



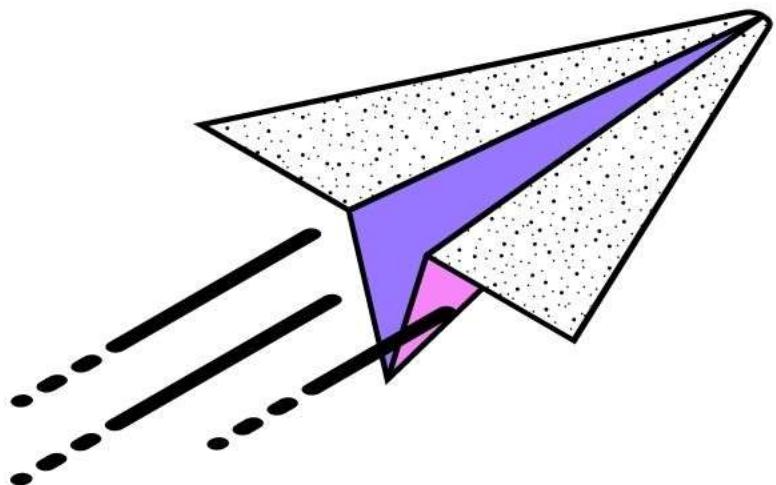
Java Standard Edition





Sección 3

Trabajar con variables primitivas



Declarar y asignar valores a variables.

Uso de constantes.

Descripción de tipos de datos primitivos como int, float, double, boolean.

Utilizar operadores aritméticos para modificar valores.

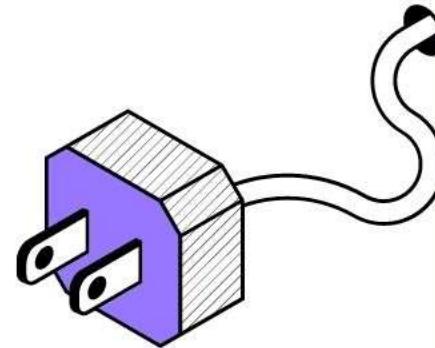
Declaración e inicialización de variables de campo.



¿Qué es una variable?

Definición de variable

- Las variables se utilizan para representar valores que pueden cambiar durante la ejecución de un programa.
- Las variables tienen un nombre, que es un identificador para hacer referencia a ellas en nuestros programas.



x
 \neq
 y

$f(x)$



Reglas para definir el nombre de una variable en Java

- Un identificador es una secuencia de caracteres que consta de letras, dígitos, guiones bajos (_) y signos de dólar (\$).
- Un identificador debe comenzar con una letra, un guion bajo (_) o un signo de dólar (\$). No puede comenzar con un dígito.
- Un identificador no puede ser una palabra reservada.
- La sintaxis para declarar una variable es **tipo_de_dato nombre_variable;**

Notas y tips para definir nombres de variables.

- Dado que Java distingue entre mayúsculas y minúsculas, **area**, **Area** y **AREA** son identificadores diferentes.
- Los identificadores descriptivos hacen que los programas sean fáciles de leer.



Declaración variables

Ejemplo donde se declaran tres variables.

```
● ● ●

public class Prueba{

    public static void main(String[] args){

        //Ejemplo DECLARACION de variables
        int contador;
        double radio;
        double tasaInteres;

    }
}
```



Asignar valor a una variable

- Una declaración de asignación designa un valor para una variable. Una declaración de asignación se puede utilizar como una expresión en Java.
- Después de declarar una variable, puede asignarle un valor. En Java, el signo igual (=) se utiliza como operador de asignación.
- Un identificador no puede ser una palabra reservada.
- La sintaxis para asignar valor a una variable es *nombre_variable = expresión o valor;*

Notas y tips para expresiones de asignación.

- Una expresión representa un cálculo que involucra valores, variables y operadores que, tomándolos juntos, se evalúa como un valor.



Conclusión

Las **variables** son espacios en la memoria temporal, dónde podemos guardar datos y accederlos a través de un nombre. También las podríamos pensar que son como cajas nombradas de la memoria.





Ejercicio

a = 10

b = 20

c = 5

a = a + 3

b = b + 4 - a

c = a + b + c

a = a + c

b = 4

c = c + 3 - b + 2

Qué valores quedan almacenados en las variables a, b y c ?



Ejercicio

a = 5

b = 18

c = 15

d = 25

a = a + 10

b = b + 5 - c

c = c + 4 + b

d = d + b + a

a = a + 1

b = b + c

c = b + c

d = b + b

Qué valores quedan almacenados en las variables a, b y c ?



Asignar valor a una variable

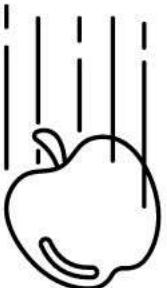
Ejemplo donde se declaran tres variables y se les asigna un valor.

```
public class Prueba{  
  
    public static void main(String[] args){  
  
        //Ejemplo ASIGNACION a variables  
        int contador = 1;  
        double radio = 1.0;  
        int x = 5 * (3 / 2);  
        double area;  
        contador = x + 1;  
        area = radius * radius * 3.14159;  
  
    }  
}
```



π

e



Constantes

- Una constante es un identificador que representa un valor permanente.
- El valor de una variable puede cambiar durante la ejecución de un programa, pero una constante, representa datos permanentes que nunca cambian.
- La sintaxis para asignar valor a una variable es **final datatype NOMBRE_CONSTANTE = valor;**

Notas y tips para expresiones de asignación.

- Una constante debe declararse e inicializarse en la misma instrucción.



Declaración Constante

Ejemplo donde se declaran cuatro constantes y sus valores.

```
public class Prueba{  
    public static void main(String[] args){  
        //Ejemplo declaracion de constantes  
        final double PI = 3.14159;  
        final double E = 2.7182;  
        final int VALOR_MINIMO = 1;  
        final int VALOR_MAXIMO = 20;  
    }  
}
```



Tipo de datos primitivos

Diferencia entre tipos primitivos y objetos.



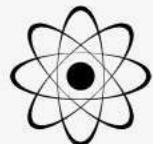
No poseen atributos

Los tipos de datos no poseen atributo alguno, estos solo guardan un valor en alguna dirección en memoria.



No poseen métodos

Los tipos de datos no poseen métodos, de hecho estos solo son utilizados por los métodos en partes del programa.



Son indivisibles, atómicos

Con baja latencia hay menos retrasos; da una reacción genuina en tiempo real.



Tipo de datos numéricos

Entre los tipos numéricos tenemos: **enteros** y de **punto flotante**

byte

-2^7 to $2^7 - 1$ (-128 to 127)

8 bits, con signo

int

-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)

32 bits, con signo

float

32 bits, IEEE 754

Negative range: $-3.4028235E+38$ to $-1.4E-45$
Positive range: $1.4E-45$ to $3.4028235E+38$

short

16 bits, con signo

-2^{15} to $2^{15} - 1$ (-32768 to 32767)

long

64 bits, con signo

-2^{63} to $2^{63} - 1$

(i.e., -9223372036854775808 to 9223372036854775807)

double

64 bits, IEEE 754

Negative range: $-1.7976931348623157E+308$ to $-4.9E-324$
Positive range: $4.9E-324$ to $1.7976931348623157E+308$





Tipo de datos no numéricos

Entre los no numéricos, tenemos los de carácter y los booleanos.

boolean

Un tipo de datos booleano declara una variable con el valor **true** (verdadero) o **false** (falso)



A

char

Un tipo de datos de carácter representa un único carácter.



"String"

Una cadena es una secuencia de caracteres.
No es primitivo, pero ...



Operadores aritméticos

Los operadores aritméticos se utilizan para modificar variables.

Simbolo	Nombre	Ejemplo	Resultado
+	Adición	34 + 1	35
-	Sustracción	34.0 - 0.1	33.9
*	Multiplicación	300 * 30	9000
/	División	1.0 / 2.0	0.5
%	residuo o modulo	20 % 3	2



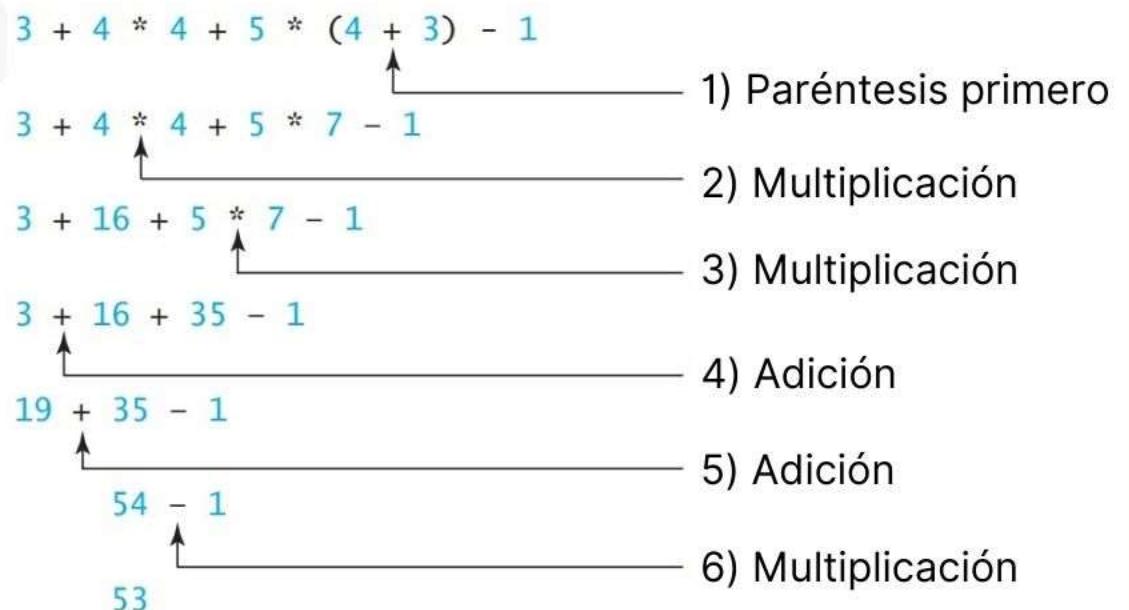


Jerarquía de los operadores aritméticos

Los operadores aritméticos se utilizan para modificar variables.

