

ESCUELA DE CÓDIGO

MÓDULO

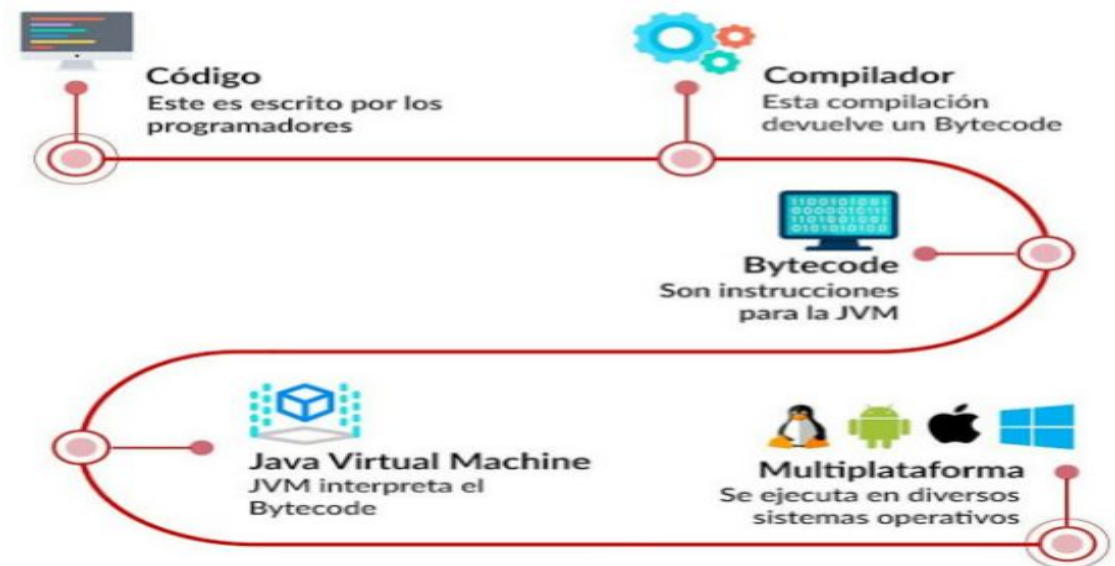
PROGRAMADOR JUNIOR-JAVA



¿Qué es Java?

- ▶ programación de alto nivel, robusto, orientado a objetos y seguro. Como plataforma es una colección de programas que ayudan a los programadores a desarrollar y ejecutar aplicaciones de programación Java de manera eficiente. Incluye un motor de ejecución, un compilador y un conjunto de bibliotecas.

¿Cómo funciona Java? ☕



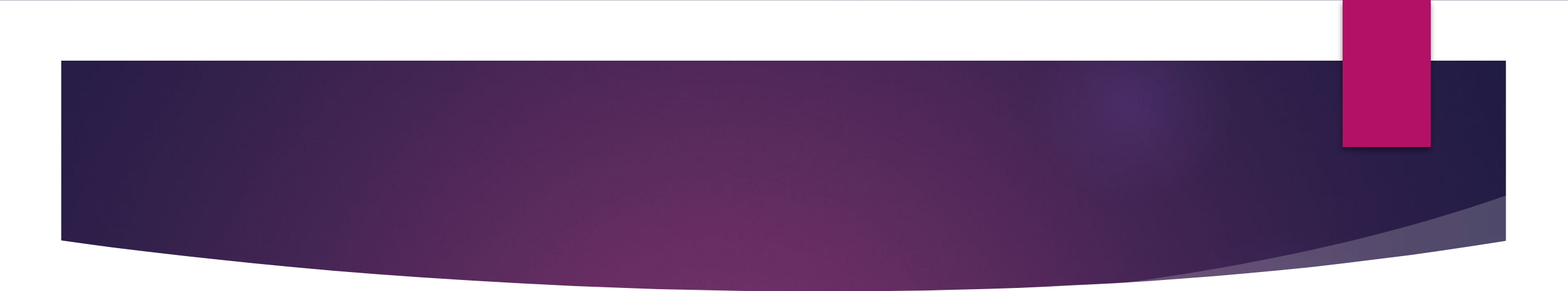
- ▶ Los lenguajes de programación se componen de **sintaxis**, las instrucciones específicas que entiende Java. Escribimos sintaxis en archivos para crear programas, que son ejecutados por la computadora para realizar la tarea deseada.

¿Para qué se usa Java?

- ▶ Utilizado para desarrollar aplicaciones de Android
- ▶ ● Ayuda a crear software empresarial
- ▶ ● Amplia gama de aplicaciones java móviles
- ▶ ● Aplicaciones de Computación Científica
- ▶ ● Uso para análisis de Big Data
- ▶ ● Programación en Java de dispositivos de Hardware
- ▶ ● Se utiliza para tecnologías del lado del servidor como Apache, JBoss, GlassFish, etc.

Componentes del lenguaje de programación Java

- ▶ Kit de desarrollo de Java (JDK). JDK es un entorno de desarrollo de software utilizado para crear applets y aplicaciones Java. El nombre completo de JDK es Java Development Kit. Los desarrolladores de Java pueden usarlo en Windows, macOS, Solaris y Linux. JDK les ayuda a codificar y ejecutar programas Java. Es posible instalar más de una versión de JDK en la misma computadora.

- 
- ▶ Máquina Virtual Java (JVM). Java Virtual Machine (JVM) es un motor que proporciona un entorno de tiempo de ejecución para controlar el código Java o las aplicaciones. Convierte el código de bytes de Java en lenguaje de máquina.
 - ▶ JVM es una parte de Java Run Environment (JRE). En otros lenguajes de programación, el compilador produce código de máquina para un sistema en particular. Sin embargo, el compilador de Java produce código para una máquina virtual conocida como máquina virtual de Java.

¿Por qué JVM?

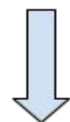
- JVM proporciona una forma independiente de la plataforma de ejecutar el código fuente de Java.
- Tiene numerosas bibliotecas, herramientas y marcos.
- Una vez que ejecuta un programa Java, puede ejecutarse en cualquier plataforma y ahorrar mucho tiempo.
- JVM viene con el compilador JIT (Just-in-Time) que convierte el código fuente de Java en un lenguaje de máquina de bajo nivel. Por lo tanto, se ejecuta más rápido que una aplicación normal..



Entorno de tiempo de ejecución de Java (JRE).

JRE es una pieza de software que está diseñada para ejecutar otro software. Contiene las bibliotecas de clases, la clase de cargador y la JVM. En términos simples, si deseas ejecutar un programa Java, necesitas JRE. Si no eres programador, no necesitas instalar JDK, solo JRE para ejecutar programas Java.

```
12 interface Video {  
13     val brightness: Int  
14  
15     val contrast: Int  
16     get() = 100  
17  
18     fun play() {  
19         println("Play")  
20     }  
21  
22     fun pause() (Fuente) .kt  
23 }
```



BYTECODE

```
2  nop  
3  new java/lang/Throwable  
4  dup  
5  invokespecial java/lang/Throwable/<init>()V  
6  invokevirtual java/lang/Throwable/getStackTrace()[Ljava/lang/StackTraceElement;  
7  astore_1  
8  iconst_0  
9  istore_0  
10 iload_0  
11 aload_1  
12 arraylength  
13 if_icmpge 60  
14 nop  
15 nop  
(Bytecode) .class
```

JRE Windows



JRE Linux



JRE Mac



Kotlin/JVM

kotlinc

JDK

javac

javadoc

jdb

jar

JRE

java

Código fuente en Java

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hola Oregoom");  
    }  
}
```



Compilar con javac
a Bytecode
-Se genera un archivo .class



Luego es interpretado
a código máquina

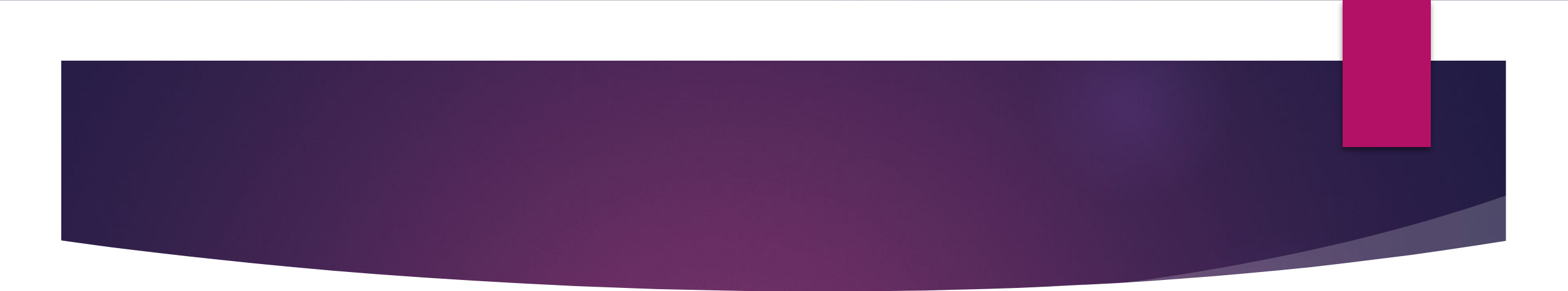
JVM

Resultado



Diferentes tipos de plataformas Java

- ▶ Plataforma Java, edición estándar (Java SE): la API de Java SE ofrece la funcionalidad principal del lenguaje de programación Java. Define toda la base de tipo y objeto para clases de alto nivel. Se utiliza para redes, seguridad, acceso a bases de datos, desarrollo de interfaz gráfica de usuario (GUI) y análisis XML.

- 
- ▶ Java Platform, Enterprise Edition (Java EE): la plataforma Java EE ofrece una API y un entorno de tiempo de ejecución para desarrollar y ejecutar aplicaciones de red altamente escalables, a gran escala, de múltiples niveles, confiables y seguras.

- 
- ▶ **Plataforma de lenguaje de programación Java, Micro Edition (Java ME):** la plataforma Java ME ofrece una API y una máquina virtual de tamaño reducido que ejecuta aplicaciones del lenguaje de programación Java en dispositivos pequeños, como teléfonos móviles.
 - ▶ **Java FX:** JavaFX es una plataforma para desarrollar aplicaciones ricas de Internet utilizando una API de interfaz de usuario liviana. Usa motores gráficos y de medios acelerados por hardware que ayudan a Java a aprovechar los clientes de mayor rendimiento y una apariencia moderna y API de alto nivel para conectarse a fuentes de datos en red.

COMENTARIOS

Existen tres estilos permitidos de comentarios en programas de Java, son:

```
// comentario en una línea

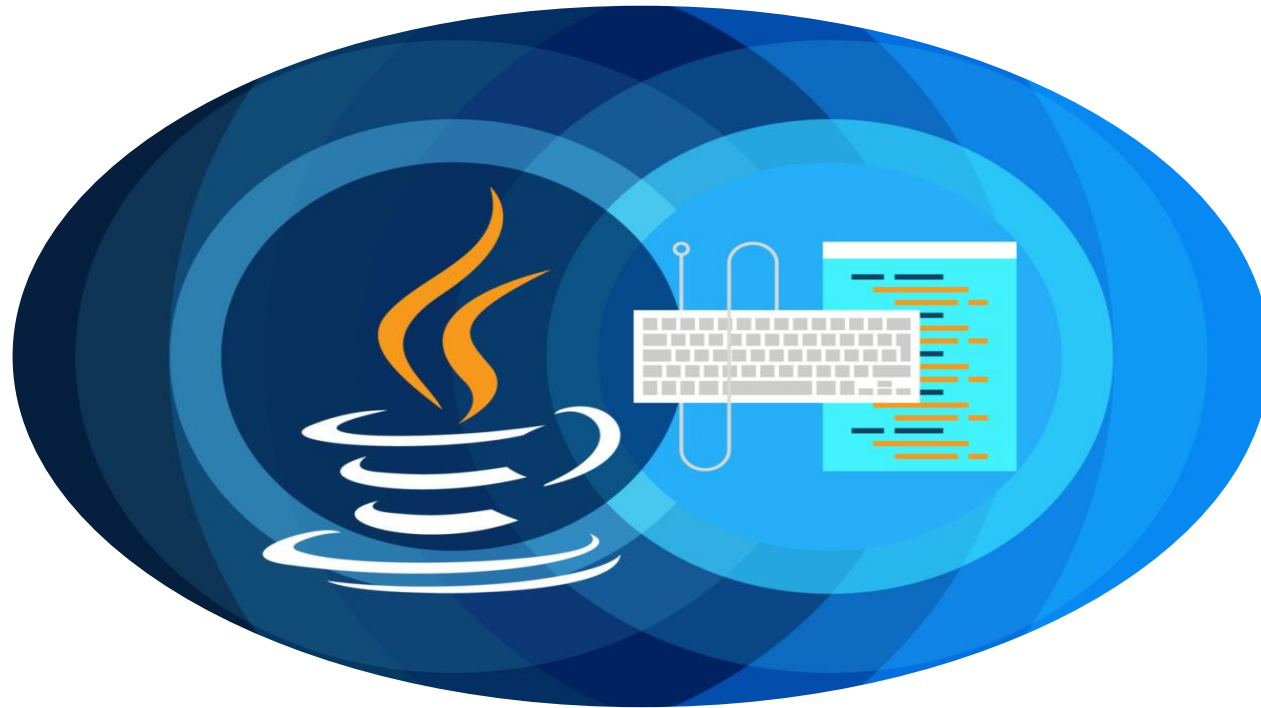
/* comentario en una
    o mas líneas */

/** comentario de documentación */
```

- Proporcionan notas legibles para humanos que aclaran el pensamiento.

Primer programa en Java

HOLA MUNDO



TEMA 2. VARIABLES

- ▶ Almacenamos información en variables, ubicaciones con nombre en la memoria.
- ▶ Nombrar una pieza de información nos permite usar ese nombre más tarde, accediendo a la información que almacenamos.
- ▶ Las variables también dan contexto y significado a los datos que almacenamos. El valor 42 podría ser la edad de alguien, el peso en libras o la cantidad de pedidos realizados.
- ▶ Con un nombre, sabemos que el valor 42 es edad, peso en kilos o números de pedidos hechos.

```
int velocidadLimite = 80;
```

```
int velocidadLimite;  
velocidadLimite = 80;
```

Algo importante que debes saber es que Java es un lenguaje de tipo estático. Significa que todas las variables deben declararse antes de que puedan usarse.

- ▶ Las variables pueden comenzar con una letra o un guión bajo, _ o signo de pesos,
- ▶ \$. Por ejemplo,

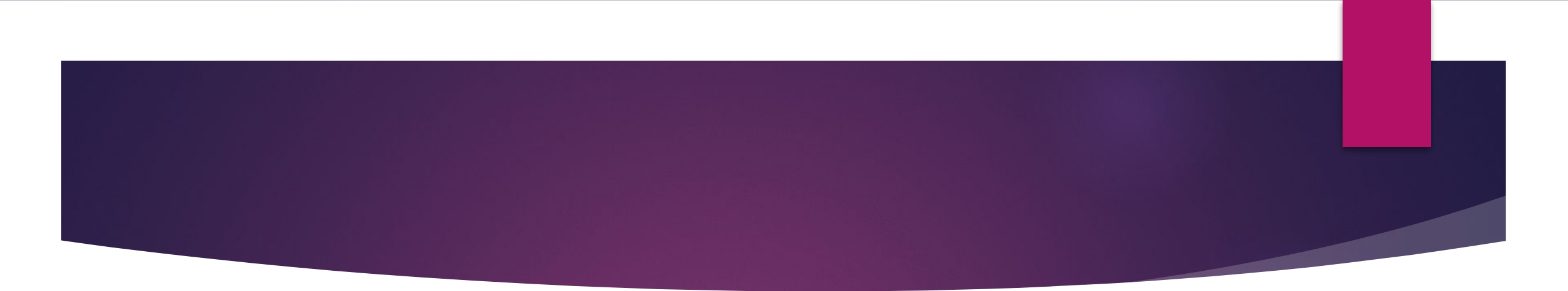
```
int edad; // nombre válido y buena práctica  
int _edad; // válido pero mala práctica  
int $edad; // válido pero mala práctica
```

- ▶ Los nombres de variables no pueden comenzar con números. Por ejemplo,

```
int 1edad; // variable inválida
```

- ▶ Los nombres de variables no pueden usar espacios en blanco. Por ejemplo,

```
int mi edad; // variable inválida
```

En este caso, si necesitas usar nombres de variables que tengan más de una palabra, usa todas las letras en minúsculas para la primera palabra y escribe en mayúscula la primera letra de cada palabra subsiguiente. Por ejemplo, miEdad.

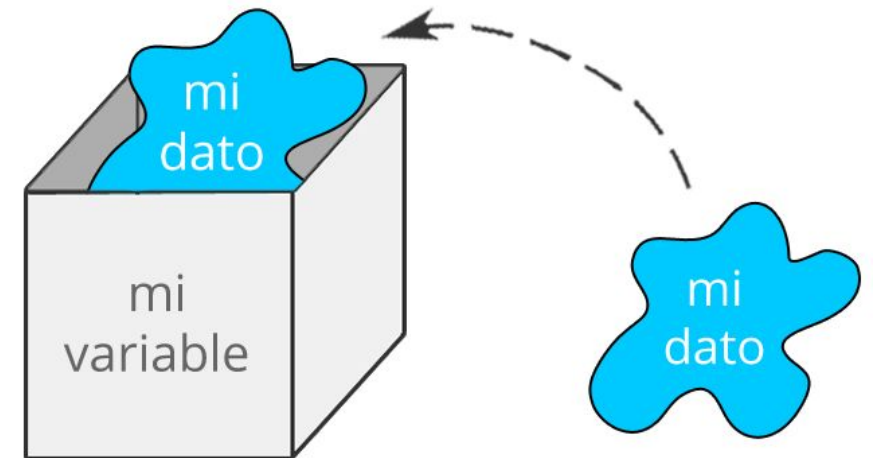
- Al crear variables, elije un nombre que tenga sentido.

Por ejemplo, puntuacion,numero, piso tiene más sentido que nombres de variables como s, n y p.

- Si eliges nombres de variables de una sola palabra, utiliza todas las letras en minúsculas. Por ejemplo, es mejor usar velocidad en lugar de VELOCIDAD o vELOCIDAD.

Hay tres tipos de variables en Java:

1. Variable local. Una variable declarada dentro del cuerpo de un método (method) se llama variable local. Puedes usar esta variable solo dentro de ese método y los otros métodos en la clase (class) ni siquiera sabrán que la variable existe. Una variable local no se puede definir con la palabra clave "static".
2. Variable de instancia. Una variable declarada dentro de la clase (class) pero fuera del cuerpo de un método (method) se denomina variable de instancia. No se declara como estático. Se denomina variable de instancia porque su valor es específico de la instancia y no se comparte entre instancias.
3. Variable estática. Una variable que se declara como static se denomina variable estática. No puede ser local. Puedes crear una sola copia de la variable estática y compartirla entre todas las instancias de la clase. La asignación de memoria para variables estáticas ocurre solo una vez cuando la clase se carga en la memoria.



TIPOS DE DATOS EN JAVA

- ▶ Tipos de datos primitivos: incluye int, double, boolean, char, byte, short, long y float
- ▶ ● Tipos de datos no primitivos: como String, Arrays y Classes.



Tipos de datos primitivos

➤ Los tipos de datos primitivos en java son ocho:

Grupos	Tipo de dato primitivo	Valores que permite almacenar en una variable
Enteros	byte	Enteros entre -128 y 127.
	short	Enteros entre -32768 y 32767.
	int	Enteros grandes. Enteros entre -2 147 483 648 y 2 147 483 647.
	long	Enteros muy grandes. Enteros entre -9 223 372 036 854 775 808 y 9 223 372 036 854 775 807.
Decimales	float	Decimales usando punto decimal. Ej: 54.23
	double	Decimales con el doble de precisión que el float.
Caracter	char	Un único carácter entre comillas simples. Ej: 'p'.
Verdad o falsedad	boolean	true o false. El valor true indica verdadero y el valor false indica falso .

DATOS INT

- ▶ En Java, los números enteros se almacenan en el tipo de datos primitivo int.
- ▶ ● ints contienen números positivos, números negativos y cero. No almacenan fracciones o números con decimales en ellos.
- ▶ ● El tipo de datos int Almacena números enteros desde -2,147,483,648 hasta 2,147,483,647
- ▶ ● El valor por default de int es 0.

DATOS DOUBLE

- ▶ El tipo de datos primitivo double puede ayudar.
- ▶ ● double puede contener decimales, así como números muy grandes y muy pequeños.
- ▶ ● El tipo de datos double es un punto flotante de precisión doble de 64 bits.
- ▶ ● El valor por default de double es 0.0d

DATOS BOOLEANO

A menudo, nuestros programas enfrentan preguntas que solo pueden responderse con un sí o un no.

Declaramos variables booleanas usando la palabra clave boolean antes del nombre

