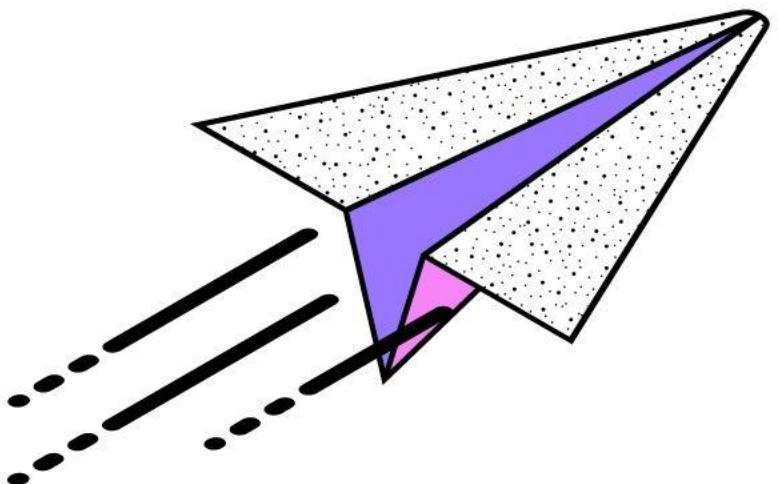




# Sección 3

Creación y uso de arreglos.



Acceso a un valor en un arreglo o ArrayList.

Declarar, instanciar e inicializar un arreglo unidimensional.

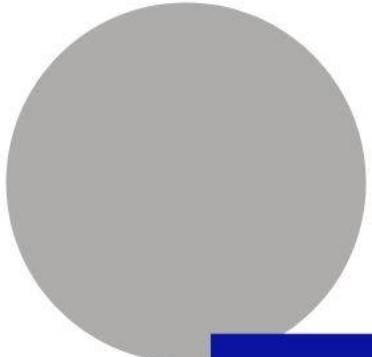
Uso de la sentencia import para trabajar con API's existentes de java.

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

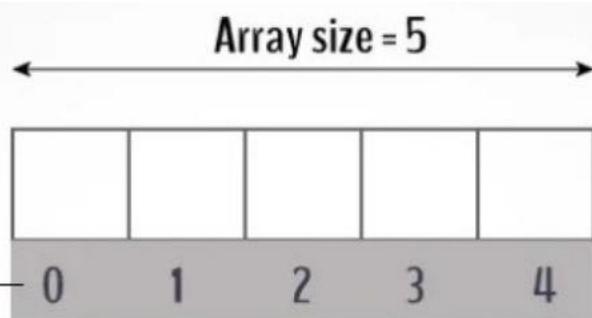
Es una colección de datos de un mismo tipo. Una sola variable de tipo arreglo o array puede hacer referencia a una gran colección de datos.

## 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];
```

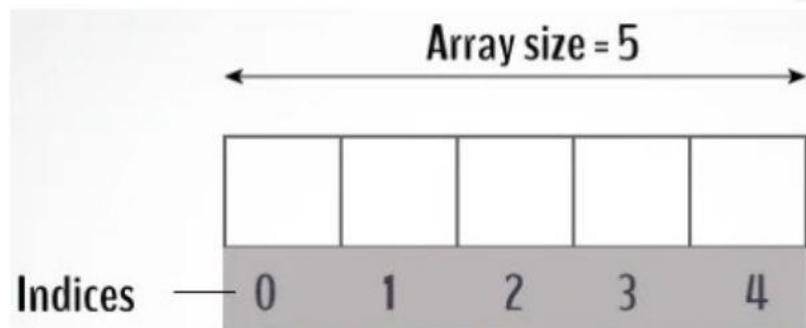
Se crea un arreglo, que guardara datos de tipo double, con una capacidad de 10. El arreglo tiene valores por defecto.

## 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;
```



## ArrayList

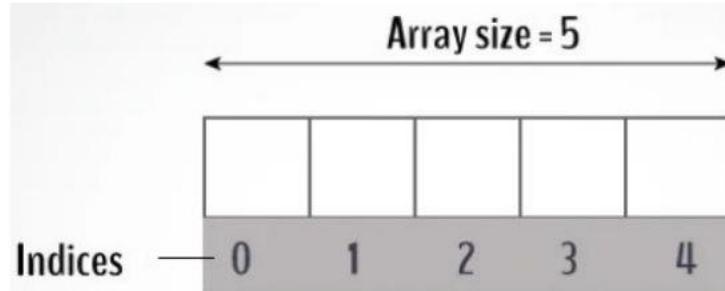
Una ArrayList se implementa usando un arreglo. En un arreglo convencional el tamaño de este era fijo una vez definido, y este no podía cambiar, en un ArrayList esto no es así, ya que el arreglo crece de manera dinámica, es decir su tamaño puede variar.

