicarse expresiones condicionales, rangos de valores o listas de valores. (otros lenguajes de programación sí lo permiten). Habrá que asociar una cláusula `case` a cada uno de los valores que deban ser tenidos en cuenta.
- La cláusula `default` será utilizada para indicar un caso por defecto, las sentencias asociadas a la cláusula `default` se ejecutarán si ninguno de los valores indicados en las cláusulas `case` coincide con el resultado de la evaluación de la expresión de la estructura `switch`.
- La cláusula `default` puede no existir, y por tanto, si ningún `case` ha sido activado finalizaría el `switch`.
- Cada cláusula `case` puede llevar asociadas una o varias sentencias, sin necesidad de delimitar dichos bloques por medio de llaves.
- En el momento en el que el resultado de la evaluación de la expresión coincide con alguno de los valores asociados a las cláusulas `case`, se ejecutarán todas las instrucciones asociadas hasta la aparición de una sentencia `break` de ruptura. (la sentencia `break` se analizará en epígrafes posteriores)

##### 9.3.2.1. EXPRESIONES SWITCH MEJORADAS

En las [novedades de Java 12](#) se añadió la posibilidad de los `switch` fueran expresiones que retornan un valor en vez de sentencias y se evita el uso de la palabra reservada `break`.

## Sentencia Switch mejorada

```

1 int entero = 5;
2
3 String numericString = switch (entero) {
4 case 0 -> "cero";
5 case 1, 3, 5, 7, 9 -> "impar";
6 case 2, 4, 6, 8, 10 -> "par";
7 default -> "error";
8 };
9 System.out.println(numericString); //impar

```

En Java 13 en vez de únicamente el valor a retornar se permite crear bloques de sentencias para cada rama `case` y retornar el valor con la palabra reservada `yield`. En los bloques de sentencias puede haber algún cálculo más complejo que directamente retornar el valor deseado.

```

1 int entero2 = 4;
2
3 String numericString2 = switch (entero2) {
4 case 0 -> {
5 String value = calculaCero();
6 yield value;
7 }
8 case 1, 3, 5, 7, 9 -> {
9 String value = calculaImpar();
10 yield value;
11 }
12
13 case 2, 4, 6, 8, 10 -> {
14 String value = calculaPar();
15 yield value;
16 }
17
18 default -> {
19 String value = calculaDefecto();
20 yield value;
21 }
22 };
23 System.out.println(numericString); //calculaPar()

```

En resumen, se ha de comparar el valor de una expresión con un conjunto de constantes, si el valor de la expresión coincide con algún valor de dichas constantes, se ejecutarán los bloques de instrucciones asociados a cada una de ellas. Si no existiese coincidencia, se ejecutarían una serie de instrucciones por defecto.

## 9.4. Estructuras de repetición

Nuestros programas ya son capaces de controlar su ejecución teniendo en cuenta determinadas condiciones, pero aún hemos de aprender un conjunto de estructuras que nos permita repetir una secuencia de instrucciones determinada. La función de estas estructuras es repetir la ejecución de una serie de instrucciones teniendo en cuenta una condición.

A este tipo de estructuras se las denomina estructuras de repetición, estructuras repetitivas, bucles o estructuras iterativas. En Java existen cuatro clases de bucles:

- Bucle `for` (repite para)
- Bucle `for/in` (repite para cada), aka `for each`
- Bucle `while` (repite mientras)
- Bucle `do while` (repite hasta)

Los bucles `for` y `for/in` se consideran bucles controlados por contador. Por el contrario, los bucles `while` y `do...while` se consideran bucles controlados por sucesos.

La utilización de unos bucles u otros para solucionar un problema dependerá en gran medida de las siguientes preguntas:

- ¿Sabemos a priori cuántas veces necesitamos repetir un conjunto de instrucciones?
- ¿Sabemos si hemos de repetir un conjunto de instrucciones si una condición satisface un conjunto de valores?
- ¿Sabemos hasta cuándo debemos estar repitiendo un conjunto de instrucciones?
- ¿Sabemos si hemos de estar repitiendo un conjunto de instrucciones mientras se cumpla una condición?

Estas y otras preguntas tendrán su respuesta en cuanto analicemos cada una de estructuras repetitivas en detalle.

### Definición

Estudia cada tipo de estructura repetitiva, conoce su funcionamiento y podrás llegar a la conclusión de que algunos de estos bucles son equivalentes entre sí. Un mismo problema, podrá ser resuelto empleando diferentes tipos de bucles y obtener los mismos resultados.

#### 9.4.1. Estructura `for`

Hemos indicado anteriormente que el bucle `for` es un bucle controlado por contador. Este tipo de bucle tiene las siguientes características:

- Se ejecuta un número determinado de veces.
- Utiliza una variable contadora que controla las iteraciones del bucle.

En general, existen tres operaciones que se llevan a cabo en este tipo de bucles:

- Se inicializa la variable contadora.
- Se evalúa el valor de la variable contador, por medio de una comparación de su valor con el número de iteraciones especificado.
- Se modifica o actualiza el valor del contador a través de incrementos o decrementos de éste, en cada una de las iteraciones.

### Ejemplo

La inicialización de la variable contadora debe realizarse correctamente para garantizar que el bucle lleve a cabo, al menos, la primera repetición de su código interno. La condición de terminación del bucle debe variar en el interior del mismo, de no ser así, podemos caer en la creación de un bucle infinito. Cuestión que se debe evitar por todos los medios. Es necesario estudiar el número de veces que se repite el bucle, pues debe ajustarse al número de veces estipulado.

#### Sintaxis estructura `for` con una única sentencia:

```
1 for (inicialización; condición; iteración)
2 sentencia;
```

#### Sintaxis estructura `for` con un bloque de sentencias:

```
1 for (inicialización; condición; iteración) {
2 sentencia1;
3 sentencia2;
4 ...
5 sentenciaN;
6 }
```

Donde....:

- `inicialización` es una expresión en la que se inicializa una variable de control, que será la encargada de controlar el final del bucle.
- `condición` es una expresión que evaluará la variable de control. Mientras la condición sea falsa, el cuerpo del bucle estará repitiéndose. Cuando la condición se cumpla, terminará la ejecución del bucle.
- `iteración` indica la manera en la que la variable de control va cambiando en cada iteración del bucle. Podrá ser mediante incremento o decremento, y no solo de uno en uno.

#### 9.4.2. Estructura `for / in`

Junto a la estructura `for`, `for / in` también se considera un bucle controlado por contador. Este bucle es una mejora incorporada en la versión 5.0 de Java.

Este tipo de bucles permite realizar recorridos sobre arrays y colecciones de objetos. Los arrays son colecciones de variables que tienen el mismo tipo y se referencian por un nombre común. Así mismo, las colecciones de objetos son objetos que se dice son iterables, o que se puede iterar sobre ellos.

Este bucle es nombrado también como bucle `for` mejorado, o bucle `foreach`. En otros lenguajes de programación existen bucles muy parecidos a este.

#### La sintaxis `for` es la siguiente:

```
1 for (declaración: expresión) {
2 sentencia1;
3 ...
4 sentenciaN;
5 }
```

Donde....:

- `expresión` es un array o una colección de objetos.
- `declaración` es la declaración de una variable cuyo tipo sea compatible con `expresión`. Normalmente, será el tipo y el nombre de la variable a declarar.

El funcionamiento consiste en que para cada elemento de la expresión, guarda el elemento en la variable declarada y haz las instrucciones contenidas en el bucle. Después, en cada una de las iteraciones del bucle tendremos en la variable declarada el elemento actual de la expresión. Por tanto, para el caso de los arrays y de las colecciones de objetos, se recorrerá desde el primer elemento que los forma hasta el último.

Observa el contenido del código representado en la siguiente imagen, puedes apreciar cómo se construye un bucle de este tipo y su utilización sobre un array.

Los bucles `for / in` permitirán al programador despreocuparse del número de veces que se ha de iterar, pero no sabremos en qué iteración nos encontramos salvo que se añada artificialmente alguna variable contadora que nos pueda ofrecer esta información.

#### Información

Esta estructura tomará sentido cuando avancemos en el curso y veamos los Arrays y las colecciones de Objetos.

#### 9.4.3. Estructura `while`

El bucle `while` es la primera de las estructuras de repetición controladas por sucesos que vamos a estudiar. La utilización de este bucle responde al planteamiento de la siguiente pregunta: ¿Qué podemos hacer si lo único que sabemos es que se han de repetir un conjunto de instrucciones mientras se cumpla una determinada condición?.

La característica fundamental de este tipo de estructura repetitiva estriba en ser útil en aquellos casos en los que las instrucciones que forman el cuerpo del bucle podría ser necesario ejecutarlas o no. Es decir, en el bucle `while` siempre se evaluará la condición que lo controla, y si dicha condición es cierta, el cuerpo del bucle se ejecutará una vez, y se seguirá ejecutando mientras la condición sea cierta. Pero si en la evaluación inicial de la condición ésta no es verdadera, el cuerpo del bucle no se ejecutará.

#### Atención!

Es imprescindible que en el interior del bucle `while` se realice alguna acción que modifique la condición que controla la ejecución del mismo, en caso contrario estaríamos ante un bucle infinito.

#### Sintaxis estructura `while` con una única sentencia:

```
1 while (condición)
2 sentencia;
```

### Sintaxis estructura `while` con un bloque de sentencias:

```

1 while (condición) {
2 sentencia1;
3 ...
4 sentenciaN;
5 }
```

**Funcionamiento:** Mientras la condición sea cierta, el bucle se repetirá, ejecutando la/s instrucción/es de su interior.

En el momento en el que la condición no se cumpla, el control del flujo del programa pasará a la siguiente instrucción que exista justo detrás del bucle `while`.

La condición se evaluará siempre al principio, y podrá darse el caso de que las instrucciones contenidas en él no lleguen a ejecutarse nunca si no se satisface la condición de partida.

#### 9.4.4. Estructura `do while`

La segunda de las estructuras repetitivas controladas por sucesos es `do while`. En este caso, la pregunta que nos planteamos es la siguiente: ¿Qué podemos hacer si lo único que sabemos es que se han de ejecutar, al menos una vez, un conjunto de instrucciones y seguir repitiéndose hasta que se cumpla una determinada condición?.

La característica fundamental de este tipo de estructura repetitiva estriba en ser útil en aquellos casos en los que las instrucciones que forman el cuerpo del bucle necesitan ser ejecutadas, al menos, una vez y repetir su ejecución hasta que la condición sea verdadera. Por tanto, en esta estructura repetitiva siempre se ejecuta el cuerpo del bucle una primera vez.

Es imprescindible que en el interior del bucle se realice alguna acción que modifique la condición que controla la ejecución del mismo, en caso contrario estaríamos ante un bucle infinito.

### Sintaxis estructura `while` con una única sentencia:

```

1 do
2 sentencia;
3 while (condición);
```

### Sintaxis estructura `while` con un bloque de sentencias:

```

1 do {
2 sentencia1;
3 ...
4 sentenciaN;
5 } while (condición);
```

#### Funcionamiento:

El cuerpo del bucle se ejecuta la primera vez, a continuación se evaluará la condición y, si ésta es falsa, el cuerpo del bucle volverá a repetirse. El bucle finalizará cuando la evaluación de la condición sea verdadera.

En ese momento el control del flujo del programa pasará a la siguiente instrucción que exista justo detrás del bucle `do-while`. La condición se evaluará siempre después de una primera ejecución del cuerpo del bucle, por lo que no se dará el caso de que las instrucciones contenidas en él no lleguen a ejecutarse nunca.

roadrunner

#### 9.4.5. Bucle infinito

Uno de los errores más comunes al implementar cualquier tipo de bucle es que nunca pueda salir, es decir, el bucle se ejecuta durante un número infinito de veces.

Podemos provocarlo intencionadamente como en estos dos ejemplos equivalentes:

### NO RECOMENDABLE

```

1 for(;;){
2 //sentencias
3 }
```

```

1 while(true){
2 //sentencias
3 }
```

O sucede cuando la condición falla por alguna razón, como en el siguiente ejemplo:

### Bucle infinito

```

1 //Programa Java para ilustrar varias trampas de bucles.
2 public class BucleInfinito{
3
4 public static void main(String[] args)
5 {
6 // bucle infinito porque la condición no es apta
7 // la condición; debería haber sido i>0.
8 for (int i = 5; i != 0; i -= 2){
9 System.out.println(i);
10 }
11
12 int x = 5;
13 // bucle infinito porque la actualización
14 // no se proporciona
15 while (x == 5)
16 {
17 System.out.println("En el bucle");
18 }
19 }
20 }
```

Otro inconveniente es que puede estar agregando algo en su objeto de colección a través de un bucle y puede **quedarse sin memoria**. Si intenta ejecutar el siguiente programa, después de un tiempo, se producirá una excepción de falta de memoria. En este ejemplo se hace uso de la colección ArrayList, pero de momento solo necesitamos saber que se comporta como un casillero al que vamos asignando elementos (que evidentemente ocupan memoria)

### Provocar falta de memoria:

```

1 //Programa Java para la excepción de falta de memoria.
2 import java.util.ArrayList;
3 public class HeapSpace
4 {
5 public static void main(String[] args)
6 {
7 ArrayList<Integer> ar = new ArrayList<>();
8 for (int i = 0; i < Integer.MAX_VALUE; i++)
9 {
10 ar.add(i);
11 }
12 }
13 }
```

Salida:

```

1 Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
2 at java.util.Arrays.copyOf(Unknown Source)
3 at java.util.Arrays.copyOf(Unknown Source)
4 at java.util.ArrayList.grow(Unknown Source)
5 at java.util.ArrayList.ensureCapacityInternal(Unknown Source)
6 at java.util.ArrayList.add(Unknown Source)
7 at article.Integer1.main(Integer1.java:9)
```

## 9.5. Estructuras de salto

¿Saltar o no saltar? he ahí la cuestión. En la gran mayoría de libros de programación y publicaciones de Internet, siempre se nos recomienda que prescindamos de sentencias de salto incondicional, es más, se desaconseja su uso por provocar una mala

estructuración del código y un incremento en la dificultad para el mantenimiento de los mismos. Pero Java incorpora ciertas sentencias o estructuras de salto que es necesario conocer y que pueden sernos útiles en algunas partes de nuestros programas.

Estas estructuras de salto corresponden a las sentencias `break`, `continue`, las etiquetas de salto y la sentencia `return`. Pasamos ahora a analizar su sintaxis y funcionamiento.

#### 9.5.1. Sentencias `break` y `continue`

Se trata de dos instrucciones que permiten modificar el comportamiento de otras estructuras o sentencias de control, simplemente por el hecho de estar incluidas en algún punto de su secuencia de instrucciones.

La sentencia `break` incidirá sobre las estructuras de control `switch`, `while`, `for` y `do while` del siguiente modo:

- Si aparece una sentencia `break` dentro de la secuencia de instrucciones de cualquiera de las estructuras mencionadas anteriormente, dicha estructura terminará inmediatamente.
- Si aparece una sentencia `break` dentro de un bucle anidado sólo finalizará la sentencia de iteración más interna, el resto se ejecuta de forma normal.

Es decir, que `break` sirve para romper el flujo de control de un bucle, aunque no se haya cumplido la condición del bucle. Si colocamos un `break` dentro del código de un bucle, cuando se alcance el `break`, automáticamente se saldrá del bucle pasando a ejecutarse la siguiente instrucción inmediatamente después de él.

La sentencia `continue` incidirá sobre las sentencias o estructuras de control `while`, `for` y `do while` del siguiente modo:

- Si aparece una sentencia `continue` dentro de la secuencia de instrucciones de cualquiera de las sentencias anteriormente indicadas, dicha sentencia dará por terminada la iteración actual y se ejecuta una nueva iteración, evaluando de nuevo la expresión condicional del bucle.
- Si aparece en el interior de un bucle anidado solo afectará a la sentencia de iteración más interna, el resto se ejecutaría de forma normal.

Es decir, la sentencia `continue` forzará a que se ejecute la siguiente iteración del bucle, sin tener en cuenta las instrucciones que pudiera haber después del `continue`, y hasta el final del código del bucle.

#### 9.5.2. Etiquetas de salto

##### Atención!

Los saltos incondicionales y en especial, saltos a una etiqueta son totalmente **desaconsejables**.

Java permite asociar etiquetas cuando se va a realizar un salto. De este modo puede conseguirse algo más de legibilidad en el código.

Las estructuras de salto `break` y `continue`, pueden tener asociadas etiquetas. Es a lo que se llama un `break` etiquetado o un `continue` etiquetado. Pero sólo se recomienda su uso cuando se hace necesario salir de bucles anidados hacia diferentes niveles. ¿Y cómo se crea un salto a una etiqueta? En primer lugar, crearemos la etiqueta mediante un identificador seguido de dos puntos (`:`). A continuación, se escriben las sentencias Java asociadas a dicha etiqueta encerradas entre llaves. Por así decirlo, la creación de una etiqueta es como fijar un punto de salto en el programa para poder saltar a él desde otro lugar de dicho programa.

¿Cómo se lleva a cabo el salto? Es sencillo, en el lugar donde vayamos a colocar la sentencia `break` o `continue`, añadiremos detrás el identificador de la etiqueta. Con ello, conseguiremos que el salto se realice a un lugar determinado.

La sintaxis será:

```
1 break <etiqueta>;
```

### Curiosidad

Quizá a aquellos/as que han programado en HTML les suene esta herramienta, ya que tiene cierta similitud con las anclas que pueden crearse en el interior de una página web, a las que nos llevará el hiperenlace o link que hayamos asociado. También para aquellos/as que han creado alguna vez archivos por lotes o archivos batch bajo MSDOS es probable que también les resulte familiar el uso de etiquetas, pues la sentencia GOTO que se utilizaba en este tipo de archivos, hacía saltar el flujo del programa al lugar donde se ubicaba la etiqueta que se indicara en dicha sentencia.

#### 9.5.3. `return`

Ya sabemos cómo modificar la ejecución de bucles y estructuras condicionales múltiples, pero ¿Podríamos modificar la ejecución de un método? ¿Es posible hacer que éstos detengan su ejecución antes de que finalice el código asociado a ellos?. Sí es posible, a través de la sentencia `return` puede utilizarse de dos formas:

- Para terminar la ejecución del método donde esté escrita, con lo que transferirá el control al punto desde el que se hizo la llamada al método, continuando el programa por la sentencia inmediatamente posterior.
- Para devolver o retornar un valor; siempre que junto a `return` se incluya una expresión de un tipo determinado. Por tanto, en el lugar donde se invocó al método se obtendrá el valor resultante de la evaluación de la expresión que acompaña al método.

### Importante

En general, una sentencia `return` suele aparecer al final de un método, de este modo el método tendrá una entrada y una salida. También es posible utilizar una sentencia `return` en cualquier punto de un método, con lo que éste finalizará en el lugar donde se encuentre dicho `return`. No será recomendable incluir más de un `return` en un método y por regla general, deberá ir al final del método como hemos comentado.

El valor de retorno es opcional, si lo hubiera debería de ser del mismo tipo o de un tipo compatible al tipo del valor de retorno definido en la cabecera del método, pudiendo ser desde un entero a un objeto creado por nosotros. Si no lo tuviera, el tipo de retorno sería `void`, y `return` serviría para salir del método sin necesidad de llegar a ejecutar todas las instrucciones que se encuentran después del `return`.

## 9.6. Excepciones

A lo largo de nuestro aprendizaje de Java nos hemos topado en alguna ocasión con errores, pero éstos suelen ser los que nos ha indicado el compilador. Un punto y coma por aquí, un nombre de variable incorrecto por allá, pueden hacer que nuestro compilador nos avise de estos descuidos.

Cuando los vemos, se corrigen y obtenemos nuestra clase compilada correctamente.

Pero, ¿Sólo existen este tipo de errores? ¿Podrían existir errores no sintácticos en nuestros programas?. Está claro que sí, un programa perfectamente compilado en el que no existen errores de sintaxis, puede generar otros tipos de errores que quizás aparezcan en tiempo de ejecución. A estos errores se les conoce como **excepciones**.

Aprenderemos a gestionar de manera adecuada estas excepciones y tendremos la oportunidad de utilizar el potente sistema de manejo de errores que Java incorpora. La potencia de este sistema de manejo de errores radica en:

1. Que el código que se encarga de manejar los errores, es perfectamente identificable en los programas. Este código puede estar separado del código que maneja la aplicación.
2. Que Java tiene una gran cantidad de errores estándar asociados a multitud de fallos comunes, como por ejemplo divisiones por cero, fallos de entrada de datos, etc. Al tener tantas excepciones localizadas, podemos gestionar de manera específica cada uno de los errores que se produzcan.

En Java se pueden preparar los fragmentos de código que pueden provocar errores de ejecución para que si se produce una excepción, el flujo del programa es lanzado (`throw`) hacia ciertas zonas o rutinas que han sido creadas previamente por el programador y cuya finalidad será el tratamiento efectivo de dichas excepciones. Si no se captura la excepción, el programa se detendrá con toda probabilidad.

En Java, las excepciones están representadas por clases. El paquete `java.lang.Exception` y sus subpaquetes contienen todos los tipos de excepciones. Todas las excepciones derivarán de la clase `Throwable`, existiendo clases más específicas. Por debajo de la clase `Throwable` existen las clases `Error` y `Exception`. `Error` es una clase que se encargará de los errores que se produzcan en la máquina virtual, no en nuestros programas. Y la clase `Exception` será la que a nosotros nos interese conocer, pues gestiona los errores provocados en los programas.

Java lanzará una excepción en respuesta a una situación poco usual. Cuando se produce un error se genera un objeto asociado a esa excepción. Este objeto es de la clase `Exception` o de alguna de sus herederas. Este objeto se pasa al código que se ha definido para manejar la excepción. Dicho código puede manipular las propiedades del objeto `Exception`.

El programador también puede lanzar sus propias excepciones. Las excepciones en Java serán objetos de clases derivadas de la clase base `Exception`. Existe toda una jerarquía de clases derivada de la clase base `Exception`. Estas clases derivadas se ubican en dos grupos principales:

- Las excepciones en tiempo de ejecución, que ocurren cuando el programador no ha tenido cuidado al escribir su código.
- Las excepciones que indican que ha sucedido algo inesperado o fuera de control.