Constantes

- La jerarquía funciona tal y como es en la aritmética.
- Primero los paréntesis, pudiendo estos estar anidados.
- Después multiplicación, división y resto, modulo o residuo. Si una expresión contiene varios operadores de multiplicación, división y modulo , se aplican de izquierda a derecha.
- Por ultimo adición y sustracción. Si una expresión contiene varios operadores de suma y resta, se aplican de izquierda a derecha.



Ejercicios practicos



- Dado un numero entero de segundos, calcular a cuantos minutos equivale y cuantos segundos restantes quedan. Ejemplo :
500 segundos equivalen a 8 minutos y 20 segundos.
- Dado un numero de grados Fahrenheit, convertirlos a Celsius.



Operadores de asignación aumentados.

Los operadores `+`, `-`, `*`, `/` y `%` se pueden combinar con el operador de asignación para formar operadores aumentados.

Operador	Nombre	Ejemplo	Equivalente
<code>+=</code>	Asignación de adición	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	Asignación de resta	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	Asignación de multiplicación	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	Asignación de división	<code>i /= 8</code>	<code>i = i / 8</code>
<code>%=</code>	Asignación de modulo	<code>i %= 8</code>	<code>i = i % 8</code>



Operadores de incremento y decrecimiento.

Los operadores de incremento (++) y decremento (--) son para incrementar y decrementar una variable en 1.

Operador	Nombre	Descripción	Ejemplo (i = 1)
<code>++var</code>	preincremento	Incrementa var en 1 y usa el nuevo valor de var en la declaración	<code>int j = ++i;</code> <code>// j es 2, i es 2</code>
<code>var++</code>	postincremento	Incrementa var en 1, pero use el valor original de var en la declaración	<code>int j = i++;</code> <code>// j es 1, i es 2</code>
<code>--var</code>	predecremento	Disminuye var en 1 y usa el nuevo valor de var en la declaración	<code>int j = --i;</code> <code>// j es 0, i es 0</code>
<code>var--</code>	postdecremento	Disminuye var en 1 y usa el valor original de var en la declaración	<code>int j = i--;</code> <code>// j es 1, i es 0</code>



Variables de campo

Declaración e inicialización de variables de campo

```
public class Circulo{  
  
    //declaracion e inicializacion  
    //de una constante  
    final double PI = 3.1416;  
  
    //variables de campo  
  
    //declaracion e inicializacion  
    double radio = 1;  
  
    //declaracion  
    double area;  
    double perimetro;  
  
    void computeArea(){  
        area = radio * radio * PI;  
    }  
  
    void computePerimetro(){  
        perimetro = 2 * radio * PI;  
    }  
  
    void setRadio(double newRadio ){  
        radio = newRadio;  
    }  
}
```

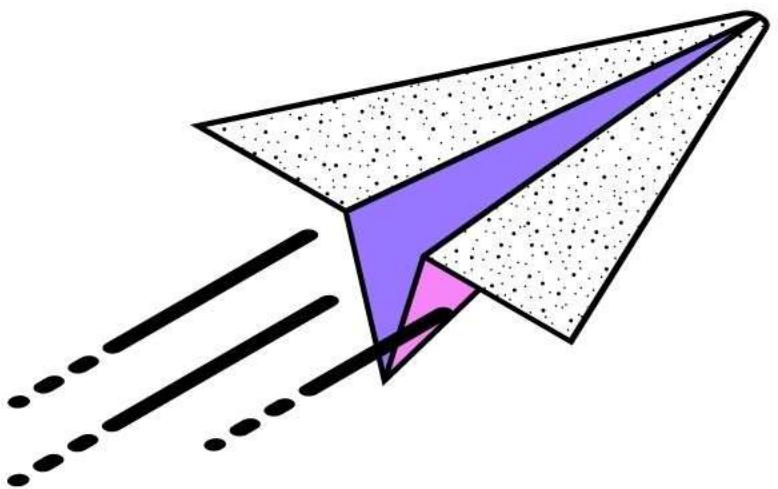
Todo lo que este dentro de los parentesis que definen la clase pero fuera de los parentesis de algun metodo , son variables de campo, es decir atributos de la clase.

Las variables definidas dentro de los parentesis de los metodos, no son variables de campo.
(aqui no hay ninguna declaracion)



Sección 3

Operadores y construcción de sentencias lógicas



Creación de estructuras if, if/else.

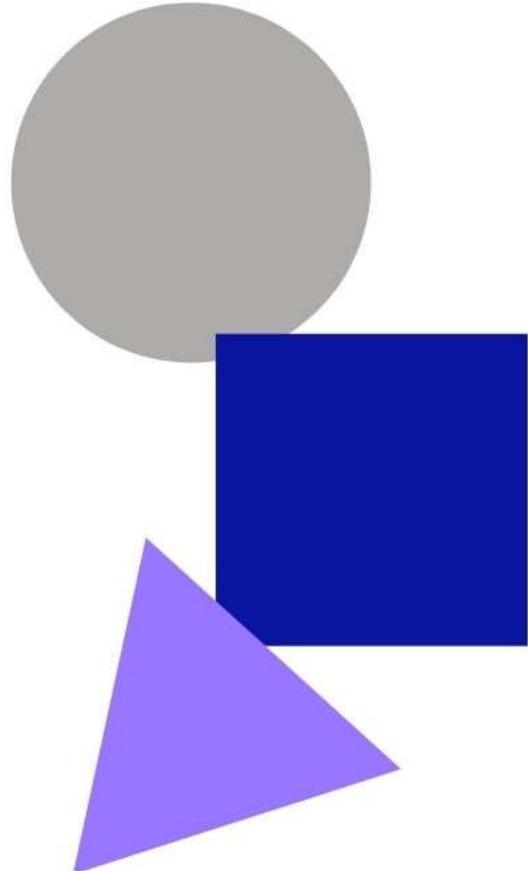
Sentencias condicionales anidadas y encadenadas.

Igualdad entre cadenas (Strings).

Uso de operadores relacionales y condicionales.

Sentencia **switch**.

Evaluación de diferentes condiciones en un programa para determinar un algoritmo.



Sentencia if

Una sentencia **if** ejecuta las instrucciones si la condición es verdadera.

Que sentencias que permite Java

Java tiene varios tipos de sentencias de selección: sentencias if simples, sentencias if-else, sentencias if anidadas, sentencias switch y expresiones condicionales.

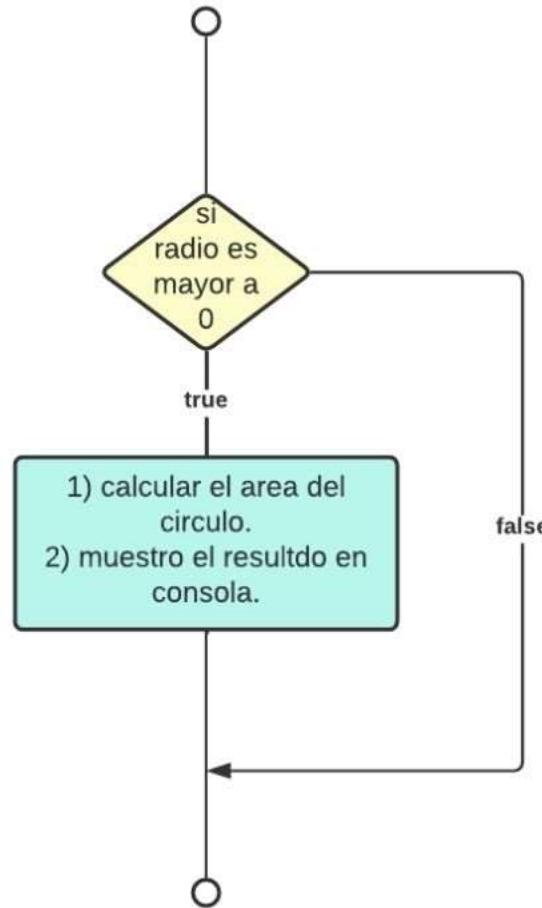
Como luce una sentencia if

```
if (expresión-booleana) {  
    instrucción(es);  
}
```



Diagrama de flujo

como se comporta una sentencia if





Sentencia if-else

Una sentencia **if** ejecuta las instrucciones delimitadas por ella si la condición es verdadera, en caso contrario ejecutara las contenidas en el bloque delimitado por la palabra **else**.



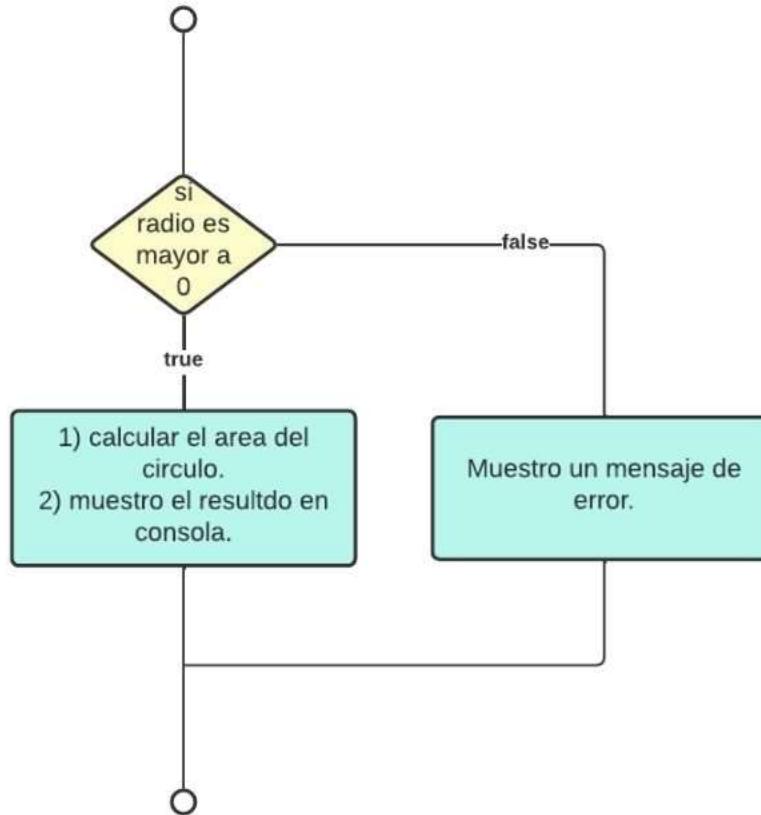
Como luce una sentencia if-else

```
if (expresión-booleana) {  
    instrucción(es); //para el caso true  
}else{  
    instrucción(es); //para el caso false  
}
```



Diagrama de flujo

como se comporta una sentencia if-else





Como luce una sentencia if anidada.

```
if (expresión-booleana 1) {  
    if(expresión-booleana 2){  
        instrucción(es); // para el caso en que ambas condiciones  
        // resulten verdaderas  
    }  
}
```

Como luce una sentencia if encadenada.