## 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.

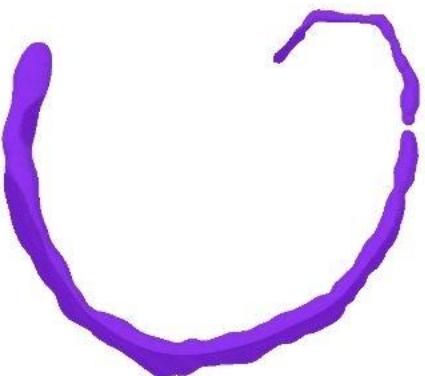
```
java nombrePrograma [argumento1, argumento2 ... ]
```

## 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.

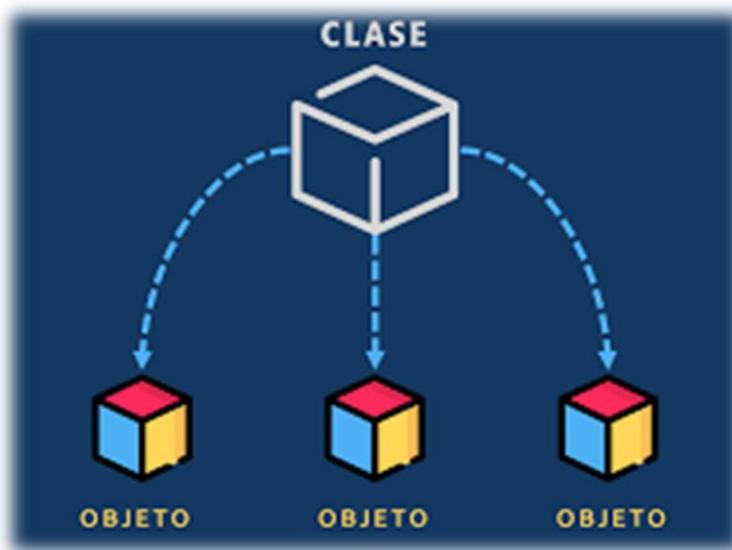
## Ejemplo de uso del ciclo for con un arreglo.