En la siguiente imagen te ofrecemos una aproximación a la jerarquía de las excepciones en Java.

```
classDiagram
 class Object
 class Throwable
 Object <|-- Throwable
 namespace Comprobadas_Checked {
 class Exception
 class IOException
 class UserExceptions
 class Other1["..."]
 class Other2["..."]
 class Other3["..."]
 }
 }
 namespace No_Comprobadas_Unchecked {
 class RuntimeException
 }
 class Error
 class ArithmeticException
 class ArrayIndexOutOfBoundsException
 class Other4["..."]
 }
 Exception <|-- Other1
 Exception <|-- IOException
 IOException <|-- Other2
 IOException <|-- Other3
 Exception <|-- UserExceptions
 Exception <|-- RuntimeException
 RuntimeException <|-- ArithmeticException
 RuntimeException <|-- ArrayIndexOutOfBoundsException
 RuntimeException <|-- Other4
 Throwable <|-- Exception
 Throwable <|-- Error
```

Y aquí tenemos una lista de las más habituales con su explicación:

| NOMBRE                                | DESCRIPCIÓN                                                                                                             |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>FileNotFoundException</b>          | Lanza una excepción cuando el fichero no se encuentra.                                                                  |
| <b>ClassNotFoundException</b>         | Lanza una excepción cuando no existe la clase.                                                                          |
| <b>EOFException</b>                   | Lanza una excepción cuando llega al final del fichero.                                                                  |
| <b>ArrayIndexOutOfBoundsException</b> | Lanza una excepción cuando se accede a una posición de un array que no existe.                                          |
| <b>NumberFormatException</b>          | Lanza una excepción cuando se procesa un numero pero este es un dato alfanumérico.                                      |
| <b>NullPointerException</b>           | Lanza una excepción cuando intentando acceder a un miembro de un objeto para el que todavía no hemos reservado memoria. |
| <b>IOException</b>                    | Generaliza muchas excepciones anteriores. La ventaja es que no necesitamos controlar cada una de las excepciones.       |
| <b>Exception</b>                      | Es la clase padre de IOException y de otras clases. Tiene la misma ventaja que IOException.                             |
| <b>ArithmaticException</b>            | Se lanza por ejemplo, cuando intentamos dividir un número entre cero.                                                   |

#### 9.6.1. El manejo de excepciones

Como hemos comentado, siempre debemos controlar las excepciones que se puedan producir o de lo contrario nuestro software quedará expuesto a fallos. Las excepciones pueden tratarse de dos formas:

- **Interrupción.** En este caso se asume que el programa ha encontrado un error irrecuperable. La operación que dio lugar a la excepción se anula y se entiende que no hay manera de regresar al código que provocó la excepción. Es decir, la operación que dio originó el error, se anula.

- **Reanudación.** Se puede manejar el error y regresar de nuevo al código que provocó el error.

Java emplea la primera forma, pero puede simularse la segunda mediante la utilización de un bloque `try` en el interior de un `while`, que se repetirá hasta que el error deje de existir. En la sección de ejemplos de puedes ver como poner el `try-catch` dentro de un `do while`.

### 9.6.2. Capturar una excepción

Para poder capturar excepciones, emplearemos la estructura de captura de excepciones `try-catch-finally`.

Básicamente, para capturar una excepción lo que haremos será declarar bloques de código donde es posible que ocurra una excepción. Esto lo haremos mediante un bloque `try` (intentar). Si ocurre una excepción dentro de estos bloques, se lanza una excepción. Estas excepciones lanzadas se pueden capturar por medio de bloques `catch`. Será dentro de este tipo de bloques donde se hará el manejo de las excepciones.

#### Sintaxis try-catch :

```

1 try {
2 //código que puede generar excepciones;
3 } catch (Tipo_excepcion_1 objeto_excepcion) {
4 //Manejo de excepción de Tipo_excepcion_1;
5 } catch (Tipo_excepcion_2 objeto_excepcion) {
6 //Manejo de excepción de Tipo_excepcion_2;
7 }
8 ...
9 finally {
10 //instrucciones que se ejecutan siempre
11 }
```

En esta estructura, la parte `catch` puede repetirse tantas veces como excepciones diferentes se deseen capturar. La parte `finally` es opcional y, si aparece, solo podrá hacerlo una vez.

Cada `catch` maneja un tipo de excepción. Cuando se produce una excepción, se busca el `catch` que posea el manejador de excepción adecuado, será el que utilice el mismo tipo de excepción que se ha producido. Esto puede causar problemas si no se tiene cuidado, ya que la clase `Exception` es la superclase de todas las demás. Por lo que si se produjo, por ejemplo, una excepción de tipo `Aritmetic Exception` y el primer `catch` capture el tipo genérico `Exception`, será ese `catch` el que se ejecute y no los demás.

Por eso el último `catch` debe ser el que capture excepciones genéricas y los primeros deben ser los más específicos. Lógicamente si vamos a tratar a todas las excepciones (sean del tipo que sean) igual, entonces basta con un solo `catch` que capture objetos `Exception`.

#### Recuerda

En Java, cuando un bloque de código puede provocar una excepción pero no se maneja adecuadamente, se produce lo que se conoce como una "excepción no controlada" o "excepción no capturada". Cuando ocurre una excepción no controlada, Java sigue un conjunto de reglas específicas para manejarla:

1. **Propagación de excepciones:** Java busca en la pila de llamadas (el seguimiento de la ejecución del programa) para ver si el método actual maneja la excepción. Si el método actual no maneja la excepción, la excepción se "propaga" hacia arriba en la pila de llamadas. (Piensa en una burbuja de aire en el fondo del mar intentando buscar una salida)
2. **Búsqueda de un manejador de excepciones:** La excepción propagada continúa buscando un manejador de excepciones adecuado a medida que se retrocede a través de los métodos que llamaron al método actual. Si se encuentra un bloque `try-catch` que puede manejar la excepción, se ejecutará el código del bloque `catch` correspondiente.
3. **Si no se encuentra un manejador adecuado:** Si la excepción llega a la parte superior de la pila de llamadas y no se encuentra un manejador de excepciones adecuado, el programa se detendrá y se imprimirá un mensaje de error en la consola, que contiene información sobre la excepción, como su tipo, mensaje y seguimiento de pila (`stack trace`).

### 9.6.3. Delegación de excepciones con throws

¿Puede haber problemas con las excepciones al usar llamadas a métodos en nuestros programas? Efectivamente, si se produjese una excepción es necesario saber quién será el encargado de solucionarla. Puede ser que sea el propio método llamado o el código que hizo la llamada a dicho método.

Quizá pudiéramos pensar que debería ser el propio método el que se encargue de sus excepciones, aunque es posible hacer que la excepción sea resuelta por el código que hizo la llamada. Cuando un método utiliza una sentencia que puede generar una excepción, pero dicha excepción no es capturada y tratada por él, sino que se encarga su gestión a quién llamó al método, decimos que se ha producido delegación de excepciones.

Para establecer esta delegación, en la cabecera del método se declara el tipo de excepciones que puede generar y que deberán ser gestionadas por quien invoque a dicho método. Utilizaremos para ello la sentencia `throws` y tras esa palabra se indica qué excepciones puede provocar el código del método. Si ocurre una excepción en el método, el código abandona ese método y regresa al código desde el que se llamó al método. Allí se buscará el `catch` apropiado para esa excepción. Su sintaxis es la siguiente:

#### Sintaxis throws :

```
1 public class Delegacion_Excepciones {
2 ...
3 public int leeAnio(BufferedReader lector) throws IOException, NumberFormatException{
4 String linea = teclado.readLine();
5 return Integer.parseInt(linea);
6 }
7 ...
8 }
```

Donde `IOException` y `NumberFormatException`, serían dos posibles excepciones que el método `leeAnio` podría generar, pero que no gestiona. Por tanto, un método puede incluir en su cabecera un listado de excepciones que puede lanzar, separadas por comas.

### 9.6.4. Crear y lanzar excepciones de usuario

Las excepciones de usuario son subclases de la clase `Exception` que podemos crear y lanzar en nuestros programas para avisar sobre determinadas situaciones.

#### 9.6.4.1. CREAR UNA NUEVA EXCEPCIÓN

Para crear una nueva excepción tenemos que crear una clase derivada (subclase) de la clase `Exception`.

La clase `Exception` tiene dos constructores, uno sin parámetros y otro que acepta un `String` con un texto descriptivo de la excepción. Todas las excepciones de usuario las crearemos de la siguiente forma:

#### Constructores de Exception:

```
1 class NombreExcepcion extends Exception {
2 public NombreExcepcion(){
3 super();
4 }
5 public NombreExcepcion(String msg){
6 super(msg);
7 }
8 }
```

#### 9.6.4.2. LANZAR UNA EXCEPCIÓN

Las excepciones se lanzan mediante la instrucción `throw`. La sintaxis es:

#### Instrucción throw :

```
1 throw new NombreExcepcion("Mensaje descriptivo de la situación inesperada");
```

Ya que se tratará de una excepción comprobada, en la cabecera del método que lanza la excepción habrá que propagarla.

### 9.6.5. Excepciones checked y unChecked

En Java, las excepciones se dividen en dos categorías principales: excepciones "checked" (comprobadas) y excepciones "unchecked" (no comprobadas).

- 1. Excepciones Comprobadas (Checked Exceptions):** - Las excepciones comprobadas son aquellas que el compilador obliga a manejar. Esto significa que, si un método puede lanzar una excepción comprobada, el programador está obligado a manejarla de alguna manera, ya sea mediante la declaración del método con `throws` o mediante el manejo directo con un bloque `try-catch`. - Ejemplos de excepciones comprobadas incluyen `IOException` y `SQLException`. - Estas excepciones suelen representar situaciones en las que un programa no puede continuar normalmente y se espera que el código las maneje de manera adecuada.

#### Excepción comprobada:

```

1 import java.io.FileReader;
2 import java.io.FileNotFoundException;
3
4 public class EjemploCheckedException {
5 public static void main(String[] args) {
6 try {
7 FileReader file = new FileReader("archivo.txt");
8 } catch (FileNotFoundException e) {
9 System.out.println("Archivo no encontrado: " + e.getMessage());
10 }
11 }
12 }
```

- 1. Excepciones No Comprobadas (Unchecked Exceptions):** - Las excepciones no comprobadas son aquellas que el compilador no requiere que se manejen explícitamente. Normalmente, son subclases de `RuntimeException`. - Estas excepciones suelen deberse a errores de programación, como acceder a un índice fuera de los límites de un array (`ArrayIndexOutOfBoundsException`) o intentar convertir un objeto a un tipo incompatible (`ClassCastException`). - Aunque no se requiere que se manejen explícitamente, es buena práctica manejarlas para evitar que el programa termine abruptamente.

#### Excepción no comprobada:

```

1 public class EjemploUncheckedException {
2 public static void main(String[] args) {
3 int[] numeros = {1, 2, 3};
4 System.out.println(numeros[4]); // Esto lanzará ArrayIndexOutOfBoundsException
5 }
6 }
```

#### 9.6.5.1. ¿COMO SÉ SI UNA EXCEPCIÓN ES DE UN TIPO O DE OTRO?

La principal diferencia radica en la obligación del compilador de manejar o declarar excepciones. Las excepciones comprobadas deben ser manejadas o declaradas en el código, mientras que las excepciones no comprobadas no tienen esta obligación y generalmente se deben a errores de programación.

En Java, puedes distinguir entre excepciones comprobadas y no comprobadas principalmente por el tipo de clase que heredan. Aquí hay algunas pautas generales:

- 1. Excepciones Comprobadas (Checked Exceptions):** - Las excepciones comprobadas suelen ser subclases directas de la clase `Exception` (o alguna de sus subclases), pero no heredan de `RuntimeException` ni de sus subclases. - Ejemplos comunes incluyen `IOException`, `SQLException`, y cualquier excepción que herede directamente de `Exception` (pero no de `RuntimeException`).
- 2. Excepciones No Comprobadas (Unchecked Exceptions):** - Las excepciones no comprobadas suelen ser subclases directas de la clase `RuntimeException`. - Ejemplos comunes incluyen `NullPointerException`, `ArrayIndexOutOfBoundsException`, y cualquier excepción que herede directamente de `RuntimeException`.

#### Importante

Ten en cuenta que estas son pautas generales y puede haber excepciones personalizadas o situaciones específicas en las que estas reglas no se apliquen estrictamente. Para obtener información precisa sobre un tipo de excepción específico, puedes consultar la documentación de Java o examinar la jerarquía de clases y herencia de la excepción en cuestión.

## 9.7. Aserciones ( Assertions )

Una aserción (afirmación) permite probar la exactitud de cualquier suposición que se haya hecho en el programa. Una afirmación se logra utilizando la declaración de `assert` en Java. Al ejecutar una aserción, se cree que es cierta. Si falla, JVM genera un error denominado `AssertionError`. Se utiliza principalmente con fines de prueba durante el desarrollo.

La declaración de afirmación se usa con una expresión booleana y se puede escribir de dos maneras diferentes.

### Primera forma:

```
1 assert expression;
```

### Segunda forma:

```
1 assert expression1 : expression2;
```

### Ejemplo:

```
1 import java.util.Scanner;
2
3 public class P7_Assertions {
4 // Programa Java para demostrar el uso de las assertions
5 public static void main(String[] args) {
6 Scanner entrada = new Scanner(System.in);
7 System.out.print("Introduce tu edad: ");
8 int age = entrada.nextInt();
9 assert (age >= 18) : "No puedes votar";
10 System.out.println("La edad del votante es de " + age);
11 }
12 }
```

Salida sin assertions:

```
1 Introduce tu edad: 14
2 La edad del votante es de 14
```

Después de habilitar las assertions:

### Saber más...

Puedes habilitar las assertions añadiendo los parámetros de la JVM en IntelliJ:

- `-ea` : Enable Assertions (habilitar aserciones)
- `-da` : Disable Assertions (deshabilitar aserciones, que es la opción por defecto)

Puedes consultar este enlace para saber donde agregar estas opciones: <https://stackoverflow.com/questions/68848158/java-assertions-in-intellij-idea-community>

Salida:

```
1 Introduce tu edad: 14
2 Exception in thread "main" java.lang.AssertionError: No puedes votar
3 at UD03.P7_Assertions.main(P7_Assertions.java:11)
```

### Otro ejemplo:

```

1 package UD03;
2
3 public class P7_Assertions2 {
4 public static void main(String[] args) {
5 System.out.println("Probando Aserciones...");
6 assert true : "Nunca veremos esto.";
7 assert false : "Esto solo lo veremos si activamos las aserciones.";
8 }
9 }
```

Ejecución sin aserciones:

```
1 Probando Aserciones...
```

Y con aserciones:

```

1 Probando Aserciones...
2 Exception in thread "main" java.lang.AssertionError: Esto solo lo veremos si activamos las aserciones.
3 at UD03.P7_Assertions2.main(P7_Assertions2.java:7)
```

#### 9.7.1. ¿Por qué utilizar aserciones?

Dondequiero que un programador quiera ver si sus suposiciones son erróneas o no.

- Para asegurarse de que un código que parece inalcanzable sea realmente inalcanzable.
- Para asegurarse de que las suposiciones escritas en los comentarios sean correctas.
- Para asegurarse de que no se alcance el caso default del switch.
- Para comprobar el estado del objeto.
- Al comienzo del método.
- Después de la invocación del método.

#### 9.7.2. Aserción o Excepciones

Las aserciones se utilizan principalmente para comprobar situaciones lógicamente imposibles. Por ejemplo, se pueden utilizar para comprobar el estado que espera un código antes de empezar a ejecutarse o el estado después de que termine de ejecutarse. A diferencia del manejo normal de excepciones/errores, las aserciones generalmente están deshabilitadas en tiempo de ejecución.

¿Dónde utilizarlas?:

- Argumentos para los métodos privados. Los argumentos para los métodos privados los proporciona únicamente el código del desarrollador y es posible que este deseé comprobar sus suposiciones sobre los argumentos.
- Casos condicionales.
- Condiciones al inicio de cualquier método.

¿Dónde **NO** utilizar aserciones?:

- Las aserciones no deben usarse para reemplazar mensajes de error
- Las aserciones no deben usarse para verificar argumentos en los métodos públicos, ya que pueden ser proporcionados por el usuario.
- Para manejar los errores proporcionados por los usuarios usaremos las excepciones.
- Las aserciones no deben usarse en argumentos de línea de comando.

## 9.8. Ejemplos UD03

### 9.8.1. if e if-else

Para completar la información que debes saber sobre las estructuras `if` e `if-else`, observa el siguiente código. En él podrás analizar el programa que realiza el cálculo de la nota de un examen de tipo test. Además de calcular el valor de la nota, se ofrece como salida la calificación no numérica de dicho examen. Para obtenerla, se combinarán las diferentes estructuras condicionales aprendidas hasta ahora.

Presta especial atención a los comentarios incorporados en el código fuente, así como a la forma de combinar las estructuras condicionales y a las expresiones lógicas utilizadas en ellas.

```

1 package UD03;
2 public class Sentencias_Condicionales {
3 /*Vamos a realizar el cálculo de la nota de un examen
4 * de tipo test. Para ello, tendremos en cuenta el número
5 * total de pregunta, los aciertos y los errores. Dos errores
6 * anulan una respuesta correcta.
7 *
8 * Finalmente, se muestra por pantalla la nota obtenida, así
9 * como su calificación no numérica.
10 *
11 * La obtención de la calificación no numérica se ha realizado
12 * combinando varias estructuras condicionales, mostrando expresiones
13 * lógicas compuestas, así como anidamiento.
14 */
15 public static void main(String[] args) {
16 // Declaración e inicialización de variables
17 int num_aciertos = 12;
18 int num_errores = 3;
19 int num_preguntas = 20;
20 float nota = 0;
21 String calificacion = "";
22 //Procesamiento de datos
23 nota = ((num_aciertos - (num_errores / 2)) * 10) / num_preguntas;
24
25 if (nota < 5) {
26 calificacion = "INSUFICIENTE";
27 } else {
28 /* Cada expresión lógica de estos if está compuesta por dos
29 * expresiones lógicas combinadas a través del operador Y o AND
30 * que se representa con el símbolo &&. De tal manera, que para
31 * que la expresión lógica se cumpla (sea verdadera) la variable
32 * nota debe satisfacer ambas condiciones simultáneamente
33 */
34 if (nota >= 5 && nota < 6) {
35 calificacion = "SUFICIENTE";
36 } else if (nota >= 6 && nota < 7) {
37 calificacion = "BIEN";
38 } else if (nota >= 7 && nota < 9) {
39 calificacion = "NOTABLE";
40 } else if (nota >= 9 && nota <= 10) {
41 calificacion = "SOBRESALIENTE";
42 }
43 }
44 //Salida de información
45 System.out.println("La nota obtenida es: " + nota);
46 System.out.println("y la calificación obtenida es: " + calificacion);
47 }
48 }
```

### 9.8.2. switch

Comprueba el siguiente fragmento de código en el que se resuelve el cálculo de un examen de tipo test, utilizando la estructura `switch`.

```
1 package UD03;
2
3 public class P3_2_condicional_switch {
4
5 /*
6 * Vamos a realizar el cálculo de la nota de un examen de tipo test. Para
7 * ello, tendremos en cuenta el número total de preguntas, los aciertos y
8 * los errores. Dos errores anulan una respuesta correcta.
9 *
10 * La nota que vamos a obtener será un número entero.
11 *
12 * Finalmente, se muestra por pantalla la nota obtenida, así como su
13 * calificación no numérica.
14 *
15 * La obtención de la calificación no numérica se ha realizado utilizando la
16 * estructura condicional múltiple o switch.
17 *
18 */
19 public static void main(String[] args) {
20 // Declaración e inicialización de variables
21 int num_aciertos = 17;
22 int num_errores = 3;
23 int num_preguntas = 20;
24 int nota = 0;
25 String calificacion = "";
26 //Procesamiento de datos
27 nota = ((num_aciertos - (num_errores / 2)) * 10) / num_preguntas;
28 switch (nota) {
29 case 5:
30 calificacion = "SUFICIENTE";
31 break;
32 case 6:
33 calificacion = "BIEN";
34 break;
35 case 7:
36 calificacion = "NOTABLE";
37 break;
38 case 8:
39 calificacion = "NOTABLE";
39 break;
40 case 9:
41 calificacion = "SOBRESALIENTE";
42 break;
43 case 10:
44 calificacion = "SOBRESALIENTE";
45 break;
46 default:
47 calificacion = "INSUFICIENTE";
48 }
49 //Salida de información
50 System.out.println("La nota obtenida es: " + nota);
51 System.out.println("y la calificación obtenida es: " + calificacion);
52 }
```

```

1 //Expresiones switch mejoradas JAVA 12
2 int entero = 5;
3
4 String numericString = switch (entero) {
5 case 0 -> "cero";
6 case 1, 3, 5, 7, 9 -> "impar";
7 case 2, 4, 6, 8, 10 -> "par";
8 default -> "error";
9 };
10 System.out.println(numericString); //impar
11
12 //Expresiones switch mejoradas JAVA 13
13
14 int entero2 = 4;
15
16 String numericString2 = switch (entero2) {
17 case 0 -> {
18 String value = calculaCero();
19 yield value;
20 }
21 case 1, 3, 5, 7, 9 -> {
22 String value = calculaImpar();
23 yield value;
24 }
25
26 case 2, 4, 6, 8, 10 -> {
27 String value = calculaPar();
28 yield value;
29 }
30
31 default -> {
32 String value = calculaDefecto();
33 yield value;
34 }
35 };
36 System.out.println(numericString); //calculaPar()
37 }
38 static String calculaCero() {return "";}
39 static String calculaImpar() {return "";}
40 static String calculaPar() {return "";}
41 static String calculaDefecto() {return "";}
42 }
```

### 9.8.3. for

Observa el siguiente archivo Java y podrás analizar un ejemplo de utilización del bucle for para la impresión por pantalla de la tabla de multiplicar del siete. Lee atentamente los comentarios incluidos en el código, pues aclaran algunas cuestiones interesantes sobre este bucle.

```

1 package UD03;
2
3 public class Repetitiva_For {
4 /* En este ejemplo se utiliza la estructura repetitiva for
5 * para representar en pantalla la tabla de multiplicar del siete
6 */
7 public static void main(String[] args) {
8 // Declaración e inicialización de variables
9 int numero = 7;
10 int contador;
11 int resultado = 0;
12 //Salida de información
13 System.out.println("Tabla de multiplicar del " + numero);
14 System.out.println(".....");
15 //Utilizamos ahora el bucle for
16 for (contador = 1; contador <= 10; contador++) {
17 /* La cabecera del bucle incorpora la inicialización de la variable
18 * de control, la condición de multiplicación hasta el 10 y el
19 * incremento de dicha variable de uno en uno en cada iteración del
20 * bucle.
21 * En este caso contador++ incrementará en una unidad el valor de
22 * dicha variable.
23 */
24 resultado = contador * numero;
25 System.out.println(numero + " x " + contador + " = " + resultado);
26 /* A través del operador + aplicado a cadenas de caracteres,
27 * concatenamos los valores de las variables con las cadenas de
28 * caracteres que necesitamos para representar correctamente la
29 * salida de cada multiplicación.
30 */
31 }
32 }
33 }
```