A graphic on the left side of the slide shows two simple 3D white human figures standing next to a grey signpost. One figure is pointing towards the signpost, which has three arrows pointing in different directions. This visual metaphor represents the flow of control or decisions in a program.

```
if (expresión-booleana 1) {  
  
    instrucción(es); // para el caso verdadero  
  
}else if(expresión-booleana 2){  
  
    instrucción(es); // para el caso falso en 1 y verdadero en 2.  
}  
  
else{  
  
    instrucción(es); // para el caso falso tanto en 1 como en 2.  
}
```



Comparación de cadenas con el operador ==

- Se utiliza para comparar cadenas y objetos en general.
- **==** es un **operador**
- **==** compara la referencia o dirección de memoria del objeto en el Heap, verifica que la dirección sea la misma o no.



Comparacion de cadenas con el metodo .equals()

- Se utiliza también para comparar cadenas y objetos en general.
- **.equals()** es un método.
- Este no verifica dirección de memoria, este se enfoca en el contenido, en este caso en las letras que componen la cadena.



Operadores relacionales.

Los operadores relacionales son de utilidad para realizar comparaciones y así determinar si se ejecuta o no cierta sentencia.

Operador	Nombre	Ejemplo var = 5	Resultado
<	Menor que	var < 0	false
<=	Menor o igual que	var <= 0	false
>	Mayor que	var > 0	true
>=	Mayor o igual que	var >= 0	true
==	Igual a	var == 0	false
!=	Distinto de	var != 0	true



Operadores lógicos.

Los operadores lógicos son de utilidad para realizar para evaluar expresiones booleana compuestas de mas de una expresión.

Operador	Nombre	Descripción
!	not	negación lógica
&&	and	conjunción lógica
	or	disyunción lógica
^	exclusive or	exclusión lógica



Sentencia switch

Una sentencia **switch** ejecuta instrucciones basadas en el valor de una variable.

Como luce una sentencia if

```
switch (expresión-switch){  
    case valor1: instrucción(es)1;  
        break;  
  
    case valor2: instrucción(es)2;  
        break;  
  
    case valorN: instrucción(es)N;  
        break;  
  
    default: instrucción(es); // ningún caso se cumplió  
}
```



Ejercicios:

- 1. Declarar una variable entera y asignarle un valor determinar si es par.**

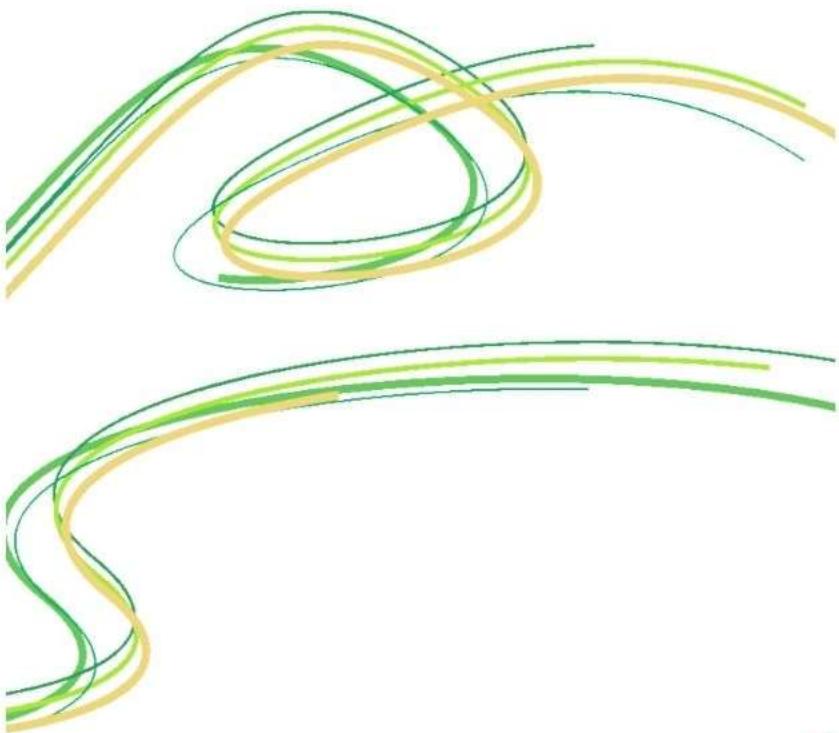
- 2. Declarar una variable entera y asignarle un valor determinar si es par.**

- 3. Declarar una variable tipo double y asignarle un valor, determinar si sus decimales son mayores o menores a 0.5.**
Ejemplo: 3.8 su parte decimal es mayor a 0.5 mientras que 3.45 su parte decimal es menor a 0.5.



Sección 3

Uso de ciclos



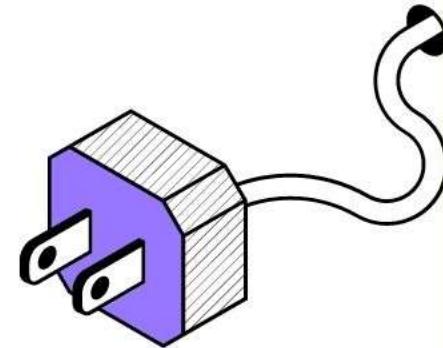
Crear ciclos while y ciclos while anidados.

Alcance de las variables.

Crear ciclos do-while

Crear ciclos for

Utilizar un ArrayList con el ciclo for.

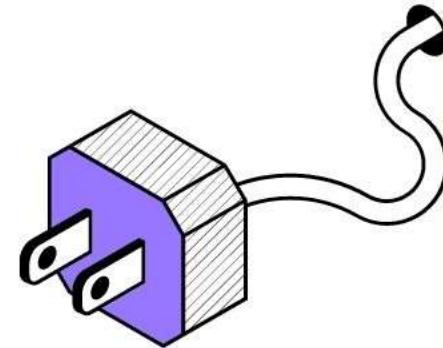


Ciclo while

Características

- Un bucle o ciclo while ejecuta declaraciones repetidamente mientras una condición es verdadera.
- La sintaxis del ciclo while es:
while (condición){
 //cuerpo del ciclo
 Instrucción (es);
}





Ciclos while anidados

Características

- Es posible definir un ciclo **while** dentro de otro.
- La sintaxis de ciclos while anidados es:
while (condición1){
 while (condición2){
 //cuerpo del ciclo
 Instrucción (es);
 }
}





Alcance de las variables.

Como funciona



- El alcance de una variable es la parte del programa donde se puede hacer referencia a la variable.
- Una variable definida dentro de un método se denomina variable local.
- El alcance de una variable local comienza desde su declaración y continúa hasta el final del bloque que contiene la variable.
- Una variable local debe declararse y asignarle un valor antes de que pueda ser utilizada.



Alcance de las variables.

```
public static void method1() {  
    .  
    .  
    for (int i = 1; i < 10; i++) {  
        .  
        .  
        int j;  
        .  
        .  
    }  
}
```

Alcance de *i* →

Alcance de *j* →

A diagram illustrating variable scope. A vertical rectangle represents the scope of the variable *i*, which starts at the opening brace of the for-loop and ends at its closing brace. Inside this scope, the variable *j* is declared. Another vertical rectangle represents the scope of *j*, which starts at its declaration and ends at the closing brace of the entire method *method1()*. Arrows from the text "Alcance de i" and "Alcance de j" point to the start of these respective scopes.

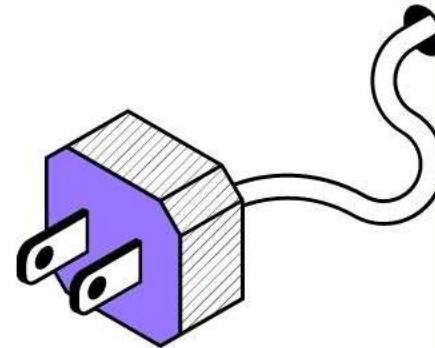
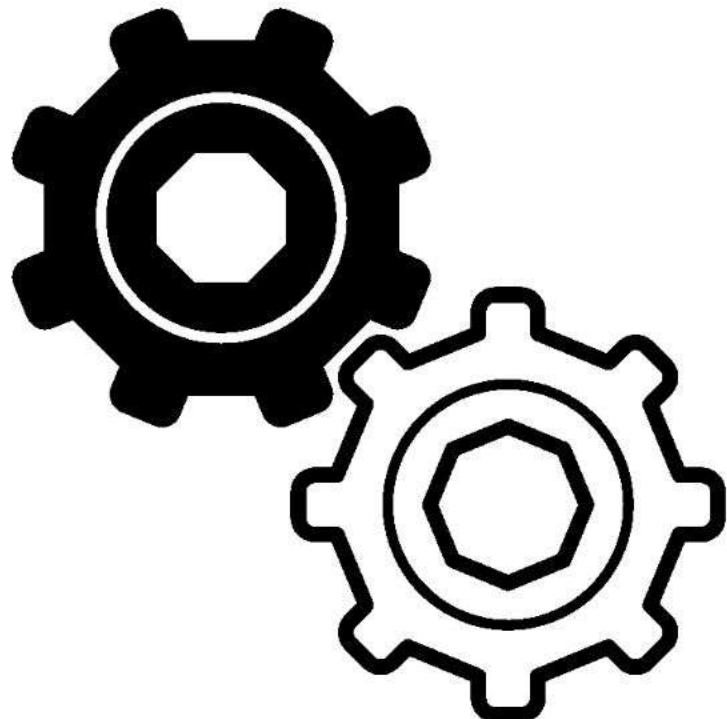