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]);  
}
```



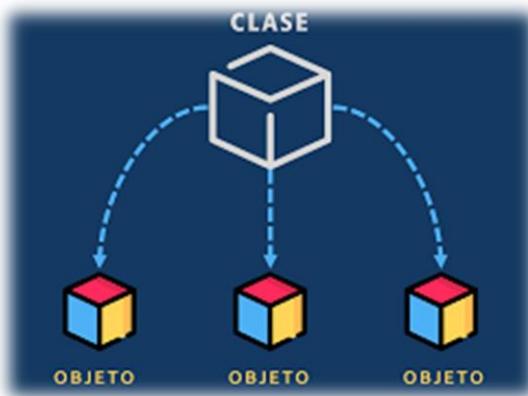
# Introducción a la programación orientada a objetos;





## Introducción a la programación orientada a objetos;

Establece que todos los problemas deberían ser resueltos mediante entidades que absorban un estado interno y un 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*.



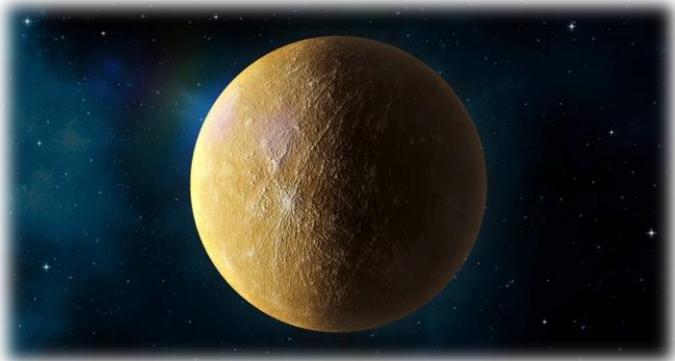
# Clase;

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

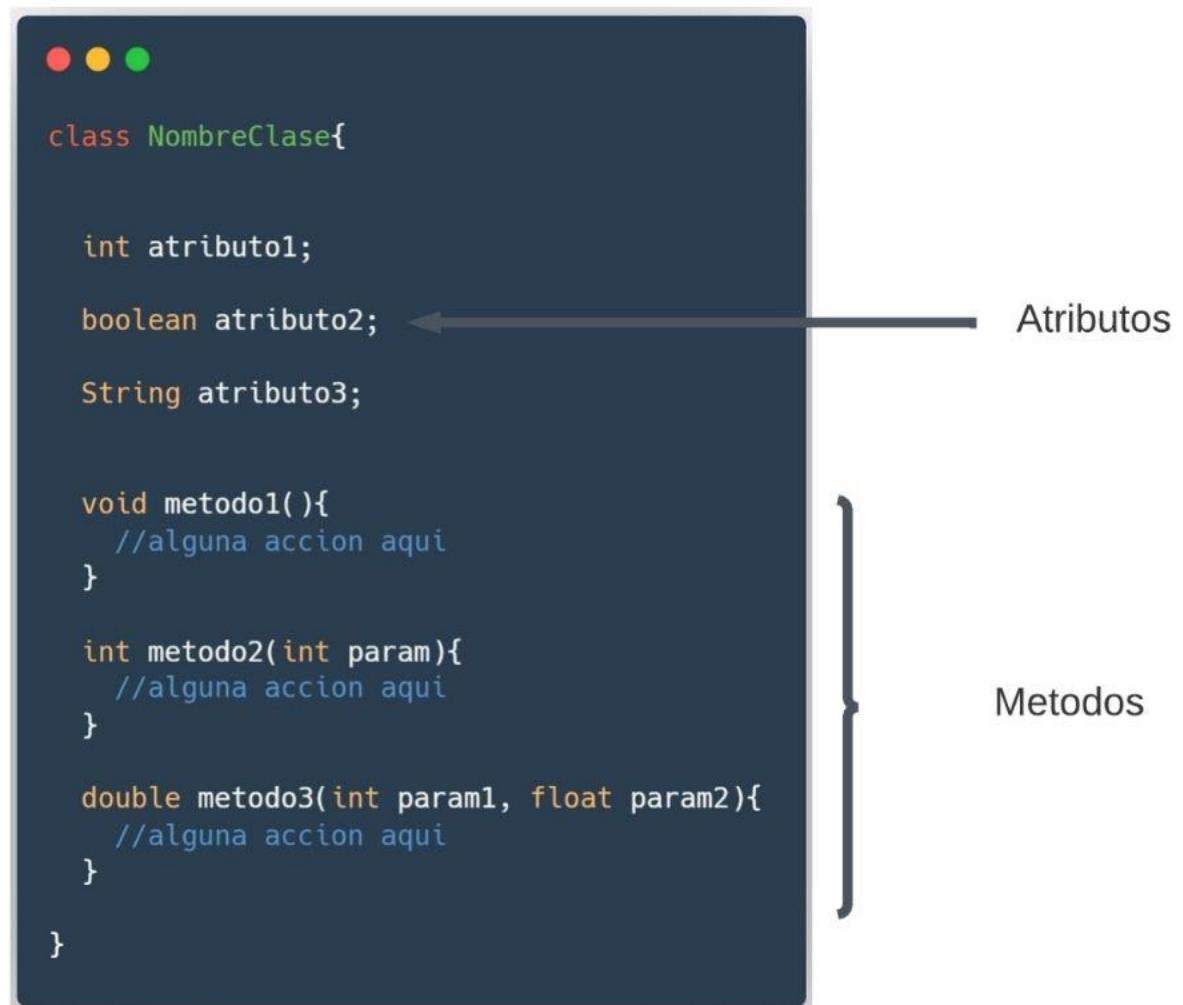


Chevrolet Corvette