#### 9.8.4. while

Observa el siguiente código java y podrás analizar un ejemplo de utilización del bucle `while` para la impresión por pantalla de la tabla de multiplicar del siete. Lee atentamente los comentarios incluidos en el código, pues aclaran algunas cuestiones interesantes sobre este bucle. Como podrás comprobar, el resultado de este bucle es totalmente equivalente al obtenido utilizando el bucle `for`.

```

1 package UD03;
2
3 public class Repetitiva_While {
4
5 public static void main(String[] args) {
6 // Declaración e inicialización de variables
7 int numero = 7;
8 int contador;
9 int resultado = 0;
10 //Salida de información
11 System.out.println("Tabla de multiplicar del " + numero);
12 System.out.println(".....");
13 //Utilizamos ahora el bucle while
14 contador = 1; //inicializamos la variable contadora
15 while (contador <= 10){ //Establecemos la condición del bucle
16 resultado = contador * numero;
17 System.out.println(numero + " x " + contador + " = " + resultado);
18 //Modificamos el valor de la variable contadora, para hacer que el
19 //bucle pueda seguir iterando hasta llegar a finalizar
20 contador++;
21 }
22 }
23 }
```

#### 9.8.5. do while

Ahora podrás analizar un ejemplo de utilización del bucle `do while` para la impresión por pantalla de la tabla de multiplicar del siete. Lee atentamente los comentarios incluidos en el código, pues aclaran algunas cuestiones interesantes sobre este bucle. Como podrás comprobar, el resultado de este bucle es totalmente equivalente al obtenido utilizando el bucle `for` y el bucle `while`.

```

1 package UD03;
2
3 public class Repetitiva_DoWhile {
4
5 public static void main(String[] args) {
6 // Declaración e inicialización de variables
7 int numero = 7;
8 int contador;
9 int resultado = 0;
10 //Salida de información
11 System.out.println("Tabla de multiplicar del " + numero);
12 System.out.println(".....");
13 //Utilizamos ahora el bucle do-while
14 contador = 1; //inicializamos la variable contadora
15 do {
16 resultado = contador * numero;
17 System.out.println(numero + " x " + contador + " = " + resultado);
18 //Modificamos el valor de la variable contadora, para hacer que el
19 //bucle pueda seguir iterando hasta llegar a finalizar
20 contador++;
21 } while (contador <= 10); //Establecemos la condición del bucle
22 }
23 }
```

#### 9.8.6. break

Aunque no es recomendable su uso aquí tienes un ejemplo de la estructura `break`

```

1 package UD03;
2
3 public class Sentencia_Break {
4
5 public static void main(String[] args) {
6 int contador;
7 for (contador=1;contador<=10;contador++){
8 if (contador==7){
9 break;
10 System.out.println("Valor: " + contador);
11 }
12 System.out.println("Fin del programa");
13 /*
14 * El bucle solo se ejecutará en 6 ocasiones, ya que cuando
15 * la variable contador sea igual a 7 encontraremos un break que
16 * romperá el flujo del bucle, transfiriéndonos a la sentencia que
17 * imprime el mensaje de Fin del programa.
18 */
19 }
20 }

```

### 9.8.7. continue

Aunque no es recomendable su uso aquí tienes un ejemplo de la estructura `continue`

```

1 package UD03;
2
3 public class Sentencia_Continue {
4
5 public static void main(String[] args) {
6 int contador;
7 System.out.println("Imprimiendo los números pares que hay del 1 al 10...");
8 for (contador = 1; contador <= 10; contador++) {
9 if (contador % 2 != 0) {
10 continue;
11 }
12 System.out.println(contador + " ");
13 }
14 System.out.println("\nFin del programa");
15 /*
16 * Las iteraciones del bucle que generarán la impresión de cada uno de
17 * los números pares, serán aquellas en las que el resultado de calcular
18 * el resto de la división entre 2 de cada valor de la variable
19 * contador, sea igual a 0.
20 */
21 }
22 }

```

### 9.8.8. Etiquetas de salto

A continuación, te ofrecemos un ejemplo de declaración y uso de etiquetas en un bucle. Como podrás apreciar, las sentencias asociadas a cada etiqueta están encerradas entre llaves para delimitar así su ámbito de acción.

```

1 package UD03;
2
3 public class EtiquetasSalto {
4
5 public static void main(String[] args) {
6 for (int i = 1; i < 3; i++) {
7 bloque_uno:
8 {
9 bloque_dos:
10 {
11 System.out.println("Iteración: " + i);
12 if (i == 1) {
13 break bloque_uno;
14 }
15 if (i == 2) {
16 break bloque_dos;
17 }
18 }
19 System.out.println("después del bloque dos");
20 }
21 System.out.println("después del bloque uno");
22 }
23 System.out.println("Fin del bucle");
24 }
25 }

```

### 9.8.9. Sentencia return

En el siguiente archivo java encontrarás el código de un programa que obtiene la suma de dos números, empleando para ello un método sencillo que retorna el valor de la suma de los números que se le han pasado como parámetros. Presta atención a los comentarios y fíjate en las conversiones a entero de la entrada de los operandos por consola.

```

1 package UD03;
2
3 import java.io.*;
4
5 public class Sentencia_Return {
6
7 private static BufferedReader stdin = new BufferedReader(
8 new InputStreamReader(System.in));
9
10 public static int suma(int numero1, int numero2) {
11 int resultado;
12 resultado = numero1 + numero2;
13 return resultado; //Mediante return devolvemos el resultado de la suma
14 }
15
16 public static void main(String[] args) throws IOException {
17 //Declaración de variables
18 String input; //Esta variable recibirá la entrada de teclado
19 int primer_numero, segundo_numero; //Estas variables almacenarán los operandos
20 // Solicitamos que el usuario introduzca dos números por consola
21 System.out.print("Introduce el primer operando:");
22 input = stdin.readLine(); //Leemos la entrada como cadena de caracteres
23 primer_numero = Integer.parseInt(input); //Transformamos a entero lo introducido
24 System.out.print("Introduce el segundo operando: ");
25 input = stdin.readLine(); //Leemos la entrada como cadena de caracteres
26 segundo_numero = Integer.parseInt(input); //Transformamos a entero lo introducido
27 //Imprimimos los números introducidos
28 System.out.println("Los operandos son: " + primer_numero + " y " + segundo_numero);
29 System.out.println("obteniendo su suma... ");
30 //Invocamos al método que realiza la suma, pasándole los parámetros adecuados
31 System.out.println("La suma de ambos operandos es: " +
32 suma(primer_numero, segundo_numero));
33 }
34 }
```

### 9.8.10. Excepciones

Vamos a realizar un programa en Java en el que se solicite al usuario la introducción de un número por teclado comprendido entre el 0 y el 100. Utilizando manejo de excepciones, controlaremos la entrada de dicho número y volver a solicitarlo en caso de que ésta sea incorrecta.

```

1 package UD03;
2
3 import java.io.*;
4 import java.util.Scanner;
5
6 public class P6_1_Excepciones {
7
8 public static void main(String[] args) {
9 int numero = -1;
10 int intentos = 0;
11 String linea;
12 Scanner teclado = new Scanner(System.in);
13 do {
14 try {
15 System.out.print("Introduzca un número entre 0 y 100: ");
16 linea = teclado.nextLine();
17 numero = Integer.parseInt(linea);
18 } catch (NumberFormatException e) {
19 System.out.println("Debe introducir un número entre 0 y 100.");
20 } catch (Exception e) {
21 System.out.println("Error al leer del teclado.");
22 } finally {
23 intentos++;
24 }
25 } while (numero < 0 || numero > 100);
26 System.out.println("El número introducido es: " + numero);
27 System.out.println("Número de intentos: " + intentos);
28 }
29 }
```

En este programa se solicita repetidamente un número utilizando una estructura `do while`, mientras el número introducido sea menor que 0 y mayor que 100. Como al solicitar el número pueden producirse los errores siguientes:

- De entrada de información a través de la excepción `Exception` generada por el método `nextLine()` de la clase `Scanner`.
- De conversión de tipos a través de la excepción `NumberFormatException` generada por el método `parseInt()`.

Entonces se hace necesaria la utilización de bloques `catch` que gestionen cada una de las excepciones que puedan producirse. Cuando se produce una excepción, se compara si coincide con la excepción del primer `catch`. Si no coincide, se compara con la del segundo `catch` y así sucesivamente. Si se encuentra un `catch` que coincide con la excepción a gestionar, se ejecutará el bloque de sentencias asociado a éste.

Si ningún bloque `catch` coincide con la excepción lanzada, dicha excepción se lanzará fuera de la estructura `try-catch-finally`.

El bloque `finally`, se ejecutará tanto si `try` terminó correctamente, como si se capturó una excepción en algún bloque `catch`. Por tanto, si existe bloque `finally` éste se ejecutará siempre.

#### 9.8.10.1. EJEMPLO DE LA PROPAGACIÓN DE EXCEPCIONES

Aquí tienes este otro ejemplo para comprender cómo se propaga una excepción hacia arriba en la pila de ejecución en Java

```

1 package UD03;
2
3 import java.util.Scanner;
4
5 public class P6_2_PropagacionExcepciones {
6 public static void main(String[] args) {
7 Scanner teclado = new Scanner(System.in);
8 try {
9 System.out.print("Introduzca un número entre 0 y 100: ");
10 String linea = teclado.nextLine();
11 int numero = Integer.parseInt(linea);
12 metodoA(numero);
13 } catch (Exception e) {
14 System.out.println("Excepción atrapada en el método main: " + e.getMessage());
15 }
16 }
17
18 public static void metodoA(int numero) {
19 try {
20 metodoB(numero);
21 } catch (ArithmeticException e) {
22 System.out.println("Excepción atrapada en el método A: " + e.getMessage());
23 }
24 }
25
26 public static void metodoB(int divisor) {
27 int resultado = 10 / divisor;
28 }
29}
30

```

#### 9.8.10.2. EJEMPLO DE EXCEPCIÓN PERSONALIZADA

En este ejemplo definimos nuestra propia clase de Excepciones (en una clase interna), la lanzamos en un método de manera personalizada, y posteriormente capturamos la excepción en el método `main`.

```

1 import java.util.Scanner;
2
3 public class ValidadorEdad {
4
5 public static void validarEdad(int edad) throws EdadInvalidaException {
6 // Validaciones específicas con mensajes personalizados
7 if (edad < 0) {
8 throw new EdadInvalidaException("La edad no puede ser negativa");
9 } else if (edad > 150) {
10 throw new EdadInvalidaException("La edad parece incorrecta (muy alta)");
11 } else if (edad < 18) {
12 throw new EdadInvalidaException("Acceso denegado: Menor de edad");
13 }
14 }
15
16 public static void main(String[] args) {
17 Scanner sc = new Scanner(System.in);
18 System.out.print("Dime una edad para validar: ");
19 int edadInput = sc.nextInt();
20
21 try {
22 System.out.println("Validando edad: " + edadInput);
23 validarEdad(edadInput);
24 // Si llegamos hasta aquí, la edad es válida
25 System.out.println("Edad válida. Acceso permitido.");
26 } catch (EdadInvalidaException e) {
27 // Capturamos y manejamos nuestra excepción personalizada
28 System.out.println("Excepción capturada: " + e.getMessage());
29 } catch (Exception e) {
30 // Capturamos cualquier otra excepción genérica
31 System.out.println("Error inesperado: " + e.getMessage());
32 }
33 }
34 //Clase interna, ni public, ni static, ni private
35 class EdadInvalidaException extends Exception {
36 // Constructor que recibe un mensaje personalizado
37 public EdadInvalidaException(String s) {
38 //Llamamos al constructor de la clase base Exception y pasamos el mensaje que recibimos
39 super(s);
40 }
41 // Constructor por defecto
42 public EdadInvalidaException() {
43 super("Edad inválida");
44 }
45 }
}

```

## 9.9. Píldoras informáticas relacionadas

Videos de Makigas al respecto:

- [Java: introducción a las excepciones](#)
- [Java: throw y throws, usos y diferencias](#)

⌚ 8 de enero de 2026

## 10. 4.2 Ejercicios de la UD03

### 10.1. Retos

1. (Descuento) modifique el programa para que, en lugar de realizar un descuento del 8% si la compra es de 100 € o más, aplique una penalización de 2 € si el precio es inferior a 30 €.

```

1 import java.util.Scanner;
2 //Un programa que calcula descuentos.
3
4 public class _1_Descuento{
5 public static final float DESCUENTO= 8;
6 public static final float COMPRA_MIN = 100;
7
8 public static void main(String[] args) {
9 Scanner lector = new Scanner(System.in);
10 System.out.print("¿Cuál es el precio del producto, en euros?");
11 float precio= lector.nextFloat();
12 lector.nextLine();
13 if (precio<= COMPRA_MIN) {
14 float descuentoHecho= precio * DESCUENTO / 100;
15 precio = precio - descuentoHecho;
16 }
17 System.out.println("El precio final a pagar es de "+ precio +" euros.");
18 }
19 }
```

2. (Adivina) modifique el programa para que, en lugar de un único valor secreto, haya dos. Para ganar, basta con acertar uno de los dos. La condición lógica que necesitará ya no se puede resolver con una expresión compuesta por una única comparación. Será más compleja.

#### Acción

Para pasar satisfactoriamente los tests, la variable `VALOR_SECRETO` debe ser renombrada a `VALOR_SECRET01`, y la nueva debe llamarse `VALOR_SECRET02` (deberá generarse aleatoriamente) y aunque el nombre sea de constante, no puede ser final.

```

1 import java.util.Scanner;
2
3 public class Adivina{
4
5 public static final int VALOR_SECRETO = 4;
6
7 public static void main(String[] args) {
8 Scanner lector = new Scanner(System.in);
9 System.out.println("Empecemos el juego.");
10 System.out.print("Adivina el valor entero, entre 0 y 10: ");
11 int valorUsuario = lector.nextInt();
12 lector.nextLine();
13 if (VALOR_SECRETO == valorUsuario) {
14 System.out.println("¡Exactamente! Era " + VALOR_SECRETO + ".");
15 } else {
16 System.out.println("¡Te has equivocado!");
17 }
18 System.out.println("Hemos terminado el juego.");
19 }
20 }
```

3. (AdivinaCorrecto) modifique el ejemplo anterior (Adivina) para que comprueben que el valor que ha introducido el usuario se encuentra dentro del rango de valores correcto (entre 0 y 10).
4. (AdivinaControlErroresEntrada) aplique el mismo tipo de control sobre los datos de la entrada del ejemplo siguiente al ejercicio del reto 1.

#### Acción

Para pasar satisfactoriamente los tests, el mensaje de error cuando no se introduzca un entero debe contener la palabra "ERROR"

```

1 import java.util.Scanner;
2
3 public class AdivinaControlErroresEntrada{
4
5 public static final int VALOR_SECRETO = 4;
6
7 public static void main(String[] args) {
8 Scanner lector = new Scanner(System.in);
9 System.out.println("Empecemos el juego.");
10 System.out.print("Adivina el valor entero, entre 0 y 10: ");
11 boolean tipoCorrecto = lector.hasNextInt();
12 if (tipoCorrecto) {
13 //Se ha escrito un entero correctamente. Ya puede leerse.
14 int valorUsuario = lector.nextInt();
15 lector.nextLine();
16 if (VALOR_SECRETO == valorUsuario) {
17 System.out.println("Exacto! Era " + VALOR_SECRETO + ".");
18 } else {
19 System.out.println("Te has equivocado!");
20 }
21 System.out.println("Hemos terminado el juego.");
22 } else {
23 //No se ha escrito un entero.
24 System.out.println("El valor introducido no es un entero.");
25 }
26 }
27 }
```

5. (Linea) Modifique el ejemplo para que primero pregunte al usuario cuántos caracteres "-" quiere escribir por pantalla, y entonces los escriba. Cuando pruebe el programa, no introduzca un número muy alto!

```

1 //Un programa que escribe una linea con 100 caracteres '-'.
2
3 public class Linea {
4
5 public static void main(String[] args) {
6 //Inicializamos un contador
7
8 int i = 0;
9 //¿Ya hemos hecho esto 100 veces?
10 while (i < 100) {
11 System.out.print("-");
12 //Lo hemos hecho una vez, sumamos 1 al contador
13
14 i = i + 1;
15 }
16 //Forzamos un salto de linea
17 System.out.println();
18 }
19 }
```

6. (TablaMultiplicar) un contador tanto puede empezar a contar desde 0 e ir subiendo, como desde el final e ir disminuyendo como una cuenta atrás. Modifique este programa para que la tabla de multiplicar comience mostrando el valor para 10 y vaya bajando hasta el 1.

```

1 import java.util.Scanner;
2
3 public class TablaMultiplicar{
4
5 public static void main(String[] args) {
6 Scanner lector = new Scanner(System.in);
7 System.out.print("¿Qué tabla de multiplicar quieres? ");
8 int tabla = lector.nextInt();
9 lector.nextLine();
10 int i = 1;
11 while (i <= 10) {
12 int resultado = tabla * i;
13 System.out.println(tabla + " * " + i + " = " + resultado);
14 i = i + 1;
15 }
16 System.out.println("Ésta ha sido la tabla del " + tabla);
17 }
18 }
```

7. (Modulo) el uso de contadores y acumuladores no es excluyente, sino que puede ser complementario. Piense cómo se podría modificar el programa para calcular el resultado del módulo y la división entera a la vez. Recuerde que la división entera simplemente sería contar cuántas veces se ha podido restar el divisor.

```

1 import java.util.Scanner;
2
3 public class Modulo{
4
5 public static void main(String[] args) {
6 Scanner lector = new Scanner(System.in);
7 System.out.print("¿Cuál es el dividendo? ");
8 int dividendo = lector.nextInt();
9 lector.nextLine();
10 System.out.print("¿Cuál es el divisor? ");
11 int divisor = lector.nextInt();
12 lector.nextLine();
13 while (dividendo >= divisor) {
14 dividendo = dividendo - divisor;
15 System.out.println("Bucle: por ahora el dividendo vale " + dividendo + ".");
16 }
17 System.out.println("El resultado final es" + dividendo + ".");
18 }
19 }
```

## 10.2. Ejercicios

### 10.2.1. if else

1. (MenorDeDos) Escribir un método (`menorDeDos`) que devuelva el menor de dos números enteros introducidos por teclado.
2. (MenorDeTres) Escribir dos métodos que devuelven el menor de tres números recibidos por parámetros. Haz dos versiones:
  - `menorDeTresV1`: utilizando los operadores de comparación (`<`, `>`, `==`) y lógicos (`&&`, `||`, ... ) necesarios
  - `menorDeTresV2`: sin utilizar ninguno de los operadores lógicos (habrá que usar sentencias `if` `else` anidadas)
3. (IntermedioDeTres) Escribir un método (`intermedioDeTres`) que devuelva el intermedio de tres números recibidos por parámetros.
4. (NotasTexto) Escribir un método (`notas2Texto`) que recibe un valor numérico (se supone entre 1 y 10 y puede contener decimales) y devuelva el literal correspondiente a dicha nota según ("insuficiente", "suficiente", "bien", "notable", "sobresaliente" y "matrícula de honor").
5. (Division) Escribir un método (`division`) que reciba dos números enteros e imprima el resultado de la división. Tener en cuenta que si dividimos un número por cero se producirá un error de ejecución y debemos evitarlo. Ejemplos de ejecución:

```

1 Vamos a dividir dos números.
2 Introduce el valor del dividendo: 15
3 Introduce el valor del divisor: 3
4 La división es: 5.0
5
6 Vamos a dividir dos números.
7 Introduce el valor del dividendo: 4
8 Introduce el valor del divisor: 0
9 No se puede realizar la división porque el divisor es 0
```

6. (Raiz) Se desea calcular la raíz cuadrada real de un número real cualquiera pedido inicialmente al usuario. Como dicha operación no está definida para los números negativos es necesario tratar, de algún modo, dicho posible error sin que el programa detenga su ejecución. Debes escribir un método `raiz` que recibe un entero. Ejemplos de ejecución:

```

1 Vamos a calcular la raíz cuadrada de un número.
2 Introduce el valor del número: 81
3 La raíz es: 9.0
4
5 Vamos a calcular la raíz cuadrada de un número.
6 Introduce el valor del número: -52
7 No se puede realizar la raíz porque el número es negativo
```

7. (Hora12) Escribir un método (`formatoDoce`) que recibe dos enteros (hora y minutos) en notación de 24 horas y la devuelve como `String` en notación de 12 horas. Por ejemplo, si la entrada es 13 horas 45 minutos, la salida será "1:45 PM". La hora y los minutos se leerán de teclado de forma separada, primero la hora y luego los minutos. Si la hora pasada al método no es correcta devolverá: "Error de conversión"

```

1 Vamos a convertir una hora a formato 12H.
2 Introduce el valor para la hora: 51
3 Introduce el valor para los minutos: 6
4 Error de conversión.
5
6 Vamos a convertir una hora a formato 12H.
7 Introduce el valor para la hora: 13
8 Introduce el valor para los minutos: 45
9 01:45 PM
10
11 Vamos a convertir una hora a formato 12H.
12 Introduce el valor para la hora: 4
13 Introduce el valor para los minutos: 53
14 04:53 AM

```

8. (Bisiesto) Escribir un método (`esBisiesto`) que devuelva si un año introducido por teclado es o no bisiesto. Un año es bisiesto si es múltiplo de 4 (por ejemplo 1984). Sin embargo, los años múltiples de 100 no son bisiestos, salvo que sean múltiplos de 400, en cuyo caso si lo son (por ejemplo 1800 no es bisiesto y 2000 si lo es). Para hacer el programa, implementa un método dentro de la clase que reciba un año y devuelva true si el año es bisiesto y false en caso de que no lo sea.

9. (Fechas) Escribir un método (`imprimeMenor`) que reciba dos fechas (día, mes y año) (6 parámetros en total), que se suponen correctas, y le muestre la menor de ellas. La fecha se mostrará en formato dd/mm/año. Utiliza un método `mostrarFecha`, para mostrar la fecha por pantalla. La fecha se mostrará siempre con dos dígitos para el día, dos para el mes y cuatro para el año.

```

1 Vamos a ver que fecha es menor.
2 Introduce el valor para el Fecha1.
3 - Dia: 4
4 - Mes: 5
5 - Año: 1999
6
7 Introduce el valor para el Fecha2.
8 - Dia: 3
9 - Mes: 5
10 - Año: 1999
11
12 La fecha menor es:
13 03/05/1999

```

10. (DiasDelMes) Escribir un método (`diasMes`) que recibe el número de un mes (1 a 12) y devuelve el número de días que tiene el mes. Hacerlo utilizando sentencias `if else`. Si el mes recibido no es correcto, debe devolver 0.

```

1 Vamos a ver cuantos días tiene un mes.
2 Introduce el valor para el mes: 8
3 Este mes tiene 31 días.
4
5 Vamos a ver cuantos días tiene un mes.
6 Introduce el valor para el mes: 54
7 Este mes tiene 0 días.