Ciclo do while

Características

- Un ciclo do-while es lo mismo que un ciclo while excepto que primero ejecuta el cuerpo del ciclo y luego verifica la condición de continuación del ciclo.
- La sintaxis del ciclo do while es:

```
do{  
    //cuerpo del ciclo  
    Instrucción (es);  
}while ( condición );
```



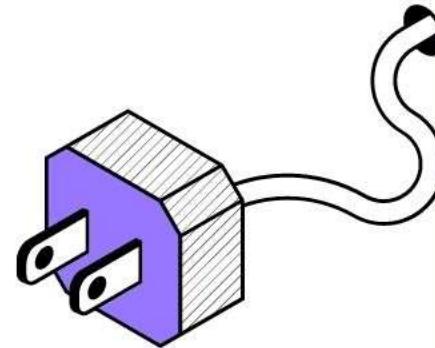
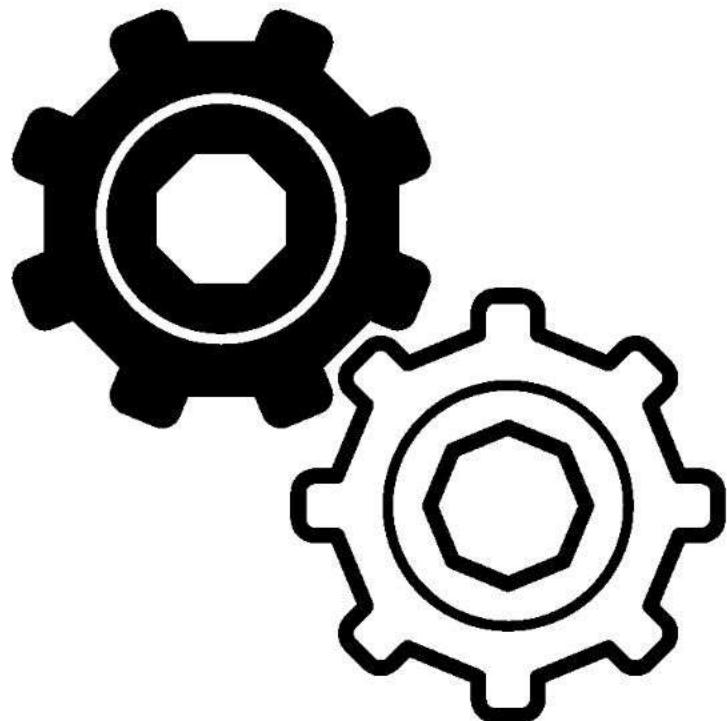


Ciclo for

Características

- Un ciclo for también tiene una condición, pero esta condición, se sabe desde un principio cuando dejará de ser verdadera.
- La sintaxis del ciclo for es:

```
for (int i = valorInicial; i < valorFinal; i++) {  
    // Loop body ...  
}
```





Ejemplo de uso del ciclo for con un ArrayList.

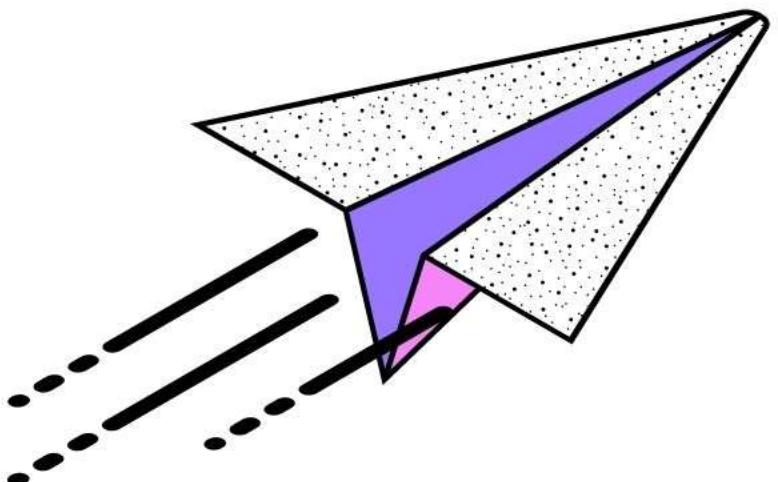
Suponiendo que tenemos definido un ArrayList que guardara elementos de tipo entero y que agregamos tres elementos, es posible procesar o manipular un objeto ArrayList con ciclos, tal y como se hizo con un arreglo convencional.

```
ArrayList<Integer> arrlist = new ArrayList<Integer>();  
  
arrlist.add(14);  
arrlist.add(7);  
arrlist.add(39);  
  
for (int i= 0; i< arrlist.size(); i++)  
{  
    System.out.println(arrlist.get(i));  
}  
  
int j = 0;  
while (arrlist.size() > j) {  
    System.out.println(arrlist.get(j));  
    j++;  
}
```



Sección 3

Creación y uso de arreglos.



Acceso a un valor en un arreglo o ArrayList.

Declarar, instanciar e inicializar un arreglo unidimensional.

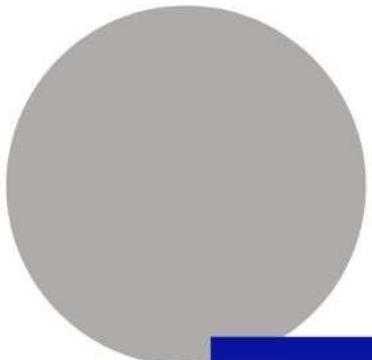
Uso de la sentencia import para trabajar con API's existentes de java.

Declarar, instanciar e inicializar un arreglo bidimensional.

Crear e inicializar un ArrayList.

Utilizar el vector de argumentos. (args Array)

Uso de un ciclo **for** para procesar un arreglo



Que es un arreglo

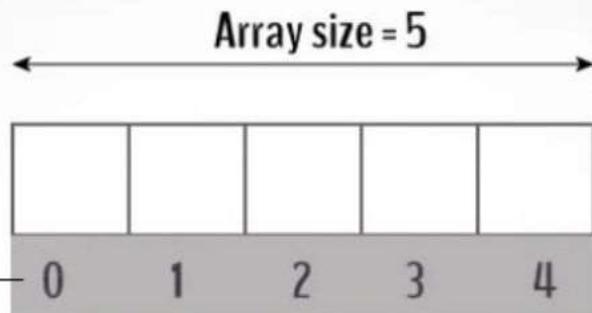
Es una colección de datos de un mismo tipo. Una sola variable de tipo arreglo o array puede hacer referencia a una gran colección de datos.

Conceptos de arreglos

- Una vez que se crea un arreglo, su tamaño es fijo.
- Una variable de referencia de matriz se utiliza para acceder a los elementos de una matriz mediante un índice.

Como se declara un arreglo en Java

tipoElemento variableReferencia[] = new tipoElemento [tamaño]





Creación de un arreglo

```
double[] miArray = new double[10];
```

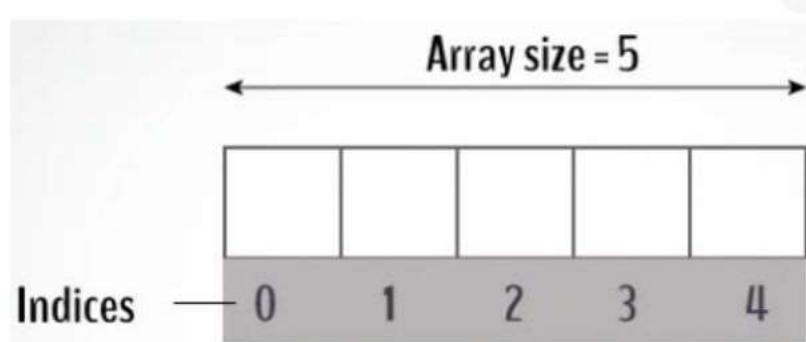
Se crea un arreglo, que guardara datos de tipo double, con una capacidad de 10. El arreglo tiene valores por defecto.

Creación de un arreglo inicializado

```
double[] miArray= {1.9, 2.9, 3.4, 3.5};
```

Se crea un arreglo, que guarda los valores que están entre las llaves. El arreglo contiene valores específicos desde su creación.