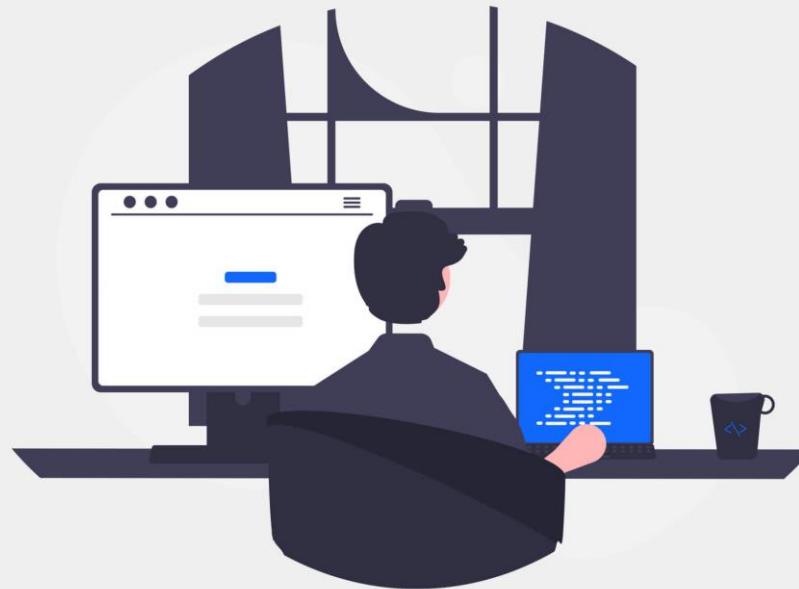


# Componentes de una clase

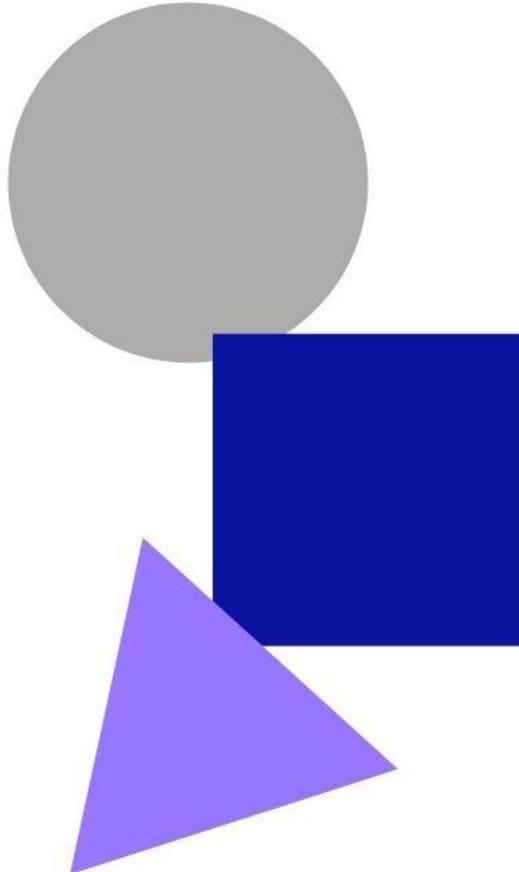
todas las clases tienen esta estructura básica



# Ejercicios prácticos



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**.

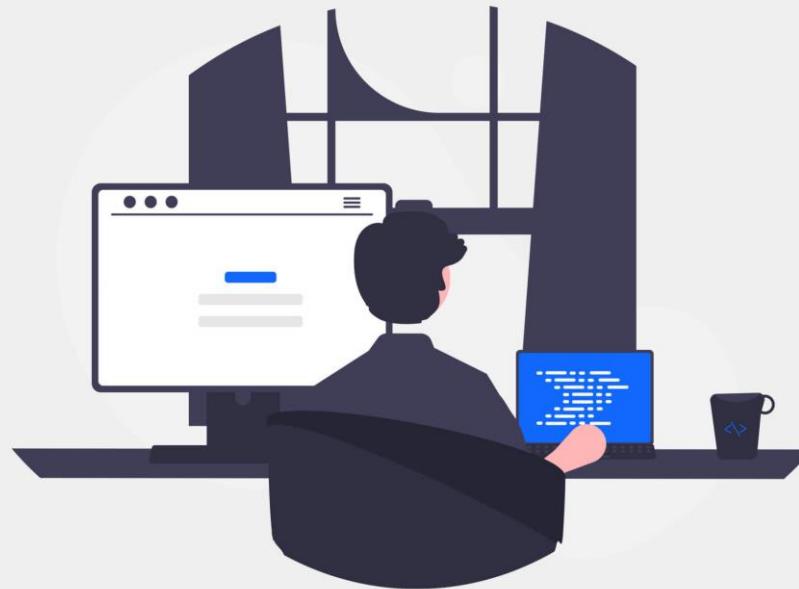


## Ejemplo de creación de objetos

- 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 "  
        circle1.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 "  
        circle2.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 "  
        circle1.radio + " es " + circulo1.getArea());  
  
    }  
}
```

