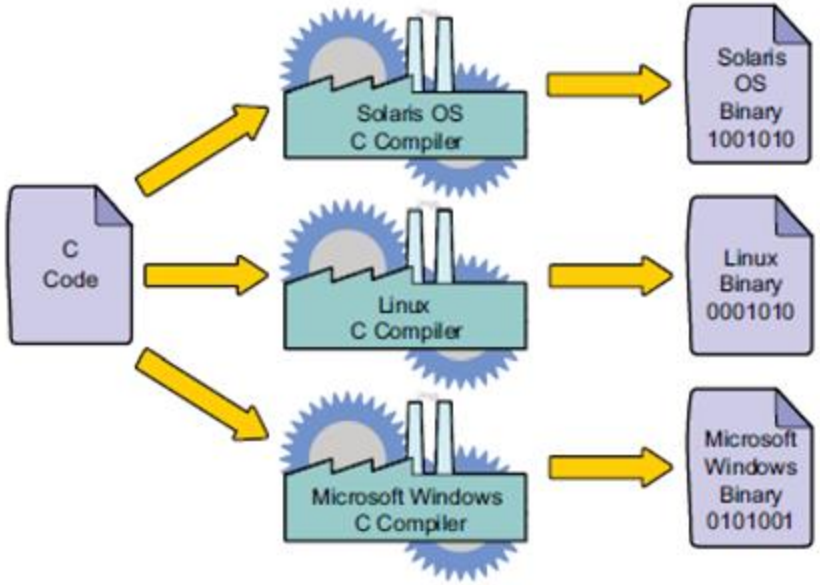

Java™

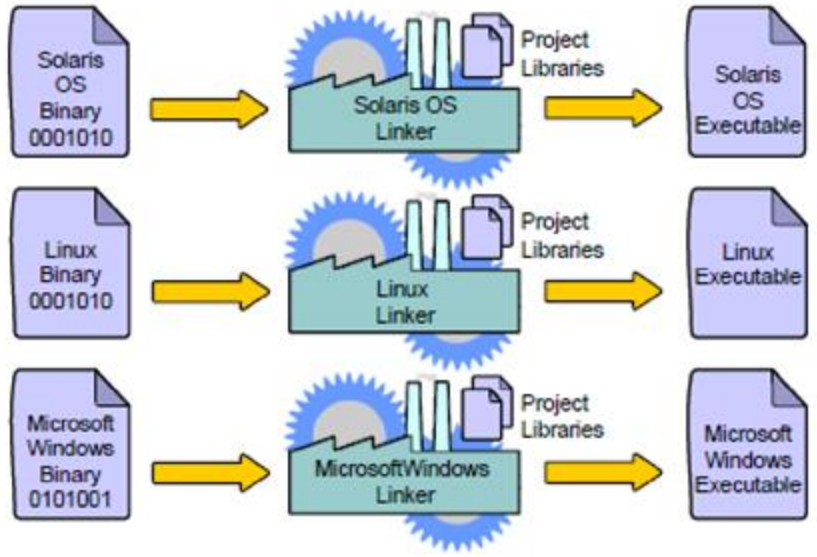
Módulo 1: Iniciación a Java

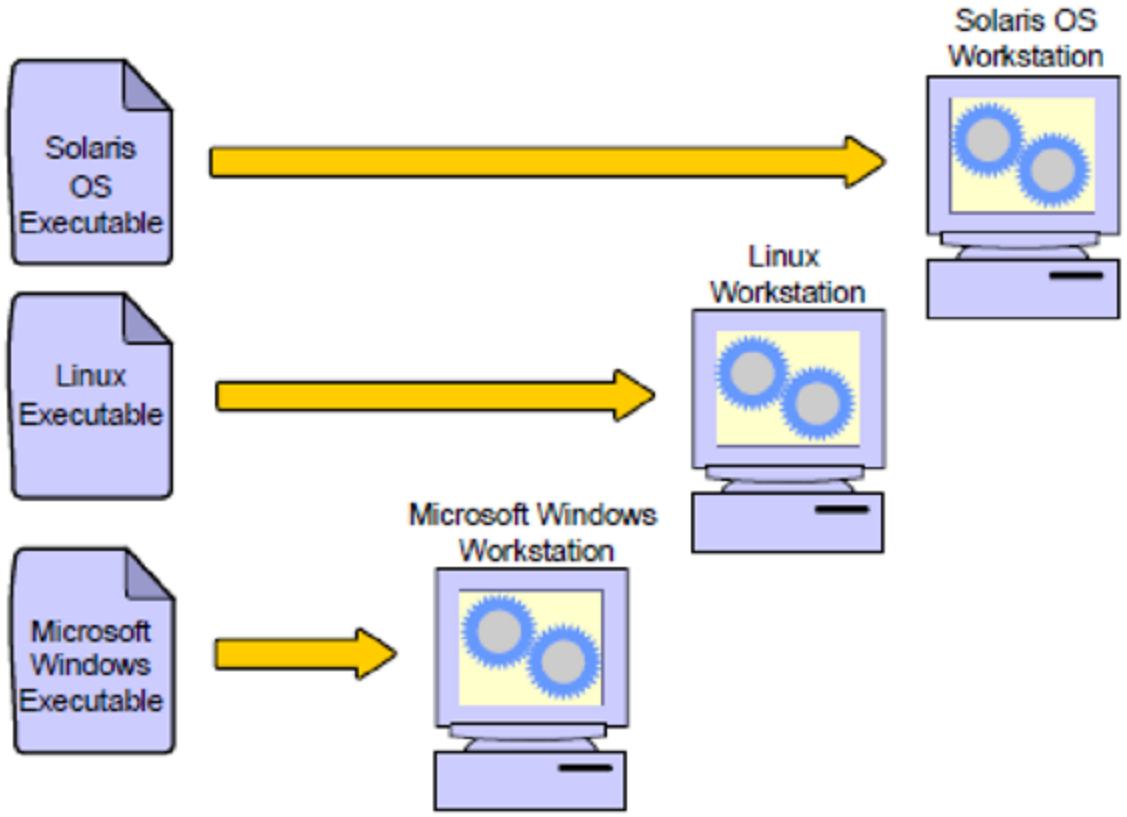
- **Unidad 1:** Introducción al lenguaje Java
- **Unidad 2**: Tipos de datos y operadores
- **Unidad 3**: Sentencias de control
- **Unidad 4**: Vectores y cadenas de texto
- **Unidad 5**: Introducción a la Programación Orientada a Objetos: clases, objetos y métodos
- **Unidad 6**: Herencia
- **Unidad 7**: Uso de interfaces
- **Unidad 8**: Excepciones
- **Unidad 9**: Módulos