Asignar valores a un arreglo con valores por defecto.



```
miArreglo[0] = 4.0;  
miArreglo[1] = 34.33;  
miArreglo[2] = 34.0;  
miArreglo[3] = 45.45;  
miArreglo[4] = 99.993;  
miArreglo[5] = 11123;
```



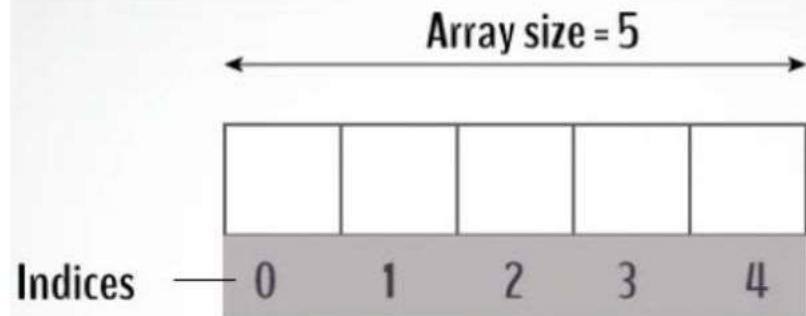
Creación de un arreglo bidimensional.

```
int matriz[ ][ ] = new int[3][3];
```

Creación de un arreglo bidimensional inicializado

```
int[ ][ ] miArreglo= {{1, 2, 3},  
                      {4, 5, 6},  
                      {7, 8, 9},  
                      {10, 11, 12}};
```

Dar valores a elementos de un arreglo bidimensional.



```
miArreglo[0][0] = 1;  
miArreglo[0][1] = 47;  
miArreglo[0][2] = 88;  
miArreglo[1][0] = 19;  
miArreglo[1][1] = 17;  
miArreglo[1][2] = 189;
```



Que son los paquetes de java

- Los paquetes son el mecanismo que usa Java para facilitar la modularidad del código.
- Un paquete puede contener una o más definiciones de interfaces y clases, distribuyéndose habitualmente como un archivo.
- Para utilizar los elementos de un paquete es necesario importar este en el módulo de código en curso, usando para ello la sentencia



Como utilizo los paquetes

- Para utilizar los elementos de un paquete es necesario importar este en el módulo de código en curso, usando para ello la sentencia **import**.



uso de import

- Importando una sola clase:

```
import java.util.Date;
```

- Importando un paquete completo de clases:

```
import java.util.*;
```

- Para utilizar los elementos de un paquete es necesario importar este en el módulo de código en curso, usando para ello la sentencia



Utilidades con import

- puedes incluir múltiples sentencias **import**:

```
import java.util.Date;
```

```
import java.util.Calendar;
```

- Por default siempre se importa `java.lang.*`

- No necesitas importar las clases que estén en el mismo paquete.



ArrayList

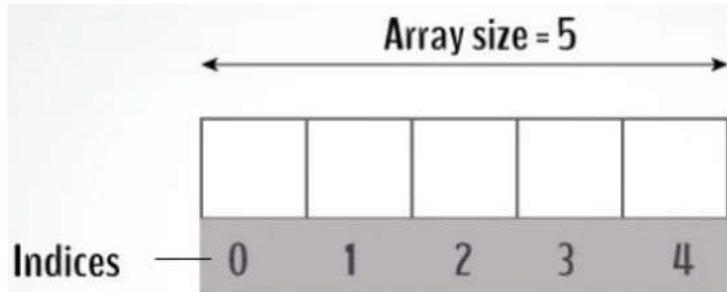
Una ArrayList se implementa usando un arreglo. En un arreglo convencional el tamaño de este era fijo una vez definido, y este no podía cambiar, en un ArrayList esto no es así, ya que el arreglo crece de manera dinámica, es decir su tamaño puede variar.

Creación de un ArrayList

```
import java.util.*;  
  
ArrayList<String> miArrayList = new ArrayList<>();
```

Agregar valores a un ArrayList

```
miArrayList.add("Arturo");  
miArrayList.add("Maria");  
miArrayList.add("Pedro");  
miArrayList.add("Fernanda");
```





Arreglo de argumentos (`String[] args`)

Es un arreglo que guarda objetos de tipo cadena (String), este es pasado justo cuando ejecutamos nuestro programa en la linea de comandos.

Pasando valores al arreglo (`String[] args`)

Cuando se ejecuta un programa, es posible pasar valores para que estos sean almacenados en el arreglo args.

```
java nombrePrograma [argumento1, argumento2 ... ]
```

Ejemplo

Suponiendo que tenemos un programa que recibe nombres de personas mediante la linea de comandos, para después imprimir un saludo para estas, la ejecucion seria la siguiente.

```
java SaludoPersonas Arturo Sofia Jorge Pedro Camila
```





Procesado de un arreglo de manera eficaz.

- Existen maneras de realizar tareas que son repetitivas.
Para el procesado de un arreglo, tenemos al ciclo **for**.
- Con el ciclo **for** es posible recorrer los elementos de un arreglo, para acceder a sus datos o modificarlos de una manera eficaz.

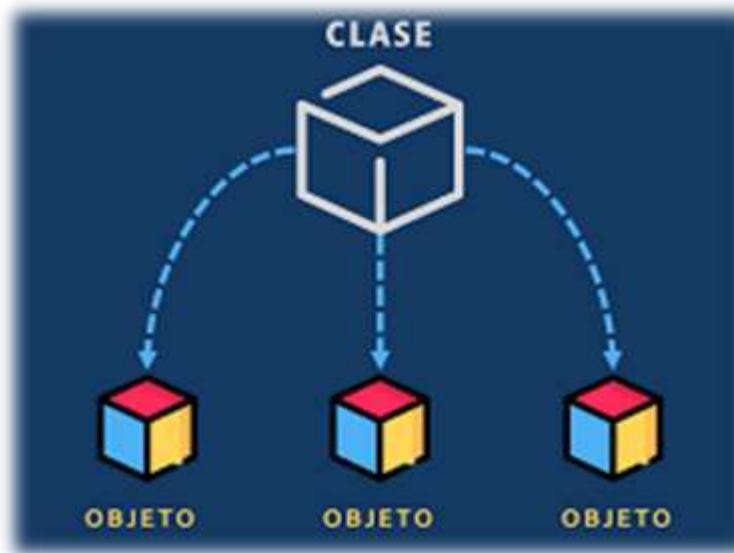
Ejemplo de uso del ciclo for con un arreglo.

Suponiendo que tenemos definido un arreglo de tamaño 5 y que la referencia se llama miArreglo, es posible recorrerlo con la siguiente instrucción:

```
for ( int i = 0; i < miArreglo.length; i++ ){  
    System.out.println(miArreglo[i]);  
}
```



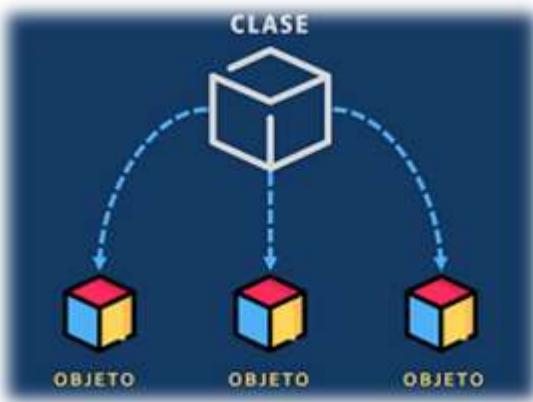
Introducción a la programación orientada a objetos;





Introducción a la programación orientada a objetos;

Establece que todos los problemas deberían ser resueltos mediante entidades que absorban un estado interno y las tareas posibles sobre estado (el objeto), con un diseño que permita entender la arquitectura de cada entidad (la clase).





Objetos;

En el mundo real un objeto es una entidad, un nombre o cualquier cosa que tenga una identidad propia.





Objetos;





Objetos;





Objetos;





Objetos;





Objetos;

Los objetos se componen de ***atributos*** y ***métodos*** que pueden ser accesibles desde dentro del objeto o desde fuera, según los niveles de acceso diseñados.



La entidad en cuestión será la **Persona**. Cada persona podrá retener datos que le describan, por ejemplo, el **nombre**, la **edad**, su **correo**, etc. A estos datos les llamaremos *los atributos de la entidad*.



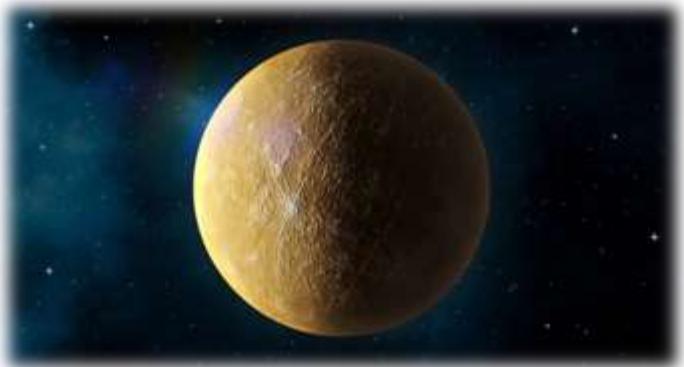
Clase;

Es un prototipo (plantilla) que contiene una colección de datos (atributos) y un conjunto de métodos (comportamiento) que manipulan los datos o que interactúan con objetos relacionados.

A los datos y métodos se les conoce como miembros de la clase.