# Ejercicios prácticos



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.

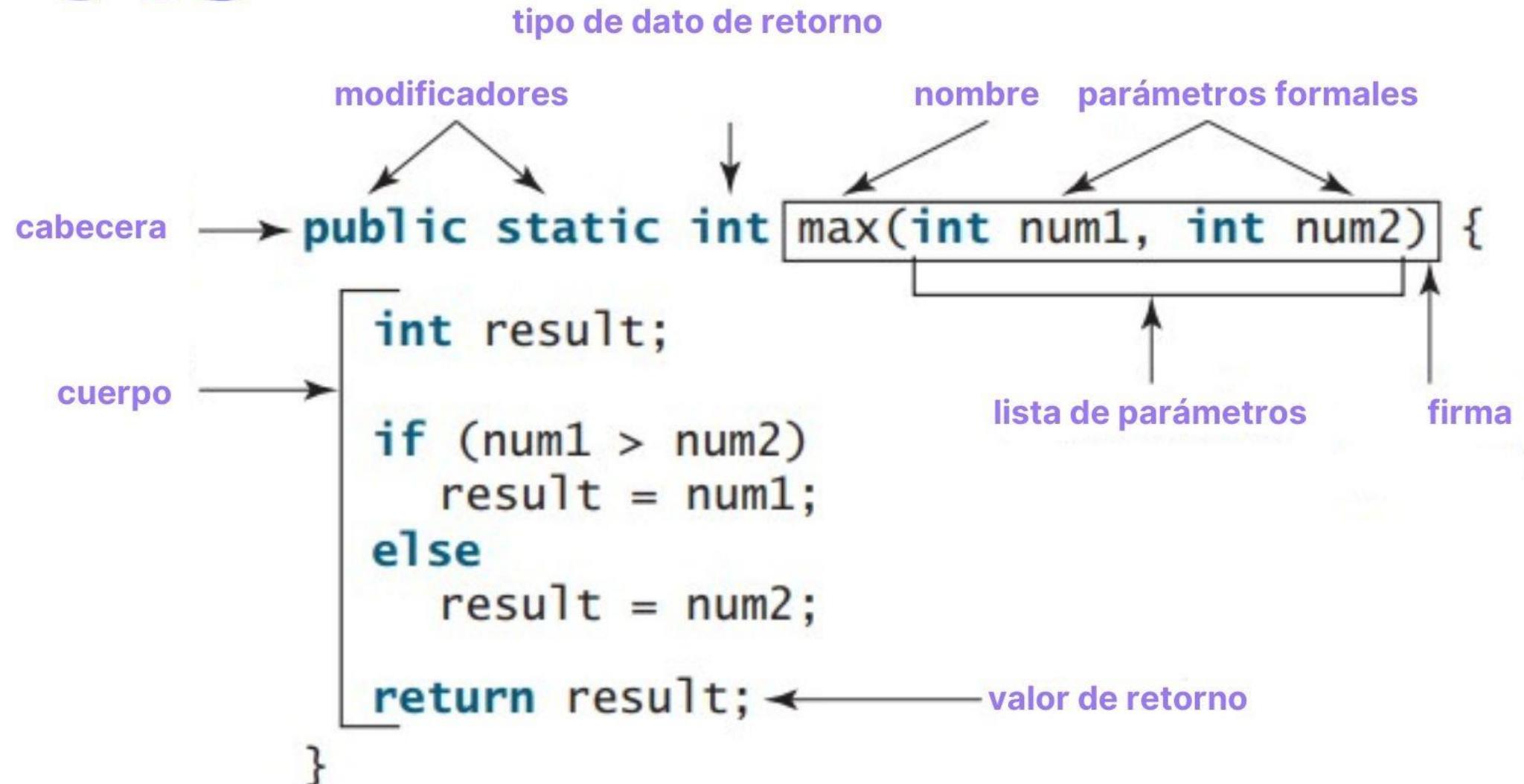
## Que define el encabezado

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**.