```

11. (NombreDelMes) Escribir un programa que lea de teclado el número de un mes (1 a 12) y visualice el nombre del mes (enero, febrero, etc). Hacerlo utilizando sentencias `if else`. Para hacer un programa, implementa un método en la clase que reciba un número de mes y devuelva el nombre del mes

12. (Salario) Escribir un método (`salario`) que recibe las horas trabajadas por un empleado en una semana y calcula y devuelve su salario neto semanal, sabiendo que:

- Las horas ordinarias se pagan a 6 €.
- Las horas extraordinarias se pagan a 10 €.
- Los impuestos a deducir son:
  - Un 2 % si el salario bruto semanal es menor o igual a 350 €
  - Un 10 % si el salario bruto semanal es superior a 350 €
  - La jornada semanal ordinaria son 40 horas. El resto de horas trabajadas se considerarán horas extra.

13. (Signo) Dados dos números enteros, num1 y num2, realizar un programa que escriba uno de los dos mensajes:

- "el producto de los dos números es positivo o nulo" o bien
- "el producto de los dos números es negativo".

Resolverlo sin calcular el producto, sino teniendo en cuenta únicamente el signo de los números a multiplicar.

14. (Calculadora) Escribir un programa para simular una calculadora. Considera que los cálculos posibles son del tipo num1 operado num2, donde num1 y num2 son dos números reales cualesquiera y operador es una de entre: +, -, \* y /. El programa pedirá al usuario en primer lugar el valor num1, a continuación el operador y finalmente el valor num2. Resolver utilizando instrucciones `if else`

15. (Comercio) Un comercio aplica un descuento del 8% por compras superiores a 40 euros. El descuento máximo será de 12 euros. Escribir un programa que solicite al usuario el importe de la compra y muestre un mensaje similar al siguiente:

- Importe de la compra 100 €
- Porcentaje de descuento aplicado: 8%
- Descuento aplicado: 8 €
- Cantidad a pagar: 92 €

16. (Editorial) Una compañía editorial dispone de 2 tipos de publicaciones: libros y revistas. El precio de cada pedido depende del número de elementos solicitados al cual se le aplica un determinado descuento, que es diferente para libros y para revistas. La siguiente tabla muestra los descuentos a aplicar en función del número de unidades y del tipo de producto:

| Cantidad pedida         | Libros            | Revistas          |
|-------------------------|-------------------|-------------------|
| Hasta 5 unidades        | 0 % de descuento  | 0 % de descuento  |
| De 6 a 10 unidades      | 10 % de descuento | 15 % de descuento |
| De 11 a 20 unidades     | 15 % de descuento | 20 % de descuento |
| A partir de 20 unidades | 20 % de descuento | 25 % de descuento |

Escribe un método `calcularCoste` que, recibiendo el tipo de publicación (`String`), que puede ser "libro" o "revista", el precio individual (`double`) y el número de unidades solicitado (`int`), devuelva el coste del pedido (aplicando el descuento correspondiente). Escribe un programa en el que el usuario deba introducir cantidad y precio de revistas y cantidad y precio de libros que incluye un pedido, y muestre el coste del pedido.

Si el tipo especificado es diferente de "libro" o "revista", el método devolverá -1.

```

1 Vamos a calcular el coste del pedido.
2
3 Introduce los datos respecto a LIBROS.
4 Precio unitario: 12,3
5 Cantidad: 4
6
7 Introduce los datos respecto a REVISTAS.
8 Precio unitario: 2,6
9 Cantidad: 5
10
11 El coste total de pedido será de 62,20€.

```

17. (Taxi) Se desea calcular el coste del trayecto realizado en taxi en función de los kilómetros recorridos en las carreras metropolitanas de Valencia. Según las tarifas vigentes para el 2012, el coste se calcula de la siguiente manera:

- Días laborables en horario diurno (de 6:00 a antes de las 22:00h): 0.73 €/km.
- Días laborables en horario nocturno: 0.84 €/km.
- Sábados y domingos: 0.93 €/km.
- Además, la tarifa mínima diurna es de 2.95€ y la mínima nocturna de 4€.

Escribir un método que recibe:

- La hora (`hora` y `minutos`) en que se realizó el trayecto.
- El día de la semana (se supone que el usuario introduce un valor entre 1 para lunes y 7 para domingo)
- Los kilómetros recorridos.

El método devuelve el coste del trayecto. Escribe un método `main` que solicite al usuario la introducción de los datos y llamando al método muestre el resultado:

```

1 Vamos a calcular el coste del trayecto.
2 Introduce la hora: 15
3 Introduce los minutos: 20
4 Introduce el dia de la semana (1 para lunes y 7 para domingo): 6
5 Introduce los kilometros recorridos: 3,5
6
7 El coste total del trayecto es de 3,26€.

```

18. (Nombre) Escribir un método (`coincidenPrimeraYUltima`) que recibe una cadena de texto. El método devolverá si la primera y la última letra del nombre coinciden o no.

Haz que funcione aunque las letras tengan diferente CASE (pista: lowercase i uppercase).

Escribe el método `main` que solicite al usuario un nombre de persona y llamando al método diga si coinciden o no, pruébalo con "Ana", "ana", "Angel", "Amanda" y "David":

```

1 Vamos a ver si la primera y la última letra de tu nombre son iguales.
2 Introduce un nombre: Ana
3 Si coinciden.
4
5 Vamos a ver si la primera y la última letra de tu nombre son iguales.
6 Introduce un nombre: Angel
7 No coinciden.

```

19. (Validar) Se desea implementar un programa que determine si dos datos `x` e `y` de entrada son válidos. Un par de datos es válido si es uno de los que aparecen en la siguiente tabla:

| <b>x :</b>       | <b>a</b> | <b>a</b> | <b>a</b> | <b>a</b> | <b>a</b> | <b>b</b> |
|------------------|----------|----------|----------|----------|----------|----------|
| <code>y :</code> | 1        | 3        | 5        | 7        | 9        | 2        |

Se pide implementar un método (`esValido`) que recibe los dos valores (`x` e `y`), que devuelva si la combinación es válida o no.

Escribe también un método `main` que lea de teclado el valor de `x` y el valor de `y`, e indique por pantalla "VALIDOS" o "NO VALIDOS". Se pide hacerlo de forma que no se utilice ninguna estructura condicional (if, switch,...), es decir, se calculará una expresión booleana que determine si `x` e `y` son válidos. Se procurará que la expresión booleana propuesta sea breve y concisa.

### 10.2.2. Bucles simples

1. (SencillosWhile) Crear una clase llamada `SencillosWhile` y crear en él métodos que realicen las siguientes tareas.

- (imparesHastaN) Recibe un nº entero `n`, devuelve los números impares que hay entre 1 y `n`. Por ejemplo, si `n` es 8 devolverá "1 3 5 7"
- (nImpares) Recibe un nº entero `n`, devuelve los `n` primeros números impares. Por ejemplo, si `n` es 3 devolverá "1 3 5" (3 primeros impares)
- (cuentaAtras) Recibe un entero `n`, devuelve una cuenta atrás partiendo de `n`: `n`, `n-1`, .... Por ejemplo, si `n` es 5 devolverá "5 4 3 2 1 0"
- (sumaNPrimeros) Recibe un entero `n`, devuelve la suma de los números entre 1 y `n`. Por ejemplo, si `n` es 5 devolverá 15
- (mostrarDivisoresN) Recibe un entero `n`, devuelve todos sus divisores, incluidos el 1 y el mismo `n`. Por ejemplo, si `n` es 12 mostraría "1 2 3 4 6 12"
- (sumaDivisoresN) Recibe un entero `n`, devuelve la suma de todos sus divisores, sin incluir al propio `n`. Por ejemplo, si `n` es 12 sumará 1, 2, 3, 4 y 6 = 16

Ejemplo de ejecución con todos los métodos:

```

1 Esta realiza varios While sencillos:
2 Introduce un número: 15
3 Imprimimos:
4 Todos los números impares menores que 15: 1 3 5 7 9 11 13 15
5 Los primeros 15 números impares: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29
6 Cuenta atrás desde 15: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
7 La suma de los 15 primeros números resulta: 120
8 Los divisores del número 15 son: 1 3 5 15
9 La suma de todos los divisores del número 15 (sin él mismo) son: 9

```

2. (SencillosFor) Crear una clase llamada "SencillosFor" y crear en él los mismos métodos que en el ejercicio anterior, pero utilizando la sentencia `for` en lugar de `while`:

```

1 Esta realiza varios For sencillos:
2 Introduce un número: 15
3 Imprimimos:
4 Todos los números impares menores que 15: 1 3 5 7 9 11 13 15
5 Los primeros 15 números impares: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29
6 Cuenta atrás desde 15: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
7 La suma de los 15 primeros números resulta: 120
8 Los divisores del número 15 son: 1 3 5 15
9 La suma de todos los divisores del número 15 (sin él mismo) son: 9

```

3. (PotenciasDe2) Crea un método (`potencias`) que recibe un entero `n` positivo y devuelve las `n` primeras potencias de 2. Es decir,  $2^0, 2^1, 2^2, 2^3, \dots, 2^n$ . Crea un método `main` que solicite un número al usuario y muestre por pantalla todas las potencias de 2. Soluciona el ejercicio sin utilizar `Math.pow`. Ten en cuenta que, por ejemplo,  $2^3 = 1 * 2 * 2 * 2$  o que  $2^4 = 1 * 2 * 2 * 2 * 2$ :

```

1 Dime n: 20
2 1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072 262144 524288 1048576

```

4. (Etapas) El ser humano pasa por una serie de etapas en su vida que, con carácter general se asocian a las edades que aparecen en la tabla siguiente.

| Etapa        | Rango                   |
|--------------|-------------------------|
| Infancia     | Hasta los 10 años       |
| Pubertad     | De 11 a 14 años         |
| Adolescencia | De 15 a 21 años         |
| Adulvez      | De 22 a 54 años         |
| Vejez        | De 55 a 70 años         |
| Ancianidad   | A partir de los 71 años |

Escribe un método `main` en el que el usuario introduzca las edades de una serie de personas y calcule y muestre que porcentaje de personas que se encuentran en cada etapa. En primer lugar el programa pedirá el número de personas que participan en la muestra y a continuación solicitará la edad de cada una de ellas. El resultado será similar al siguiente:

```

1 ¿Cuántas personas hay en la muestra?: 5
2 Introduce la edad de la persona 1 de 5 : 17
3 --> adolescencia
4 Introduce la edad de la persona 2 de 5 : 21
5 --> adolescencia
6 Introduce la edad de la persona 3 de 5 : 53
7 --> adulvez
8 Introduce la edad de la persona 4 de 5 : 62
9 --> vejez
10 Introduce la edad de la persona 5 de 5 : 78
11 --> ancianidad
12 Etapa Número %
13 Infancia: 0 0.0%
14 Pubertad: 0 0.0%
15 Adolescencia: 2 40.0%
16 Adulvez: 1 20.0%
17 Vejez: 1 20.0%
18 Ancianidad: 1 20.0%
19
20 ¿Cuántas personas hay en la muestra?: 3
21 Introduce la edad de la persona 1 de 3 : 1
22 --> infancia
23 Introduce la edad de la persona 2 de 3 : 2
24 --> infancia
25 Introduce la edad de la persona 3 de 3 : 25
26 --> adulvez
27 Etapa Número %
28 Infancia: 2 66.66666666666667%
29 Pubertad: 0 0.0%
30 Adolescencia: 0 0.0%
31 Adulvez: 1 33.33333333333336%
32 Vejez: 0 0.0%
33 Ancianidad: 0 0.0%
```

5. (Primo) Escribir un método `esPrimo` que recibe un entero y devuelve si es primo o no. Escribe un método `main` en el que el usuario escriba un número entero y se le diga si se trata o no de un número primo, usando el método `esPrimo`. Recuerda que un nº primo es aquel que solo es divisible por 1 y por sí mismo (Es decir tiene SOLO y EXCLUSIVAMENTE dos divisores cuyo resto sea cero).

```

1 Este te dice si un número es primo:
2 Introduce un número: 15
3 El número es primo?: false
4
5 Este te dice si un número es primo:
6 Introduce un número: 7
7 El número es primo?: true
```

6. (Primos) Escribir un programa en el que el usuario escriba un número entero y se le diga todos los números primos entre 1 y el número introducido (Puedes usar el método `esPrimo` que has escrito en el ejercicio anterior).

7. (EsPrimoMejorada) Haz una nueva versión del programa del ejercicio anterior teniendo en cuenta lo siguiente:

- El único número par que es primo es el 2.
- Un número  $n$  no puede tener divisores mayores que  $n/2$  (o mayores que `Math.sqrt(n)`)

**A**tenCIÓN

Para pasar satisfactoriamente los tests de rendimiento, debes tener el ejercicio 6 (Primos) y 7 (EsPrimoMejorada) y el rendimiento del 7 debe ser un 20% mejor (más rápido) que el del 6

8. (Divisores) Escribir un programa que muestre los tres primeros divisores de un número n introducido por el usuario. Por ejemplo, si el usuario introduce el número 45, el programa mostrará los divisores 1, 3 y 5. Ten en cuenta que la posibilidad de que el número n tenga menos de 3 divisores. Prueba qué pasa si el usuario pide, por ejemplo, los tres primeros divisores de 7.
9. (SumaSerie) Dado un número n, introducido por el usuario, calcula y muestra por pantalla la siguiente suma  $1/1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$
10. (Cifras) Escribir un método (`numeroCifras`) que recibe un número entero cualquiera (positivo, negativo o cero) y devuelve cuantas cifras tiene. Pistas: ¿Cuántas cifras tiene el nº 25688? ¿Cuántas veces podemos dividir el nº 25688 por 10 hasta que se hace cero? Cuidado, el nº 0 tiene una cifra.

```

1 Vamos a decirte cuantas cifras tiene un número:
2 Introduce un número: 0
3 El número tiene 1 cifras.
4
5 Vamos a decirte cuantas cifras tiene un número:
6 Introduce un número: 25688
7 El número tiene 5 cifras.
8
9 Vamos a decirte cuantas cifras tiene un número:
10 Introduce un número: -14
11 El número tiene 2 cifras.

```

11. (Transportes) Una empresa de transportes cobra 30€ por cada bulto que transporta. Además, si el peso total de todos los bultos supera los 300 kilos, cobra 0.9€ por cada kg extra. Por último si el trasporte debe realizarse en sábado, cobra un plus de 60€. La empresa no realiza el pedido si hay que transportar más de 30 bultos, si el peso total supera los 1000 kg o si se solicita hacerlo en domingo. Realizar un método `main` que solicite el número de bultos, el día de la semana (valor entre 1 y 7) y el peso de cada uno de los bultos y muestre el coste del transporte en caso de que pueda realizarse o un mensaje adecuado en caso contrario:

```

1 ¿Cuantos bultos debes enviar?: 3
2 ¿Qué dia de la semana se realiza el envio (entre 1=lunes y 7=domingo)?: 7
3 No se puede realizar el envio por superar los 30 bultos o ser domingo
4
5 ¿Cuantos bultos debes enviar?: 4
6 ¿Qué dia de la semana se realiza el envio (entre 1=lunes y 7=domingo)?: 6
7 Cuanto pesa el bulto 1 de 4 : 25
8 Cuanto pesa el bulto 2 de 4 : 120
9 Cuanto pesa el bulto 3 de 4 : 35
10 Cuanto pesa el bulto 4 de 4 : 78
11 El coste del envio será de: 180.0
12
13 ¿Cuantos bultos debes enviar?: 2
14 ¿Qué dia de la semana se realiza el envio (entre 1=lunes y 7=domingo)?: 1
15 Cuanto pesa el bulto 1 de 2 : 700
16 Cuanto pesa el bulto 2 de 2 : 400
17 No se puede realizar el envio por superar los 1000Kg

```

12. (Containers) La capacidad de un buque que transporta containers está limitada tanto por la cantidad de containers como por el peso, pudiendo transportar un máximo de 100 containers y un máximo de 700 toneladas. Hacer un programa en el que se vaya introduciendo el peso de los containers (en toneladas) a medida que se cargan en el barco, hasta que se llegue al máximo de capacidad. Mostrar al final la cantidad de containers cargados y el peso total. En el momento en que se desee cargar un container que haga que la carga total supere las 700 toneladas, se dará por finalizada la carga, aunque pudieran existir containers menos pesados con posibilidad de ser cargados.
13. (Notas) Realizar un método `main` que permita introducir las notas de varios exámenes de un alumno de un curso. El usuario irá introduciendo las notas una tras otra. Se considerará finalizado el proceso de introducción de notas cuando el usuario introduzca una nota negativa. Al final, el programa le dirá: - El número de notas introducidas. - El número de aprobados (mayor o igual a 5 puntos) - La nota media

```

1 Vamos a darte información sobre tus notas, introduce una nota tras otra. Para finalizar escribe una nota negativa:
2 Introduce la nota 1: 5,5
3 Introduce la nota 2: 6,7
4 Introduce la nota 3: 8,4
5 Introduce la nota 4: 3,2
6 Introduce la nota 5: 1,5
7 Introduce la nota 6: -3
8 Has introducido un total de 5 notas,
9 de las cuales has aprobado 3,
10 con una media de 4.

```

14. (NotasExtremas) Modificar el ejercicio anterior para que además calcule la nota máxima y la nota mínima.

### 10.2.3. Bucles anidados

1. (Edades) Programa que pida al usuario la edad de cinco personas. Si la suma de las edades es inferior a 200, el programa volverá a solicitar las 5 edades.

```

1 Introduce la edad de la persona 1: 80
2 Introduce la edad de la persona 2: 42
3 Introduce la edad de la persona 3: 15
4 Introduce la edad de la persona 4: 56
5 Introduce la edad de la persona 5: 34

```

```

1 Introduce la edad de la persona 1: 15
2 Introduce la edad de la persona 2: 12
3 Introduce la edad de la persona 3: 43
4 Introduce la edad de la persona 4: 5
5 Introduce la edad de la persona 5: 4
6 La suma de edades debe ser superior a 200
7
8 Introduce la edad de la persona 1: 48
9 Introduce la edad de la persona 2: 54
10 Introduce la edad de la persona 3: 35
11 Introduce la edad de la persona 4: 12
12 Introduce la edad de la persona 5: 89

```

2. (NotasPorAlumno) Programa que pida al usuario las notas de `A` alumnos en `s` asignaturas, alumno por alumno. `A` y `s` se definirán en el programa como `CONSTANTES`.

#### Atención

Para pasar satisfactoriamente los tests, la variable `A` debe ser 3 y `s` debe ser 5.

```

1 Alumno 1
2 Introduce la nota de la asignatura 1: 5
3 Introduce la nota de la asignatura 2: 4
4 Introduce la nota de la asignatura 3: 3
5 Introduce la nota de la asignatura 4: 5
6 Introduce la nota de la asignatura 5: 0
7 Alumno 2
8 Introduce la nota de la asignatura 1: 6
9 Introduce la nota de la asignatura 2: 8
10 Introduce la nota de la asignatura 3: 9
11 Introduce la nota de la asignatura 4: 8
12 Introduce la nota de la asignatura 5: 7
13 Alumno 3
14 Introduce la nota de la asignatura 1: 0
15 Introduce la nota de la asignatura 2: 0
16 Introduce la nota de la asignatura 3: 2
17 Introduce la nota de la asignatura 4: 4
18 Introduce la nota de la asignatura 5: 5

```

3. (NotasPorAsignatura) Programa que pida al usuario las notas de `A` alumnos en `s` asignaturas, asignatura por asignatura. `A` y `s` se definirán en el programa como `CONSTANTES`.

```

1 Asignatura 1
2 Introduce nota del alumno 1:
3 Introduce nota del alumno 2:
4 ...
5 Asignatura 2
6 Introduce nota del alumno 1:
7 ...

```

4. (MediasPorAsignatura) Repite el ejercicio anterior haciendo que se muestre la media de cada asignatura

```

1 Asignatura 1
2 Introduce nota del alumno 1:
3 Introduce nota del alumno 2:
4 ...
5 Media asignatura 1: 8.5 puntos
6
7 Asignatura 2
8 Introduce nota del alumno 1:
9 ...
10 Media asignatura 2: 6.5 puntos
11 ...

```

5. (TablaMult) Escribir un programa que permita al usuario introducir un número `N` e imprima la tabla de multiplicar (del 0 al 10) de todos los números entre 1 y `N`. Ejemplo: Si el usuario introduce en número 5, el programa imprimiría

```

1 Tabla del 1:
2 1 por 0, 0
3 1 por 1, 1
4 1 por 2, 2
5 ...
6 1 por 10, 10
7
8 Tabla del 2:
9 2 por 0, 0
10 2 por 1, 2
11 ...
12 2 por 10, 20
13
14 Tabla del 3:
15 ...
16
17 Tabla del 5:
18 ...
19 5 por 10, 50

```

6. (PrimosHastaN) Programa que solicite al usuario un numero `n` y muestre todos los números primos menores o iguales que `n`. (IGUAL AL 27!!)

7. (CombinarLetras2) Escribir un programa que muestre todas las palabras de dos letras que se pueden formar con los cuatro primeros caracteres del alfabeto en minúsculas ('a', 'b', 'c', 'd'):

```

1 aa
2 ab
3 ac
4 ad
5 ba
6 bb
7 bc
8 bd
9 ...
10 da
11 db
12 dc
13 dd

```

### Acción

Para pasar satisfactoriamente los tests, en lugar de usar `System.out.println`, debes usar `System.out.print` con un `\n` al final de la cadena a imprimir.

8. (CombinarLetras3) Repite el ejercicio anterior mostrando palabras de tres letras

```

1 aaa
2 aab
3 ...
4 ddc
5 ddd

```

### Acción

Para pasar satisfactoriamente los tests, en lugar de usar `System.out.println`, debes usar `System.out.print` con un `\n` al final de la cadena a imprimir.