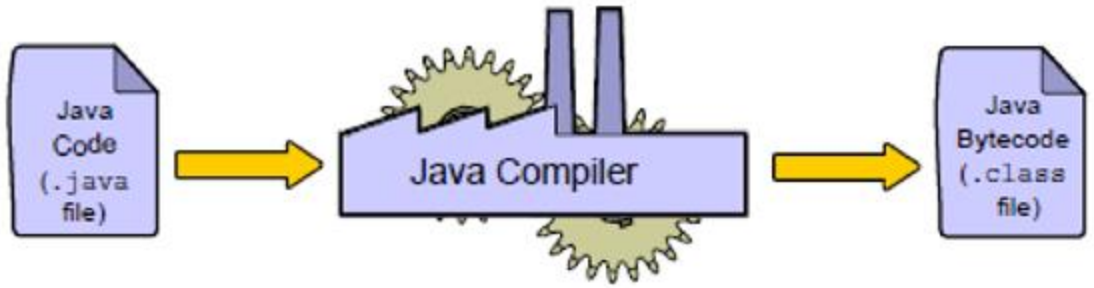
Herramientas

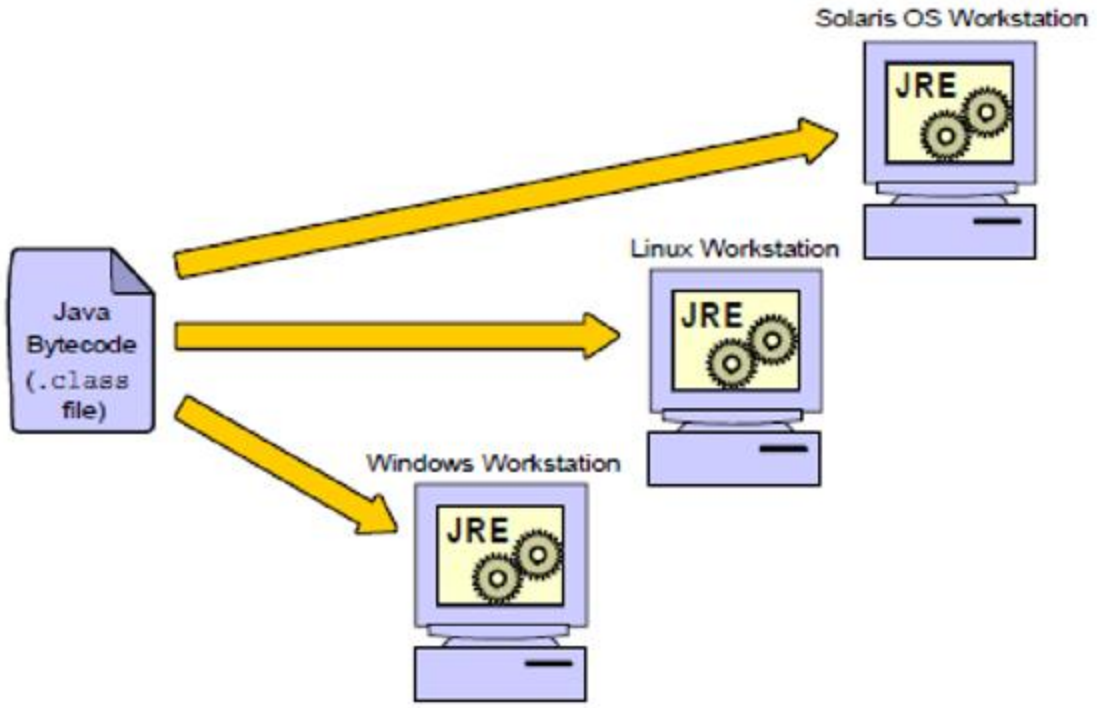
- JDK 11
- IntelliJ IDEA CE

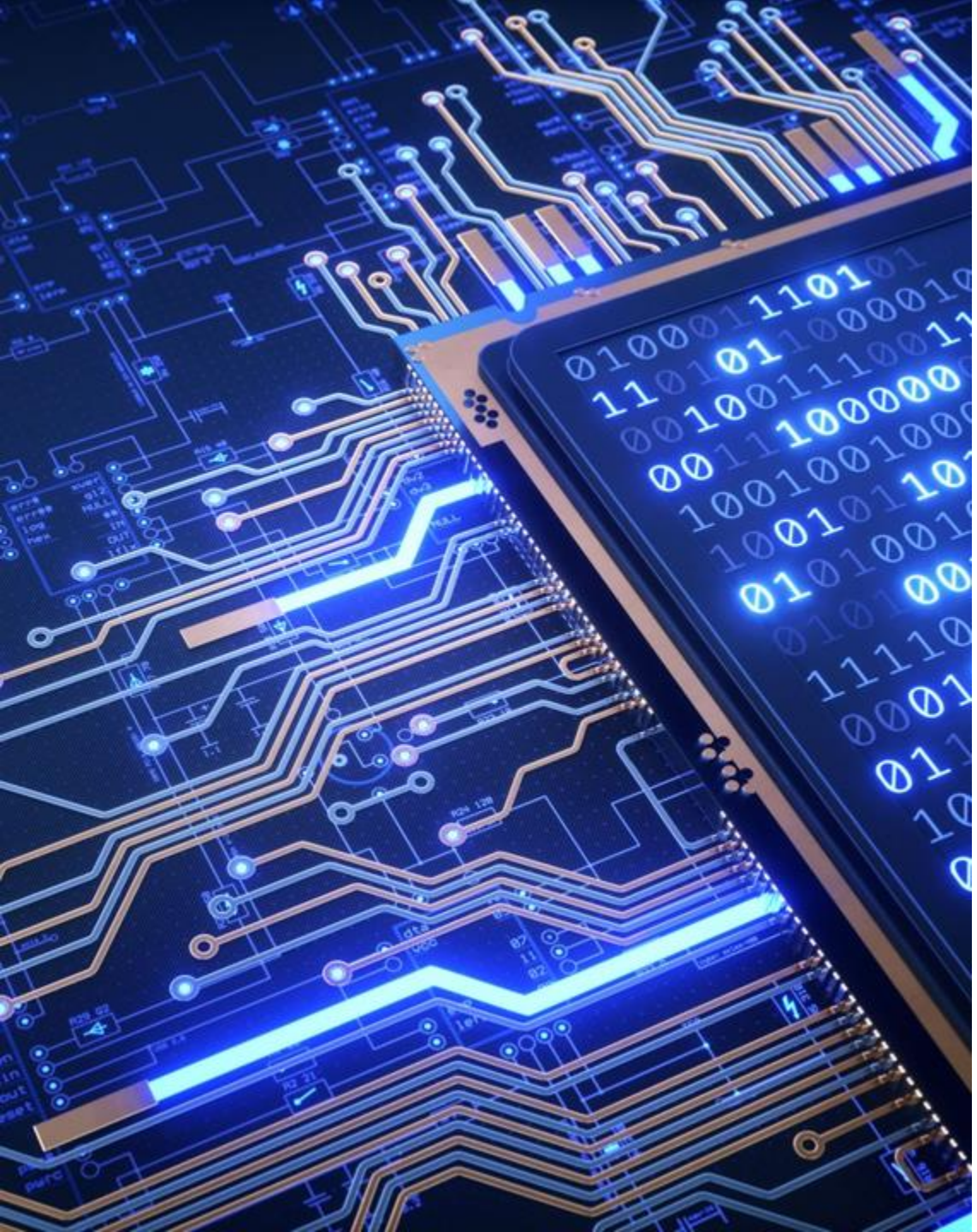

C
Code
Solaris OS
C Compiler
Linux
C Compiler
Microsoft Windows
C Compiler
Solaris
OS
Binary
1001010
Linux
Binary
0001010
Microsoft
Windows
Binary
0101001


Solaris
OS
Binary
0001010
Linux
Binary
0001010
Microsoft
Windows
Binary
0101001
Solaris OS
Linker
Linux
Linker
MicrosoftWindows
Linker
Project
Libraries
Project
Libraries
Project
Libraries
Solaris
OS
Executable
Linux
Executable
Microsoft
Windows
Executable


Solaris
OS
Executable