Objetos de la clase Planeta



Mercurio



Venus



Tierra



Marte



Objetos de la clase Automóvil



BMW i8



Ford Bronco



Subaru BRZ

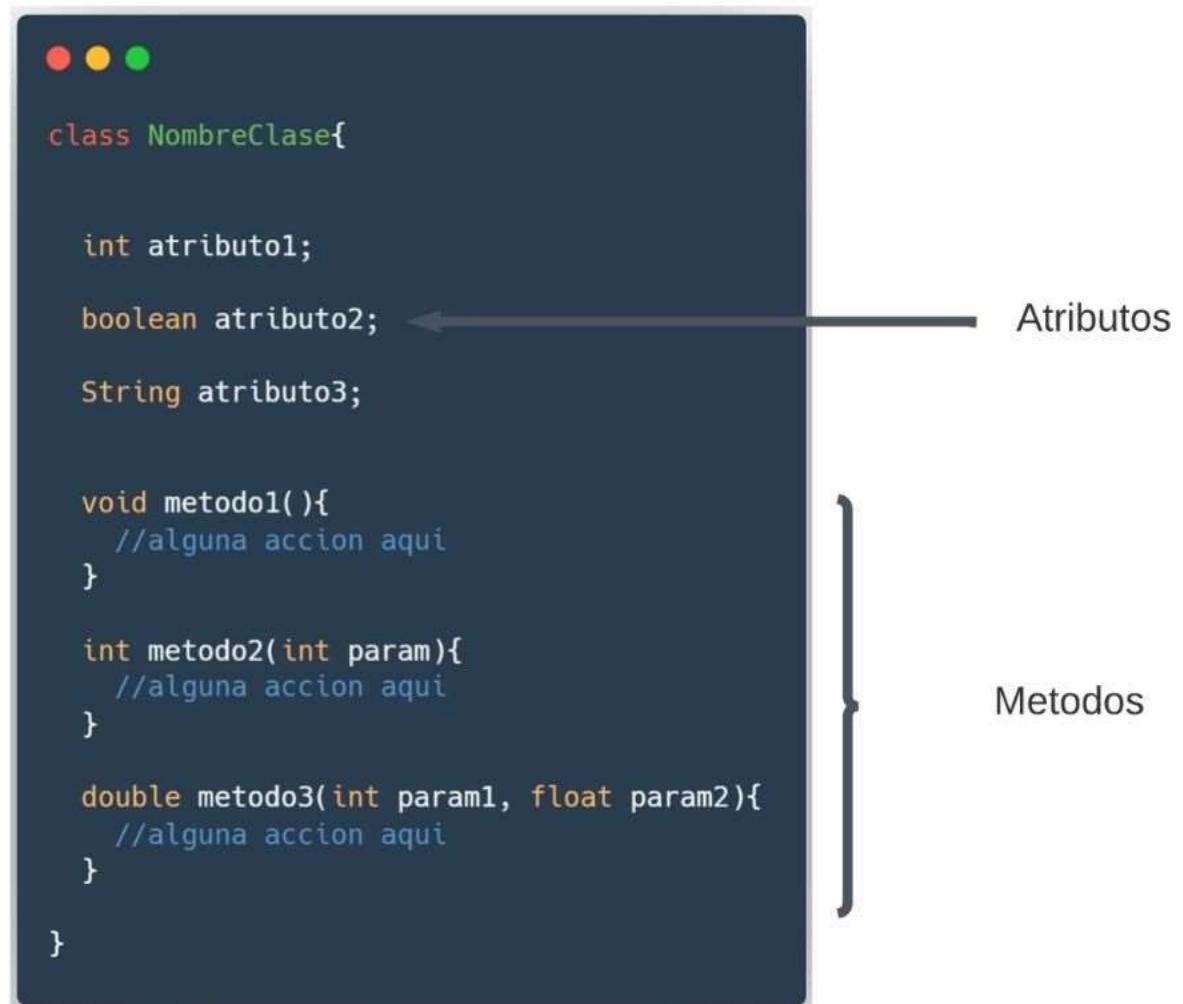


Chevrolet Corvette



Componentes de una clase

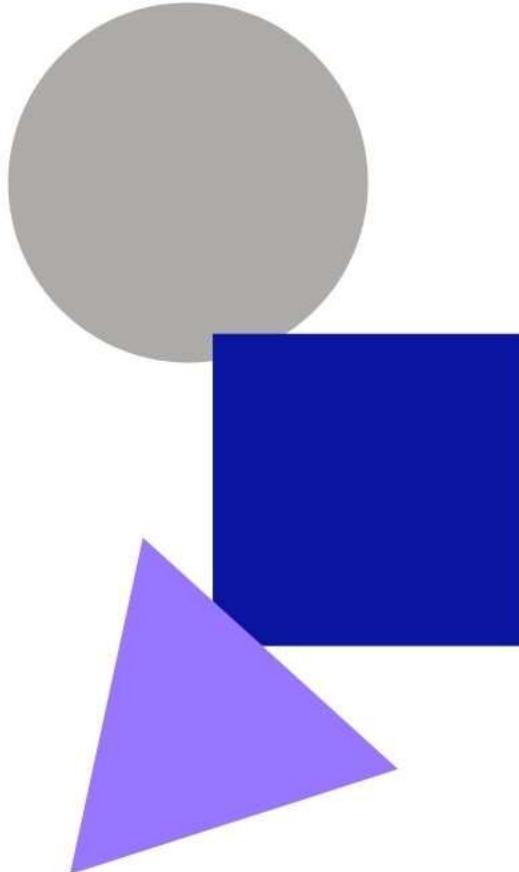
todas las clases tienen esta estructura básica



Ejercicios practicos



Diseñar una clase llamada Circulo y otra llamada Automóvil.



Estado de un Objeto

El estado de un objeto (también conocido como sus propiedades o atributos) está representado por campos de datos con sus valores actuales.

Comportamiento de un Objeto

El comportamiento de un objeto (también conocido como sus acciones) se define mediante métodos.

De donde proviene un Objeto

Los objetos del mismo tipo se definen utilizando una clase común. Una clase es una plantilla o modelo que define cuáles serán los campos de datos y métodos de un objeto.

Relación entre Clase y Objeto

Un objeto es la instancia de una clase. Puede crear muchas instancias de cierta clase. La creación de una instancia se conoce como **instanciación**.

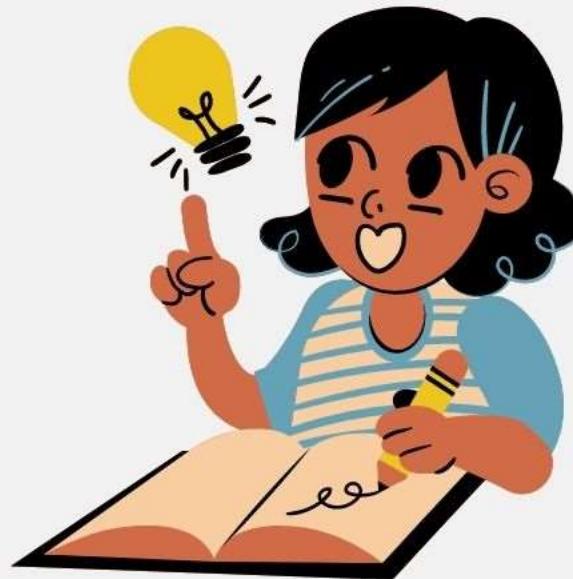


Ejemplo de creación de objetos

- Las clases son definiciones de objetos y los objetos se crean a partir de clases.

```
public class Pruebas {  
  
    /** Metodo principal */  
    public static void main(String[] args) {  
  
        // Creamos un circulo con radio = 1;  
  
        Circulo circulo1 = new Circulo();  
        System.out.println("El area del circulo 1 de radio "  
        circulo1.radio + " es " + circulo1.getArea());  
  
        // Creamos un circulo con radio = 1, y despues es cambiado;  
  
        Circulo circulo2 = new Circulo();  
        circulo2.setRadio(25);  
        System.out.println("El area del circulo 2 de radio "  
        circulo2.radio + " es " + circulo2.getArea());  
  
        // Modificamos el radio de circulo1  
        //calculamos de nuevo el area  
        circulo1.radio = 8;  
        System.out.println("El area del circulo 1 de radio "  
        circulo1.radio + " es " + circulo1.getArea());  
  
    }  
}
```

Ejercicios practicos



Crea algunos objetos de las clases Circulo y Automóvil.



Sección 3

Métodos y sobrecarga de métodos



Uso de modificadores

Paso de argumentos y retorno de valores

Creación de variables y métodos estáticos

Sobrecarga de un método

Crear e invocar un método



Que son los modificadores de java

- Se usan en clases, constructores, métodos, datos y bloques de nivel de clase), pero el modificador **final** también se puede usar en variables locales en un método.
- Un modificador que se puede aplicar a una clase se denomina modificador de clase.
- Un modificador que se puede aplicar a un método se denomina modificador de método.
- Un modificador que se puede aplicar a un campo de datos se denomina modificador de datos.



Modificadores existentes

- **public:** Una clase, constructor, método o campo de datos es visible para todos los programas de cualquier paquete.
- **private:** Un constructor, método o campo de datos solo es visible en esta clase.
- **protected:** Un constructor, método o campo de datos es visible en este paquete y en las subclases de esta clase en cualquier paquete.
- **static:** Define un método, un campo de datos o un bloque de inicialización estático.
- **final:** Una clase no puede heredar. Un método no se puede modificar en una subclase. Un campo de datos es una constante.



Métodos

Conceptos clave



En que consiste un método

La definición de un método consiste en el nombre de este, sus parámetros, su tipo de valor de retorno y el cuerpo.

Definición en Java

La definición de un método consta de un encabezado y de un cuerpo.

