



# Introducción a la Programación en Java

---

# Programación Orientada a Objeto

- El lenguaje de programación java es 100% orientado a objeto, en java todo se considera un objeto

Clase Taxi { --- > **EL NOMBRE DE LA CLASE**

**Propiedades:** --- > **También denominadas atributos o campos (fields)**

Matrícula identificativa

Distrito en el que opera

Tipo de motor diesel o gasolina

**Constructor de la clase** --- > **Definición de qué ocurre cuando se crea un objeto del tipo definido por la clase**

**Operaciones disponibles:** --- > **Métodos de la clase**

Asignar una matrícula

Asignar un distrito

Asignar un tipo de motor

}

# POO. Clase. Propiedades

---

- Para declarar las propiedades o atributos de una clase es necesario:
  - Indicar visibilidad
  - Tipo de dato

```
private int saldo;  
private int transacciones;
```

# POO. Clase. Constructores

```
public class CajaAhorro {  
  
    private int saldo;  
    private int transacciones;  
    private String codigo_cliente;  
  
    //Constructores  
    public CajaAhorro(String codigo_cliente, int saldo ) {  
        this.codigo_cliente = codigo_cliente;  
        this.saldo = saldo;  
        this.transacciones = transacciones;  
    }  
  
    public CajaAhorro(String codigo_cliente) {  
        this(codigo_cliente, 5000 );  
    }  
  
    public CajaAhorro() {  
  
    }  
}
```

Diagram illustrating the constructors of the `CajaAhorro` class:

- A red box highlights the `public` keyword in the first constructor.
- A red box highlights the `this` keyword in the first constructor.
- A red box highlights the `public` keyword in the second constructor.
- A red box highlights the `this` keyword in the second constructor.
- A red box highlights the `public` keyword in the third constructor.
- A red box highlights the `this` keyword in the third constructor.
- A red arrow points from the `public` keyword in the first constructor to the `public` keyword in the second constructor.
- A red arrow points from the `this` keyword in the first constructor to the `this` keyword in the second constructor.
- A red arrow points from the `public` keyword in the second constructor to the `public` keyword in the third constructor.
- A red arrow points from the `this` keyword in the second constructor to the `this` keyword in the third constructor.
- A red dashed line with an arrow points from the text "llama a otro constructor" to the `this` keyword in the second constructor.

# POO. Clase. Constructores (cont.)

---

- Para crear un objeto de la clase se usa la siguiente sintaxis:

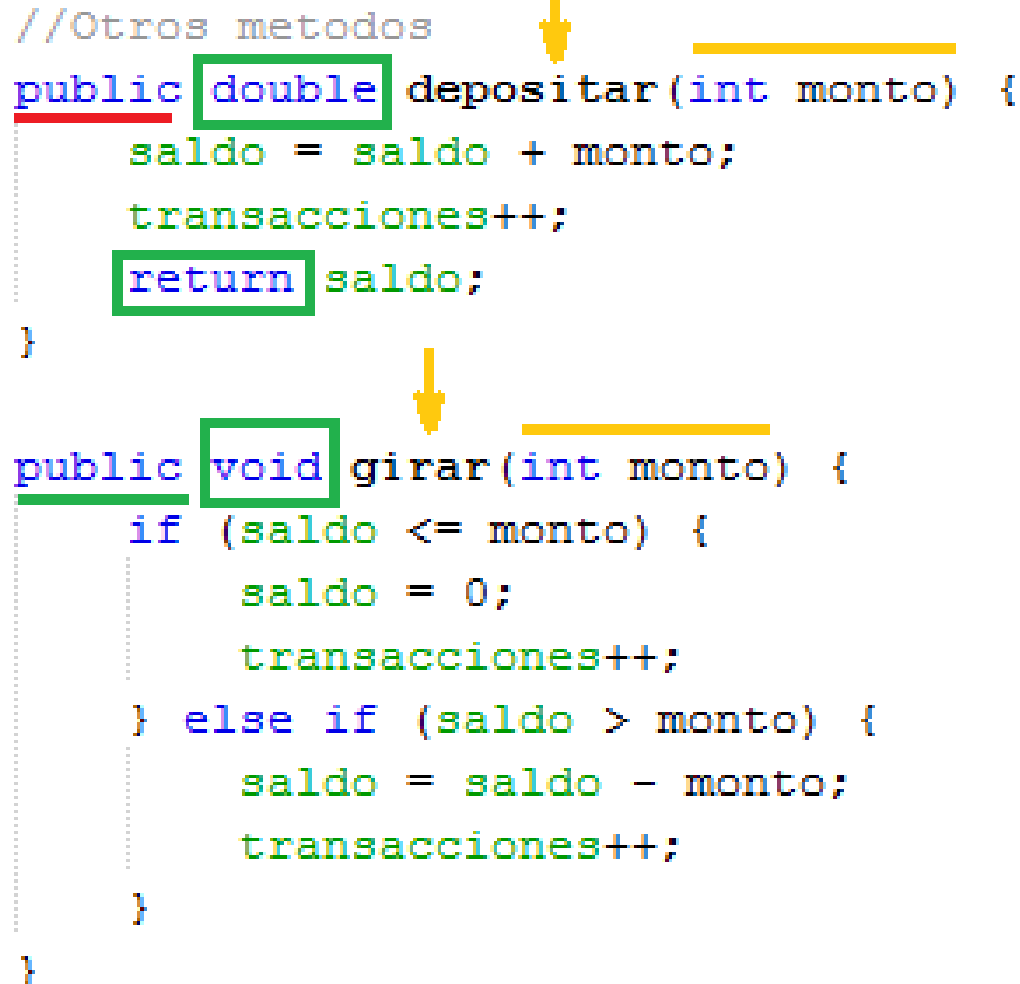
```
CajaAhorro cuenta = new CajaAhorro();
```