Linux
Executable
Microsoft
Windows
Executable
Solaris OS
Workstation
Linux
Workstation
Microsoft Windows
Workstation


Java
Code
(.java
file)
Java Compiler
Java
Bytecode
(.class
file)


Solaris OS Workstation
JRE
Linux Workstation
JRE
Java
Bytecode
(.class
file)
Windows Workstation
JRE

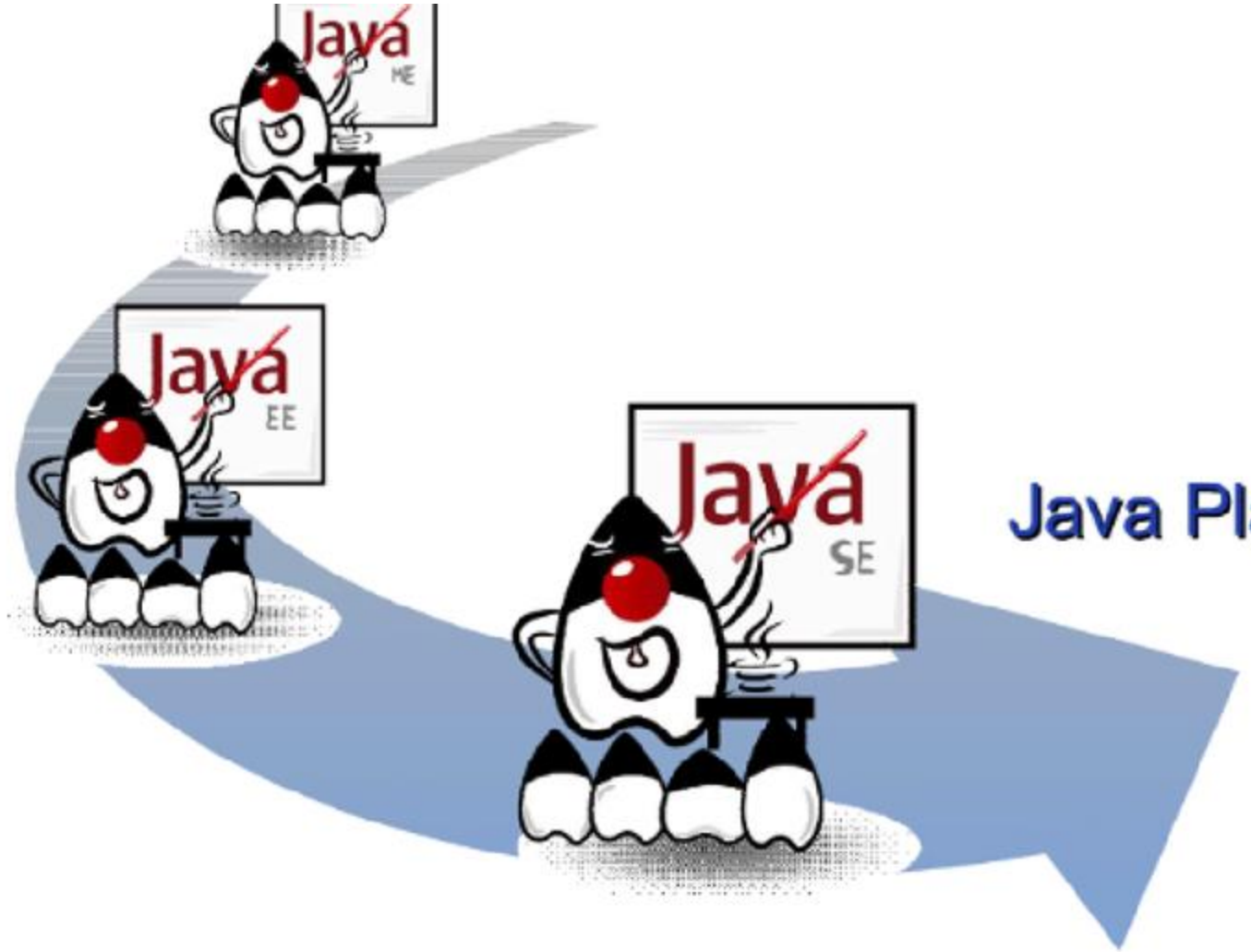


JRE Y JDK

•**JRE (Java Runtime Environment)**; Seria necesario unicamente para ejecutar una aplicacion. Esta seria la version que se deben descargar los clientes, usuarios finales de nuestra aplicacion. Incluye unicamente la JVM y un conjunto de librerias para poder ejecutar.