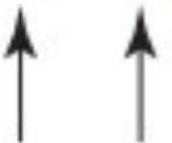
# Definición de un Método



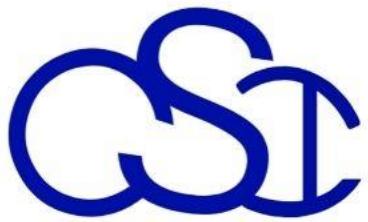


# Invocación de un Método

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



parámetros actuales (argumentos)

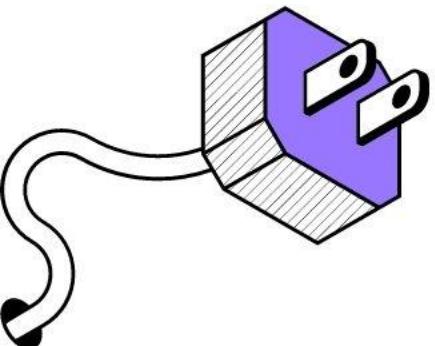


## variables y métodos estáticos

- 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.





# 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.

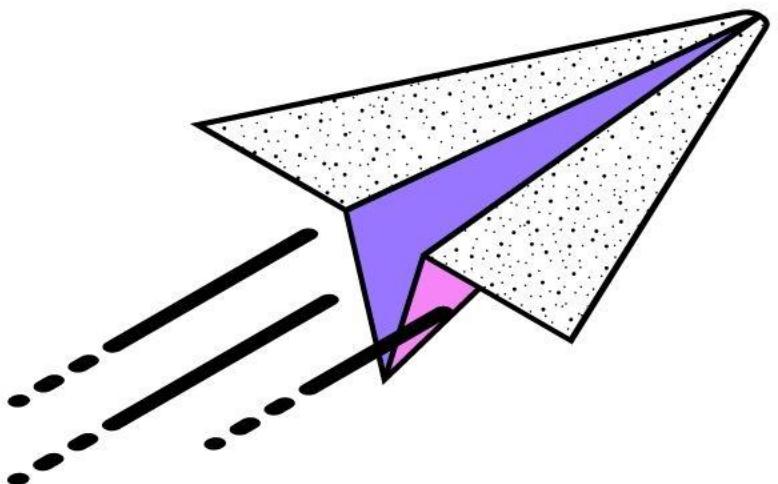


```
public class CircleWithPrivateDataFields {  
    private double radius = 1;  
  
    private static int numberofObjects = 0;  
  
    public CircleWithPrivateDataFields() {  
        numberofObjects++;  
    }  
  
    public CircleWithPrivateDataFields(double newRadius) {  
        radius = newRadius;  
        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



## 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;
```