# POO. Clase. Métodos

---

```
//Otros metodos
public double depositar(int monto) {
    saldo = saldo + monto;
    transacciones++;
    return saldo;
}

public void girar(int monto) {
    if (saldo <= monto) {
        saldo = 0;
        transacciones++;
    } else if (saldo > monto) {
        saldo = saldo - monto;
        transacciones++;
    }
}
```



# POO. Clase. Métodos (cont.)

---

- Atributos y métodos estáticos

```
public static final double PI
```

```
//Para elevar un número a una potencia n en java usando el método Math.pow.  
int radio = (int) Math.pow(6, 2);  
double AreaCirculo = Math.PI * radio;  
System.out.println("El area del circulo, es: " + AreaCirculo + " cm2");  
  
double raiz = Math.sqrt(9);  
System.out.println(raiz);
```

# POO. Clase. Método (cont.)

---

- Sintaxis métodos set y get.

```
public int GetSaldo() {  
    return saldo;  
}
```

```
public void SetSaldo(int saldo) {  
    this.saldo = saldo;  
}
```

Ver ejemplo: clase caja de ahorro





# POO

---

- La POO se basa en cuatro conceptos:
  - Encapsulación
  - Herencia
  - Polimorfismo
  - Abstracción

# POO. Encapsulación

---

- Modificadores de acceso

Modificador	Visible desde la propia <b>clase</b>	Visible desde otra clase del mismo <b>package</b>	Visible desde una <b>Subclase</b>	Acceder desde <b>cualquier</b> clase
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
No especificado	Sí	Sí	No	No
private	Sí	No	No	No

# POO. Herencia

---

```
           subclass           superclass
           |                   |
public class Jefe extends Empleado{

    private float incentivo;

    public Jefe(String nombre, int salario, float incentivo){

        super(nombre, salario);

        this.incentivo = incentivo;

    }

    public float SalarioConIncentivo() {
        int salarioBase = super.GetSalarioBase();
        return salarioBase += salarioBase*incentivo;
    }
}
```

## POO. Herencia (cont.)

---

- **Nota:** En caso que no quieras que una clase B herede de otra clase A, debes declarar la clase A como **final** lo que haces que no se pueda extender y la cadena de la herencia finalizaría.



# POO. Polimorfismo

---

```
public class Persona {  
  
    private String nombre;  
    private int edad;  
  
    public Persona(String nombre, int edad)  
    {...4 lines }  
  
    public String MostrarDatos() {  
        return "Persona, que se llama: " + nombre + " y tiene " + edad + " años de edad";  
    }  
}
```

# POO. Polimorfismo (cont.)

```
public class Estudiante extends Persona{

    private String grado;
    private int cantAsignaturas;

    public Estudiante(String nombre, int edad, String grado, int cantAsignaturas)
    {...5 lines }

    public String MostrarDatos(){
        return "Estudiante, que cursa el: " + grado + " y lleva " + cantAsignaturas
            + " asiganturas";
    }
}

public class Trabajador extends Persona{

    private String trabajadorID;
    private String empresa;
    private int agnoTrabajados;

    public Trabajador(String nombre, int edad, String trabajadorID,
        String empresa, int agnoTrabajados)
    {...6 lines }

    public String MostrarDatos(){
        return "Trabajador, que labora en la empresa " + empresa + " desde hace "
            + agnoTrabajados + " años" + " y su ID es: " + trabajadorID;
    }
}
```

Ver ejemplo de polimorfismo

# POO. Polimorfismo (cont.)

- Palabra reservada **final**: al usarla en la definición del método, implica que cuando se establezca la herencia en la subclase no se podrá crear un método con el mismo nombre.

## getTime

```
public final Date getTime()
```

Returns a Date object representing this Calendar's time value (millisecond offset from the Epoch").

### Returns:

a Date representing the time value.

### See Also:

setTime(Date), getTimeInMillis()

API Java. Clase Calendar. Método getTime  
Al declararlo como **final**, evita que se cree en otra clase, que herede de Calendar, un método con el mismo nombre.

# POO. Clases abstractas y método abstracto

```
public abstract class Animal {    Clase abstracta

    private String nombre;

    /**
     * Constructor de la clase Animal
     * @param nombre
     */
    + public Animal (String nombre) { ...4 lines }

    /**
     * Metodo Abstracto tipoAnimal, la implementación depende
     * de las clases concretas que extiendan la clase Animal
     */
    public abstract String tipoAnimal();
                                     Métodos abstractos
    public abstract String comunicarse();

}
```



# POO. Clases abstractas y método abstracto (cont.)

```
public class Perro extends Animal{
```

```
    public Perro(String nombre) { ...4 l
```

```
    @Override  
    public String tipoAnimal() {  
        return "Tipo Animal : Es un Perro";  
    }
```

```
    @Override  
    public String comunicarse() {  
        return "Metodo comunicarse : El perro Ladra";  
    }
```

```
public class Gato extends Animal{
```

```
    public Gato(String nombre) { ...4 lines }
```

```
    @Override  
    public String tipoAnimal() {  
        return "Tipo Animal : Es un Gato";  
    }
```

```
    @Override  
    public String comunicarse() {  
        return "Metodo comunicarse : El gato maulla";  
    }
```

Ver ejemplo de clase abstracta

# Hemos terminado por hoy

---