```
boolean javaEsUnLenguajeCompilado = true;
```

```
boolean javaEsUnaTazaDeCafe = false;
```

de la variable.

El valor por default de boolean es false.

- El tipo de datos booleano especifica un bit de información, pero su "tamaño" no se puede definir con precisión.

DATOS CHAR

El tipo de datos char puede contener cualquier carácter, como una letra, un espacio o un signo de puntuación.

- Debe estar entre comillas simples, ' '.
- Es un carácter Unicode de 16 bits.
- Su valor por default es '\u0000' (Java usa el sistema Unicode, no el sistema de código ASCII. El \u0000 es el rango más bajo del sistema Unicode.).

DATOS STRINGS

El tipo de datos String se utiliza para almacenar una secuencia de caracteres (texto). Los valores de String (cadena) deben estar entre comillas dobles. Las cadenas (String) en Java no son tipos primitivos, sino objetos. Ya hemos visto ejemplos de String, por ejemplo, cuando imprimimos "¡Hola Mundo!".

Operadores en Java

Los operadores son símbolos que realizan operaciones sobre variables y valores. Por ejemplo, + es un operador que se usa para sumar, mientras que * también es un operador que se usa para multiplicar.

Los operadores en Java se pueden clasificar en 5 tipos:

- ▶ 1. Operadores aritméticos
- ▶ 2. Operadores de Asignación
- ▶ 3. Operadores relacionales
- ▶ 4. Operadores lógicos
- ▶ 5. Operadores unarios

Operadores aritméticos

Operador	Operación
+	Suma
-	Sustracción
*	Multiplicación
/	División
%	Operación Módulo (resto/residuo después de la división)

Operadores de Asignación

Operador	Ejemplo	Equivalente a
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Los operadores de asignación compuesta realizan una operación aritmética en una variable y luego reasignan su valor.

Operadores relacionales

Operador	Descripción	Ejemplo
<code>==</code>	Es igual a	<code>3 == 5</code> devuelve false
<code>!=</code>	No igual a	<code>3 != 5</code> devuelve true
<code>></code>	Mayor que	<code>3 > 5</code> devuelve false
<code><</code>	Menor que	<code>3 < 5</code> devuelve true
<code>>=</code>	Mayor o igual que	<code>3 >= 5</code> devuelve false
<code><=</code>	Menor o igual que	<code>3 <= 5</code> devuelve true

Operadores lógicos

- ▶ Los operadores lógicos se utilizan para comprobar si una expresión es verdadera o falsa. Se utilizan en la toma de decisiones.

Operador	Ejemplo	Significado
&& (Logical AND)	expresion1 && expresion2	verdadero (true) solo si tanto la expresion1 como la expresion2 son verdaderas
 (Logical OR)	expresion1 expresion2	Verdadera si expresion1 o expresion2 es verdadera
! (Logical NOT)	!expresion	verdadero si la expresion es falsa y viceversa

Operadores unarios

- ▶ Los operadores unarios se utilizan con un solo operando. Por ejemplo, ++ es un operador unario que aumenta el valor de una variable en 1. Es decir, ++5 devolverá 6.

Operador	Significado
+	Más unario: no es necesario usarlo ya que los números son positivos sin usarlo
-	Menos unario: invierte el signo de una expresión
++	Operador de incremento: incrementa el valor en 1
--	Operador de disminución: disminuye el valor en 1
!	Operador de complemento lógico: invierte el valor de un booleano

TEMA EXTRA USANDO LA CLASE Scanner

Método	Explicación
boolean nextBoolean()	Lee valores lógicos booleanos introducidos por el usuario.
byte nextByte()	Lee valores byte introducidos por el usuario.
double nextDouble()	Lee valores double introducidos por el usuario.
float nextFloat()	Lee valores float introducidos por el usuario.
int nextInt()	Lee valores int introducidos por el usuario.
String nextLine()	Lee valores String introducidos por el usuario.
long nextLong()	Lee valores long introducidos por el usuario.
short nextShort()	Lee valores short introducidos por el usuario.

TEMA 4: Sentencias if/else

El compilador de Java ejecuta el código de **arriba a abajo**. Las declaraciones en el código se ejecutan según el orden en que aparecen. Sin embargo, Java proporciona **sentencias** que se pueden usar para controlar el flujo del código Java. Estas declaraciones se denominan **declaraciones de flujo** de control. Es una de las características fundamentales de Java, que proporciona un flujo fluido de programa.

TEMA 4: Sentencias if/else

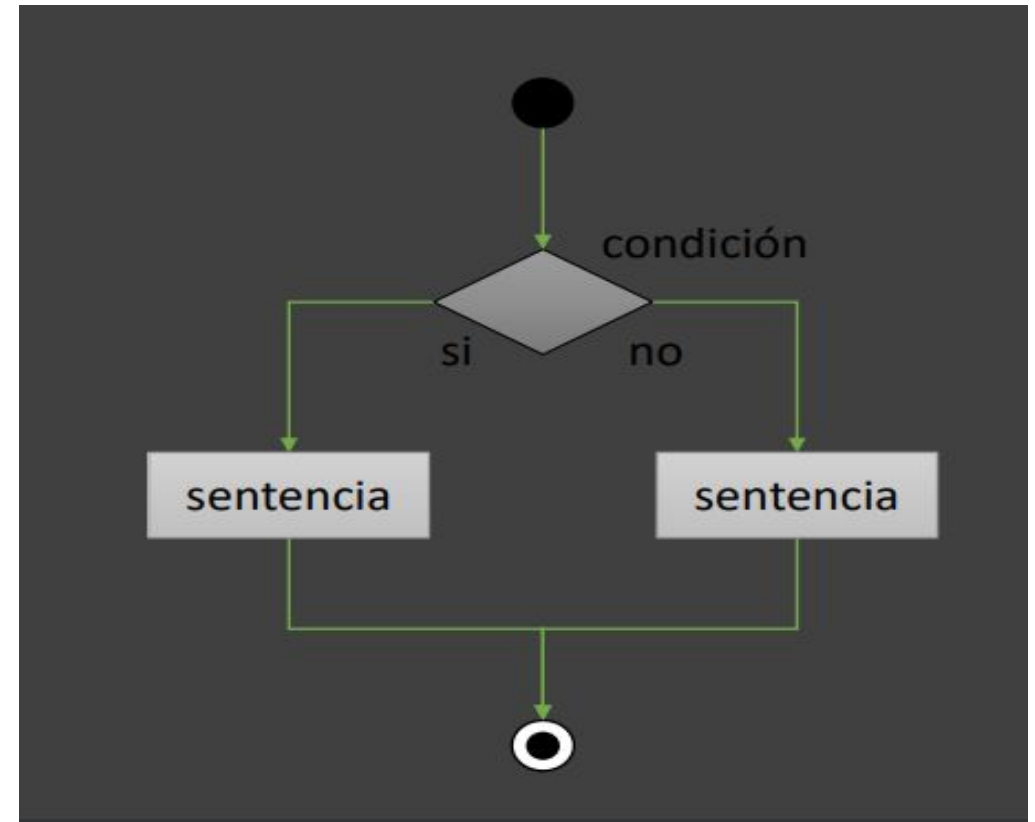
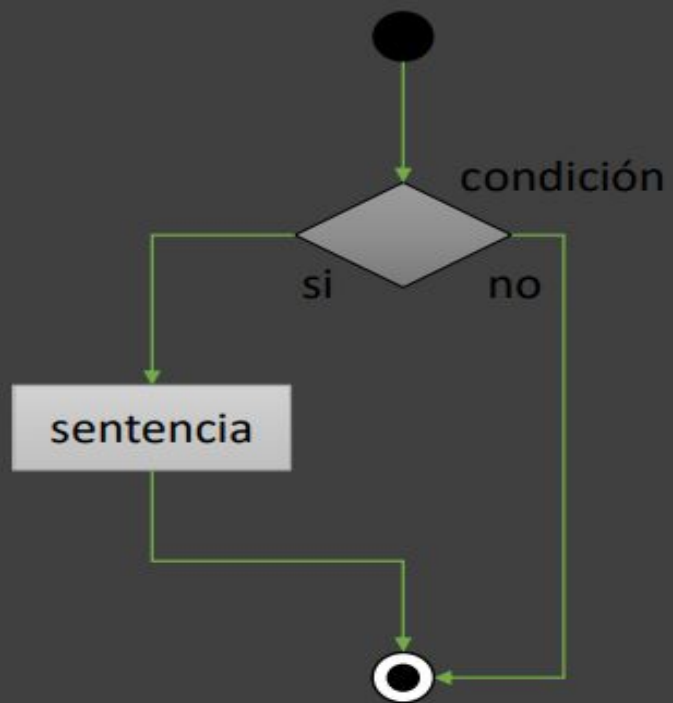
Ejecuta un bloque de código si alguna condición es verdadera

```
if(expresion) {  
    // ejecuta el bloque de código si cumple la condición  
}
```

```
if(expresion) {  
    // ejecuta el bloque de código si cumple la condición  
} else {  
    // sentencia a ejecutar si NO se cumple la condición  
}
```