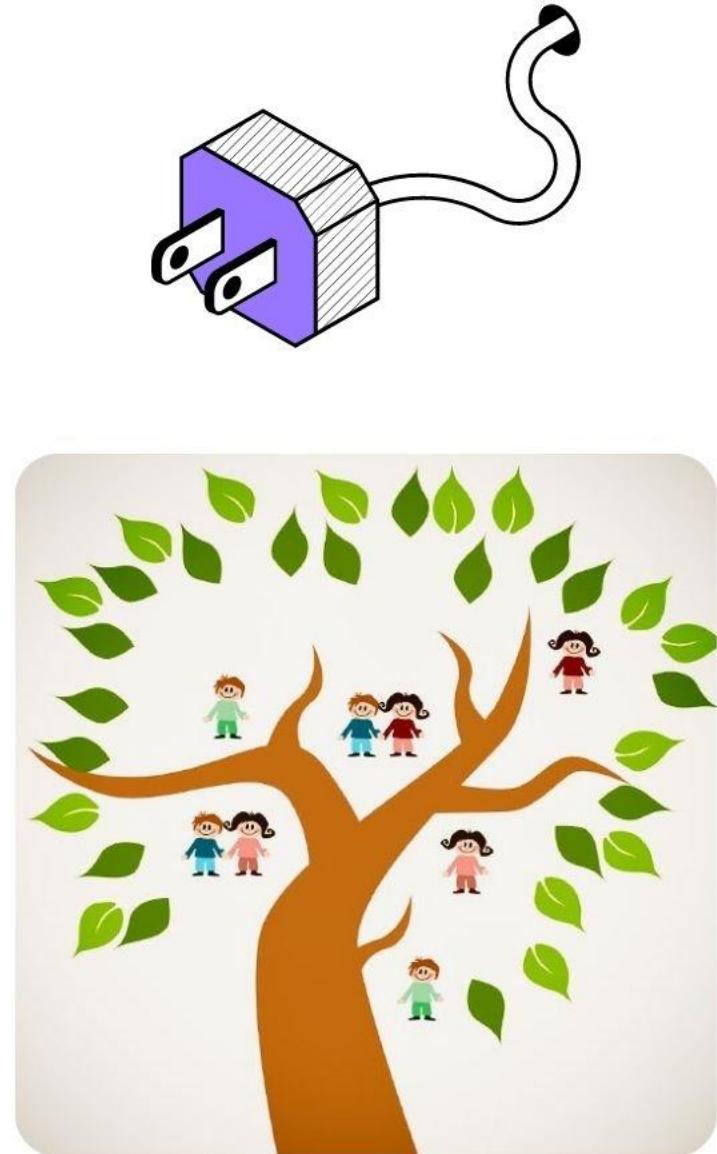
- Por default siempre se importa `java.lang.*`

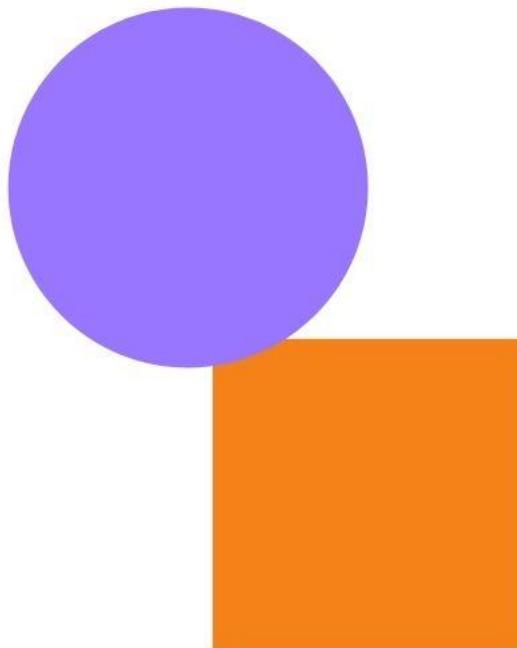
- No necesitas importar las clases que estén en el mismo paquete.

# 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

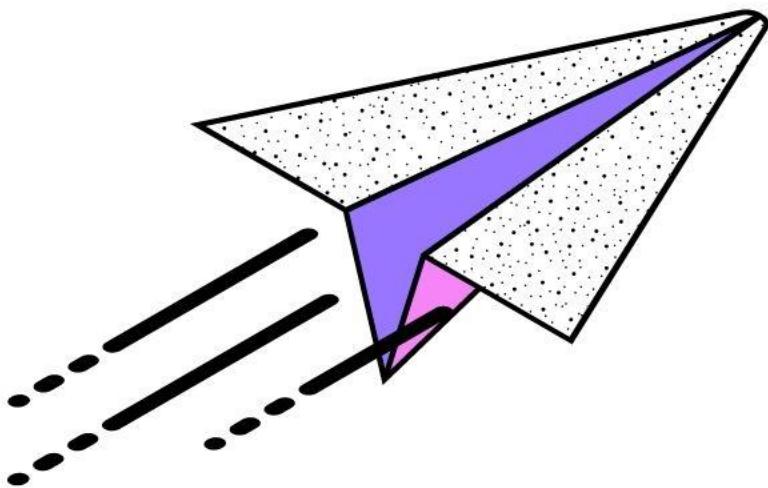
- Naturalmente se necesitan dos clases, una super clase y una subclase. La subclase heredara los métodos y atributos de la superclase.
- **public class Circulo extends ObjetoGeometrico**