9. (LetraALetra) Escribe un programa en el que se solicite al usuario un texto de forma repetida hasta que el usuario introduzca la cadena vacía. Con cada texto que introduzca el usuario se le mostrará carácter a carácter, cada carácter en una línea

```

1 Vamos a escribir textos letra a letra
2 Introduce texto: Hola
3 H
4 o
5 l
6 a
7 Introduce texto: Casa
8 C
9 a
10 s
11 a
12 Introduce texto:
13 Fin del programa

```

### Acción

Para pasar satisfactoriamente los tests, en lugar de usar `System.out.println`, debes usar `System.out.print` con un `\n` al final de la cadena a imprimir.

10. (DibujarFiguras1) Escribe una clase que contenga los métodos que se indican a continuación. En el método `main` solicita al usuario las dimensiones de las figuras necesarias en cada caso y llama al método correspondiente para que se muestre por pantalla
- a. `void dibRecAsteriscos (int ancho, int alto)` dibuja un rectángulo utilizando asteriscos, como el siguiente. En el ejemplo ancho es 7 y alto es 3

```

1 * * * * * *
2 * * * * * *
3 * * * * * *

```

```

1 Vamos a dibujar figuras.
2 Introduce el ancho: 8
3 Introduce el alto: 4
4 * * * * * * *
5 * * * * * * *
6 * * * * * * *
7 * * * * * * *

```

- b. `void dibRecNumeros1 (int ancho, int alto)` dibuja un rectángulo utilizando números, como el siguiente. En el ejemplo ancho es 7 y alto es 3

```

1 1 2 3 4 5 6 7
2 1 2 3 4 5 6 7
3 1 2 3 4 5 6 7

```

- c. `void dibRecNumeros2 (int ancho, int alto)` dibuja un rectángulo utilizando números, como el siguiente. En el ejemplo ancho es 7 y alto es 3

```

1 7 6 5 4 3 2 1
2 7 6 5 4 3 2 1
3 7 6 5 4 3 2 1

```

- d. `void dibRecNumeros3 (int ancho, int alto)` dibuja un rectángulo utilizando números, como el siguiente. En el ejemplo ancho es 7 y alto es 3

```

1 01 02 03 04 05 06 07
2 08 09 10 11 12 13 14
3 15 16 17 18 19 20 21

```

- e. `void dibDiagonal (int ancho, int alto)` dibuja un rectángulo con ceros y unos. Los 1 están en las posiciones en las que fila y columna coinciden. En el ejemplo ancho es 7 y alto es 3

```

1 1 0 0 0 0 0 0
2 0 1 0 0 0 0 0
3 0 0 1 0 0 0 0

```

- f. `void dibRecLetras (int ancho, int alto)` dibuja un rectángulo letras sucesivas comenzando por la "a". En el ejemplo ancho es 7 y alto es 3

```

1 a a a a a a a
2 b b b b b b b
3 c c c c c c c

```

g. void dibRecLetras2 (int ancho, int alto) dibuja un rectángulo letras sucesivas terminando por la "a". En el ejemplo ancho es 7 y alto es 3

```

1 c c c c c c c
2 b b b b b b b
3 a a a a a a a

```

h. void dibRecLetras3 (int ancho, int alto) dibuja un rectángulo letras sucesivas comenzando por la "a". En el ejemplo ancho es 7 y alto es 3

```

1 a b c d e f g
2 h i j k l m n
3 o p q r s t u

```

11. (dibujarFiguras2) Escribe una clase que contenga los métodos que se indican a continuación. En el método main solicita al usuario las dimensiones de las figuras necesarias en cada caso y llama al método correspondiente para que se muestre por pantalla

a. void dibRectNumeros3 (int ancho, int alto) dibuja un rectángulo utilizando números, como el siguiente. En el ejemplo ancho es 7 y alto es 3

```

1 1 2 3 4 5 6 7 7 6 5 4 3 2 1
2 1 2 3 4 5 6 7 7 6 5 4 3 2 1
3 1 2 3 4 5 6 7 7 6 5 4 3 2 1

```

b. void dibRectAsteriscos1 (int ancho, int alto) dibuja un rectángulo utilizando asteriscos (\*) y espacios en blanco, como el siguiente. En el ejemplo ancho es 7 y alto es 3

```

1 * * * * * *
2 * * * * * *
3 * * * * * *

```

c. void dibRectAsteriscos2 (int ancho, int alto) dibuja un rectángulo utilizando asteriscos (\*), espacios en blanco y el carácter '+', como el siguiente. En el ejemplo ancho es 7 y alto es 3

```

1 * + * + * + *
2 * + * + * + *
3 * + * + * + *

```

d. void dibRectAsteriscos3 (int ancho, int alto) dibuja un rectángulo utilizando asteriscos (\*) y espacios en blanco, como el siguiente. En el ejemplo ancho es 7 y alto es 3

```

1 * * * * * *
2 *
3 * * * * * *

```

e. void dibTriangulo1 (int base) dibuja un triángulo utilizando asteriscos (\*) y espacios en blanco, como el siguiente. En el ejemplo base es 5

```

1 *
2 **
3 ***
4 ****
5 *****

```

f. void dibTriangulo2 (int altura) dibuja un triángulo utilizando asteriscos (\*) y espacios en blanco, como el siguiente. En el ejemplo altura es 5

```

1 *
2 **
3 ***
4 ****
5 *****

```

g. void dibTriangulo3 ( int altura) dibuja un triángulo utilizando asteriscos (\*) y espacios en blanco, como el siguiente. En el ejemplo altura es 5

```

1 *
2 * *
3 * * *
4 * * * *
5 * * * * *

```

#### 10.2.4. switch

- (NotasTexto2) Escribir un método (`nota2TextoSwitch`) que reciba la nota de un examen (valor numérico entero entre 1 y 10) y muestre el literal correspondiente a dicha nota según (insuficiente, suficiente, bien, notable, sobresaliente, matrícula de honor). Hacerlo utilizando la sentencias switch.
- (DiasDelMes2) Escribir un método que recibe un número de mes (1 a 12) y devuelva el número de días que tiene el mes. Resolver utilizando la sentencias switch. Si el mes es erroneo, devolverá 0.

```

1 Vamos a calcular el número de días que tiene un mes.
2 Introduce el número del mes (1-12): 8
3 31
4
5 Vamos a calcular el número de días que tiene un mes.
6 Introduce el número del mes (1-12): 2
7 28
8
9 Vamos a calcular el número de días que tiene un mes.
10 Introduce el número del mes (1-12): 15
11 0

```

- (NombreDelMes2) Escribir un método (`nombreMes`) que recibe el número de un mes (1 a 12) y devuelve el nombre del mes ("enero", "febrero", etc). Resolver utilizando la sentencias switch. Si el mes no es válido, devolverá la cadena vacía "".
- (Calculadora2) Escribir un método (`calculadora`) para simular una calculadora. Considera que los cálculos posibles son del tipo `num1, operador, num2`, donde `num1` y `num2` son dos números reales cualesquiera y operador es una de entre: '+', '-', '\*' y '/'. El método recibirá los tres parámetros y devolverá el resultado de la operación. Resolver utilizando la sentencias switch. Si el `operador` no es ninguno de la lista, devuelve el `num1`.

#### 10.2.5. en papel...

- (Valor) ¿Qué valor se asignará a consumo en la sentencia `if` siguiente si velocidad es 120?

```

1 if (velocidad > 80)
2 consumo = 10;
3 else if (velocidad > 100)
4 consumo = 12;
5 else if (velocidad > 120)
6 consumo = 15;

```

- (Errores) Encuentra y corrige los errores de los siguientes fragmentos de programa.

## a. fragmento a

```

1 if (x > 25)
2 y = x
3 else
4 y = z;

```

## b. fragmento b

```

1 if (x<0)
2 System.out.println("El valor de x es" +x);
3 System.out.println ("x es negativo");
4 else
5 System.out.println ("El valor de x es"+x);
6 System.out.println ("x es positivo");

```

## c. fragmento c

```

1 if (x = 0) System.out.println ("x igual a cero");
2 else System.out.println ("x distinto de cero");

```

## 3. (SalidaExacta) Cuál es la salida exacta por pantalla del siguiente fragmento de programa

```

1 int x = 20;
2 System.out.println("Comenzamos");
3 if (x<= 20)
4 if (x>50) System.out.println("Muy grande");
5 else {
6 if (x%2 != 0) System.out.println("Impar");
7 }
8 else if (x<=20) System.out.println("Pequeño");
9 System.out.println("Terminamos");

```

## 4. (Descuentos) En una tienda, por liquidación, se aplican distintos descuentos en función del total de las compras realizadas:

- Si total < 500 €, no se aplica descuento.
- Si 500 € <= total <= 2000 €, se aplica un descuento del 30 %.
- Si total > 2000 €, entonces se aplica un descuento del 50 %

¿Cuál de los siguientes fragmentos de programa asigna a la variable `desc` el descuento correcto? Indica "Si" o "NO" al lado de cada fragmento

## a. fragmento a

```

1 double desc = 0.0;
2 if (total <= 500)
3 if (total >= 2000) desc = 30.0;
4 else desc = 50.0;
5 total = total * desc / 100.0;

```

## b. fragmento b

```

1 double desc = 0.0;
2 if (total >= 500)
3 if (total <= 2000) desc = 30.0;
4 else desc = 50.0;
5 total = total * desc / 100.0;

```

## c. fragmento c

```

1 double desc = 0.0;
2 if (total <= 2000){
3 if (total >= 500) desc = 30.0;
4 } else desc = 50.0;
5 total = total * desc / 100.0;

```

## d. fragmento d

```

1 double desc = 0.0;
2 if (total > 500)
3 if (total < 2000) desc = 30.0;
4 else desc = 50.0;
5 total = total * desc /100.0;

```

5. (Salida) ¿Qué salida producirá el siguiente fragmento de programa si la variable entera platos vale 1? ¿Y si vale 3? ¿Y si vale 0?

```

1 switch (platos) {
2 case 1: System.out.println("\nPrimer plato");
3 case 2: System.out.println ("\nSegundo plato");
4 case 3: System.out.println ("\\nBebida");
5 System.out.println ("\nPostre");
6 break;
7 default: System.out.println("\nCafé");
8 }

```

6. (ValorP) Dados tres enteros a, b y c, y un booleano p, el siguiente análisis por casos establece el valor de p en función de los valores de a, b y c:

```

1 Si a > b entonces p = cierto;
2 si a < b entonces p = falso;
3 si a = b entonces
4 si a > c entonces p = cierto;
5 si a < c entonces p = falso;
6 si a = c entonces p = falso;

```

Se pide la traducción de dicho análisis por casos a Java mediante:

- Una única instrucción `if` sin anidamientos.
- Una única instrucción, de la forma `p = ...`, que utilice el operador ternario.
- Una única instrucción, de la forma `p = ...`, sin sentencias `if` ni utilizar el operador ternario.

#### 10.2.6. Trazas

Indica cual será la salida producida por los siguientes programas, teniendo en cuenta los datos de entrada:

1. (Trazado) **Datos de entrada: 2, 5**

a.

```

1 public static void main (String[] args){
2 Scanner tec = new Scanner(System.in);
3 int x,y,a;
4 x = tec.nextInt();
5 y = tec.nextInt();
6 a = x+y;
7 System.out.println(a);
8 }

```

b.

```

1 public static void main (String[] args){
2 Scanner tec = new Scanner(System.in);
3 int x,a;
4 x = tec.nextInt();
5 x = tec.nextInt();
6 a= x+x;
7 System.out.println(a);
8 }

```

c.

```

1 public static void main (String[] args){
2 Scanner tec = new Scanner(System.in);
3 int x,y,a;
4 x = tec.nextInt();
5 y = tec.nextInt();
6 a = x+y;
7 a = x*y;
8 System.out.println(a);
9 }

```

d.

```

1 public static void main (String[] args){
2 Scanner tec = new Scanner(System.in);
3 int x,y,a;
4 x = tec.nextInt();
5 y = tec.nextInt();
6 a = x+y;
7 System.out.println(a);
8 a = x*y;
9 System.out.println(a);
10 }

```

e.

```

1 public static void main (String[] args){
2 Scanner tec = new Scanner(System.in);
3 int x,y,a;
4 x = tec.nextInt();
5 y = tec.nextInt();
6 a = x*y;
7 a = a*x+y;
8 a = a*a;
9 System.out.println(a);
10 }

```

f.

```

1 public static void main (String[] args){
2 Scanner tec = new Scanner(System.in);
3 int x,y,a;
4 x = tec.nextInt();
5 y = tec.nextInt();
6 a = x;
7 a = doble(x);
8 System.out.format ("%d\n%d\n%d",x,y,a);
9 }
10 public static int doble(int num){
11 return 2*num;
12 }

```

g.

```

1 public static void main (String[] args) {
2 Scanner tec = new Scanner(System.in);
3 int x,y,a;
4 x = tec.nextInt();
5 y = tec.nextInt();
6 a = x;
7 doble(a);
8 System.out.format("%d\n%d\n%d\n%d",x,y,a);
9 }
10 public static void doble(int x){
11 x = 2*x;
12 }

```

h.

```

1 public static void main (String[] args){
2 Scanner tec = new Scanner(System.in);
3 int x,y,a;
4 x = tec.nextInt();
5 y = tec.nextInt();
6 a = calcular(y,x);
7 System.out.format("%d\n%d\n%d\n%d",x,y,a);
8 }
9 public static int calcular (int x, int y){
10 return x-y;
11 }

```

i.

```

1 public static void main (String[] args){
2 Scanner tec = new Scanner(System.in);
3 int x,y,a;
4 x = tec.nextInt();
5 y = tec.nextInt();
6 y = calcular(x);
7 a = calcular(y);
8 System.out.format("%d\n%d\n%d\n%d",x,y,a);
9 }
10 public static int calcular (int x){
11 return x*x;
12 }

```

2. (Traza2) **Datos de entrada: 2, 5, 7**

```

1 public static void main (String[] args){
2 int k,l,m,x,y,z;
3 k = tec.nextInt();
4 l = tec.nextInt();
5 m = tec.nextInt();
6 x = k+l;
7 if (x != m) {
8 y = k*l;
9 z = 0;
10 } else {
11 y = 0;
12 z = k-l;
13 }
14 if (z < 0) z = -z;
15 System.out.format("%d\n%d\n%d\n",x,y,z);
16 }
```

3. (Traza3) **Datos de entrada: 2, 5, 7, 9, -9, -7, -5, -2**

a.

```

1 public static void main (String[] args){
2 int x,y;
3 x = 0;
4 y = tec.nextInt();
5 while(! (y<0)) {
6 x+=-y;
7 y = tec.nextInt();
8 System.out.format("%d, %d",x,y);
9 }
10 }
```

b.

```

1 public static void main (String[] args){
2 int x,y,z,a;
3 x = y = z = a = 0;
4 x = tec.nextInt();
5 while(x>0) {
6 if (y < z) y = tec.nextInt();
7 else z= tec.nextInt();
8 a = a-x+y*z;
9 x = tec.nextInt();
10 System.out.format("%d, %d, %d, %d",a,x,y,z);
11 }
12 }
```

4. (Traza4) **Datos de entrada: 5, 5, 7, -5, -4, 2**

a.

```

1 public static void main (String[] args){
2 int x, y, a=0;
3 x = 0;
4 y = 99;
5 while (x >= 0) {
6 x = tec.nextInt();
7 y = tec.nextInt();
8 a = a + x*y;
9 }
10 System.out.println(a);
11 }
```

b.

```

1 public static void main (String[] args){
2 int x, y, a=0;
3 x = 0;
4 y = 99;
5 while (x >= 0 && y >= 0) {
6 x = tec.nextInt();
7 y = tec.nextInt();
8 a = a + x*y;
9 }
10 System.out.println(a);
11 }
```

c.

```

1 public static void main (String[] args){
2 int x, y, a=0;
3 x = 0;
4 y = 99;
5 while (x >= 0 && y <= 0) {
6 x = tec.nextInt();
7 y = tec.nextInt();
8 a = a + x*y;
9 }
10 System.out.println(a);
11 }
```

d.

```

1 public static void main (String[] args){
2 int x, y, a=0;
3 x = 0;
4 y = 99;
5 while (x >= 0 || y >= 0) {
6 x = tec.nextInt();
7 y = tec.nextInt();
8 a = a + x*y;
9 }
10 System.out.println(a);
11 }
```

5. (Trazo5) **Datos de entrada: 5, 5, 7, -5, -4, 2**

```

1 public static void main(String[] args) {
2 int x, y;
3 x = 2;
4 y = 3;
5 while (x + y > 0) {
6 x = tec.nextInt();
7 y = tec.nextInt();
8 x += y;
9 y = x - y;
10 System.out.format("%d, %d", x, y);
11 }
12 }
```

6. (Trazo6) **Datos de entrada: 2, 4, 7, 5, -6, -3, 6, 6**

a.

```

1 public static void main (String[] args){
2 int a,b;
3 do{
4 a = tec.nextInt();
5 b = tec.nextInt();
6 for (int i=a ; i<=b ; i++)
7 System.out.println(i);
8 } while (a!=b)
9 }
```

b.

```

1 public static void main (String[] args){
2 int a,b;
3 a=5;
4 b=5;
5 do {
6 for (int i=a ; i<=b ; i++)
7 System.out.println(i);
8 a = tec.nextInt();
9 b = tec.nextInt();
10 } while (a!=b);
11 }
```

7. (Trazo7) **Datos de entrada: 3, 3, 5, 5, -3, -7, 2, 2**

```

1 public static void main (String[] args){
2 int x,y;
3 do {
4 x = tec.nextInt();
5 b = tec.nextInt();
6 } while (x==y);
7 if (x>y) {
8 x=y;
9 y=x;
10 }
11 System.out.format("%d %d %n",x,y);
12 }
```

8. (Trazo8) **Datos de entrada:** 3, 2, 1, 4

```

1 public static void main (String[] args){
2 int a=0,b;
3 b = tec.nextInt();
4 for(int i=1;i<=b,i++) a=(a+i)*i;
5 System.out.println(a);
6 }
```

9. (Trazo9) **Datos de entrada:** No aplica

```

1 public static void main (String[] args){
2 int x,y;
3 for (x=3;x>=1;x--){
4 for(y=1;y<=x;y++) System.out.println(x);
5 System.out.println();
6 }
7 }
```

10. (Trazo10) **Datos de entrada:** No aplica

```

1 public static void main (String[] args){
2 int x,y;
3 x=0;
4 y=0;
5 for (int i=1;i<=2;i++) {
6 for (int j=1;j<=3;j++) x=(x+i)*j;
7 y+=x;
8 }
9 System.out.println("%d %d %n",x,y);
10 }
```

11. (Trazo11) **Datos de entrada:** 4, 5, 6, 7, 8, 9

```

1 public static void main (String[] args){
2 int x,y;
3 do x = tec.nextInt();
4 while (x<=5);
5 y=0;
6 for (int i=12;i>=x;i-=2) y+=(x*i);
7 System.out.println(y);
8 }
```

**10.2.7. Excepciones**

1. (Edades) Escribe un programa que solicite al usuario la edad de cinco personas y calcule la media. La edad de una persona debe ser un valor entero comprendido en el rango [0,110]. Realiza tres versiones:

- a. (Edades\_1) Si se introduce mal la edad de una persona se vuelve a pedir la edad de esa persona.

```

1 Introduce una edad entre 0 y 110 para la persona 1: 55
2 Introduce una edad entre 0 y 110 para la persona 2: 125
3 Introduce una edad entre 0 y 110 para la persona 2: -4
4 Introduce una edad entre 0 y 110 para la persona 2: 34
5 Introduce una edad entre 0 y 110 para la persona 3: 45
6 Introduce una edad entre 0 y 110 para la persona 4: 12
7 Introduce una edad entre 0 y 110 para la persona 5: 5
8 La media de las 5 edades introducidas es: 30,20
```

- a. (Edades\_2) Si se introduce mal la edad de una persona, el programa muestra un mensaje de error, no calcula la media y termina.

```

1 Introduce una edad entre 0 y 110 para la persona 1: 55
2 Introduce una edad entre 0 y 110 para la persona 2: 125
3 java.lang.Exception: La edad introducida debe estar entre 0 y 110.

```

- a. (Edades\_3) Si se introduce mal la edad de una persona, el programa vuelve a solicitar la edad de las cinco personas (comienza el proceso).
2. (PosicionLetra) Escribe los programas que se indican a continuación. Ejecuta cada programa haciendo que la entrada del usuario provoque una excepción. Anota el nombre de la excepción que se produce y cuál es la jerarquía de objetos de la que desciende:
- Programa que solicita dos números enteros (a y b) y muestra el resultado de su división (a/b).
    - El usuario introduce 0 como valor de b.
    - El usuario introduce letras cuando el programa espera números enteros.
    - El usuario introduce un número real cuando el programa espera un entero.
  - Programa que solicita al usuario su nombre y una posición dentro del nombre. Se muestra al usuario la letra del nombre cuya posición se ha indicado. Por ejemplo:

```

1 Introduce nombre: Javi
2 Introduce posición: 2
3 En la posición 2 de Javi está la letra a

```

- El usuario introduce una posición inválida.
- (PosicionLetraMain) Repite el ejercicio anterior utilizando métodos y llamándolos desde el método `main`:

  - Un método `dividir` que devuelva el cociente de dos números que recibe como parámetro
  - Un método `letraNombre` que, dados un String `nombre` y un entero `pos`, devuelva el carácter del nombre que ocupa la posición indicada. Ejecuta los programas provocando errores (como en el ejercicio anterior) y observa los mensajes que se generan.

- (DividirArgs) Escribir un programa que divida dos números que se reciben en main en `args[0]` y `args[1]`.

Ejemplo:

```

1 $ java dividir 10 5
2 10/5 es igual a 2

```

Donde 10 y 5 son `args[0]` y `args[1]` respectivamente, es decir los parámetros con que llamamos al programa `dividir`.