•**JDK (Java Development Kit)**; Estos son los recursos que necesitamos los desarrolladores ya que incluye lo siguiente:

- JRE
- Compilador de java
- Documentacion del API (todas las librerias de Java)
- Otras utilidades por ejemplo para generar archivos .jar, crear documentacion,



Java Platforms

Java

CARACTERÍSTICAS DE JAVA

- Orientado a objetos; como se mencionaba al inicio del capitulo, en Java todo se considera como un objeto. La OO facilita la reutilizacion de codigo como se vera mas adelante.
- Distribuido; Java permite desarrollar aplicaciones distribuidas, esto significa que parte de la aplicacion puede alojarse en un servidor de Madrid y otra parte puede residir en otro servidor de Barcelona por ejemplo.
- Simple; Aunque os parezca mentira en este momento, Java es un lenguaje muy simple de programar. Ademas, contamos con muchas librerias ya desarrolladas, listas para utilizarse y lo mas importante una buena documentacion de uso.
- Multihilo; El lenguaje Java funciona mediante hilos no a traves de procesos como otros lenguajes. Esto hace que sea mas rapida y mas segura su ejecucion.
- Seguro; En el curso iremos viendo las diversas formas de implementar seguridad en aplicaciones Java.
- Independiente de la plataforma; como ya comentamos anteriormente.

```
package com.jorge.holamundo;

/*
 * Primer programa escrito en
 Java
 */
public class HolaMundo {
    public static
  void main(String
          args[])
                {
 System.out.printl
 n("Hola Mundo!");
  }
}
```

Algo de códio para empezar

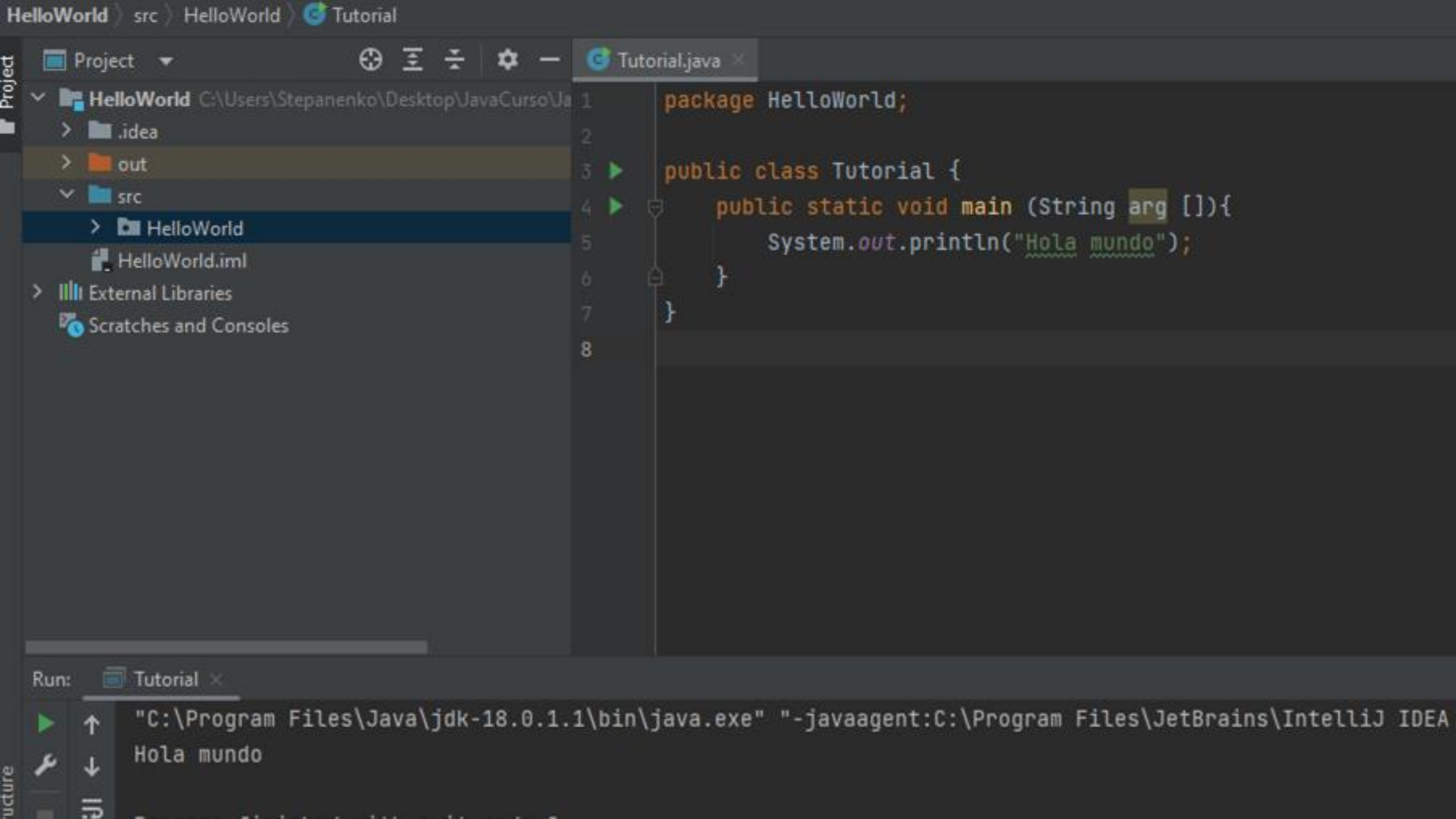
- **Los comentarios** se escriben entre los caracteres **/* y */** y pueden ocupar **varias líneas**
- Si un **comentario** sólo ocupa **una línea** se puede escribir justo después de los caracteres **//**
- La instrucción System.out.println("Hola") escribe el **texto indicado por pantalla**
- Cada **sentencia** de código se **termina** siempre con un **;**
- La **definiciones de las clases y métodos** no se consideran sentencias y no terminan con el caracter ; ya que definen nuevos bloques de código
- Cada **bloque de código** se inicia con el caracter **{ y termina con }**