## 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**

# 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

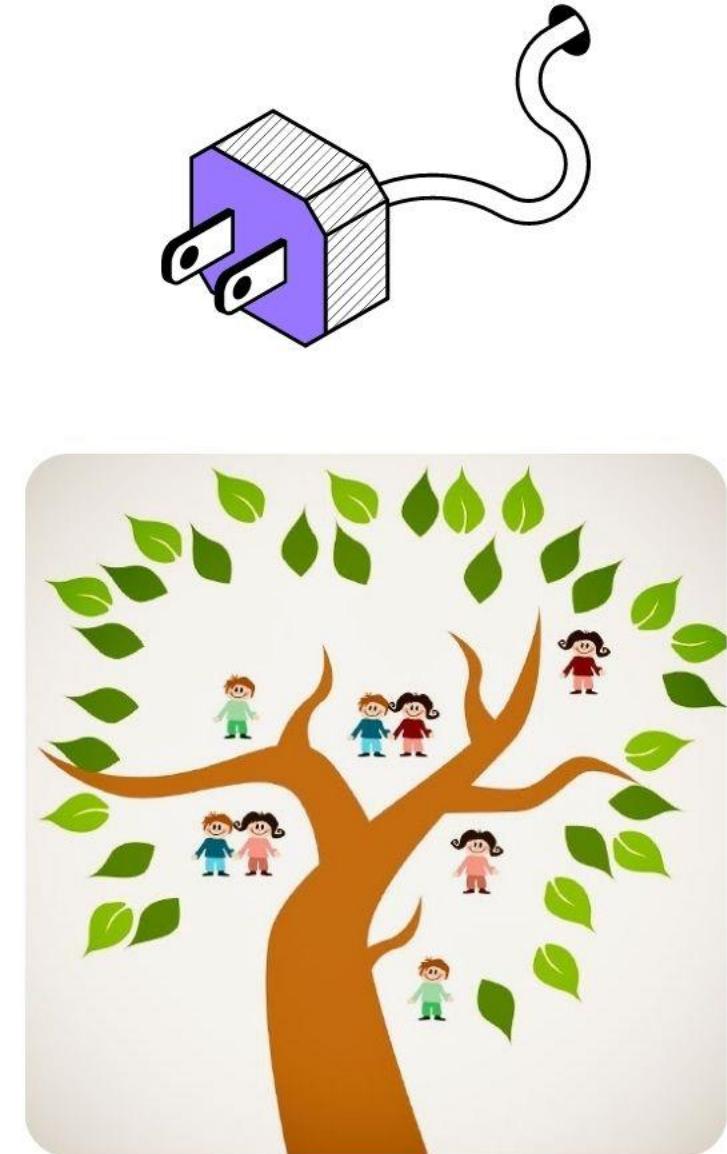
Para distinguir una interfaz de una clase, Java usa la siguiente sintaxis para definir una interfaz:

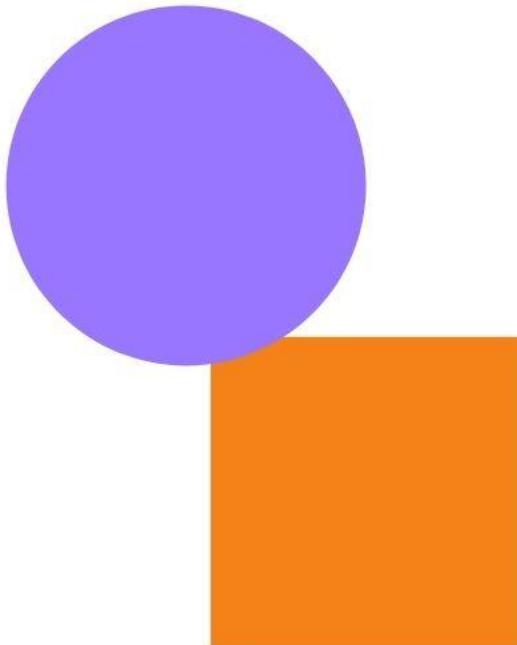
```
modifier interface InterfaceName {  
    /** Constant declarations */  
    /** Abstract method signatures */  
}
```

# 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.