5. (PorqueError) Justifica por qué se produce error en el siguiente fragmento de código

```

1 try {
2 System.out.println("Introduce edad: ");
3 int edad = tec.nextInt();
4 if (edad >= 18) {
5 System.out.println("Mayor edad");
6 } else {
7 System.out.println("Menor edad");
8 }
9 System.out.println("Introduce nif");
10 String nif = tec.next();
11 int numero = Integer.parseInt(nif.substring(0, nif.length() - 1));
12 char letra = nif.charAt(nif.length() - 1);
13 System.out.println("Número: " + numero);
14 System.out.println("Letra: " + letra);
15 } catch (Exception e){
16 System.out.println("Debias introducir un número");
17 } catch (NumberFormatException e) {
18 System.out.println("El nif es incorrecto");
19 }

```

6. (SalidaPantalla) Indica qué se mostrará por pantalla cuando se ejecute esta clase y por qué:

```

1 public class Uno {
2 private static int metodo() {
3 int valor=0;
4 try {
5 valor = valor + 1;
6 valor = valor + Integer.parseInt("42") ;
7 valor = valor + 1;
8 System.out.println("Valor al final del try: " + valor);
9 } catch(NumberFormatException e) {
10 valor = valor + Integer.parseInt ("42");
11 System.out.println("Valor al final del catch: " + valor) ;
12 }
13 finally {
14 valor = valor + 1;
15 System.out.println("Valor al final de finally: " + valor) ;
16 }
17 valor = valor + 1;
18 System.out.println ("Valor antes del return: " + valor) ;
19 return valor;
20 }
21
22 public static void main(String[] args) {
23 try {
24 System.out.println (metodo());
25 } catch (Exception e) {
26 System.err.println("Excepcion en metodo()");
27 e.printStackTrace();
28 }
29 }
30 }
```

7. (SalidaPantalla2) Indica qué se mostrará por pantalla cuando se ejecute esta clase y por qué:

```

1 public class Dos {
2 private static int metodo() {
3 int valor=0;
4 try {
5 valor = valor+1;
6 valor = valor + Integer.parseInt("W");
7 valor = valor + 1;
8 System.out.println("Valor al final del try: " + valor);
9 } catch(NumberFormatException e) {
10 valor = valor + Integer.parseInt("42");
11 System.out.println("Valor al final del catch: " + valor) ;
12 }
13 finally {
14 valor = valor + 1;
15 System.out.println("Valor al final de finally: " + valor) ;
16 }
17 valor = valor + 1;
18 System.out.println ("Valor antes del return: " + valor) ;
19 return valor ;
20 }
21
22 public static void main (String[] args) {
23 try {
24 System .out.println(metodo());
25 } catch (Exception e) {
26 System.err.println("Excepcion en metodo() ");
27 e.printStackTrace();
28 }
29 }
```

8. (SalidaPantalla3) Indica qué se mostrará por pantalla cuando se ejecute esta clase y por qué:

```

1 public class Tres {
2 private static int metodo() {
3 int valor = 0;
4 try {
5 valor = valor +1;
6 valor = valor + Integer.parseInt("W");
7 valor = valor + 1;
8 System.out.println("Valor al final del try : " + valor);
9 } catch (NumberFormatException e) {
10 valor = valor + Integer.parseInt("W");
11 System.out.println("Valor al final del catch : " + valor);
12 } finally {
13 valor = valor + 1;
14 System.out.println("Valor al final de finally: " + valor);
15 }
16 valor = valor + 1;
17 System.out.println ("Valor antes del return: " + valor);
18 return valor ;
19 }
20
21 public static void main (String[] args)
22 {
23 try {
24 System.out.println(metodo ());
25 } catch (Exception e) {
26 System.err.println("Excepcion en metodo()");
27 e.printStackTrace();
28 }
29 }
30 }

```

9. (SalidaPantalla4) Indica qué se mostrará por pantalla cuando se ejecute esta clase y por qué:

```

1 import java.io.*;
2
3 public class Cuatro
4 {
5 private static int metodo() {
6 int valor = 0;
7 try {
8 valor = valor+1;
9 valor = valor + Integer.parseInt("W");
10 valor = valor + 1;
11 System.out.println("Valor al final del try : " + valor) ;
12 throw new IOException();
13 } catch (IOException e) {
14 valor = valor + Integer.parseInt("42");
15 System.out.println("Valor al final del catch : " + valor);
16 } finally {
17 valor = valor + 1;
18 System.out.println("Valor al final de finally: " + valor);
19 }
20 valor = valor + 1;
21 System.out.println ("Valor antes del return: " + valor) ;
22 return valor ;
23 }
24
25 public static void main(String[] args) {
26 try {
27 System.out.println(metodo ());
28 } catch (Exception e) {
29 System.err.println("Excepcion en metodo()");
30 e.printStackTrace();
31 }
32 }
33 }

```

10. (SalidaPantalla5) Indica qué se mostrará por pantalla cuando se ejecute esta clase:

- a. Si se ejecuta con `java Cinco casa`
- b. Si se ejecuta con `java Cinco 0`
- c. Si se ejecuta con `java Cinco 7`

```

1 public class Cinco {
2 public static void main(String args[]) {
3 try {
4 int a = Integer.parseInt(args[0]);
5 System.out.println("a = " + a);
6 int b=42/a;
7 String c = "holo";
8 char d = c.charAt(50);
9 } catch (ArithmetricException e) {
10 System.out.println("div por 0: " + e);
11 } catch (IndexOutOfBoundsException e) {
12 System.out.println("índice del String fuera de límites: " + e);
13 } finally {
14 System.out.println("Ejecución de finally");
15 }
16 }
17 }
```

11. (SalidaPantalla6) Indica cuál será la salida del siguiente programa y por qué

```

1 public class Seis {
2 public static void procA() {
3 try {
4 System.out.println("dentro del procA"); 2
5 throw new RuntimeException("demo"); 3
6 } finally {
7 System.out.println("Finally del procA"); 4
8 }
9 }
10
11 public static void procB() {
12 try {
13 System.out.println("dentro del procB"); 6
14 return; 7
15 } finally {
16 System.out.println("finally del procB"); 8
17 }
18 }
19
20 public static void main(String args[]) {
21 try {
22 procA(); 1
23 } catch(Exception e) {
24 procB(); 5
25 }
26 }
27 }
```

12. (SalidaPantalla7) Indica cuál será la salida del siguiente programa y por qué

```

1 public class Siete {
2 public static void metodo() {
3 try {
4 throw new NullPointerException("demo"); 2
5 } catch (NullPointerException e) {
6 System.out.println("capturada en método"); 3
7 throw e; 4
8 }
9 }
10
11 public static void main (String args[]) {
12 try {
13 metodo(); 1
14 } catch(NullPointerException e) {
15 System.out.println("capturada en main " + e); 5
16 }
17 }
18 }
```

13. (DivisionPorCero) Crea un programa que intente dividir dos números enteros ingresados por el usuario y maneja la excepción de división por cero. [Aquí](#) tienes la explicación de porqué la división entre 0 no provoca excepciones para `double` y `float`.

```

1 Introduce el dividendo (entero): 85
2 Introduce el divisor (entero): 4
3 El resultado de 85/4 es: 21
4
5 Introduce el dividendo (entero): 5
6 Introduce el divisor (entero): 0
7 Error: División por cero no permitida.
```

14. (CalculadoraExcepcion) Crea una clase `calculadoraExcepcion` con un método `dividir` que acepte dos números con decimales como argumentos, devuelva su resultado o lance una excepción personalizada (`DivisionPorCeroException`) si el divisor es cero. Captura la excepción en el método `main` y muestra un mensaje de error.

```

1 Introduce el dividendo: 15,5
2 Introduce el divisor: 4
3 El resultado de 15,5/4,0 es: 3,875
4
5 Introduce el dividendo: 3,5
6 Introduce el divisor: 0
7 Error: División por cero no permitida.

```

15. (EntradaNoNumerica) Escribe un programa que lea un número entero desde el teclado. Si el usuario ingresa algo que no es un número entero, maneja la excepción y muestra un mensaje de error.
16. (RangoNumerico) Escribe un programa que solicite al usuario ingresar un número entre 1 y 100. Si el número está fuera de ese rango, lanza una excepción personalizada (`NúmeroFueraDeRangoException`) y muestra un mensaje de error.

```

1 Introduce un número entre 1 y 100: 5
2 Número válido: 5
3
4 Introduce un número entre 1 y 100: 500
5 Error: Número fuera de rango: 500

```

17. (NumeroNegativo) Crea un método que reciba dos números como argumentos y lance una excepción personalizada si uno de los números es negativo. Captura esa excepción en el método principal y muestra un mensaje de error.

```

1 Introduce el primer número: 5
2 Introduce el segundo número: 4
3 Ambos números son positivos.
4
5 Introduce el primer número: -4
6 Introduce el segundo número: 5
7 Error: Uno de los números es negativo.

```

18. (LongitudCadena) Diseña un programa que lea una cadena de caracteres desde el teclado y, si la longitud de la cadena es mayor de 10 caracteres, lance una excepción personalizada. Captura esa excepción y muestra un mensaje de error.
19. (TemperaturaInvalida) Implementa una clase `ConversorTemperatura` que tenga un método para convertir grados Celsius a Fahrenheit. Si el valor en grados Celsius es inferior a -273.15, lanza una excepción personalizada. Captura la excepción y muestra un mensaje de error en el método principal.
20. (EdadInvalida) Diseña una clase `ValidadorEdad` que tenga un método para validar si una persona tiene una edad válida (por ejemplo, entre 0 y 120 años). Si la edad no es válida, lanza una excepción personalizada y muestra un mensaje de error en el método principal.

### 10.2.8. Aserciones

1. (Aserciones1) A partir del siguiente fragmento de código, añade una linea debajo del comentario de la linea 4 que haga lo que se solicita:

```

1 class Main {
2 public static void main(String args[]) {
3 String[] finde = {"viernes", "sabado", "domingo"};
4 //Añade una aserción que compruebe que solo hay dos días en el fin de semana.
5
6 System.out.println("Solo hay " + weekends.length + " días en el fin de semana");
7 }
8 }

```

2. (Aserciones2) Escribe un método llamado `validarEdad(int edad)` que acepte como parámetro la edad de una persona. Usa una aserción para verificar que la edad sea un valor positivo y menor que 150. Si la edad es negativa o extremadamente alta, la aserción debería fallar. Dispones del siguiente `main`:

```

1 public static void main(String[] args) {
2 // Caso que debe pasar la aserción
3 validarEdad(25);
4 // Caso que debe fallar la aserción (edad negativa)
5 validarEdad(-5);
6 }

```

3. (Aserciones3) Crea un método llamado `esPar(int numero)` que devuelva `true` si el número es par y `false` en caso contrario. Luego, escribe una aserción para verificar que el resultado es `true` cuando el número proporcionado es efectivamente par. Dispones del siguiente `main`:

```

1 public static void main(String[] args) {
2 // Caso que debe devolver true
3 assert esPar(4) : "El número 4 debería ser par";
4 // Caso que debe devolver false
5 assert !esPar(3) : "El número 3 no debería ser par";
6 // Hace saltar la excepción si las aserciones están activadas y alguna falla
7 assert esPar(5) : "Hará saltar la excepción";
8 }

```

4. (Aserciones4) Implementa un método llamado `dentroDeRango(int numero, int min, int max)` que devuelva `true` si el número está en el rango `[min, max]` y `false` en caso contrario. Usa aserciones para probar que el método devuelve `true` para un número dentro del rango y `false` para uno fuera.

```

1 // Ejemplo de uso:
2 assert dentroDeRango(5, 1, 10) : "El número 5 debería estar en el rango [1, 10]";
3 assert !dentroDeRango(15, 1, 10) : "El número 15 no debería estar en el rango [1, 10]";

```

## 10.3. Actividades

1. (TransformarBucle) Transforma el siguiente bucle for en un bucle while:

```

1 for (i=5; i<15; i++) {
2 System.out.println(i);
3 }

```

2. (NumerosPares) Escribe un método (`nPares`) que recibe un número `n` y que devuelve un `String` con los `n` primeros números pares. Escribe un método `main` que pida un número al usuario y use el método `nPares` para mostrarlos en pantalla.

```

1 Introduce un número entero positivo: 15
2 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30

```

3. (Rango) Escribe un método (`rango`) que recibe dos enteros y que devuelve un `String` que comience con el primero y termine en el segundo. Se asume que el primero será menor que el segundo. Escribe un método `main` que pida los dos números al usuario y muestre el rango de números usando el método `rango`:

```

1 Introduce el primer número del rango: 200
2 Introduce el segundo número del rango: 250
3 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 2
38 239 240 241 242 243 244 245 246 247 248 249 250

```

4. (TablasMultiplicar) Escribe un método (`tabla`) que recibe un número entero positivo y devuelve un `String` con la tabla de multiplicar de dicho número con el siguiente formato:

```

1 Tabla del 2
2 2 x 1 = 2
3 2 x 2 = 4
4 ...
5 2 x 10 = 20

```

Escribe un método `main`, que usando el método `tabla` imprima por pantalla las tablas del 1 al 10.

5. (SinMultiplos5) Programa que muestre los números del 1 al 100 sin mostrar los múltiplos de 5.

```

1 ...
2 28
3 29
4 31
5 32
6 33
7 34
8 36
9 37
10 38
11 39
12 41
13 42
14 43
15 ...

```

### Acción

Para pasar satisfactoriamente los tests, en lugar de usar `System.out.println`, debes usar `System.out.print` con un `\n` al final de la cadena a imprimir.

6. (CuadradoHastaNegativo) Leer un número y mostrar su cuadrado, repetir el proceso hasta que se introduzca un número negativo.
7. (PositivoNegativo) Leer un número e indicar si es positivo o negativo. El proceso se repetirá hasta que se introduzca un 0.
8. (ParImpar) Leer números hasta que se introduzca un 0. Para cada uno indicar si es par o impar.
9. (ContarNumeros) Pedir números hasta que se teclee uno negativo, y mostrar cuántos números se han introducido.
10. (AdivinarNúmero) Realizar un juego para adivinar un número  $x$ . Para ello pedir un número  $N$ , y luego ir pidiendo números indicando "mayor" o "menor" según sea mayor o menor con respecto a  $x$ . El proceso termina cuando el usuario acierta.

```

1 Voy a pensar un número menor que 1000 para que lo adivines...
2
3 Introduce que número crees que he pensado: 500
4 El número que he pensado es menor que 500
5 Introduce que número crees que he pensado: 250
6 El número que he pensado es menor que 250
7 Introduce que número crees que he pensado: 125
8 El número que he pensado es mayor que 125
9 Introduce que número crees que he pensado: 187
10 El número que he pensado es mayor que 187
11 Introduce que número crees que he pensado: 210
12 El número que he pensado es mayor que 210
13 Introduce que número crees que he pensado: 230
14 El número que he pensado es mayor que 230
15 Introduce que número crees que he pensado: 240
16 El número que he pensado es mayor que 240
17 Introduce que número crees que he pensado: 245
18 Enhorabuena has acertado, el número que había pensado es el 245

```

11. (SumaNumeros) Pedir números hasta que se teclee un 0, mostrar la suma de todos los números introducidos.
12. (MediaNumeros) Pedir números hasta que se introduzca uno negativo, y calcular la media.
13. (NumerosHastaN) Pedir un número  $N$ , y mostrar todos los números del 1 al  $N$ .
14. (De100a0) Escribir todos los números del 100 al 0 de 7 en 7.
15. (Suma15Numeros) Pedir 15 números y escribir la suma total.
16. (ProductoImpares) Diseñar un programa que muestre el producto de los 10 primeros números impares.
17. (Factorial) Pedir un número y calcular su factorial (el factorial se representa con el simbolo  $!$ ).

Aquí tienes el factorial de los 5 primeros números enteros:

```

1 1! = 1
2 2! = 2 * 1 = 2
3 3! = 3 * 2 * 1 = 6
4 4! = 4 * 3 * 2 * 1 = 24
5 5! = 5 * 4 * 3 * 2 * 1 = 120

```

Ejemplo de ejecución del programa:

```

1 Dime el número para calcular su factorial: 6
2 6! = 6 * 5 * 4 * 3 * 2 * 1 = 720

```

18. (MediaPosNeg) Pedir 10 números. Mostrar la media de los números positivos, la media de los números negativos y la cantidad de ceros.
19. (Sueldos1000) Pedir 10 sueldos. Mostrar su suma y cuantos hay mayores de 1000€.
20. (AlumnosEdadAltura) Dadas las edades y alturas de 5 alumnos, mostrar la edad y la estatura media, la cantidad de alumnos mayores de 18 años, y la cantidad de alumnos que miden más de 1.75.
21. (TablaMultiplicar) Pide un número (que debe estar entre 0 y 10) y mostrar la tabla de multiplicar de dicho número.
22. (AprobadosSuspensos) Dadas 6 notas, escribir la cantidad de alumnos aprobados y suspensos.
23. (SueldoMaximo) Pedir un número  $N$ , introducir  $N$  sueldos, y mostrar el sueldo máximo.
24. (HayNegativo) Pedir 10 números, y mostrar al final si se ha introducido alguno negativo.
25. (HaySuspensos) Pedir 5 calificaciones de alumnos y decir al final si hay algún suspensos.
26. (Multiplo3) Pedir 5 números e indicar si alguno es múltiplo de 3.

27. (SaludoHorario) Realiza un programa que pida una hora por teclado y que muestre luego buenos días, buenas tardes o buenas noches según la hora. Se utilizarán los tramos de 6 a 12, de 13 a 20 y de 21 a 5. respectivamente. Sólo se tienen en cuenta las horas, los minutos no se deben introducir por teclado.
28. (DiaSemana) Escribe un programa en que dado un número del 1 a 7 escriba el correspondiente nombre del día de la semana.
29. (SalarioHorasExtras) Escribe un programa que calcule el salario semanal de un trabajador teniendo en cuenta que las horas ordinarias (40 primeras horas de trabajo) se pagan a 12 euros la hora. A partir de la hora 41, se pagan a 16 euros la hora.
30. (MediaTresNotas) Realiza un programa que calcule la media de tres notas.
31. (BoletinNotas) Amplía el programa anterior para que diga la nota del boletín (insuficiente, suficiente, bien, notable o sobresaliente).
32. (Horoscopo) Escribe un programa que nos diga el horóscopo a partir del día y el mes de nacimiento.
33. (Cuestionario) Realiza un minicuestionario con 4 preguntas tipo test sobre las asignaturas que se imparten en el curso. Cada pregunta acertada sumará un punto. El programa mostrará al final la calificación obtenida.
34. (NotaProgramacion) Calcula la nota de un trimestre de la asignatura Programación. El programa pedirá las dos notas que ha sacado el alumno en los dos primeros controles. Si la media de los dos controles da un número mayor o igual a 5, el alumno está aprobado y se mostrará la media. En caso de que la media sea un número menor que 5, el alumno habrá tenido que hacer el examen de recuperación que se califica como apto o no apto, por tanto se debe preguntar al usuario ¿Cuál ha sido el resultado de la recuperación? (apto/no apto). Si el resultado de la recuperación es apto, la nota será un 5; en caso contrario, la nota será 1.

Ejemplo 1:

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

Ejemplo 2:

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

Ejemplo 3:

```
1 Nota del primer control: 6 Nota del segundo control: 3
2 ¿Cuál ha sido el resultado de la recuperación? (apto/no apto): no apto
3 Tu nota de Programación es 1
```

35. (Multiplos5For) Muestra los números múltiplos de 5 entre el 0 y el 100 utilizando un bucle `for`.
36. (Multiplos5While) Muestra los números múltiplos de 5 entre el 0 y el 100 utilizando un bucle `while`.
37. (Multiplos5DoWhile) Muestra los números múltiplos de 5 entre el 0 y el 100 utilizando un bucle `do while`.
38. (ContarAtrasFor) Muestra los números del 320 al 160, contando de 20 en 20 hacia atrás utilizando un bucle `for`.
39. (ContarAtrasWhile) Muestra los números del 320 al 160, contando de 20 en 20 hacia atrás utilizando un bucle `while`.
40. (ContarAtrasDoWhile) Muestra los números del 320 al 160, contando de 20 en 20 utilizando un bucle `do-while`.
41. (CajaFuerte) Realiza el control de acceso a una caja fuerte. La combinación será un número de 4 cifras. El programa nos pedirá la combinación para abrirla. Si no acertamos, se nos mostrará el mensaje "**Esa no es la combinación correcta**" y si acertamos se nos dirá "**La caja se ha abierto correctamente**". Tendremos cuatro oportunidades para abrir la caja fuerte, si lo sobrepasamos nos dirá "**Has sobrepasado el número de intentos permitido**".

### Acción

Para pasar satisfactoriamente los tests, la combinación de apertura debe estar configurada en "1234"

42. (CuadradoCubo) Escribe un programa que muestre en tres columnas, el cuadrado y el cubo de los 5 primeros números enteros a partir de uno que se introduce por teclado.
43. (Potencia) Escribe un método (`potencia`) que reciba una base y un exponente (enteros positivos) y que calcule la potencia (sin usar `Math`). Escribe un método `main` para pedir los datos al usuario y usando el método `potencia` mostrar el resultado.

```
1 Vamos a calcular una potencia.
2 Introduce la BASE: 6
3 Introduce el EXPONENTE: 5
4 El resultado de elevar la BASE: 6 al EXPONENTE: 5 resulta en: 7776
```

44. (Suma100Siguentes) Realiza un programa que sume los 100 números siguientes a un número entero y positivo introducido por teclado. Se debe comprobar que el dato introducido es correcto (que es un número positivo).
45. (NumerosEntre7) Escribe un programa que imprima por pantalla los números enteros comprendidos entre dos números introducidos por teclado y validados como distintos, el programa debe empezar por el menor de los enteros introducidos e ir incrementando de 7 en 7.

```

1 Introduce dos números DIFERENTES!
2 Introduce el primer número: 8
3 Introduce el segundo número: 8
4 Los números no son DIFERENTES!
5
6 Introduce dos números DIFERENTES!
7 Introduce el primer número: 6
8 Introduce el segundo número: 50
9 6
10 13
11 20
12 27
13 34
14 41
15 48

```

46. (EstadisticasNumeros) Realiza un programa que vaya pidiendo números hasta que se introduzca un numero negativo y nos diga cuantos números se han introducido, la media de los impares y el mayor de los pares. El número negativo sólo se utiliza para indicar el final de la introducción de datos pero no se incluye en el cómputo.
47. (SumaHasta10000) Escribe un programa que permita ir introduciendo una serie indeterminada de números mientras su suma no supere el valor 10000. Cuando esto último ocurra, se debe mostrar el total acumulado, el contador de los números introducidos y la media.
48. (Multiplos3) Escribe un programa que muestre, cuente y sume los múltiplos de 3 que hay entre 1 y un número leído por teclado.
49. (PrecioFinal) Escribe un programa que calcule el precio final de un producto según su base imponible (precio antes de impuestos), el tipo de IVA aplicado (general, reducido o superreducido) y el código promocional. Los tipos de IVA general, reducido y superreducido son del 21%, 10% y 4% respectivamente. Los códigos promocionales pueden ser `nopro`, `mitad`, `menos5` o `5porc` que significan respectivamente que no se aplica promoción, el precio se reduce a la mitad, se descuentan 5 euros o se descuenta el 5%.