Palabras reservadas del lenguaje Java

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

Consideraciones

- Normalmente una **aplicación se corresponde con un Proyecto** (en IntelliJ IDEA en nuestro caso)
- Estructura basada en **paquetes** (carpetas) y debemos **especificar uno como mínimo**
- La **unidad mínima de un proyecto** es **la clase** (normalmente pública) y normalmente cada una se escribe en **un fichero de código**
- La **unidad mínima de ejecución** es el **método** que normalmente estará compuesto de un conjunto de instrucciones relacionadas
- Como mínimo tendrá que haber **una clase pública** con el método public static void main(String args[]), que será el
- **punto de arranque** cuando se ejecute el proyecto
- El compilador **nunca procesa los comentarios**
- **Nunca** debemos **editar/modificar** los ficheros del proyecto que **no sean código Java**


HelloWorld 〉 src 〉 HelloWorld 〉 Tutorial
Project
HelloWorld C:\Users\Stepanenko\Desktop\JavaCurso\Ja
.idea
out
src
HelloWorld
HelloWorld.iml
External Libraries
Scratches and Consoles
Tutorial.java
1 package HelloWorld;
2
3 public class Tutorial {
4 public static void main (String arg []){
5 System.out.println("Hola mundo");
6 }
7 }
8
Run: Tutorial
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Hola mundo

Proceso de creación de una aplicación

- Al compilar, **cada clase de código genera un fichero .class** y sólo una de ellas será la que inicie la aplicación
- No es raro que un proyecto pueda contener **más de 100 clases**, a lo que habría que sumar **otros recursos** (texto, imágenes, . . .)
- Lo normal es que el código compilado se empaquete como **.jar** o **.war**
- Junto con las clases compiladas se empaqueta el **manifiesto de la aplicación**, donde se especifica, entre otras cosas, cuál es la clase que contiene el método main (punto de arranque de la aplicación)
- Los IDE incorporan **menús para facilitar la creación de empaquetados** y manifiestos del proyecto
- Dependiendo de cómo esté configurado el Sistema Operativo **podemos lanzar la aplicación simplemente haciendo doble-click** sobre el empaquetado (HolaMundo.jar, por ejemplo)


DEMO

Tipo	Tamaño	Valor mínimo	Valor máximo
byte	1 byte	-128	127
short	2 bytes	-32768	32767
int	4 bytes	-2147483648	2147483647
long	8 bytes	-9223372036854775808	9223372036854775807
float	4 bytes	-3.402823e38	3.402823e38
double	8 bytes	-1.79769313486232e308	1.79769313486232e308
char	2 bytes		
boolean	1 byte	false	true

Tipos de datos primitivos

Símbolo	Significado	Ejemplo
=	Asignación	x = 4
+	Suma	x + 3
-	Resta	x - 3
/	División	x / 3
*	Multiplicación	x * 4 / 2
%	Módulo (resto entero)	20 % 3

Operadores de asignación y aritméticos

Símbolo	Significado	Ejemplo
++	Preincremento	++x
++	Posincremento	x++
--	Predecremento	--x
--	Posdecremento	x++

Operadores pre/pos incremento

Símbolo	Significado	Ejemplo
+=	Suma y asignación	x += 3
-=	Resta y asignación	x -= 4
*=	Multiplicación y asignación	x *= 10
/=	División y asignación	x /= 10
%=	Módulo y asignación	x %= 2

Operadores combinados

Símbolo	Significado	Ejemplo
==	Igual que	x == 3
!=	Distinto que	x != 3
<	Menor que	x < 3
<=	Menor o igual que	x <= 4
>	Mayor que	x > 3
>=	Mayor o igual que	x >= 6

Operadores de comparación

Símbolo	Significado	Ejemplo	Resultado
!	Negación	! (10 == 10)	false
||	Or	(10 == 10) || (10 == 3)	true
&&	And	(10 == 10) && (10 == 3)	false

Operadores lógicos

Declaración de variables

- Las variables en Java se deben declarar antes de poderse usar
- En la declaración se debe especificar el tipo y nombre. Opcionalmente se les puede asignar un valor inicial

- **int** cantidad;

boolean
terminado; **int**
cantidad = 10;

- **boolean** terminado = false;

Uso de valores literales

- Los valores literales son valores fijos que podemos asignar a las variables o bien utilizar para representar directamente por pantalla. En general hay que tener en cuenta lo siguiente:

  - Los números se escriben directamente (cantidad = 3)
  - Los caracteres (char) se escriben siempre entre comillas simples (caracter = 'c')
  - Por defecto un valor numérico literal entero se interpreta como int (cantidad = 3). **Si queremos que se interprete como un long tendremos que añadir el caracter l (**long x = 3l**)**
  - Por defecto un valor numérico literal decimal se interpreta como double (peso = 10.3). **Si queremos que se interprete como un float tendremos que añadir el caracter f (**float peso = 10.3f**)**
  - Las cadenas de texto de longitud variable se escriben entre comillas dobles (String nombre = "Tokio School") y el tipo de dato utilizado sería un String (no es un tipo de dato primitivo pero Java nos deja tratarlo como tal)

Declaración de constantes

- Las constantes son variables cuyo **valor no puede ser modificado**. Se declaran siempre con un valor por defecto que debe permanecer inalterado.
- Se declaran con la palabra reservada **final** antes del tipo de dato, y siempre se escriben en mayúsculas.
- Son útiles para representar **valores fijos** a lo largo del programa, y siempre será mejor que escribirlos literalmente en diferentes ubicaciones del código.