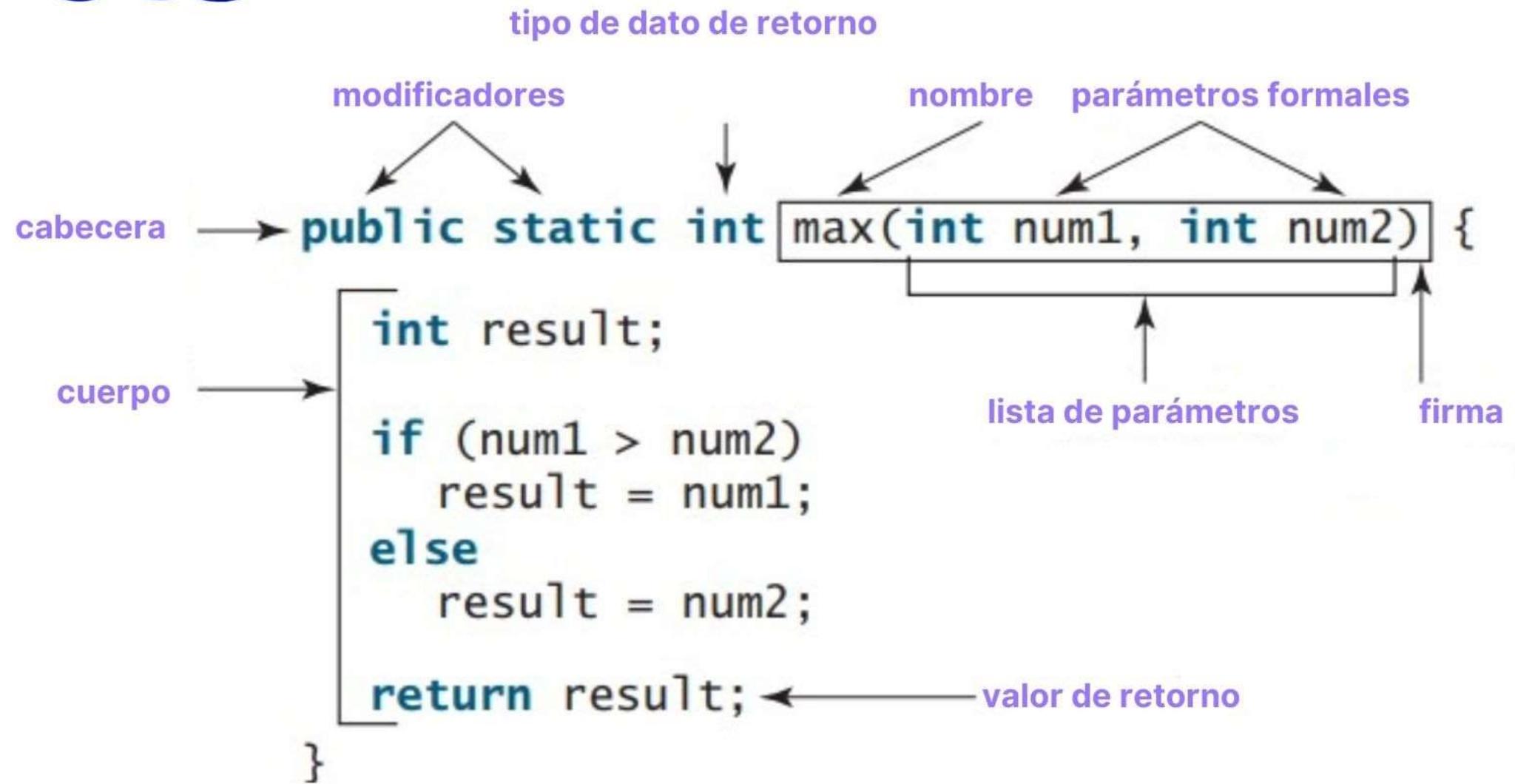
Que define el encabezado

Especifica los modificadores, el tipo de valor de retorno, el nombre del método y sus parámetros.

Si un método no devuelve ningún valor se denomina método **void**.



Definición de un Método





Invocación de un Método

```
int z = max(x, y);
```



parámetros actuales (argumentos)

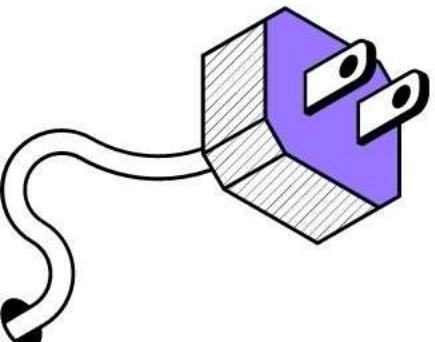


variables y métodos estáticos

- Una variable estática es compartida por todos los objetos de la clase. Un método estático no puede acceder a los miembros de instancia de la clase (sus atributos).
- Las variables estáticas almacenan valores para las variables en una ubicación de memoria común..
- Para utilizar los elementos de un paquete es necesario importar este en el módulo de código en curso, usando para ello la sentencia

Como defino una variable o método estático

- Para el caso de una variable, basta con colocar la palabra reservada **static** en su declaración.
- Para el caso de un método, basta con colocar la palabra reservada **static** en su cabecera.





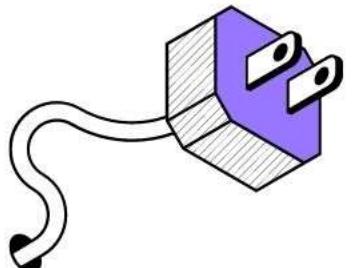
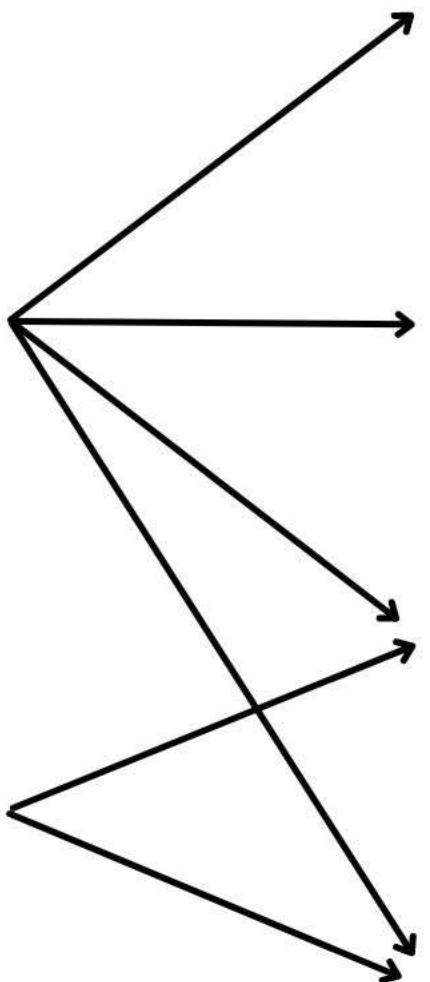
Un método de instancia

Método de instancia

Un método estático

Método estático

Un campo de datos estático





Sobrecarga de Métodos

Conceptos clave



En que consiste la sobrecarga de un método

La sobrecarga de métodos le permite definir los métodos con el mismo nombre siempre que sus firmas sean diferentes.

```
/*Retorna el maximo de dos valores enteros*/
public static int max(int num1, int num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}

/*Retorna el maximo de dos valores double*/
public static double max(double num1, double num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}

/*Retorna el maximo de tres valores double*/
public static double max(double num1, double num2, double num3) {
    return max(max(num1, num2), num3);
}
```



Sección 3

Uso de la encapsulación y constructores



Implementando la encapsulación

Crear un constructor



La encapsulación

Como funciona



- Hacer que los campos de datos sean privados protege los datos y hace que la clase sea fácil de mantener.
- En primer lugar, los datos pueden ser **manipulados**.
- En segundo lugar, la clase se vuelve difícil de mantener y vulnerable a errores.
- Para evitar modificaciones directas de los campos de datos, debe declarar los campos de datos como privados, utilizando el modificador privado. Esto se conoce como encapsulación de campos de datos.



Los constructores

Como funciona



- Se invoca un constructor para crear un objeto usando el operador new.
- Los constructores son un tipo especial de método.
- Un constructor debe tener el mismo nombre que la propia clase.
- Los constructores no tienen un tipo de retorno, ni siquiera **void**.
- Los constructores se invocan mediante el operador **new** cuando se crea un objeto. Los constructores juegan el papel de inicializar a los objetos.

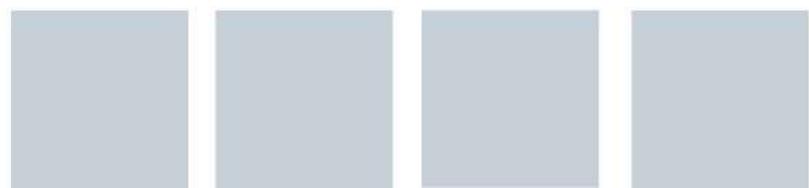
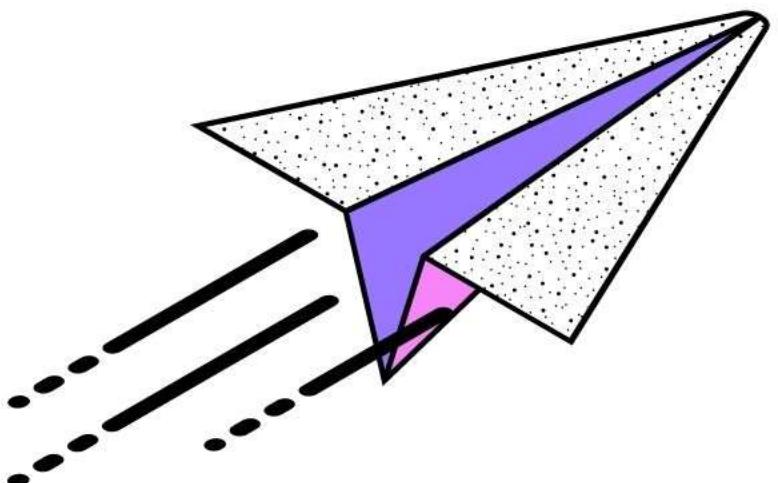


```
public class CircleWithPrivateDataFields {  
    private double radius = 1;  
  
    private static int numberofObjects = 0;  
  
    public CircleWithPrivateDataFields() {  
        numberofObjects++;  
    }  
  
    public CircleWithPrivateDataFields(double newRadius) {  
        radius = newRadius;  
        numberofObjects++;  
    }  
  
    public double getRadius() {  
        return radius;  
    }  
  
    public void setRadius(double newRadius) {  
        radius = (newRadius >= 0) ? newRadius : 0;  
    }  
  
    public static int getNumberofObjects() {  
        return numberofObjects;  
    }  
  
    public double getArea() {  
        return radius * radius * Math.PI;  
    }  
}
```



Sección 3

Conceptos avanzados del paradigma orientado a objetos



Agregando abstracción al análisis y diseño

Crear e implementar una Java Interface

Uso de la herencia

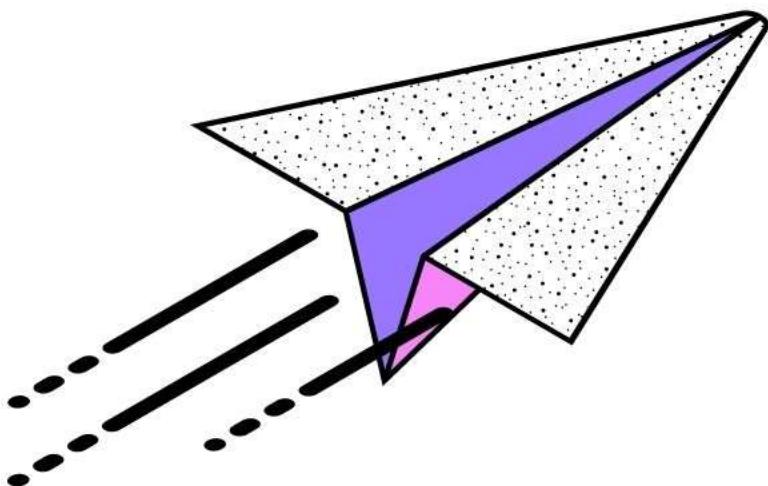
Comprender el propósito de las Java Interfaces.