Ejemplo:

```

1 Introduzca la base imponible: 25
2 Introduzca el tipo de IVA (general, reducido o superreducido): reducido
3 Introduzca el código promocional (nopro, mitad, menos5 o 5porc): mitad
4 Base imponible 25.00
5 Cód. promo. (mitad): -12.50
6 IVA (10%) 1.25
7 Precio con IVA 13.75
8 TOTAL 13.75

```

50. (AnioBisiesto) Pedir un año e indicar si es bisiesto, teniendo en cuenta que son bisiestos todos los años divisibles por 4, excluyendo los que sean divisibles por 100, pero no los que sean divisibles por 400.

En pseudocódigo se calcularía así:

```

1 SI ((año divisible por 4) Y ((año no divisible por 100) O (año divisible por 400)))ENTONCES
2 es bisiesto
3 SINO
4 no es bisiesto
5 FIN_SI

```

51. (NumeroALetras) Pedir un número de 20 a 99 y mostrarlo escrito. Por ejemplo, para 56 mostrar: cincuenta y seis.
52. (VehiculoIVA) Introducir datos de un vehículo (marca, modelo y precio). Devolver el precio con IVA del vehículo. Controlar con Excepciones que el precio del vehículo introducido son números y que el cálculo de Precio Final con IVA no devuelva error.
53. (NotaMediaAlumnos) Introducir códigos de alumnos, nombre y nota hasta que se introduzca un código de alumno negativo. Devolver la nota media de los alumnos la clase. Controlar con Excepciones que las notas introducidas son números y que si no se introducen alumnos el cálculo de la media no devuelva error.
54. (ImporteFinal) Crear una función o método llamado `impFinal`, que calcule el importe final de una compra. Los parámetros que se le pasarán a la función son el `precio` del producto, las `cantidad` de unidades compradas, el `porcentaje` de `iva` y el `porcentaje` de `descuento`. El método principal debe pedir por teclado el `precio` del producto, las `unidades` adquiridas, el `porcentaje` de `IVA` y el `porcentaje` de `descuento` y devolver el `Importe final` de la Factura.
55. (CapacidadDisco) Crear una función que calcule la capacidad de un disco. La capacidad se calcula multiplicando los Cabezales o pistas del disco por los Cilindros por los Sectores por Tamaño de Sector. El método principal debe pedir por teclado los Cabezales o Pistas del disco, los Cilindros, Sectores y Tamaño de Sector y devolver la Capacidad del disco en Gigabytes.

### Ejemplo

Por ejemplo: Calcular la capacidad de un disco teniendo en cuenta que dispone de 10 Cabezales o Pistas, 65535 Cilindros, 1024 Sectores/pista y un Tamaño de 512 bytes/sector:

Capacidad del disco =  $10 * 65535 * 1024 * 512 = 343597383680$  bytes

$343597383680$  bytes / 1024 / 1024 / 1024 = 320 Gbytes

56. (MayorDeTres) Función que devuelva el mayor de tres números. El método principal debe pedir por teclado los tres números introducidos por el teclado. La función debe recibir como parámetros los tres números y devolver el mayor.

4 de diciembre de 2025

## 11. 4.3 Talleres

---

### 11.1. Taller UD03\_01: GitHub Classroom

#### **11.1.1. Unirnos a GitHub Classroom**

Aceptamos el *Assignement* (la tarea/ejercicio) a partir del link del profesor, en este caso: <https://classroom.github.com/a/9bM3kTmM>

#### **11.1.2. Tarea**

Debes enviar tus soluciones a GitHub Classroom y superar al menos la mitad de los tests, cuantos más tests superados, mejor nota tendrás en la tarea.

 21 de noviembre de 2025

## 11.2. Taller UD03\_02: Acepta el reto

---

### **11.2.1. Inscríbete en Acepta el reto**

Inscríbete en la web [www.aceptaelreto.com](http://www.aceptaelreto.com) seleccionando nuestro instituto.

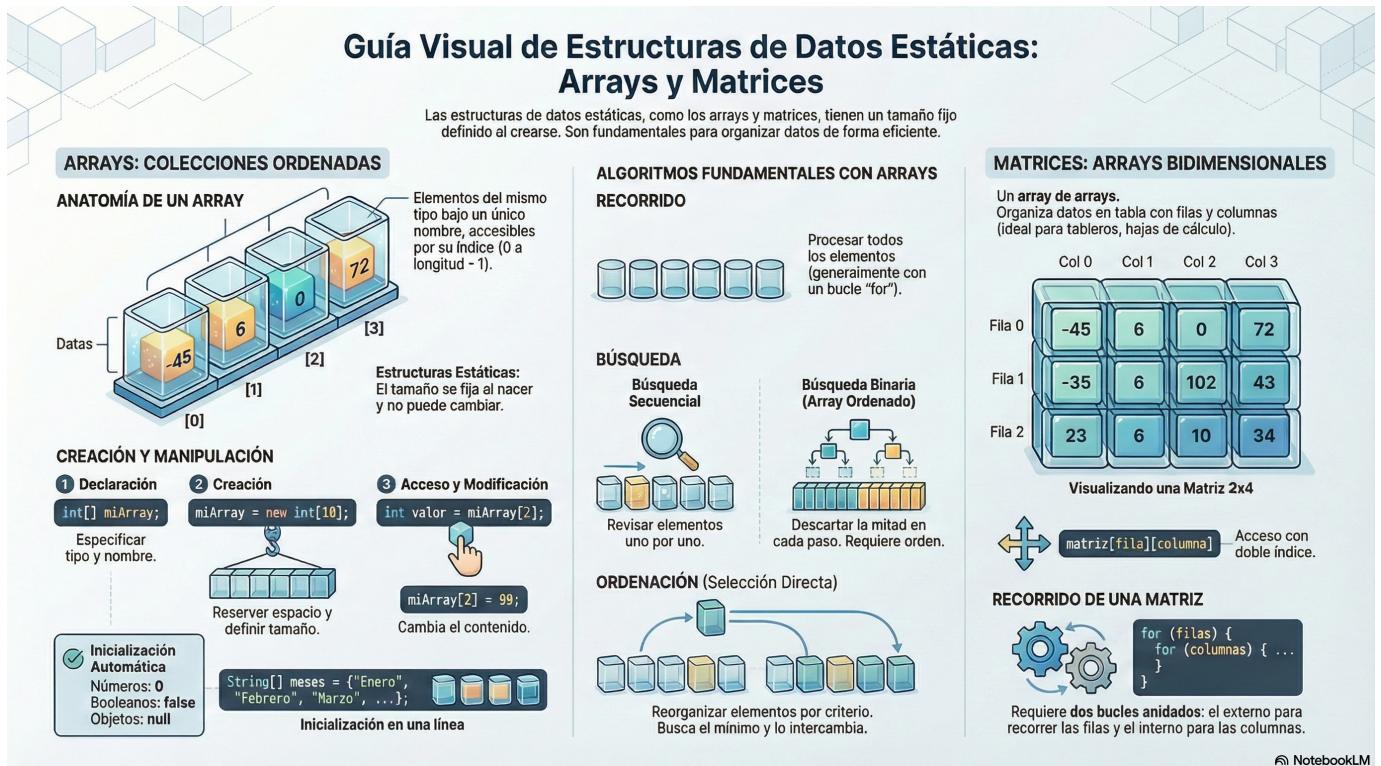
### **11.2.2. Tarea**

Haz una captura de pantalla de tu perfil una vez hecho login con tu usuario y envia la captura en un pdf al profesor.

 21 de noviembre de 2025

## 5. UD04

### 12. 5.1 Estructuras de datos: Arrays y matrices. Recursividad.



#### 12.1. Introducción

A menudo, para resolver problemas de programación, no basta con disponer de sentencias condicionales o iterativas como las que hemos visto (`if`, `switch`, `while`, `for`, ...).

También es necesario disponer de herramientas para organizar la información de forma adecuada: **las estructuras de datos**.

Los arrays son una estructura de datos fundamental, que está disponible en la mayoría de lenguajes de programación y que nos permitirá resolver problemas que, sin ellos, resultarían difíciles o tediosos de solucionar.

Imaginemos, por ejemplo, que queremos leer los datos de pluviosidad de cada uno de los 31 días de un mes. Posteriormente se desea mostrar la pluviosidad media del mes y en cuántos días las lluvias superaron la media.

Con las herramientas de que disponemos hasta ahora, nos veríamos obligados a declarar **31 variables double**, una para cada día, y a elaborar un largo programa que leyera los datos y contara cuáles superan la media. Con el uso de arrays, problemas como este tienen una solución fácil y corta.

#### 12.2. Arrays

Un array es una colección de elementos del mismo tipo, que tienen un nombre o identificador común.

Se puede acceder a cada componente del array de forma individual para consultar o modificar su valor. El acceso a los componentes se realiza mediante un subíndice, que viene dado por la posición que ocupa el elemento dentro del array.

En la siguiente figura se muestra un array `c` de enteros:

Array `c`

### Importante

El primer subíndice de un array es el cero. El último subíndice es la longitud del array menos uno.

El número de componentes de un array se establece inicialmente al crearlo y no es posible cambiarlo de tamaño. Es por esto que reciben el nombre de estructuras de datos estáticas.

#### 12.2.1. Declaración y creación

Para poder utilizar un array hay que declararlo:

```
1 tipo nombreVariable[];
```

O

```
1 tipo[] nombreVariable;
```

En la declaración se establece el nombre de la variable y el tipo de los componentes. Por ejemplo:

```
1 double lluvia1[]; // lluvia1 es un array de double
2 double[] lluvia2; // lluvia2 es un array de double <== Esta es la declaración recomendada, el [] siempre acompañando al tipo.
```

En la declaración anterior no se ha establecido el número de componentes. El número de componentes se indica en la creación, que se hace utilizando el operador `new`:

```
1 lluvia1 = new double[31];
```

Con esta instrucción se establece que el número de elementos del array `lluvia` son 31, reservando con ello el compilador espacio consecutivo para 31 componentes individuales de tipo `double`.

Las dos instrucciones anteriores se pueden unir en una sola:

```
1 // tipo[] nombreVariable = new tipo[numElementos];
2 double[] lluvia2 = new double[31];
```

El valor mediante el cual se define el número de elementos del array tiene que ser una expresión entera, pero no tiene por qué ser un literal como en el ejemplo anterior. El tamaño de un array se puede establecer durante la ejecución, como en el siguiente ejemplo:

```
1 // usamos un array para almacenar las edades de un grupo de personas
2 // la variable numPersonas contiene el número de personas del grupo
3 // y se asigna en tiempo de ejecución
4 Scanner teclado = new Scanner(System.in);
5 System.out.print("Introduce cuantos elementos debe tener el array edad[]:");
6 int numPersonas = teclado.nextInt();
7 int[] edad = new int[numPersonas];
```

#### 12.2.2. Acceso a los componentes

Como ya hemos dicho, el acceso a los componentes del array se realiza mediante subíndices. La sintaxis para referirse a un componente del array es la siguiente:

```
1 nombreVariable[subindice]
```

Tras declarar el array `lluvia`, se dispone de 31 componentes de tipo `double` numeradas desde la 0 a la 30 y accesibles mediante la notación: `lluvia[0]` (componente primera), `lluvia[1]` (componente segunda) y así sucesivamente hasta la última componente: `lluvia[30]`.

Con cada una de las componentes del array de `double` `lluvia` es posible efectuar todas las operaciones que podrían realizarse con variables individuales de tipo `double`, por ejemplo, dadas las declaraciones anteriores, las siguientes instrucciones serían válidas:

```

1 int[] edad = new int[3];
2 System.out.print("Introduce el dato para el componente 0: ");
3 edad[0] = teclado.nextInt(); //25
4 System.out.println("El componente [0] vale " + edad[0]);
5 edad[1] = edad[0] + 1;
6 edad[2] = edad[0] + edad[1];
7 edad[2]++;
8 System.out.println("El componente [1] vale " + edad[1]); //26
9 System.out.println("El componente [2] vale " + edad[2]); //52

```

Además, hay que tener en cuenta que el subíndice ha de ser una expresión entera, por lo que también son válidas expresiones como las siguientes:

```

1 int i;
2 ...
3 edad[i] = edad[i + 1];
4 edad[i + 2] = edad[i];

```

### 12.2.3. Inicialización

Cuando creamos un array, Java inicializa automáticamente sus componentes:

- Con 0 cuando los componentes son de tipo numérico.
- Con false cuando los componentes son `boolean`.
- Con el carácter de ASCII 0, cuando los componentes son `char`.
- Con `null` cuando son objetos (`Strings`, etc)

Aun así, es probable que estos no sean los valores con los que queremos inicializar el array. Tenemos entonces dos posibilidades:

- Acceder individualmente a los componentes del array para darles valor:

```

1 int edad2[] = new int[10];
2 edad2[0] = 25;
3 edad2[1] = 10;
4 ...
5 edad2[9] = 12;

```

- O inicializar el array en la declaración de la siguiente forma:

```
1 int edad3[] = {25, 10, 23, 34, 65, 23, 1, 67, 54, 12};
```

#### Más información

Es decir, enumerando los valores con los que se quiere inicializar cada componente, encerrados entre llaves. De hacerlo así, no hay que crear el array con el operador `new`. Java crea el array con tantos componentes como valores hemos puesto entre llaves. Además no es necesario indicar el número de elementos del mismo.

### 12.2.4. Un ejemplo práctico

Ya hemos resuelto en temas anteriores el problema de devolver el nombre de un mes dado su número.

Vamos a resolverlo ahora ayudándonos de arrays:

```

1 public static String nombreMes(int mes){
2 String[] nombre = {"enero", "febrero", "marzo", "abril",
3 "mayo", "junio", "julio", "agosto",
4 "septiembre", "octubre", "noviembre", "diciembre"};
5 return nombre[mes-1];
6 }

```

El método define un array de `String` que se inicializa con los nombres de los doce meses. La primera componente del array (`nombre[0]`) se deja vacía, de forma que enero quede almacenado en `nombre[1]`. Devolver el nombre del mes indicado se reduce a devolver el componente del array cuyo número indica el parámetro mes: `nombre[mes]`

### 12.2.5. Arrays como parámetros. Paso de parámetros por referencia.

Hasta el momento sólo se ha considerado el paso de parámetros por valor; de manera que cualquier cambio que el método realice sobre los parámetros formales no modifica el valor que tiene el parámetro real con el que se llama al método. En java, todos los parámetros de tipo simple (`byte`, `short`, `int`, ...) se pasan por valor.

#### Importante

Por el contrario, los arrays no son variables de tipo primitivo, y como cualquier otro objeto, se pasa siempre por referencia.

En el paso de parámetros por referencia lo que se pasa en realidad al método es la dirección de la variable u objeto. Es por esto que el papel del parámetro formal es el de ser una referencia al parámetro real; la llamada al método no provoca la creación de una nueva variable. De esta forma, las modificaciones que el método pueda realizar sobre estos parámetros se realizan efectivamente sobre los parámetros reales. En este caso, ambos parámetros (formal y real) se pueden considerar como la misma variable con dos nombres, uno en el método llamante y otro en el llamado o invocado, pero hacen referencia a la misma posición de memoria.

En el siguiente ejemplo, la variable `a`, de tipo primitivo, no cambia de valor tras la llamada al método. Sin embargo la variable `v`, array de enteros, si se ve afectada por los cambios que se han realizado sobre ella en el método:

```

1 public static void main(String[] args){
2 int a = 1;
3 int[] v = {1,1,1};
4 metodo(v,a); //Pasar un array como parámetro
5 System.out.println(a); // Muestra 1
6 System.out.println(v[0]); // Muestra 2
7 }
8
9 public static void metodo(int[] x, int y){ //recibir un array como parámetro
10 x[0]++;
11 y++;
12 }
```

#### Importante

Como podemos observar, para pasar un array a un método, simplemente usamos el nombre de la variable en la llamada. En la cabecera del método, sin embargo, tenemos que utilizar los corchetes `[]` para indicar que el parámetro es un array.

### 12.2.6. El atributo `length`

Todas las variables de tipo array tienen un atributo `length` que permite consultar el número de componentes del array. Su uso se realiza posponiendo `.length` al nombre de la variable:

```

1 double[] estatura = new double[25];
2 ...
3 System.out.println(estatura.length); // Mostrará por pantalla: 25
```

### 12.2.7. `String[] args` en el `main`

El método `main` puede recibir argumentos desde la línea de comandos. Para ello, el método `main` recibe un parámetro (`String args[]`). Vemos que se trata de un array de `Strings`. El uso del atributo `length` nos permite comprobar si se ha llamado al programa de forma correcta o no. Veamos un ejemplo para saber si es Navidad. Se habrá llamado correctamente si el array `args` contiene dos componentes (día, mes):

```

1 public class EsNavidad {
2 public static void main(String[] args) {
3 if (args.length != 2) {
4 System.out.println("ERROR:");
5 System.out.println("Llame al programa de la siguiente forma:");
6 System.out.println("java EsNavidad dia mes");
7 } else {
8 // args[0] es el dia
9 // args[1] es el mes
10 if ((Integer.valueOf(args[0]) == 25) && (Integer.valueOf(args[1]) == 12)) {
11 System.out.println("ES NAVIDAD!");
12 } else {
13 System.out.println("No es navidad.");
14 }
15 }
16 }
17 }
```

## 12.3. Problemas de recorrido, búsqueda y ordenación

Muchos de los problemas que se plantean cuando se utilizan arrays pueden clasificarse en tres grandes grupos de problemas genéricos: los que conllevan el recorrido de un array, los que suponen la búsqueda de un elemento que cumpla cierta característica dentro del array, y los que implican la ordenación de los elementos del array.

La importancia de este tipo de problemas proviene de que surgen, no sólo en el ámbito de los arrays, sino también en muchas otras organizaciones de datos de uso frecuente (como las listas, los ficheros, etc.). Las estrategias básicas de resolución que se verán a continuación son también extrapolables a esos otros ámbitos.

### 12.3.1. Problemas de recorrido

Se clasifican como problemas de recorrido aquellos que para su resolución exigen algún tratamiento de todos los elementos del array. El orden para el tratamiento de estos elementos puede organizarse de muchas maneras: ascendente, descendente, ascendente y descendente de forma simultánea, etc.

En el siguiente ejemplo se muestra un método en java para devolver, a partir de un array que contiene la pluviosidad de cada uno de los días de un mes, la pluviosidad media de dicho mes. Para ello se recorren ascendenteamente los componentes del array para ir sumándolos:

```

1 public static double pluviosidadMediaAscendente(double[] lluvia){
2 double suma = 0;
3 //Recorremos el array ascendente
4 for (int i = 0; i<lluvia.length; i++){
5 suma += lluvia[i];
6 }
7 double media = suma / lluvia.length;
8 return media;
9 }
```

La forma de recorrer el array ascendente es, como vemos, utilizar una variable entera (`i` en nuestro caso) que actúa como subíndice del array. Éste subíndice va tomando los valores `0, 1, ..., lluvia.length-1` en el seno de un bucle, de manera que se accede a todos los componentes del array para sumarlos.

El mismo problema resuelto con un recorrido descendente sería como sigue:

```

1 public static double pluviosidadMediaDescendente(double[] lluvia){
2 double suma = 0;
3 //Recorremos el array descendente
4 for (int i = lluvia.length-1; i>=0; i--){
5 suma += lluvia[i];
6 }
7 double media = suma / lluvia.length;
8 return media;
9 }
```

También realizamos un recorrido para obtener la pluviosidad máxima del mes (la cantidad de lluvia más grande caída en un día), es decir, el elemento más grande del array:

```

1 public static double pluviosidadMaxima(double[] lluvia){
2 // Suponemos que la pluviosidad máxima se produjo el primer día
3 double max = lluvia[0];
4 // Recorremos el array desde la posición 1, comprobando si hay una pluviosidad mayor
5 for (int i = 1; i < lluvia.length; i++) {
6 if(lluvia[i] > max){
7 max = lluvia[i];
8 }
9 }
10 return max;
}

```

#### 12.3.1.1. BUCLE FOR EACH (FOR-LOOP)

En el tema anterior vimos algún tipo de bucles que explicaríamos cuando los pudiésemos utilizar, en este grupo están los bucles for each o for-loops. Aquí tenemos un ejemplo de recorrido de un array con la sintaxis que ya conocemos:

```

1 int[] array = { 1, 2, 3, 4, 5, 6, 7, 8 };
2 for (int i = 0; i < array.length; i++) {
3 System.out.print(array[i] + " ");
4 }

```

el anterior fragmento genera la siguiente salida:

```
1 1 2 3 4 5 6 7 8
```

Este mismo código se puede escribir de la siguiente manera:

```

1 int[] array = { 1, 2, 3, 4, 5, 6, 7, 8 };
2 for (int i : array) { // Mentalmente podemos traducir por:
3 // "para cada entero "i" que encontraremos en el array"
4 System.out.print(i + " ");
5 }

```

la salida seguirá siendo la misma:

```
1 1 2 3 4 5 6 7 8
```



Cuidado!

Con el segundo método no tenemos acceso a la posición o índice del array, este método no serviría para métodos en los que necesitamos conocer la posición o utilizarla de alguna manera. Traducimos el método de `pluviosidadMedia` con un bucle `loop`:

```

1 public static double pluviosidadMediaLoop(double[] lluvia){
2 double suma = 0;
3 // Recorremos el array con el loop
4 for (int i : lluvia){
5 suma += i;
6 }
7 double media = suma / lluvia.length;
8 return media;
9 }

```

#### 12.3.2. Problemas de búsqueda

Se denominan problemas de búsqueda a aquellos que, de alguna manera, implican determinar si existe algún elemento del array que cumpla una propiedad dada. Con respecto a los problemas de recorrido presentan la diferencia de que no es siempre necesario tratar todos los elementos del array: el elemento buscado puede encontrarse inmediatamente, encontrarse tras haber recorrido todo el array, o incluso no encontrarse.

##### 12.3.2.1. BÚSQUEDA ASCENDENTE

Consideremos, por ejemplo, el problema de encontrar cuál fue el primer día del mes en que no llovió nada, es decir, el primer elemento del array con valor cero:

```

1 //Devolveremos el subíndice del primer componente del array cuyo valor es cero.
2 // Si no hay ningún día sin lluvias devolveremos -1
3 public static int primerDiaSinLluvia1(double[] lluvia){
4 int i=0 ;
5 boolean encontrado = false ;
6 while (i<lluvia.length && !encontrado){
7 if (lluvia[i] == 0) encontrado = true ;
8 else i++ ;
9 }
10 if (encontrado) return i ;
11 else return -1 ;
12 }

```

Hemos utilizado el esquema de búsqueda: Definimos una variable `boolean` que indica si hemos encontrado o no lo que buscamos. El bucle se repite mientras no lleguemos al final del array y no hayamos encontrado un día sin lluvias.