```
if(expresion) {  
    // bloque a ejecutar si cumple la primera condición  
} else if(expresion) {  
    // sentencia a ejecutar si cumple segunda condición  
} else {  
    // sentencia a ejecutar si NO cumple ninguna condición  
}
```

TEMA 4: Sentencias



Bucles en Java

Un bucle (loop) es una herramienta de programación que permite a los desarrolladores repetir el mismo bloque de código hasta que se cumpla alguna condición.

El compilador primero evalúa una condición booleana. Si la condición es verdadera (true), se ejecuta el cuerpo del bucle. Cuando se ejecuta la última línea del cuerpo del ciclo, la condición se vuelve a evaluar. Este proceso continúa hasta que la condición es falsa (false). Si la condición inicial es falsa (false), el ciclo nunca se ejecuta.

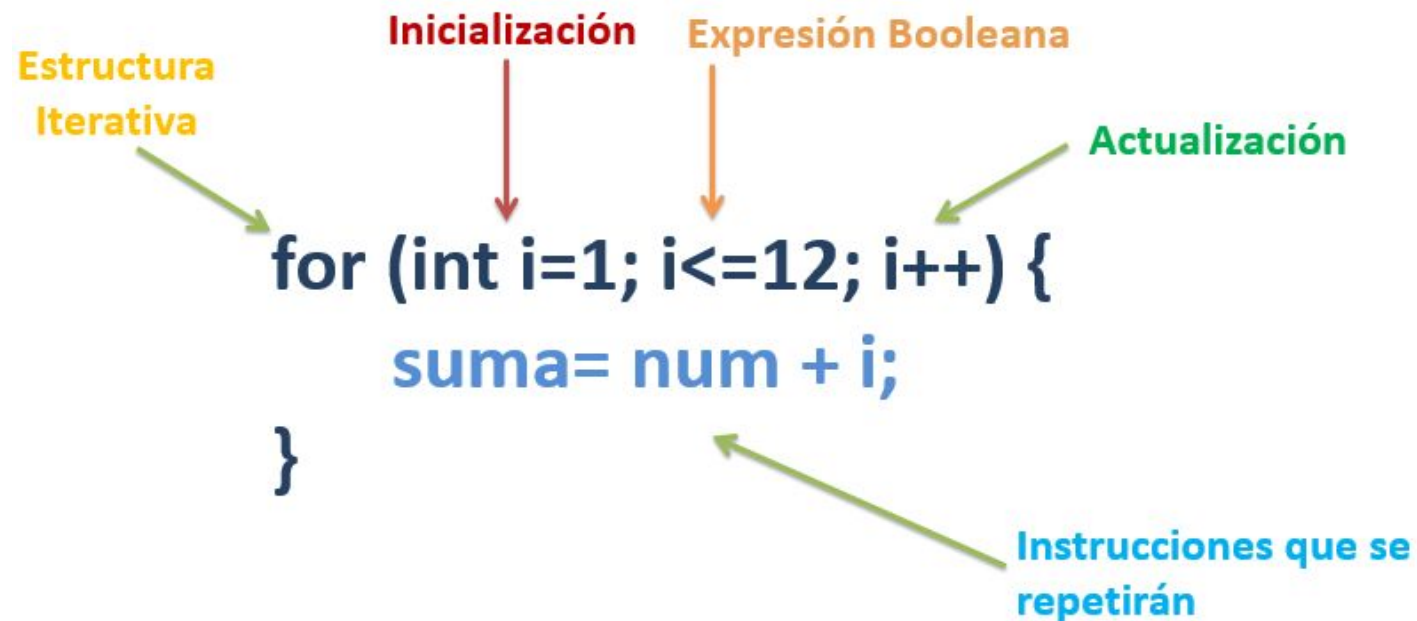
Empleamos bucles para escalar fácilmente los programas, ahorrando tiempo y minimizando los errores.

Repasaremos dos tipos de bucles que veremos en todas partes:

- ▶● el bucle for
- ▶● el bucle while

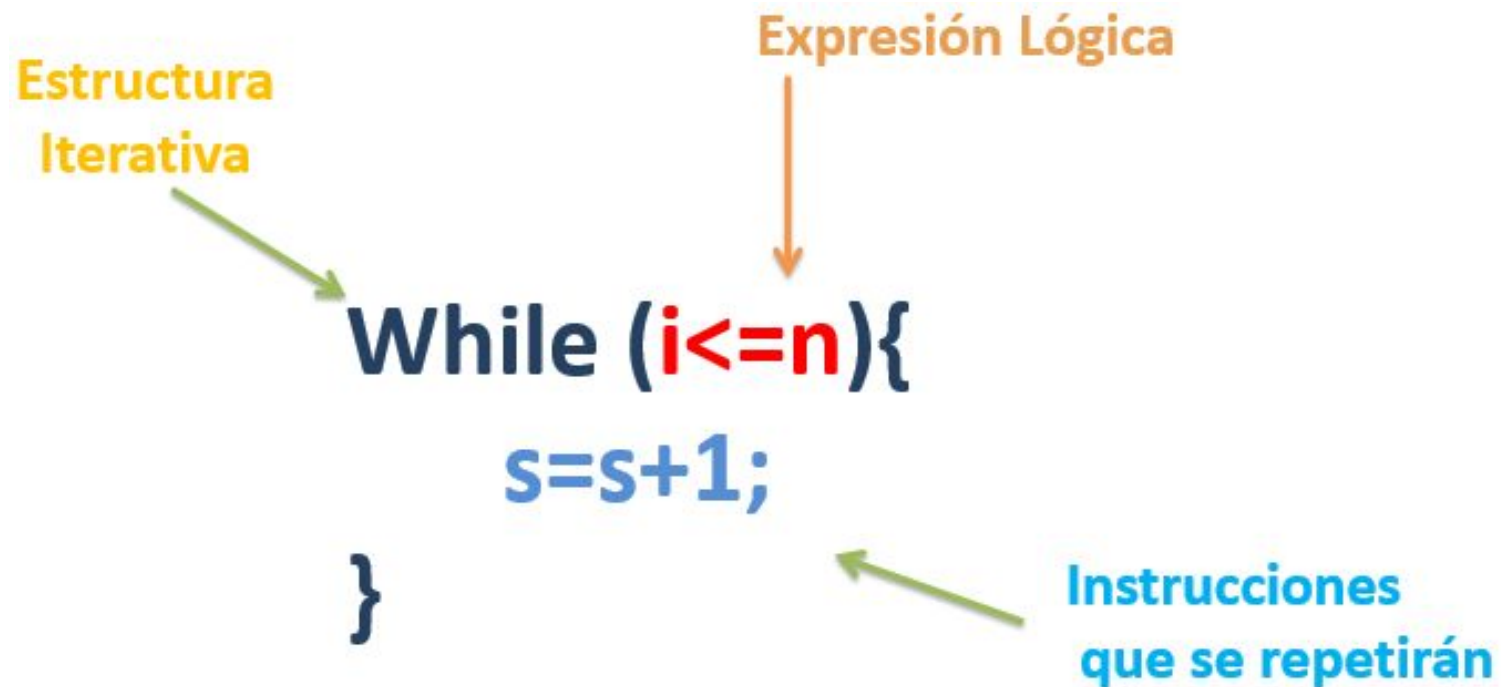
Bucle for en Java

- Incrementar con bucles es tan común en la programación que Java (como muchos otros lenguajes de programación) incluye una sintaxis específica para abordar este patrón: el bucle for.



Bucle while en Java

- El bucle while se usa para ejecutar un código específico hasta que se cumpla una condición determinada.



bucle do/while.

- ▶ El bucle siempre se ejecutará al menos una vez, incluso si la condición es falsa (false), porque el bloque de código se ejecuta antes de que se verifique la condición:

Estructura
Iterativa

do{

s=s+1;

}

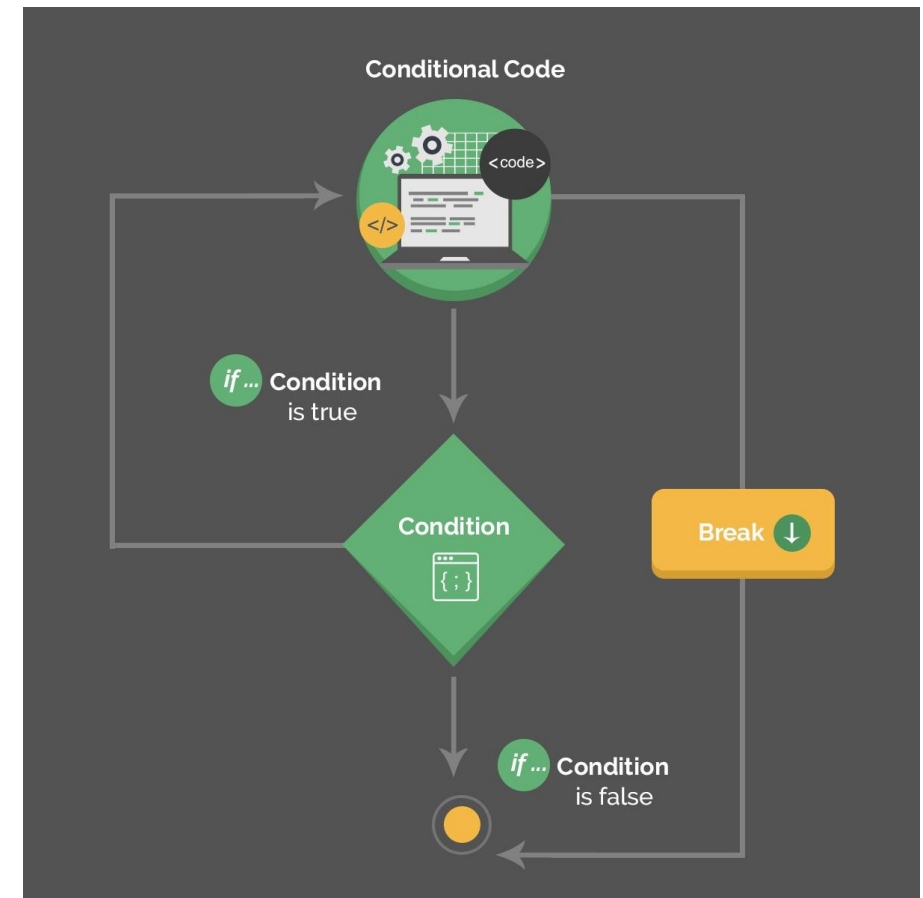
While (nota<=0 || nota>=20)

Instrucciones
que se repetirán

Expresión Lógica

Break y Continue en Java

- ▶ Si alguna vez queremos salir de un bucle antes de que finalice todas sus iteraciones o queremos omitir una de las iteraciones, podemos usar las palabras clave **break** y **continue**.
- ▶ La palabra clave **break** se utiliza para salir o interrumpir un bucle. Una vez que se ejecuta break, el bucle dejará de iterar.



Series

- ▶ Una serie es una colección numerada de objetos en la que se permiten repeticiones y el orden es importante. Las series pueden ser finitas, por ejemplo, C, A, R, L, O, S es una serie de letras con la letra 'C' primero y 'S' al final, o infinitas, como la serie de todos los números pares positivos 2, 4, 6, 8, 10, 12, 14

Serie de impares positivos

- ▶ La serie de números impares positivos es una serie formada por todos aquellos números enteros que no son múltiplos del número 2, es decir, no se puede escribir como producto del número 2 con cualquier otro número. Los primeros términos de la serie de números impares positivos son 1, 3, 5, 7, 9, 11, 13, 15
- ▶ Serie de números impares positivos: 1, 3, 5, 7, 9, 11, 13, 15, 17, ... así sigue sucesivamente.

La *serie de Fibonacci* es una serie donde el siguiente término es la suma de los dos términos anteriores. Los dos primeros términos de la serie de Fibonacci son **0** y **1**.

Serie de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... así sigue sucesivamente.

Primer término	0	
Segundo término	1	
Siguiendo término	$1 = 0 + 1$	término 3
	$2 = 1 + 1$	término 4
	$3 = 2 + 1$	término 5
	$5 = 3 + 2$	término 6
	$8 = 5 + 3$	término 7
	13	término 8
	21	término 9
	34	término 10
	...así continúa sucesivamente	

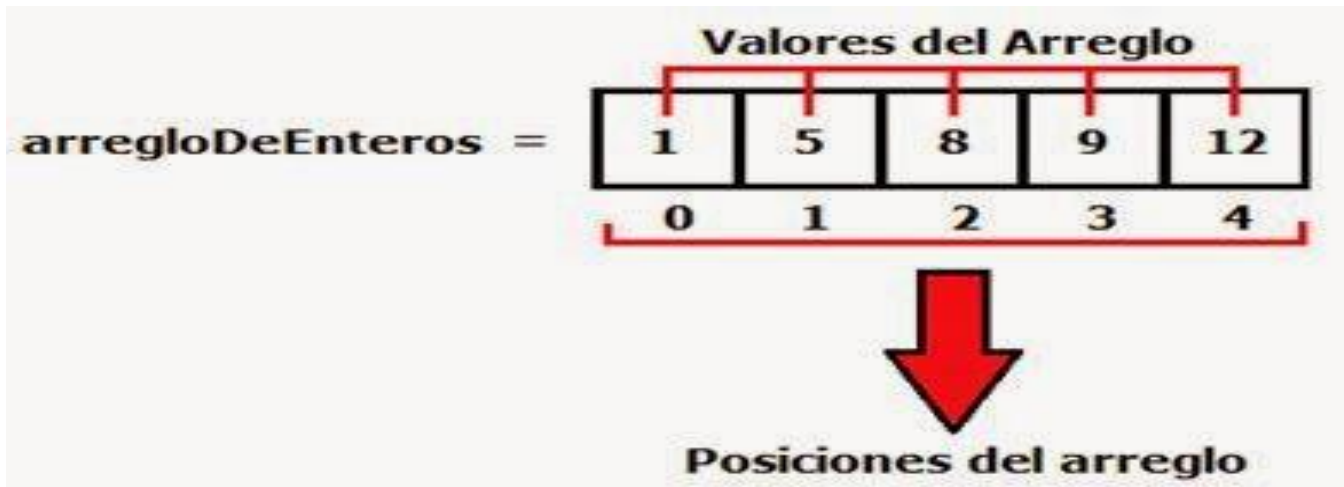
TEMA 5 . Instrucciones

- ▶ Una alternativa a encadenar condiciones if-then-else juntas es usar la sentencia switch.
- ▶ Este condicional verificará un valor dado contra cualquier número de condiciones y ejecuta el bloque de código donde haya una coincidencia.