- final int CANTIDAD = 10;

Pueden formar parte de expresiones junto con otras variables:

- float precioFinal = precioUnidad * CANTIDAD;

Pero nunca podrán ser modificadas:

- ~~CANTIDAD += 10;~~

Utilizar cadenas de texto

Consideraciones:

- El tipo String es realmente una **clase**, asi que las cadenas de texto son realmente **objetos Java**
- Al tratarse de un objeto podemos realizar **operaciones** sobre la cadena (invocar a sus **métodos**).
- Java nos permite utilizarlo **como si fuera un tipo primitivo** (es un tipo de dato muy utilizado)
- Como es un objeto, su **valor por defecto** es **null**
- **Cualquier valor** puede ser representado como una cadena, por lo que normalmente es el tipo destino para el input de usuario
- El operador suma (+) aplicado a cadenas de texto **concatena** las cadenas operando para formar una cadena más larga
    - **String** nombreApellidos = nombre + " " + apellidos;

Conversión de tipos

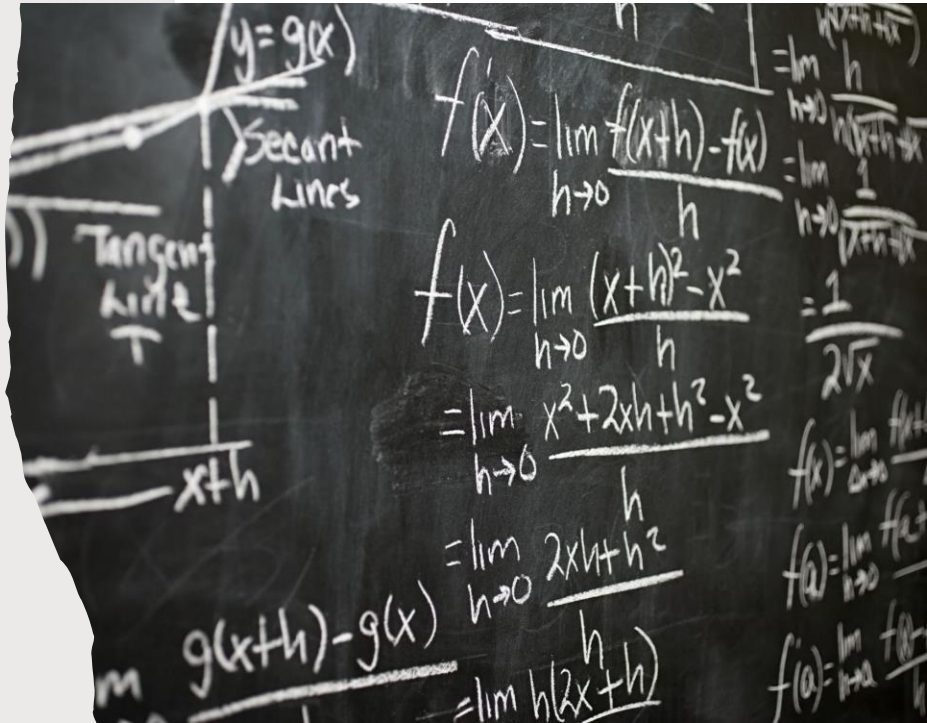
- **Conversión implícita**: La hace automáticamente el compilador siempre que sea posible y no suponga posible pérdida de información

```
float numeroDecimal = 10;
```

- **Conversión explícita**: La hace explícitamente el programador asumiendo cualquier posible pérdida de información

```
float numeroDecimal = 3.4f;
int cantidad = (int) numeroDecimal;
```

- **Parseo de valores**: No se convierte el tipo (no son compatibles) sino que se extrae el valor para almacenarlo en una variable de otro tipo. Conviene tener en cuenta que si la cadena está vacía se producirá una Excepción (finalización forzada del programa por un fallo)




DEMO

EJERCICIO

Operador condicional

Es un operador ternario que permite asignar un valor u otro a una variable en función a que se cumpla o no una condición. Es una forma reducida de utilizar una sentencia condicional if .. else

La síntaxis del operador es:

- resultado = condicion ? valorSiVerdadero: valorSiFalso

El siguiente ejemplo realiza la asignación del valor dependiendo de si un número es par o impar en función del resultado de la operación módulo con el número 2

- int numero = 10;
- String cadena = 10 % 2 == 0 ? "Es número par" : "Es número impar";

En el siguiente, se parsea la cadena de forma segura:

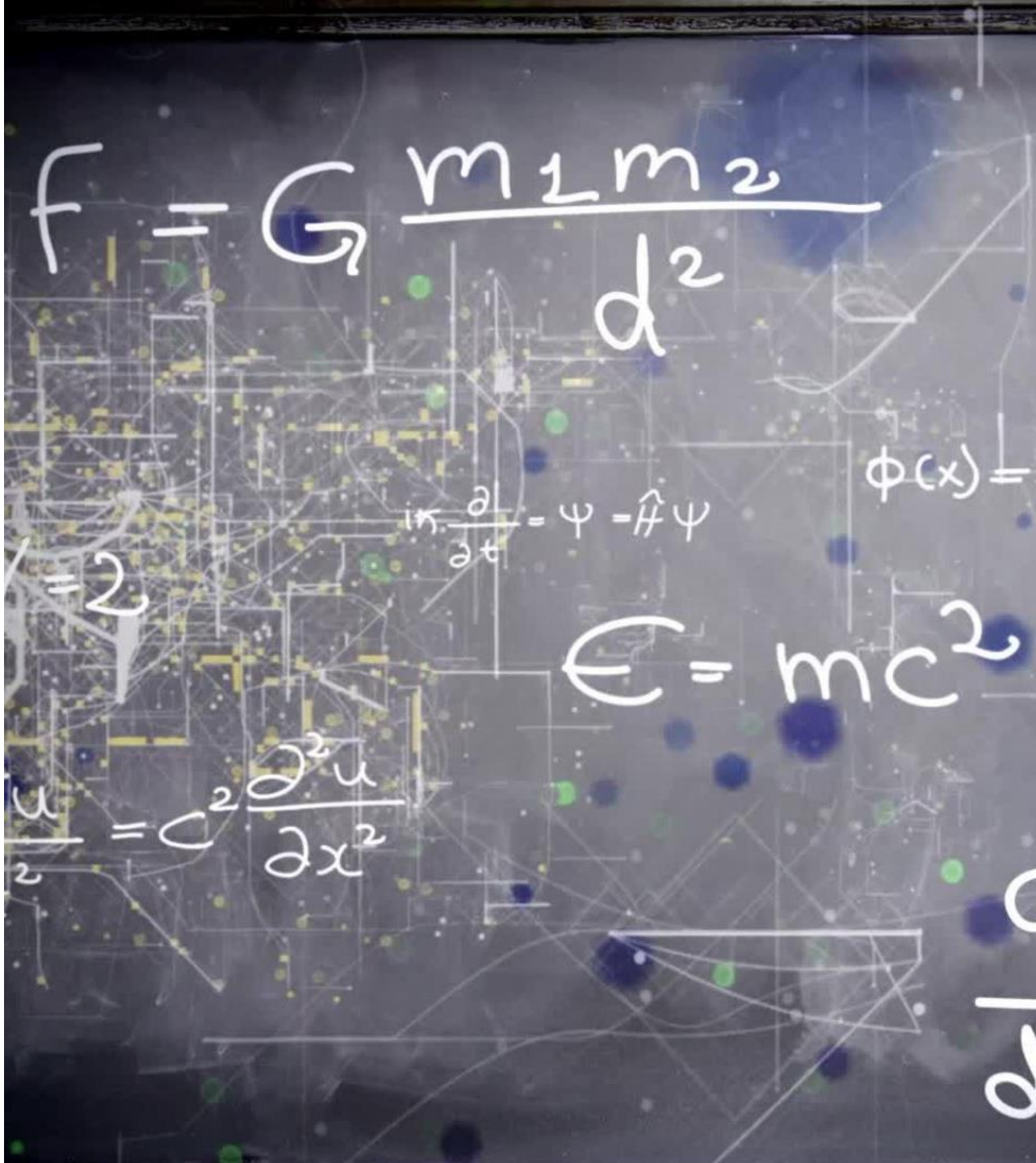
- String valor = "30";
- int cantidad = valor.equals("") ? 0 : Integer.parseInt(valor);