Uso del polimorfismo, sobreescritura

Uso de superclases y subclases

Interfaces

conceptos clave



Que es

Una interfaz es similar a una clase, esta contiene solo constantes y métodos abstractos.

Proposito

Su intención es especificar un comportamiento común para objetos de clases relacionadas o clases no relacionadas.

Como se implementa en Java

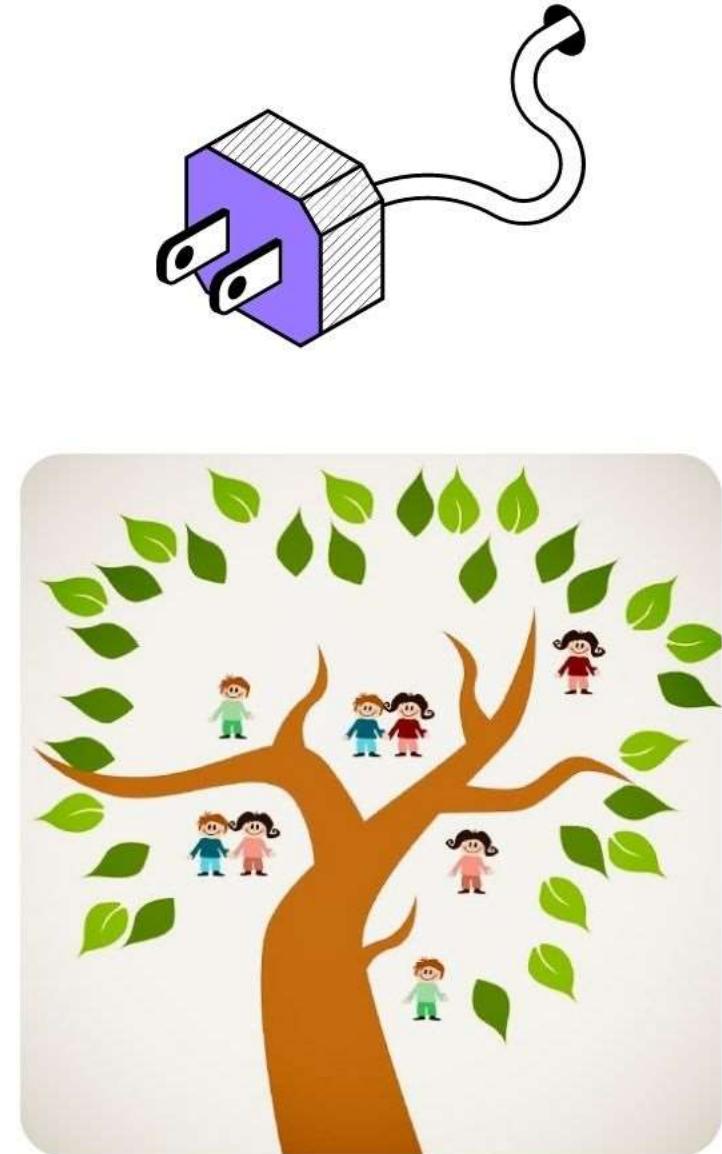
Para distinguir una interfaz de una clase, Java usa la siguiente sintaxis para definir una interfaz:

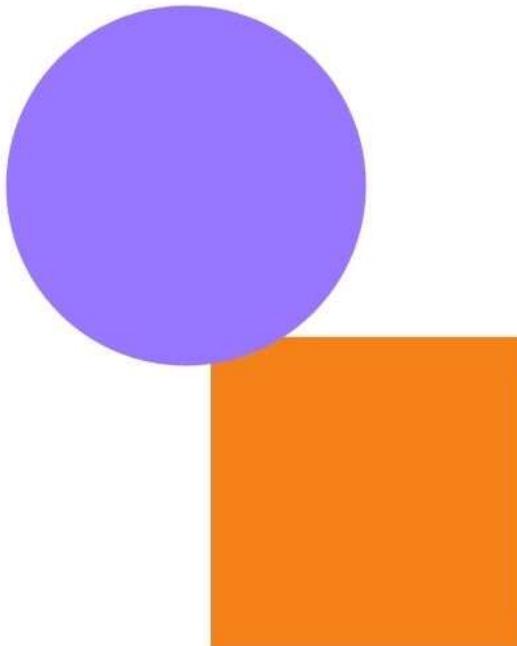
```
modifier interface InterfaceName {  
    /** Constant declarations */  
    /** Abstract method signatures */  
}
```

Herencia

Conceptos clave

- La programación orientada a objetos permite definir nuevas clases a partir de clases existentes. Esto se llama herencia.
- La herencia le permite definir una clase general (por ejemplo, una superclase) y luego extenderla a clases más especializadas (por ejemplo, subclases).
- Una clase para modelar objetos del mismo tipo. Diferentes clases pueden tener algunas propiedades y comportamientos comunes, que pueden generalizarse en una clase que puede ser compartida por otras clases





Como se lleva a cabo la herencia en Java

- Naturalmente se necesitan dos clases, una super clase y una subclase. La subclase heredara los métodos y atributos de la superclase.
- **public class Circulo extends ObjetoGeometrico**

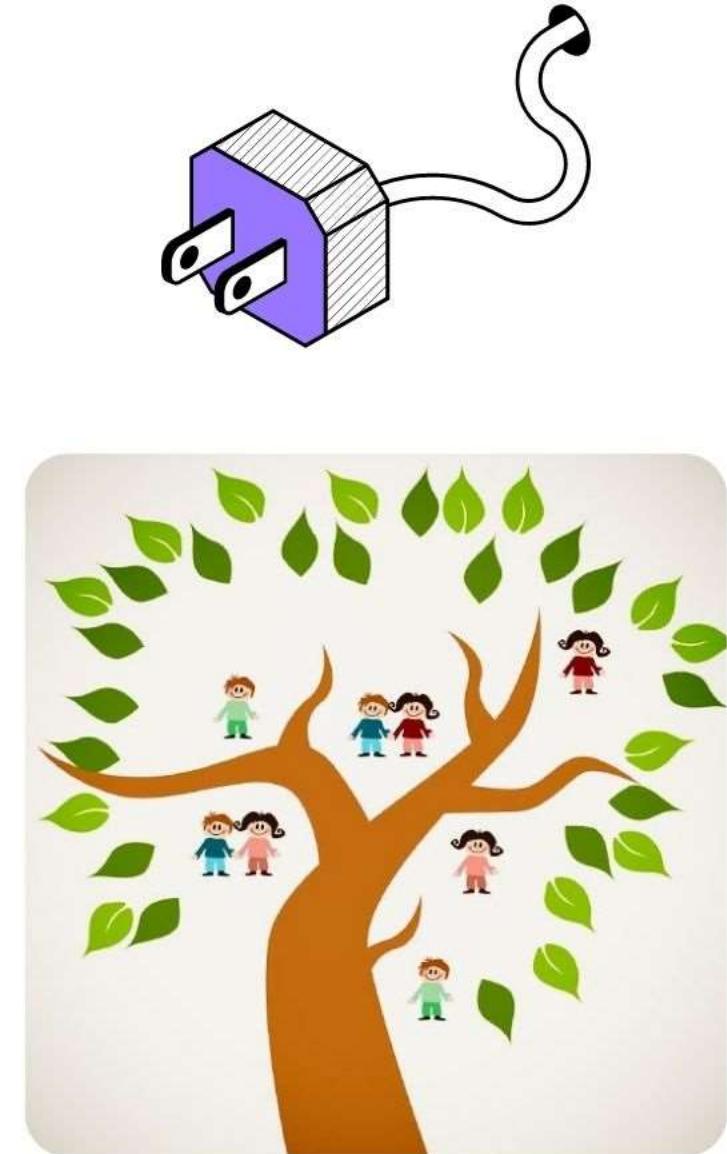
Tenga en cuenta los siguientes puntos con respecto a la herencia

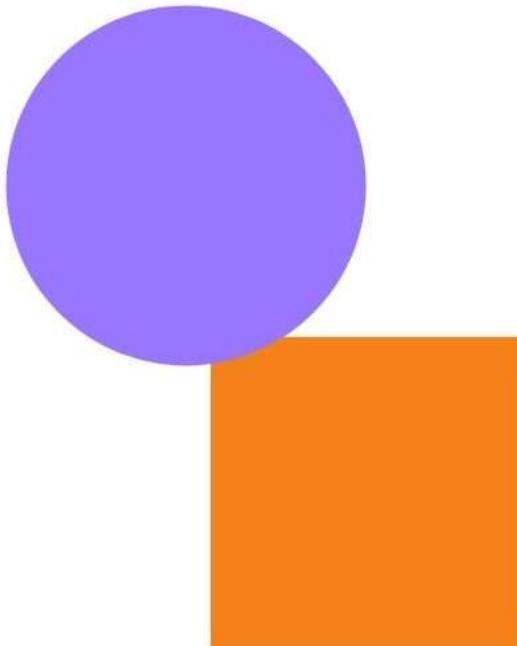
- Los campos de datos privados en una superclase no son accesibles fuera de la clase
- La herencia se utiliza para modelar la relación **es-un**

Polimorfismo

Conceptos clave

- Etimológicamente significa muchas formas.
- En programación orientada a objetos, polimorfismo es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su invocación.





Sobre escritura de un método

- Para sobre escribir un método, el método debe definirse en la subclase utilizando la misma firma y el mismo tipo de retorno que en su superclase.
- Una subclase hereda métodos de una superclase. A veces es necesario que la subclase modifique la implementación de un método definido

Tenga en cuenta los siguientes puntos con respecto a la sobre escritura

- Un método de instancia puede sobre escribirse solo si es accesible. Por lo tanto, un método privado no puede sobre escribirse porque no es accesible fuera de su propia clase.
- Al igual que un método de instancia, un método estático se puede heredar. Sin embargo, un método estático no se puede sobre escribir. Si un método estático definido en la superclase se redefine en una subclase, el método definido en la superclase se oculta.