```
switch (expresión) {  
    case valor1:  
        // código  
        break;  
    case valor2:  
        // código  
        break;  
    ...  
    ...  
    default:  
        // declaraciones predeterminadas  
}
```

TEMA 6 . Arreglos

- ▶ Nos sirven para almacenar un grupo de datos.



Arreglos unidimensionales

- ▶ Si estuviéramos almacenando 5 números de boletos de lotería, por ejemplo, tendríamos que crear una variable diferente para cada valor:

```
int primerNumero = 4;
```

```
int segundNumero = 8;
```

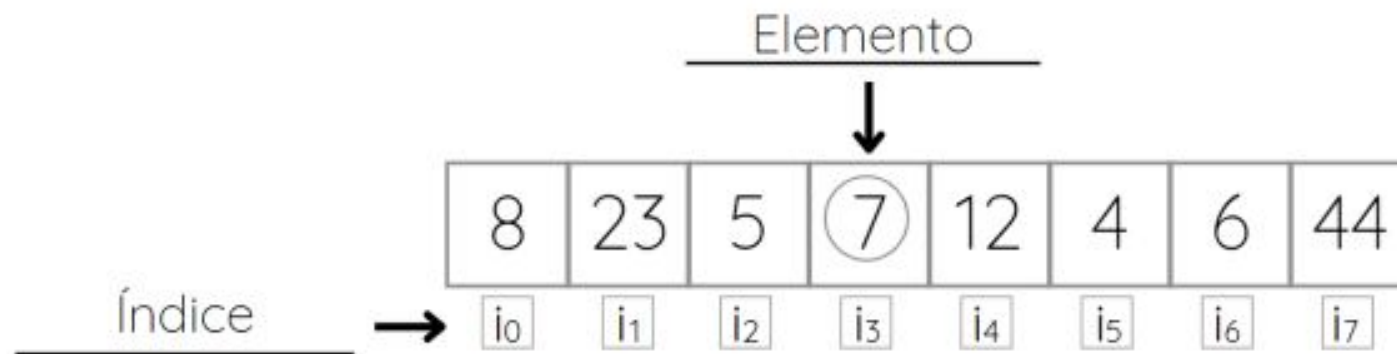
```
int tercerNumero = 15;
```

```
int cuartoNumero = 16;
```

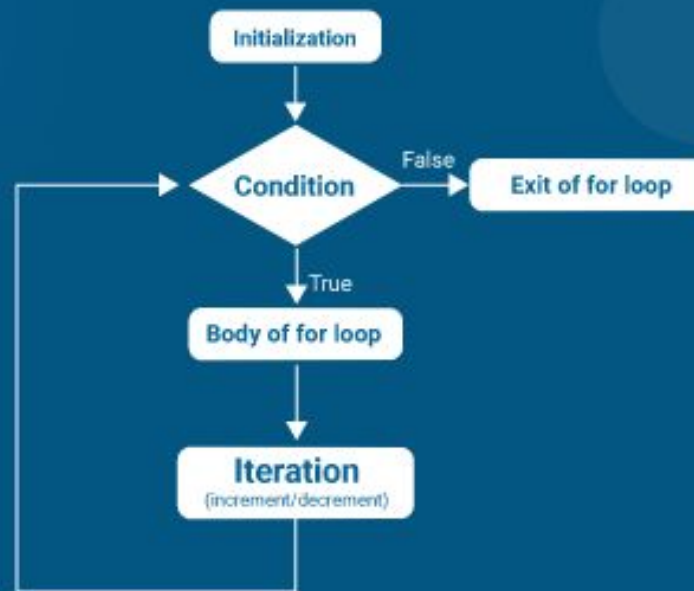
```
int quintoNumero = 23;
```

Arreglos unidimensionales

- ▶ En un arreglo, cada ubicación de memoria está asociada con un número. El número se conoce como un índice del arreglo. Cada índice de un arreglo se corresponde con un valor diferente. Aquí hay un diagrama de un arreglo lleno de valores enteros:



For-Each loop in Java



Arreglos bidimensionales

- ▶ Una matriz bidimensional es una tabla rectangular de números, símbolos o expresiones, dispuestos en filas (renglones) y columnas,

Por ejemplo, la matriz bidimensional A tiene dos renglones y tres columnas.

También se puede decir que esta es una matriz bidimensional 2 x 3, dos renglones (filas) y tres columnas.

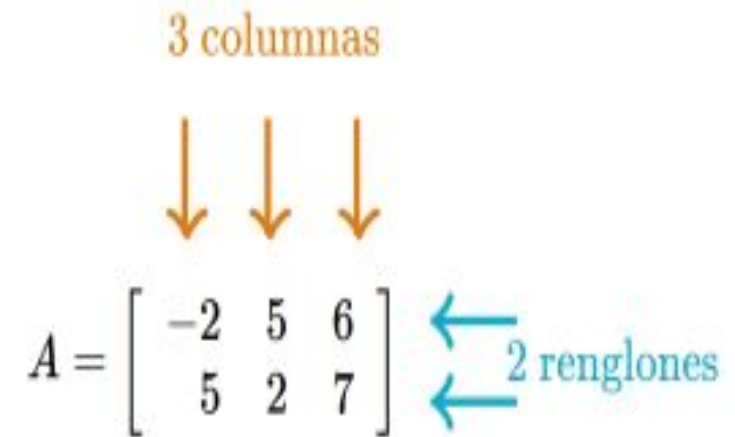
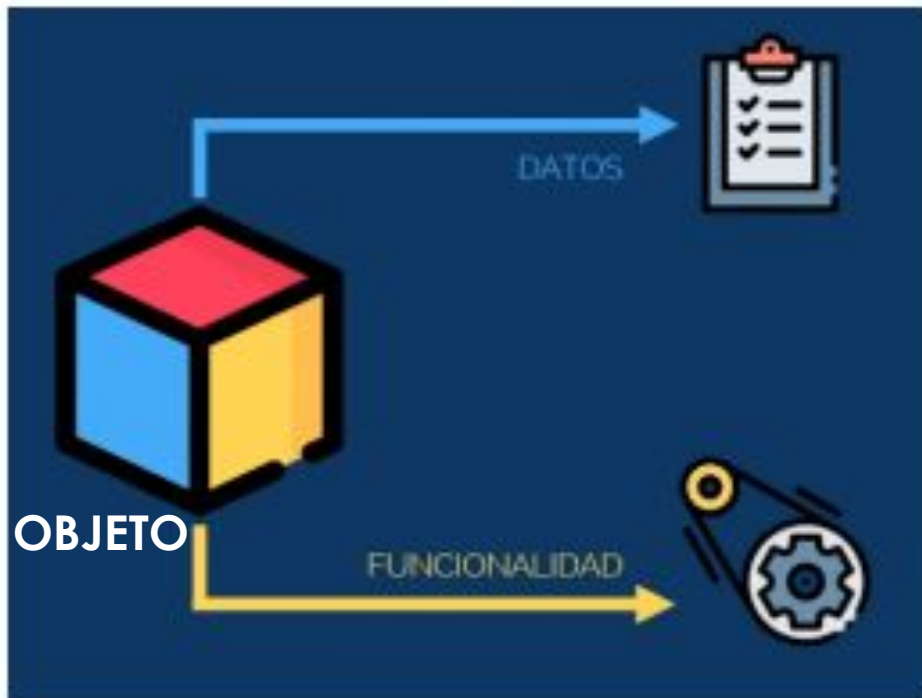


Diagram illustrating the dimensions of matrix A. The matrix is shown as $A = \begin{bmatrix} -2 & 5 & 6 \\ 5 & 2 & 7 \end{bmatrix}$. Above the matrix, three orange arrows point down to each column, labeled "3 columnas". To the right of the matrix, two blue arrows point left to each row, labeled "2 renglones".

VII. Programación Orientada a Objetos

- ▶ En la programación existe algo llamado paradigmas de programación. Un paradigma de programación es un estilo o forma de programación.

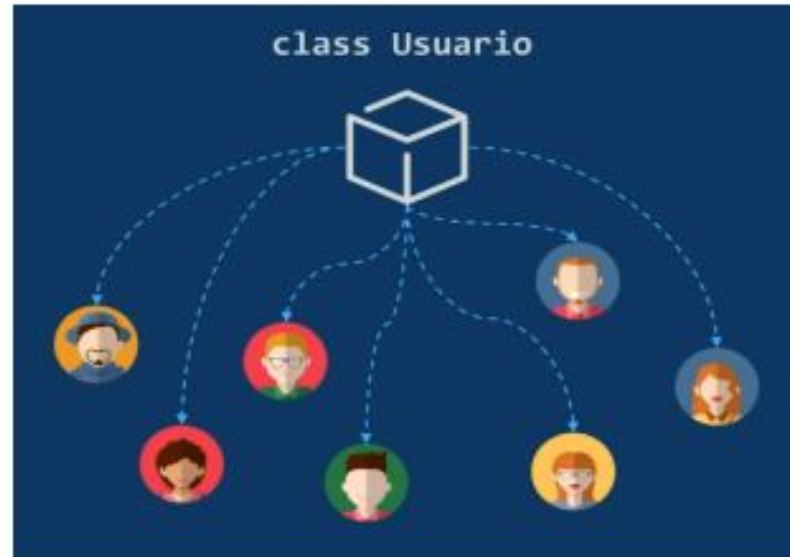


Programación
estructurada



Los objetos se comunican entre ellos.

NOTA: Como dijimos antes los **objetos** tienen **datos y funcionalidades**. Los **datos** en Programación Orientada a Objetos se llaman **atributos** y las **funcionalidades** se llaman **métodos**, así, cada objeto tiene sus atributos y tiene sus métodos.



- ▶ Como paradigma la Programación Orientada a Objetos se basa en cuatro pilares:
- ▶ abstracción, encapsulamiento, polimorfismo y herencia.

La **abstracción** es un proceso de interpretación y diseño que implica reconocer y enfocarse en las características importantes de una situación o elemento, y filtrar o ignorar todas las particularidades no esenciales.

- Dejar a un lado los detalles y definir características específicas.
- Centrarse en lo que es y lo que hace.
- Hacer más énfasis en qué hace que cómo lo hace

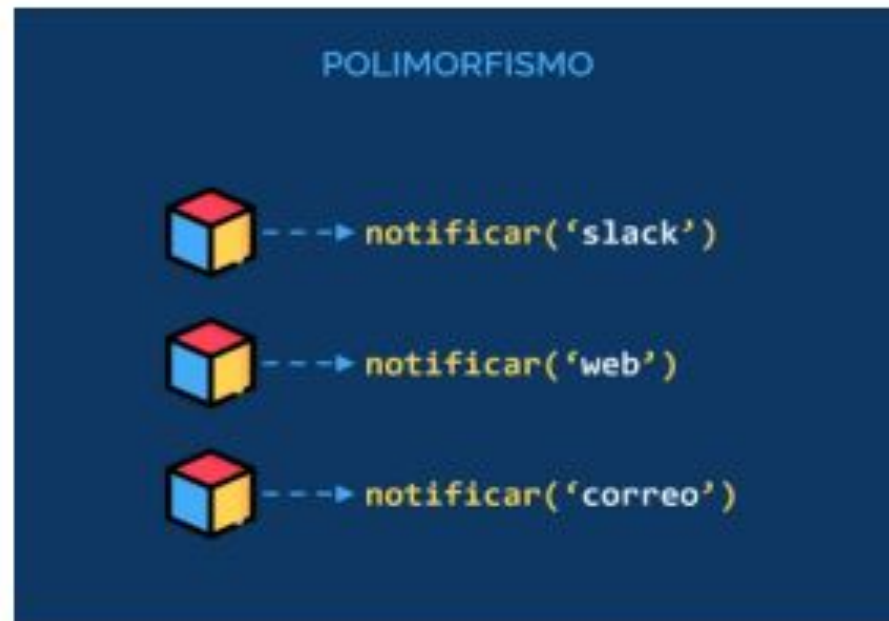
El proceso de abstracción es identificar qué atributos y qué métodos va a tener una clase.

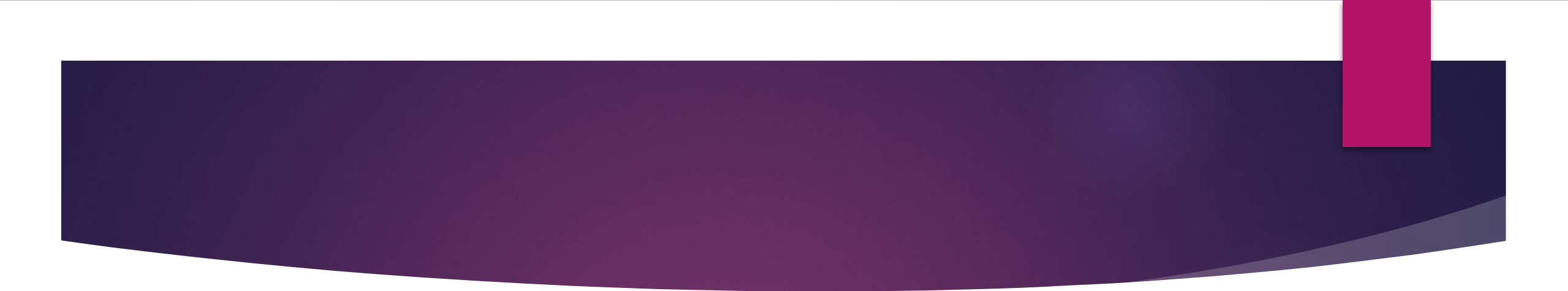


- ▶ proteger la información de manipulaciones no autorizadas de tal manera que cuando se comunican los objetos hay cierto tipo de acceso a información que puede tener uno del otro a través de métodos para acceder, lo que evita cambios indebidos. A esto se le llama el **encapsulamiento**



- **Polimorfismo** es poder dar la misma instrucción a diferentes objetos de tal manera que cada uno de ellos responda a su propia manera.



- 
- ▶ En la Programación Orientada a Objetos, tenemos una clase padre y las clases hijas que **heredan** funcionalidades y atributos de esa clase padre pero sin embargo no son idénticas, solamente aprovechan eso que ya existe para después añadir nuevas cosas.

Clases en POO

Una clase (class) es un prototipo o modelo para el objeto. Antes de crear un objeto, primero necesitamos definir la clase.

Podemos pensar en la clase (class) como un boceto (prototipo) de una casa. Contiene todos los detalles sobre los pisos, puertas, ventanas, etc., basados en estas descripciones es que construimos la casa. La casa es el objeto.