if-else

- El flujo del programa tomará **un camino** u otro **en base a la condición** que antes se cumpla
- Sólo se ejecuta **un camino** (el primero cuya condición se cumpla), incluso aunque varias condiciones puedan cumplirse
- La parte **else if** es **opcional**
- La parte **else** es **opcional**
- Si dentro de la parte **if, else if o else** sólo se va a escribir una instrucción, se pueden **omitir las llaves**, aunque **no** se recomienda
- Se **recomienda añadir siempre un caso else**, incluso cuando parezca que no es necesario

switch-case

- El flujo del programa tomará **un camino** u otro **en b ase a la condición** que antes se cumpla
- Existe la posibilidad de que se ejecuten **varios casos**, siempre y cuando no se rompa la ejecución del primero con la instrucción break;
- El caso **default** es **opcional**
- Se recomienda utilizar para aquellos casos en los qu e haya más de dos posibilidades. En caso contrario se recomienda utilizar una sentenc ia **if-else**
- Se **recomienda añadir siempre un caso default**, in cluso cuando no parece que se necesite.



EJERCICIO

while

- Ejecuta el conjunto de instrucciones **mientras se cumpla la condición**
- **La condición debe cumplirse para que el bucle se ejecute al menos una vez**

do-while

- Ejecuta el conjunto de instrucciones **mientras se cumpla la condición**
- Siempre **se ejecuta, al menos, una vez**, se cumpla o no la condición de permanencia

for

- Se ejecuta el conjunto de instrucciones mientras se cumpla la condición (segunda expresión de la definición del for)
- Ideal en los **casos en que sabemos el número de veces** que algo debe ejecutarse

for-each

- Se ejecuta el conjunto de instrucciones **tantas veces como elementos haya en la colección que se recorre**. Además, en cada iteración se sirve cada elemento de la lista en la variable que se define en el for-each
- Ideal para **recorrer colecciones de datos** y cuando se necesita recorrerla **de inicio a fin**

break

- La instrucción **break** permite **salir repentinamente del bucle**
- Siempre **debe ir dentro de un bloque condicional** if puesto que de otra manera se ejecutaría siempre y el bucle no tendría sentido
- Es una forma de **terminar un bucle bajo condiciones excepcionales** que quizás complicarían la definición de la condición de permanencia si se definieran alli.

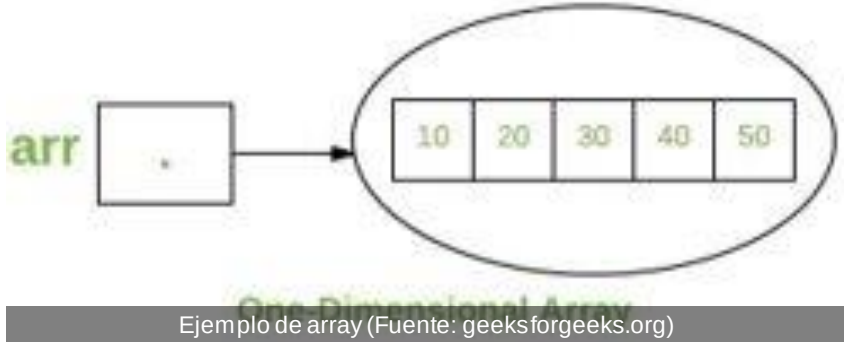
continue

- La instrucción **continue** permite **terminar la iteración actual** de un bucle y fuerza que se continúe con la siguiente
- Siempre **debe ir dentro de un bloque condicional** if puesto que de otra manera se ejecutaría siempre y el bucle no tendría sentido
- Es una **forma de terminar una iteración concreta bucle bajo condiciones excepcionales** que quizás complicaría la lógica dentro de dicho bucle.