También es posible una solución sin utilizar la variable `boolean`:

```

1 public static int primerDiaSinLluvia2(double[] lluvia){
2 int i=0 ;
3 while (i<lluvia.length && lluvia[i] != 0)
4 i++;
5 if (i == lluvia.length) return -1 ;
6 else return i;
7 }

```

En este caso el subíndice `i` se incrementa mientras estemos dentro de los límites del array y no encontremos un día con lluvia `0`. Al finalizar el bucle hay que comprobar por cual de las dos razones finalizó: ¿Se encontró un día sin lluvias o se recorrió todo el array sin encontrar ninguno? En esta comprobación es importante no acceder al array si existe la posibilidad de que el subíndice esté fuera de los límites del array. La siguiente comprobación sería **incorrecta**:

```

1 if (lluvia[i] == 0) return i;
2 else return -1;

```

ya que, si se ha finalizado el bucle sin encontrar ningún día sin lluvia, `i` valdrá `lluvia.length`, que no es una posición válida del array, y al acceder a `lluvia[i]` se producirá la excepción `ArrayIndexOutOfBoundsException` (índice del array fuera de los límites)

Por otra parte, el mismo problema se puede resolver utilizando la sentencia `for`, como hemos hecho otras veces. Sin embargo la solución parece menos intuitiva porque el cuerpo del `for` quedaría vacío:

```

1 public static int primerDiaSinLluvia3(double[] lluvia){
2 int i;
3 for (i=0; i<lluvia.length && lluvia[i] != 0; i++) /*Nada*/ ;
4 if (i == lluvia.length) return -1 ;
5 else return i;
6 }

```

Otra opción más:

```

1 public static int primerDiaSinLluvia4(double[] lluvia){
2 int i=0 ;
3 while (i<lluvia.length){
4 if (lluvia[i++] == 0) return i ;
5 }
6 return -1 ;
7 }

```

#### 12.3.2.2. BÚSQUEDA DESCENDENTE

En los ejemplos de búsqueda anteriores hemos iniciado la búsqueda en el elemento cero y hemos ido ascendiendo hasta la última posición del array. A esto se le llama búsqueda ascendente.

Si queremos encontrar el último día del mes en que no llovió podemos realizar una búsqueda descendente, es decir, partiendo del último componente del array y decrementando progresivamente el subíndice hasta llegar a la posición cero o hasta encontrar lo buscado:

```

1 public static int ultimoDiaSinLluvia(double[] lluvia){
2 int i=lluvia.length-1;
3 boolean encontrado = false ;
4 while (i>=0 && !encontrado){
5 if (lluvia[i] == 0) encontrado = true ;
6 else i-- ;
7 }
8 if (encontrado) return i ;
9 else return -1 ;
10 }

```

### 12.3.2.3. BÚSQUEDA EN UN ARRAY ORDENADO: BÚSQUEDA BINARIA

Suponga que una amiga apunta un número entre el 0 y el 99 en una hoja de papel y vosotros debéis adivinarlo. Cada vez que conteste, le dirá si el valor que ha dicho es mayor o menor que el que ha de adivinar. ¿Qué estrategia seguiría para lograrlo? Hay que pensar un algoritmo a seguir para resolver este problema.

Una aproximación muy ingenua podría ser ir diciendo todos los valores uno por uno, empezando por 0. Está claro que cuando llegue al 99 lo habréis adivinado. En el mejor caso, si había escrito el 0, acertará en la primera, mientras que en el peor caso, si había escrito el 99, necesitaríais 100 intentos. Si estaba por medio, tal vez con 40-70 basta. Este sería un algoritmo eficaz (hace lo que tiene que hacer), pero no muy eficiente (lo hace de la mejor manera posible). Ir probando valores al azar en lugar de hacer esto tampoco mejora gran cosa el proceso, y viene a ser lo mismo.

Si alguna vez habeis jugado a este juego, lo que habreis hecho es ser un poco más astutos y empezar por algún valor del medio. En este caso, por ejemplo, podría ser el 50. Entonces, en caso de fallar, una vez está seguro de si el valor secreto es mayor o menor que su respuesta, en el intento siguiente probar un valor más alto o más bajo , e ir haciendo esto repetidas veces.

Generalmente, la mejor estrategia para adivinar un número secreto entre 0 y N sería primer probar  $N/2$ . Si no se ha acertado, entonces si el número secreto es más alto se intenta adivinar entre  $(N/2 + 1)$  y N. Si era más bajo, se intenta adivinar el valor entre 0 y  $N-1$ . Para cada caso, se vuelve a probar el valor que hay en el medio del nuevo intervalo. Y así sucesivamente, haciendo cada vez más pequeño el intervalo de búsqueda, hasta adivinarlo. En el caso de 100 valores, esto garantiza que, en el peor de los casos, en 7 intentos seguro que se adivina. Esto es una mejora muy grande respecto al primer algoritmo, donde hacían falta 100 intentos, y por tanto, este sería un algoritmo más eficiente. Concretamente, siempre se adivinará en  $\log_2(N)$  intentos como máximo.

Si os fijáis, el ejemplo que se acaba de explicar, en realidad, no es más que un esquema de búsqueda en una secuencia de valores, como puede ser dentro de un array, partiendo de la condición que todos los elementos estén ordenados de menor a mayor. De hecho, hasta ahora, para hacer una búsqueda de un valor dentro de un array se ha usado el sistema "ingenuo", mirando una por una todas las posiciones. Pero si los elementos están ordenados previamente, se podría usar el sistema "astuto" para diseñar un algoritmo mucho más eficiente, y hasta cierto punto, más "inteligente".

El algoritmo basado en esta estrategia se conoce como **búsqueda binaria o dicotómica**.

Para ello iniciaremos la búsqueda en la posición central del array.

- Si el elemento central es el buscado habremos finalizado la búsqueda.
- Si el elemento central es mayor que el buscado, tendremos que continuar la búsqueda en la mitad izquierda del array ya que, al estar éste ordenado todos los elementos de la mitad derecha serán también mayores que el buscado.
- Si el elemento central es menor que el buscado, tendremos que continuar la búsqueda en la mitad derecha del array ya que, al estar éste ordenado todos los elementos de la mitad izquierda serán también menores que el buscado.

En un solo paso hemos descartado la mitad de los elementos del array. Para buscar en la mitad izquierda o en la mitad derecha utilizaremos el mismo criterio, es decir, iniciaremos la búsqueda en el elemento central de dicha mitad, y así sucesivamente hasta encontrar lo buscado o hasta que descubramos que no está.

Supongamos por ejemplo que, dado un array que contiene edades de personas, ordenadas de menor a mayor queremos averiguar si hay alguna persona de 36 años o no.

El siguiente método soluciona este problema realizando una búsqueda binaria:

```

1 public static boolean hayAlguienDe36(int[] edad) {
2 // Las variables izq y der marcarán el fragmento del array en el que
3 // realizamos la búsqueda. Inicialmente buscamos en todo el array.
4 final int NUMERO_BUSCADO = 36;
5 int izq = 0;
6 int der = edad.length - 1;
7 boolean encontrado = false;
8 while (izq <= der && !encontrado) {
9 // Calculamos posición central del fragmento en el que buscamos
10 int m = (izq + der) / 2;
11 if (edad[m] == NUMERO_BUSCADO) // Hemos encontrado una persona de 36
12 {
13 encontrado = true;
14 } else if (edad[m] > NUMERO_BUSCADO) {
15 // El elemento central tiene más de 36.
16 // Continuamos la búsqueda en la mitad izquierda. Es decir,
17 // entre las posiciones izq y m-1
18 der = m - 1;
19 } else {
20 // El elemento central tiene menos de 36.
21 // Continuamos la búsqueda en la mitad derecha. Es decir,
22 // entre las posiciones m+1 y der
23 izq = m + 1;
24 } // del if
25 } // del while
26 return encontrado; // if (encontrado) return true; else return false;
27 }

```

La búsqueda finaliza cuando encontramos una persona con 36 años (`encontrado==true`) o cuando ya no es posible encontrarla, circunstancia que se produce cuando `izq` y `der` se cruzan (`izq>der`).

### 12.3.3. Problemas de ordenación

Con frecuencia necesitamos que los elementos de un array estén ordenados (por ejemplo para usar la búsqueda binaria).

Existen multitud de algoritmos que permiten ordenar los elementos de un array, entre los que hay soluciones **iterativas** y soluciones **recursivas**.

Entre los algoritmos **iterativos** tenemos, por ejemplo, el **método de la burbuja**, el **método de selección directa** y el **método de inserción directa**.

Entre los **recursivos**, son conocidos el algoritmo **mergesort** y el **quickSort**, que realizan la ordenación más rápidamente que los algoritmos iterativos que hemos nombrado.

Como ejemplo vamos a ver como se realiza la ordenación de un array de enteros utilizando el método de **selección directa**:

```

1 public static void seleccionDirecta(int[] v) {
2 for (int i = 0; i < v.length-1; i++) {
3 //Localizamos elemento que tiene que ir en la posición i
4 int posMin = i;
5 //buscar el menor a la derecha
6 for (int j = i + 1; j < v.length; j++) {
7 if (v[j] < v[posMin]) {
8 posMin = j;
9 }
10 }
11 //al llegar aquí posMin tendrá la posición del elemento menor
12 //Intercambiamos los elementos de las posiciones i y posMin
13 //v[i]<=>v[posMin];
14 int aux = v[posMin];
15 v[posMin] = v[i];
16 v[i] = aux;
17 }
18 }

```

El método consiste en recorrer el array ascendente a partir de la posición cero.

En cada posición (`i`) localizamos el elemento que tiene que ocupar dicha posición cuando el array esté ordenado, es decir, el menor de los elementos que quedan a su derecha.

Cuando se ha determinado el menor se coloca en su posición realizando un intercambio con el elemento de la posición `i`. Con ello, el array queda ordenado hasta la posición `i`.

Y a modo de curiosidad os dejo por aquí el método de inserción directa:

1. Comenzamos considerando el primer elemento como la parte ordenada.
2. Luego, tomamos un elemento de la parte no ordenada y lo insertamos en la posición correcta dentro de la parte ordenada, desplazando los elementos mayores que él hacia la derecha.

3. Repetimos este proceso hasta que todos los elementos estén en la parte ordenada.

```

1 public static void insercionDirecta(int[] array) {
2 for (int i = 1; i < array.length; i++) {
3 int key = array[i];
4 int j = i - 1;
5
6 // Mover los elementos mayores que key hacia la derecha
7 while (j >= 0 && array[j] > key) {
8 array[j + 1] = array[j];
9 j--;
10 }
11
12 // Insertar key en su posición correcta
13 array[j + 1] = key;
14 }
15 }
```

### Más información

Ejemplos visuales de distintos métodos de ordenación, con distintos tipos de entradas: <https://www.toptal.com/developers/sorting-algorithms> Otro ejemplo de visualizador de algoritmos: <http://algorithm-visualizer.org/>

## 12.4. Arrays bidimensionales: matrices

Los arrays bidimensionales, también llamados matrices, son muy similares a los arrays que hemos visto hasta ahora: También son una colección de elementos del mismo tipo que se agrupan bajo un mismo nombre de variable. Sin embargo:

- Sus elementos están organizados en filas y columnas. Tienen, por tanto una altura y una anchura, y por ello se les llama bidimensionales.
- A cada componente de una matriz se accede mediante dos subíndices: el primero se refiere al número de fila y el segundo al número de columna. En la siguiente figura, `m[0][0]` es 2, `m[0][3]` es 9, `m[2][0]` es 57

Screenshot 20210821 085938

- Como vemos, filas y columnas se numeran a partir del 0.

Si se quisiera extender el tratamiento el estudio de la pluviosidad, para abarcar no solo los días de un mes sino los de todo un año, se podría definir, por ejemplo, un array de 366 elementos, que mantuviera de forma correlativa los datos de pluviosidad de una zona día a día. Con ello, por ejemplo, el dato correspondiente al día 3 de febrero ocuparía la posición 34 del array, mientras que el correspondiente al 2 de julio ocuparía el 184.

Una aproximación más conveniente para la representación de estos datos consistiría en utilizar una matriz con 12 filas (una por mes) y 31 columnas (una por cada día del mes). Esto permitiría una descripción más ajustada a la realidad y, sobre todo, simplificaría los cálculos de la posición real de cada día en la estructura de datos. El elemento `[0][3]` correspondería, por ejemplo, a las lluvias del 4 de enero.

### 12.4.1. Matrices en Java

#### Definición

En Java, una matriz es, en realidad un array en el que cada componente es, a su vez, un array. Dicho de otra manera, una matriz de enteros es un array de arrays de enteros.

Esto, que no es igual en otros lenguajes de programación, tiene ciertas consecuencias en la declaración, creación y uso de las matrices en Java:

- Una matriz, en Java, puede tener distinto número de elementos en cada fila.
- La creación de la matriz se puede hacer en un solo paso o fila por fila.
- Si `m` es una matriz de enteros...
- `m[i][j]` es el entero de la fila `i`, columna `j`
- `m[i]` es un array de enteros.
- `m.length` es el número de filas de `m`.

- `m[i].length` es el número de columnas de la fila `i`
- Podríamos dibujar la matriz `m` del ejemplo anterior de una forma más cercana a cómo Java las representa internamente:

Screenshot 20210821 090234

#### 12.4.2. Declaración de matrices.

El código siguiente declara una matriz (array bidimensional) de elementos de tipo `double`, y la crea para que tenga 5 filas y 4 columnas (matriz de 5x4):

```
1 double[][] m1 = new double[5][4];
```

La siguiente declaración es equivalente a la anterior aunque en la práctica es menos utilizada a no ser que queramos que cada fila tenga un número distinto de elementos:

```
1 double[][] m2 = new double [5][];
2 m2[0] = new double[4];
3 m2[1] = new double[4];
4 m2[2] = new double[4];
5 m2[3] = new double[4];
6 m2[4] = new double[4];
```

Es posible inicializar cada uno de los subarrays con un tamaño diferente (aunque el tipo base elemental debe ser siempre el mismo para todos los componentes). Por ejemplo:

```
1 double[][] m3 = new double [5][];
2 m3[0] = new double[3];
3 m3[1] = new double[4];
4 m3[2] = new double[14];
5 m3[3] = new double[10];
6 m3[4] = new double[9];
```

#### 12.4.3. Inicialización.

La forma de inicializar una matriz de enteros de por ejemplo [4][3] seria:

```
1 int[][] m4 = {{7,2,4},{8,2,5},{9,4,3},{1,2,4}};
2 //aunque se entiende mejor de este modo:
3 int[][] m4 = {
4 {7,2,4},
5 {8,2,5},
6 {9,4,3},
7 {1,2,4}
8 };
```

|   |   |   |
|---|---|---|
| 7 | 2 | 4 |
| 8 | 2 | 5 |
| 9 | 4 | 3 |
| 1 | 2 | 4 |

#### 12.4.4. Recorrido

El recorrido se hace de forma similar al de un array aunque, dado que hay dos subíndices, será necesario utilizar dos bucles anidados: uno que se ocupe de recorrer las filas y otro que se ocupe de recorrer las columnas.

El siguiente fragmento de código recorre una matriz `m4` para imprimir sus elementos uno a uno.

```

1 //recorrido por filas
2 System.out.println("\nRecorrido por filas: ");
3 for (int f = 0; f < m4.length; f++) {
4 for (int c = 0; c < m4[f].length; c++) {
5 System.out.print(m4[f][c] + " ");
6 }
7 System.out.println("");
8 }
9 //Recorrido por filas:
10 //7 2 4
11 //8 2 5
12 //9 4 3
13 //1 2 4

```

El recorrido se ha hecho por filas, es decir, se imprimen todos los elementos de una fila y luego se pasa a la siguiente. Como habíamos indicado anteriormente, `m.length` representa el número de filas de `m`, mientras que `m[i].length` el número de columnas de la fila `i`.

También es posible hacer el recorrido por columnas: imprimir la columna 0, luego la 1, etc:

```

1 System.out.println("\nRecorrido por columnas: ");
2 int numFilas = m4.length;
3 int numColumnas = m4[0].length;
4 for (int c = 0; c < numColumnas; c++) {
5 for (int f = 0; f < numFilas; f++) {
6 System.out.print(m4[f][c] + " ");
7 }
8 System.out.println("");
9 }
10 //Recorrido por columnas:
11 //7 8 9 1
12 //2 2 4 2
13 //4 5 3 4

```

o, directamente ...

```

1 System.out.println("\nRecorrido por columnas versión 2: ");
2 for (int c = 0; c < m4[0].length; c++) {
3 for (int f = 0; f < m4.length; f++) {
4 System.out.print(m4[f][c] + " ");
5 }
6 System.out.println("");
7 }
8 //Recorrido por columnas versión 2:
9 //7 8 9 1
10 //2 2 4 2
11 //4 5 3 4

```

En este caso, para un funcionamiento correcto del recorrido sería necesario que todas las columnas tuvieran igual número de elementos, pues en el bucle externo, se toma como referencia para el número de columnas la longitud de `m[0]`, es decir el número de elementos de la primera fila.

## 12.5. Arrays multidimensionales

En el punto anterior hemos visto que podemos definir arrays cuyos elementos son a la vez arrays, obteniendo una estructura de datos a la que se accede mediante dos subíndices, que hemos llamado arrays bidimensionales o matrices.

Este *anidamiento* de estructuras se puede generalizar, de forma que podríamos construir arrays de más de dos dimensiones. En realidad Java no pone límite al número de subíndices de un array. Podríamos hacer declaraciones como las siguientes:

```

1 int[][][] notas = new int[10][5][3]; //Notas de 10 alum. en 5 asign. en 3 eval.
2 notas[2][3][1]=5;//El alumno 2, para la asignatura 3 de la primera evaluación ha sacado un 5
3 double[][][][][] w = new double [2][7][10][4][10];

```

### Más información

Sin embargo, encontrar ejemplos en los que sean necesarios arrays de más de tres dimensiones es bastante raro, y aún cuando los encontramos solemos utilizar arrays de uno o dos subíndices porque nos resulta menos complejo manejarlos.

## 12.6. Recursividad

# Entendiendo la Recursividad

Una técnica de programación poderosa donde un método se invoca a sí mismo para resolver problemas complejos dividiéndolos en versiones más simples hasta alcanzar una solución directa.

### Fundamentos de la Recursividad

#### Un método que se invoca a sí mismo

Resuelve un problema complejo dividiéndolo en versiones más simples de si mismo.

#### El Caso Base: La condición de salida

Es la condición donde el problema es tan simple que se resuelve directamente, deteniendo las llamadas.

#### El Caso Recursivo: La llamada a sí mismo

Llama al mismo método, pero con datos que se acercan progresivamente al caso base.

### Aplicación y Precauciones

#### Ejemplo Clásico: Cálculo del Factorial (n!)

La definición recursiva es  $n! = n * (n-1)!$ , con un caso base de  $0! = 1$ .

| Operación                  | Retorno      |
|----------------------------|--------------|
| Llamada factorial(3)       | Esperando... |
| Operación 3 * factorial(2) | Esperando... |
| Llamada factorial(2)       | Esperando... |
| Operación 2 * factorial(1) | Esperando... |
| Llamada factorial(1)       | Esperando... |
| Operación 1 * factorial(0) | Esperando... |
| Llamada factorial(0)       | Esperando... |
| Operación Caso Base        | Retorno 1    |

|              |              |
|--------------|--------------|
| Operación    | Retorno      |
| Esperando... | Esperando... |
| Retorno      | Esperando... |
| Esperando... | Esperando... |
| Retorno      | Esperando... |
| Esperando... | Esperando... |

#### Riesgo Principal: Desbordamiento de Pila (Stack Overflow)

Ocurre si el caso base nunca se alcanza, agotando la memoria con llamadas infinitas.



© NotebookLM

A la hora de crear programas complejos, uno de los aspectos que diferencia el buen programador del aficionado es su capacidad de hacer algoritmos eficientes. O sea, que sean capaces de resolver el problema planteado en el mínimo de pasos. En el caso de un programa, esto significa la necesidad de ejecutar el mínimo número de instrucciones posible. Ciertamente, si el resultado tiene que ser exactamente el mismo, siempre será mejor hacer una tarea en 10 pasos que en 20, intentando evitar pasos que en realidad son innecesarios. Por lo tanto, la etapa de diseño de un algoritmo es bastante importante y hay que pensar bien una estrategia eficiente. Ahora bien, normalmente, los algoritmos más eficientes también son más difíciles de pensar y codificar, ya que no siempre son evidentes.

### 12.6.1. Aplicación de la recursividad

A menudo encontrareis que explicar de palabra la idea general de una estrategia puede ser sencillo, pero traducirla a instrucciones de Java ya no lo es tanto. Retomamos ahora el caso de la búsqueda dicotómica o binaria, dado que hay que ir repitiendo unos pasos en sucesivas iteraciones, está más o menos claro que el problema planteado para realizar búsquedas eficientes se basa en una estructura de repetición. Pero no se recorren todos los elementos y el índice no se incrementa uno a uno, sino que se va cambiando a valores muy diferentes para cada iteración. No es un caso evidente. Precisamente, este ejemplo no se ha elegido al azar, ya que es un caso en el que os puede ir bien aplicar un nuevo concepto que permite facilitar la definición de algoritmos complejos donde hay repeticiones.

#### Definición

La **recursividad** es una forma de describir un proceso para resolver un problema de manera que, a lo largo de esta descripción, se usa el proceso mismo que se está describiendo, pero aplicado a un caso más simple.

De hecho, tal vez sin darse cuenta de ello en, ya se ha usado recursividad para describir cómo resolver un problema. Para ver qué significa exactamente la definición formal apenas descrita, se repetirá el texto en cuestión, pero remarcando el aspecto recursivo de la descripción:

*"Generalmente, la mejor estrategia para adivinar un número secreto entre 0 y N sería primero probar N/2. Si no se ha acertado, entonces si el número secreto es más alto se intenta adivinar entre (N/2 + 1) y N. Si era más bajo, se intenta adivinar el valor entre 0 y N-1. Para cada caso, se vuelve a probar el valor que hay en el centro del nuevo intervalo. Y así sucesivamente, hasta adivinarlo."*

O sea, **el proceso de adivinar un número se basa en el proceso de intentar adivinar un número!** Esto parece hacer trampas, ya es como usar la misma palabra que se quiere definir a su propia definición. Pero fíjese en un detalle muy importante. Los nuevos usos del proceso de "adivinar" son casos más simples, ya que primero se adivina entre N valores posibles, luego entre N/2 valores, después entre N/4, etc. Este hecho no es casual y de él depende poder definir un proceso recursivo de manera correcta.