```
class nombreDeClase{  
    // atributos  
    // métodos  
}
```

Objetos en POO

- Un objeto se llama una instancia de una clase. Por ejemplo, supongamos que bicicleta es una clase como antes, entonces bicicletaMontana, bicicletaDeportiva, bicicletaTurismo, etc. pueden considerarse como objetos provenientes de la clase bicicleta.

Sintaxis

```
nombreClase objeto = new nombreClase();
```

Ejemplo.

```
// Para la clase bicicleta  
bicicleta bicicletaDeportiva= new bicicleta();  
  
bicicleta bicicletaTurismo = new bicicleta();
```


Accediendo a las partes de una clase

```
class bicicleta
{
    // atributo o datos
    int velocidades = 5;

    // método o funcionalidad
    void frenado()
    {
        ...
    }
}

// Crear un objeto
bicicleta bicicletaDeportiva = new bicicleta();

// campo de acceso y método
bicicletaDeportiva.velocidades;
bicicletaDeportiva.frenado();
```

Métodos en POO

- ▶ Un método es un bloque de código que realiza una tarea en específico.

Supongamos que necesitas escribir un programa que cree un círculo y lo coloree. Puedes crear dos métodos para resolver este problema:

- Un método para dibujar el círculo.
- Un método para colorear el círculo.

Dividiendo un problema complejo en situaciones más manejables hará que tu programa sea más fácil de entender y reutilizable.

- ▶ ● Métodos definidos por el usuario: Podemos crear nuestro propio método en función de nuestras necesidades.
- ▶ ● Métodos estándar de la biblioteca: Estos son métodos incorporados en Java que están disponibles para usarse.

```
tipoDatoDevolucion nombreMetodo() {  
    // bloque de código del método
```

tipoDatoRegresa: especifica qué tipo de valor regresa un método, por ejemplo, si un método tiene un tipo de dato de devolución (tipoDatoDevolucion) int, entonces devuelve un valor entero. Si el método no devuelve un valor, su tipo de devolución es vacío (void).

- nombreMetodo: es un identificador que se usa para referirse a un método particular en un programa.

- Bloque de código del método: incluye las declaraciones en el programa que se utilizan para realizar algunas tareas. El bloque de código del método está delimitado por llaves { }.

- 
- modificador: define los tipos de acceso si el método es público (public), privado (private), etc.
 - static: si usamos la palabra clave static, se puede acceder sin crear objetos.

Por ejemplo, el método `sqrt()` de la clase Matemáticas estándar es estática. Por lo tanto, podemos llamar directamente `Math.sqrt` sin crear una instancia de clase `Math`.

- parametro1 / parametro2 - Estos son valores que se transfieren al método.

► Podemos pasar cualquier número de argumentos a un método.

EJEMPLO

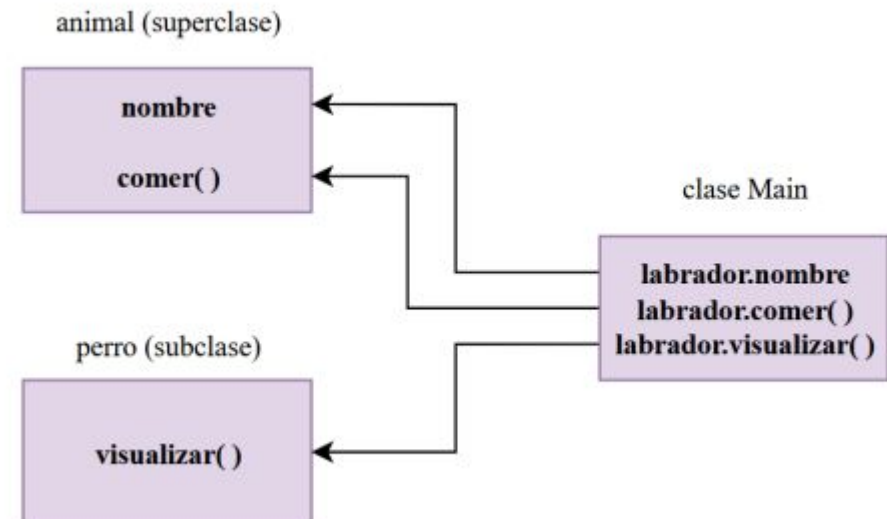
```
class Main {  
  
    // creación de un método  
    public int sumarNumeros(int a, int b) {  
        int suma = a + b;  
        // valor de devolución  
        return suma;  
    }  
  
    public static void main(String[] args) {  
  
        int num1 = 25;  
        int num2 = 15;  
  
        // creación de un objeto de Main  
        Main obj = new Main();  
        // invocando el método  
        int resultado = obj.sumarNumeros(num1, num2);  
        System.out.println("La suma es: "+ resultado);  
    }  
}
```

```
class Main {  
  
    // creamos un método  
    public static int cuadrado(int num) {  
  
        // return statement  
        return num * num;  
    }  
  
    public static void main(String[] args) {  
        int resultado;  
  
        // se invoca al método  
        // se almacena el valor devuelto en resultado  
        resultado = cuadrado(10);  
  
        System.out.println("El valor del cuadrado de 10 es: " + resultado);  
    }  
}
```

Herencia en POO

- ▶ La nueva clase que se crea se conoce como subclase (clase hija o derivada) y la clase existente desde donde se deriva la clase hija se conoce como superclase (clase padre o base).
- ▶ La palabra clave `extends` se utiliza para realizar la herencia en Java. Por ejemplo,

```
class animal {  
    // Aquí irían definidos los métodos y atributos de la clase animal  
}  
  
// uso de la palabra clave extends para llevar acabo la herencia  
class perro extends animal {  
    // métodos y atributos de perro  
}
```





► Hay cinco tipos de herencia:

1. Herencia Única. En este tipo de herencia, una única subclase se extiende desde una única superclase.
2. Herencia Multinivel. En la herencia multinivel, una subclase se extiende desde una superclase y luego la misma subclase actúa como una superclase para otra clase.
3. Herencia Jerárquica. En la herencia jerárquica, múltiples subclases se extienden desde una sola superclase.
4. Herencia Múltiple. En múltiples herencias, una sola subclase se extiende desde múltiples superclases.
5. Herencia Híbrida. La herencia híbrida es una combinación de dos o más tipos de herencia.

Polimorfismo en POO

El polimorfismo es un concepto importante en Programación Orientada a Objetos. Simplemente significa más de una forma.

Es decir, la misma entidad (método, operador u objeto) puede realizar diferentes operaciones en diferentes escenarios.

Polimorfismo--Overriding el Método

- ▶ En este caso, el mismo método realizará una operación en la superclase y otra operación en la subclase.

Encapsulación en POO

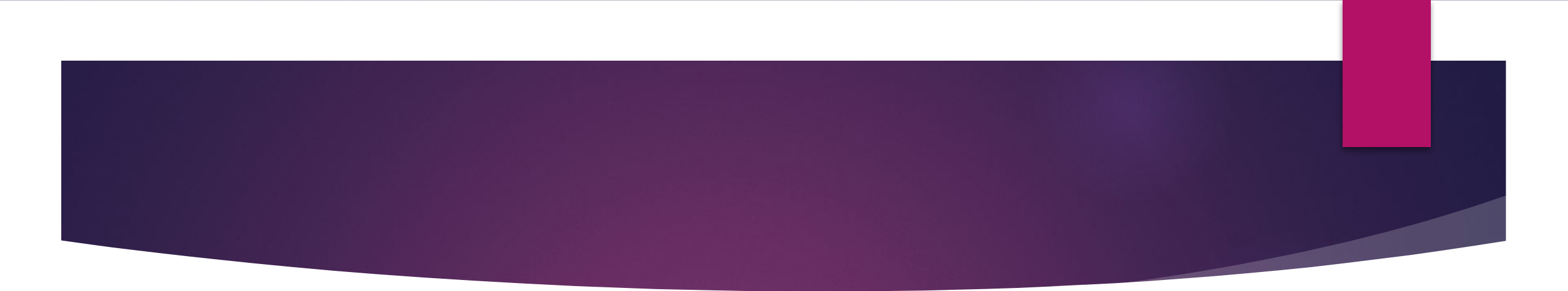
- ▶ La encapsulación se refiere a la agrupación de atributos y métodos dentro de una sola clase.
- ▶ Evita que las clases externas accedan y cambien los atributos y métodos de una clase. Esta también ayuda a ocultar datos.

Sin embargo, es posible

acceder a ellas si proporcionamos métodos públicos (public) get y set.

El método get devuelve el valor de la variable y el método set establece un valor.

La sintaxis de ambos es que comienzan con get o set, seguido del nombre de la variable, con la primera letra en mayúscula:



```
class Persona {  
    private String nombre; // private = acceso restringido  
  
    // Getter  
    public String getNombre() {  
        return nombre;  
    }  
  
    // Setter  
    public void setNombre (String nuevoNombre) {  
        this.nombre = nuevoNombre;  
    }  
}
```

VIII. Variables de Texto

- ▶ La clase Scanner que se encuentra dentro del paquete java.util se usa para leer datos de entrada de diferentes fuentes como flujos de entrada, usuarios, archivos, etc.

```
import java.util.Scanner;

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

        // Crea un objeto a partir de la clase Scanner
        Scanner entrada = new Scanner(System.in);

        System.out.print("Ingresa tu nombre: ");

        // captura la entrada del teclado
        String nombre = entrada.nextLine();

        //imprime el nombre
        System.out.println("Mi nombre es " + nombre);

        // cierra la clase Scanner
        entrada.close();
    }
}
```

La clase **Scanner** proporciona varios métodos que nos permiten leer entradas de diferentes tipos.

Método	Función
nextBoolean()	Lee valores lógicos booleanos introducidos por el usuario.
nextByte()	Lee valores byte introducidos por el usuario.
nextDouble()	Lee valores double introducidos por el usuario.
nextFloat()	Lee valores float introducidos por el usuario.
nextInt()	Lee valores int introducidos por el usuario.
nextLine()	Lee valores String introducidos por el usuario.
nextLong()	Lee valores long introducidos por el usuario.
nextShort()	Lee valores short introducidos por el usuario.

Concatenación

- ▶ El método `concat()` concatena (junta) dos cadenas (string's) y las devuelve.

```
class Main {  
    public static void main(String[] args) {  
        String str1 = "Programación en ";  
        String str2 = "Java";  
  
        // concatenación de str1 y str2  
        System.out.println(str1.concat(str2));  
    }  
}
```

Mayúsculas

- ▶ El método `toUpperCase()` convierte todos los caracteres de una cadena (String) en caracteres en mayúsculas.
- ▶ La sintaxis del método `toUpperCase()` es:

```
cadena.toUpperCase()
```

Ejemplo.

```
class Main {  
    public static void main(String[] args) {  
        String str1 = "Aprender Java es divertido";  
        String str2 = "Java123";  
  
        // convierte en mayúsculas los caracteres de la cadena  
        System.out.println(str1.toUpperCase());  
        System.out.println(str2.toUpperCase());  
    }  
}
```

Minúsculas

- ▶ El método `toLowerCase()` convierte todos los caracteres de la cadena (String) en caracteres en minúsculas.
- ▶ La sintaxis del método `toLowerCase()` es:

```
cadena.toLowerCase()
```

Ejemplo.

```
class Main {  
    public static void main(String[] args) {  
        String str1 = "PROGRAMACIÓN EN JAVA";  
  
        // convierte en minúsculas los caracteres de la cadena  
        System.out.println(str1.toLowerCase());  
    }  
}
```


Longitud

- ▶ El método `length()` devuelve la longitud de la cadena (string).
- ▶ La sintaxis del método `length()` es:

```
cadena.length()
```

Ejemplo.

```
class Main {  
    public static void main(String[] args) {  
        String str1 = "Programar en Java es divertido";  
        //cuenta número de caracteres en una cadena contando los espacios en blanco  
        System.out.println(str1.length());  
    }  
}
```

Conversión

- ▶ Podemos convertir un tipo de dato String en un tipo de dato int en java usando el método `Integer.parseInt()`.
- ▶ Generalmente se usa si tenemos que realizar operaciones matemáticas en una cadena (string) que contiene un número. Cada vez que recibimos datos de `TextField` o `TextArea`, los datos ingresados se reciben como una cadena. Si los datos ingresados están en formato de número, necesitaríamos convertir la cadena a un dato de tipo int. Para hacerlo, usamos el método `Integer.parseInt()`.
- ▶ Podemos convertir un tipo de dato int en un tipo de dato String en java usando el método `String.valueOf()`. Generalmente se usa cuando tenemos que mostrar un número en un

Leer textos en Java

- ▶ La clase Reader del paquete java.io es una clase abstracta para leer transmisión de caracteres.
- ▶ Dado que Reader es una clase abstracta, no es útil por sí misma. Sin embargo, sus subclases se pueden usar para leer datos.
- ▶ Para usar la funcionalidad de Reader, podemos hacer uso de su subclase **FileReader**.
- ▶ Para crear un Reader, primero debemos importar el paquete **java.io.Reader**. Una vez que importamos el paquete, así es como podemos crear el lector.

LinkedList en Java

- ▶ La clase LinkedList del marco de colecciones Java proporciona la funcionalidad en la estructura de datos de listas enlazadas.
- ▶ Agregar elementos a una lista enlazada
- ▶ Acceso a los elementos de una lista enlazada
- ▶ Cambiar elementos de una lista enlazada
- ▶ Eliminar elemento de una lista enlazada

Escribir archivo de texto en Java

- ▶ La clase `Writer` del paquete `java.io` es una clase abstracta que representa un flujo de caracteres.
- ▶ Dado que `Writer` es una clase abstracta, no es útil por sí misma. Sin embargo, sus subclases se pueden usar para escribir datos.
- ▶ Para crear un `Writer`, primero debemos importar el paquete `java.io.Writer`. Una vez que importamos el paquete, así es como podemos crear el `Writer`.

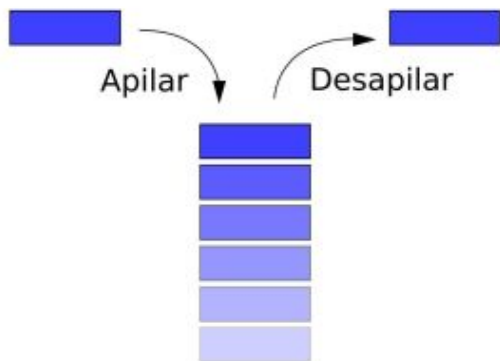
IX. Pilas

La pila (stack) es una estructura de datos lineal que se utiliza para almacenar la recopilación de objetos, siendo el modo de acceso a sus elementos de tipo LIFO (del inglés Last In, First Out, «último en entrar, primero en salir»)

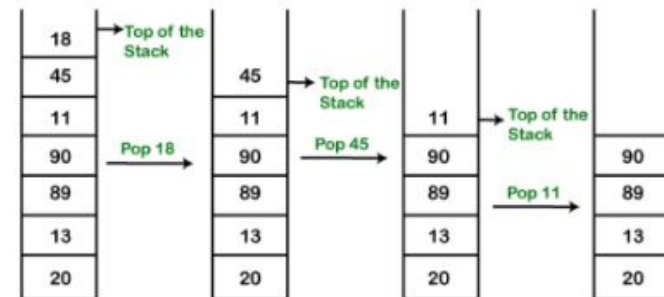
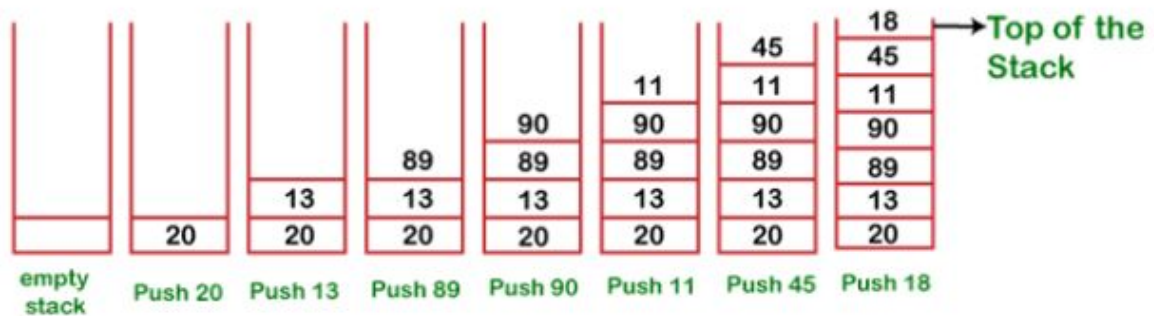
Java Collection Framework proporciona muchas interfaces y clases para almacenar la colección de objetos. Una de ellas es la clase de pila (stack) que proporciona diferentes operaciones, como push, pop, peek, etc.

Las dos operaciones más importantes de la estructura de datos de pila son push (apilar, colocar) y pop (retirar, desapilar). La operación push inserta un elemento en la pila (stack) y pop elimina un elemento de la parte superior de la pila (top of the stack).

Veamos cómo funcionan en las pilas.



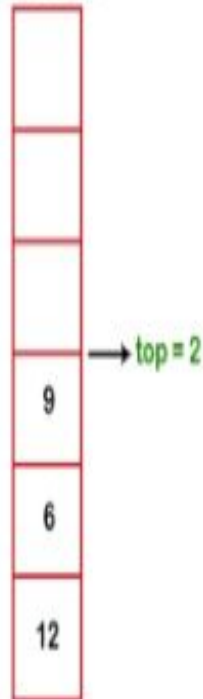
Apilemos (**push**) 20, 13, 89, 90, 11, 45, 18, respectivamente en la pila de datos.



Pila vacía (**empty stack**): una pila que no tiene ningún elemento se le conoce como una pila vacía (**empty stack**). Cuando la pila está vacía, el valor de la variable de la parte superior es -1.



Cuando apilamos (**push**) un elemento en la pila, la parte superior aumenta en **1**. Por ejemplo, en la siguiente figura tenemos que para:
push 12, top=0, push 6, top=1 y push 9, top=2



Java collections framework tiene una clase llamada **Stack** que proporciona la funcionalidad de la estructura de datos apilados.

Para crear una pila, primero debemos importar el paquete **java.util.stack**. Una vez que importamos el paquete, así es como podemos crear una pila en Java:

```
Stack<Type> pila = new Stack<>();
```

Aquí, el **type** indica el tipo de datos que se almacenan en la pila. Por ejemplo,