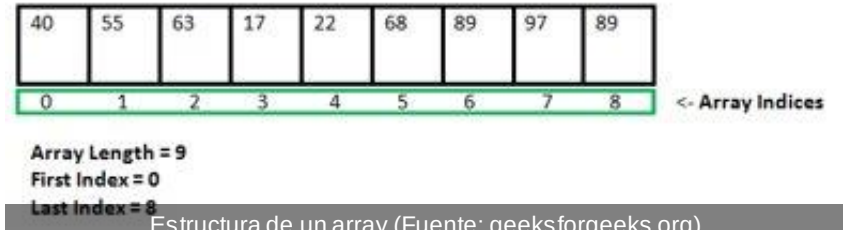
EJERCICIO

Vector / Array

- Un vector o array es una estructura de memoria que permite almacenar un **conjunto agrupado de valores de mismo tipo**.
- Para definirlo se especifica el **tipo de dato** de los valores que contendrá y la **capacidad** de éste, que será el número de valores que podrá almacenar.
- Cada elemento del array es **accesible** a través del **nombre** del array y la **posición** que ocupa dentro de éste
- La **primera posición** de una array siempre será la posición **0**



Ejemplo de array (Fuente: geeksforgeeks.org)



Estructura de un array (Fuente: geeksforgeeks.org)

Declarar un vector

- Vector de **enteros**. **Sin tamaño**. **Sin instanciar**
    - **int**[] notas;
- Vector de **enteros.** Tamaño **2, inicializado** (valores por defecto = 0)
    - **int**[] notas = **new int**[2];
- Vector de **cadenas**. Tamaño **2**, **inicializado** con valores
    - **String**[] palabras = **new String**[]{"una", "dos"};

Cómo acceder a los valores de un vector

- Es posible acceder a todos los valores de un vector utilizando un bucle for para acceder posición a posición (mayor control)

```
for (int i = 0; i < notas.length; i++) {

System.out.println(notas[i]);

}
```

- Es posible acceder a cada uno de los elementos de un vector utilizando un bucle for-each (de principio a fin)

```
for (int nota : notas)
{ System.out.println(nota)
;

}
```

Cómo acceder a los valores de un vector

- También es posible acceder a cualquier posición directamente

- **int**[]
notas =
new int[2];
notas[0] =
10;
- notas[1] = 5;
- System.out.println("Nota primer examen: " + notas[0]);
- System.out.println("Nota segundo examen: " + notas[1]);

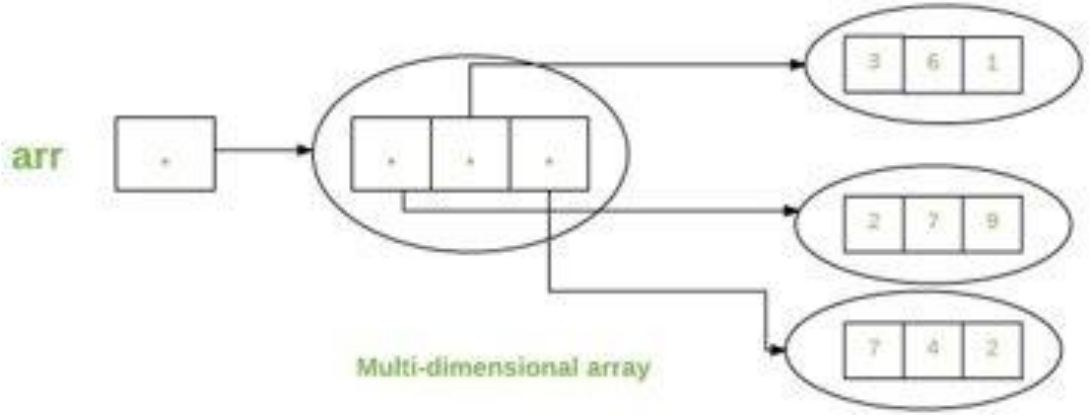
Operaciones con vectores

- En Java existe la clase Arrays para realizar operaciones con vectores:

  - **int** binarySearch(vector[], valor): Busca el valor especificado en el vector y devuelve su posición. En caso de que no lo encuentre devolverá el valor -1
  - vector[] copyOf(vector[], nuevaLongitud): Recorta o alarga el vector que se pasa como parámetro para que se
  - ajuste a la nueva longitud. Devuelve el nuevo vector modificado.
  - fill(vector[], valor): Rellena el vector que se pasa como parámetro con el valor especificado en todas sus posiciones
  - sort(vector[]): Ordena todos los valores del vector
  - String toString(vector[]): Devuelve el contenido del vector representado por una cadena

Vectores de dos dimensiones

- A los vectores de dos dimensiones se les conoce también con el nombre de matrices
- Funcionan igual que los vectores uni-dimensionales pero en este caso hay dos posiciones que indicar: fila y columna
    - **int**[][] valores = **new int**[2][5]
- También se pueden considerar como *"vectores de vectores"*, puesto que cada posición del vector contiene otro vector



String

- Una cadena de texto es una **secuencia de caracteres** (cualquiera)
- En Java se definen como el tipo **String** y su tamaño es **variable** (y no es necesario especificarlo)
- En Java se permite asignar valor a un String c**omo si se tratara de un tipo primitivo** pero realmente **es un objeto** (más adelante hablaremos de ello pero conviene conocer este dato ya)
- La cadena de texto es un **objeto inmutable** por lo que todas las operaciones que invoquemos sobre ella nunca la modifican, sino que devuelven su valor modificado (que tendrá que volver a ser almacenado sobre si misma u otra cadena)

Declarar un vector

- Cadena de texto con **valor nulo** (es la forma de especificar que no tiene valor)
    - String nombre = null;
- Cadena de **texto vacía** (es un valor válido)
    - String nombre = “”;
- Cadena de texto con **valor**
    - String nombre = “Santiago Faci”;
- Cadena de texto c**ompuesta de dígitos** (sigue siendo una cadena de texto)
    - String numero = “123”;

Cuándo utilizar una cadena de texto

- Por supuesto, usaremos una cadena cuando queramos almacenar texto.
- A veces, incluso cuando el valor que se quiere almacenar está compuesto exclusivamente de dígitos, éste debe almacenarse como cadena de texto, puesto que no son valores numéricos realmente ya que no soportan ninguna operación matemática ni se espera que se haga ninguna con ellos.
- Estos son algunos ejemplos
  - DNI
  - Número de la Seguridad Social
  - Cuenta Bancaria
  - Teléfono

Casos de uso con cadenas

- Un usuario introduce su nombre y apellidos en una caja de texto
  - Convertir a mayúscula/minúscula y eliminar los espacios a inicio y final
- Un usuario introduce su número de cuenta
  - Convertir la cadena en un array de caracteres para realizar el cálculo del digito de control
- Un usuario introduce su número de DNI (con letra)
  - Convertimos el valor a un número para hacer los cálculos y luego lo almacenamos todo junto (con letra) concatenando.
- Un usuario introduce un precio en un formulario
  - Tras las comprobaciones (caja de texto vacía o valor sólo compuesto por dígitos) tendremos que parsearlo al tipo de dato que corresponda (float en este caso)

EJERCICIO