```
// crea una pila de datos de tipo entero
Stack<Integer> pila = new Stack<>();

// crea una pila de datos de tipo cadena o String
Stack<String> pila = new Stack<>();
```


El método push()

- ▶ Para apilar un elemento a la parte superior de la pila, usamos el método push ().

El método pop()

- ▶ Para eliminar un elemento de la parte superior de la pila (top of the stack), usamos el método pop(). A continuación tenemos un ejemplo de su aplicación:

El método peek()

- ▶ El método peek() devuelve el objeto desde la parte superior de la pila. Por ejemplo,

El método search()

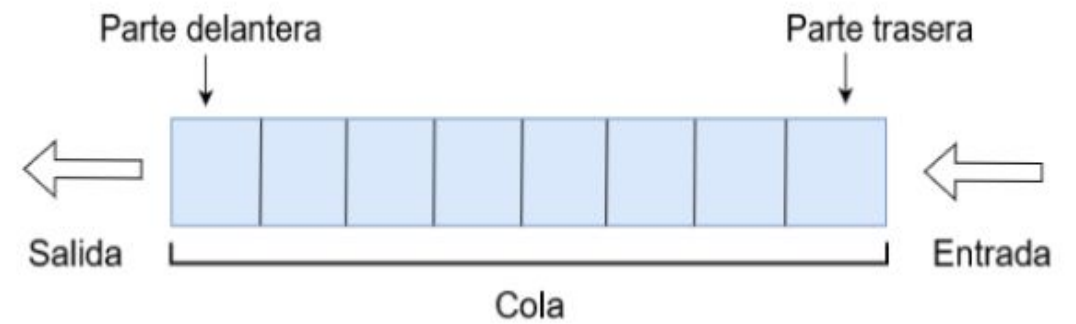
- ▶ Para buscar un elemento en la pila, usamos el método search(). Devuelve la posición del elemento contando desde la parte superior de la pila. Por ejemplo,

El método `empty()`

- ▶ Para verificar si una pila está vacía o no, usamos el método `empty()`. Por ejemplo,

X. Colas

- Una cola (queue) es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pull por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir.



Sintaxis

```
import java.util.Queue;  
  
Queue<TipoDato> cola = new LinkedList<TipoDato>();
```

Donde el TipoDato es el tipo de datos que se almacenará en la cola, y la clase de cola es una clase que implementa la interfaz de cola. En Java, debemos importar el paquete `java.util.queue` para usar la cola (queue).

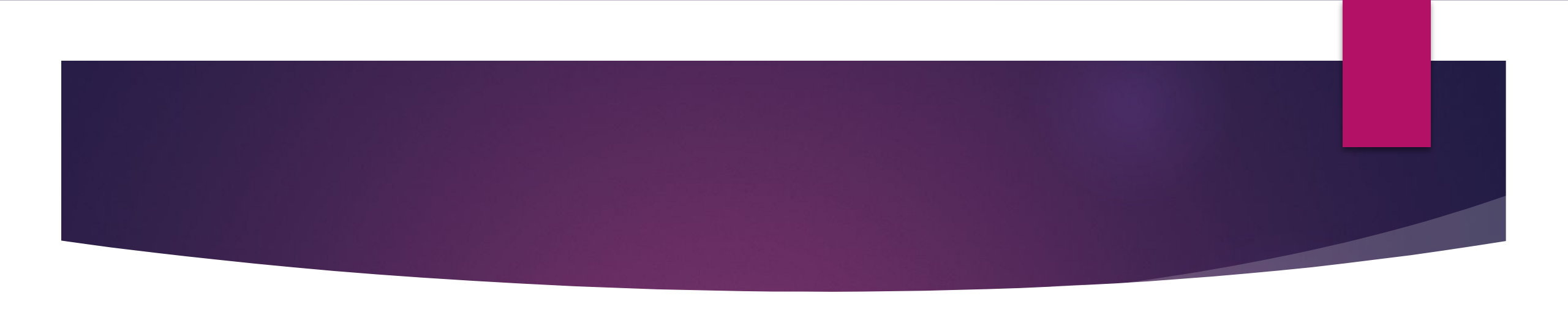
métodos comúnmente utilizados por la interfaz cola (queue)

- ▶ ● `add()`: inserta el elemento especificado en la cola. Si el proceso es exitoso, `add()` devuelve verdadero (`true`), si no, lanza una excepción.
- ▶ ● `offer()`: inserta el elemento especificado en la cola. Si el proceso es exitoso, `offer()` devuelve verdadero (`true`), si no devuelve falso (`false`).
- ▶ ● `element()`: devuelve la parte delantera de la cola. Lanza una excepción si la cola está vacía.
- ▶ ● `peek()`: devuelve la parte delantera de la cola. Devuelve `null` si la cola está vacía.
- ▶ ● `remove()`: devuelve y elimina la parte delantera de la cola. Lanza una excepción si la cola está vacía.
- ▶ ● `poll()`: devuelve y elimina la parte delantera de la cola. Devuelve `null` si la cola está vacía.

XI. Métodos de Ordenamiento




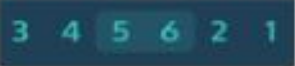

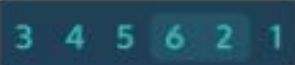

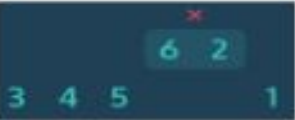
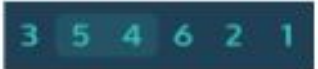
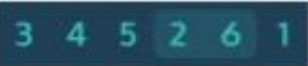
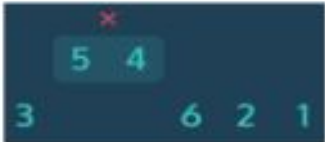





► Ordenamiento de burbuja

El ordenamiento de burbuja es un algoritmo simple para ordenar una lista de N números en orden ascendente. El ordenamiento de burbuja compara dos elementos adyacentes y los intercambia si están en el orden ascendente equivocado.



Este algoritmo consiste en una iteración externa y una iteración interna. En la iteración interna, el primer y el segundo elementos se comparan y se intercambian de tal manera que el segundo elemento tiene un valor más grande que el primero. Esto se repite para los subsecuentes, segundo y tercero, par de elementos, y así sucesivamente hasta que se compara el último par de elementos ($N-2$, $N-1$). Al final de la iteración interna, el elemento más grande aparece al último. Esto se repite para todos los elementos de la lista en la iteración externa.

Ejemplo de la iteración interna para el primer ciclo externo:

1. 	9. 
2. 	10. 
3. 	11. 
4. 	12. 
5. 	13. 
6. 	14. 
7. 	15. 
8. 	16. 

Ordenamiento por inserción

- ▶ El ordenamiento por inserción es un algoritmo de ordenamiento que construye un arreglo ordenado al finalizar procediendo de a un elemento a la vez. En cada iteración a través de un arreglo dado, el algoritmo de inserción toma un elemento y encuentra la ubicación al que pertenece y lo inserta.



El ordenamiento por inserción funciona dividiendo el arreglo de entrada en dos listas virtuales: una sub lista ordenada y una sub lista desordenada.

La sub lista ordenada contiene inicialmente el primer elemento del arreglo de entrada. El resto de los elementos constituyen la sub lista desordenada.

El ordenamiento por inserción irá a través de los elementos de la sub lista desordenada de a uno por uno, esencialmente eliminando un elemento de la sub lista desordenada e insertando este en la posición correcta en la sub lista ordenada cambiando todos los elementos en la sub lista ordenada que sean más grandes que el elemento que se está ordenando.

7 | 2 5 8 -3
ordenado | desordenado

7 | [] 5 8 -3
2

2 7 | 5 8 -3
ordenado | desordenado

2 7 | [] 8 -3
5

2 5 7 | 8 -3
ordenado | desordenado

2 5 7 | [] -3
8

2 5 7 8 | -3
ordenado | desordenado

2 5 7 8 | []
-3

-3 2 5 7 8
ordenado

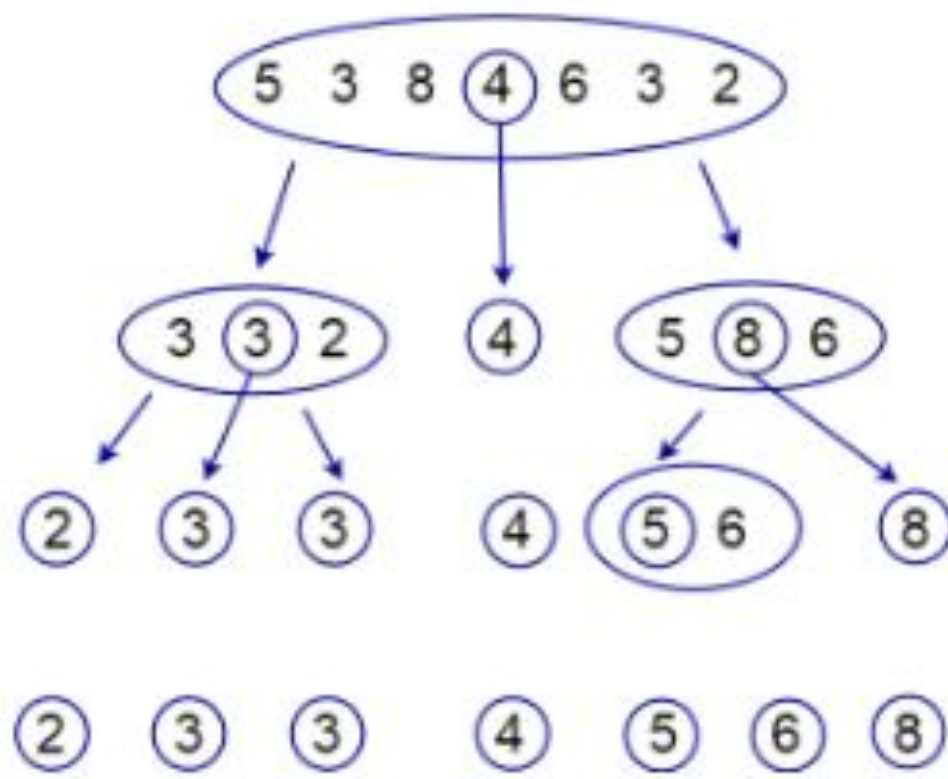
Ordenamiento rápido

- ▶ El ordenamiento rápido es un algoritmo recursivo eficiente para ordenar arreglos o listas de valores. El algoritmo es un tipo de ordenamiento, donde los valores se ordenan mediante una operación de comparación como $>$ o $<$.



El ordenamiento rápido es un método para ordenar un arreglo al dividirlo repetidamente en sub-arreglos por medio de:

1. Seleccionando un elemento del arreglo inicial. Este elemento se llama elemento pivote.
2. Comparando cada elemento en el arreglo con el elemento pivote, dividimos los elementos en dos sub arreglos, los que son mayores y los que son menores al elemento pivote.
3. Este proceso continúa hasta que cada subarreglo contiene un solo elemento.
4. En este punto, los elementos ya están ordenados. Finalmente, los elementos se combinan para formar un arreglo ordenado.



XII. Entorno de Desarrollo para crear aplicaciones gráficas