

Programación

Unidad 3: Java y POO

Unidad 3

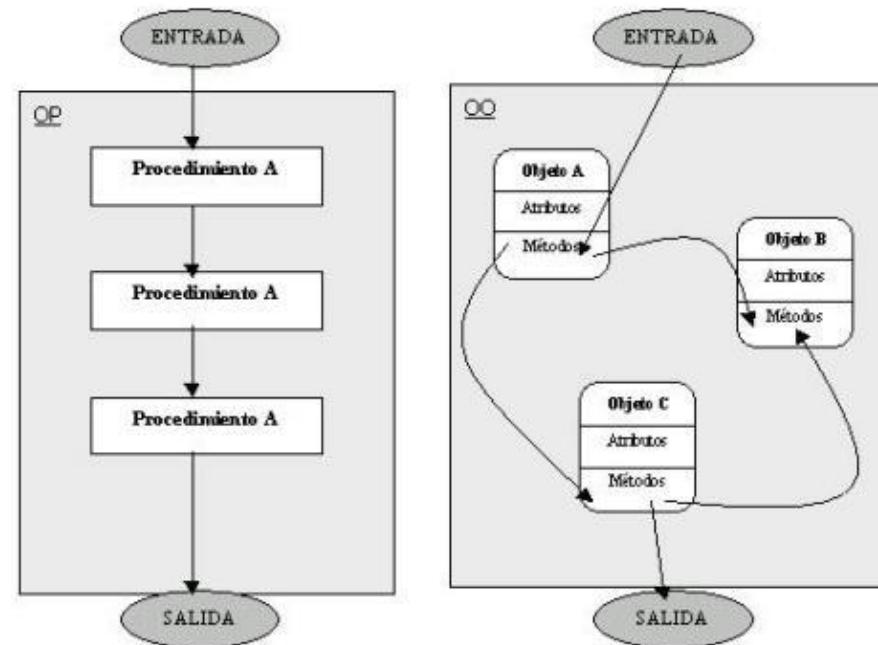
- Programación Orientada a Objetos
- Herencia
- Polimorfismo
- Clases en Java
- Atributos y métodos
- Constructores
- Convenciones en Java
- Objetos

Programación Orientada a Objetos (POO - OOP)

Unidad 3: Java y POO

Junto con el paradigma de la **orientación a procedimientos**, son las dos filosofías generales de diseño más importantes.

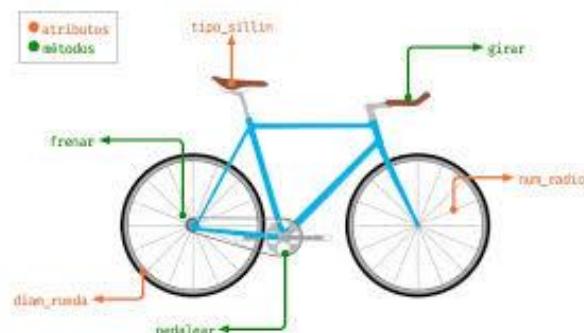
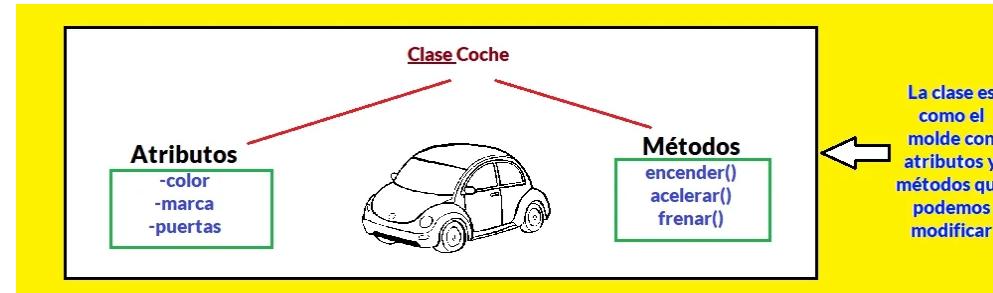
A diferencia de la orientación a procedimientos (OP), **la orientación a objetos** (OO) no concibe los procesos como una secuencia de procedimientos con su entrada y salida, sino que se basa en un conjunto de objetos interactuando:



Unidad 3: Java y POO

Es importante distinguir entre los conceptos de clase y objeto:

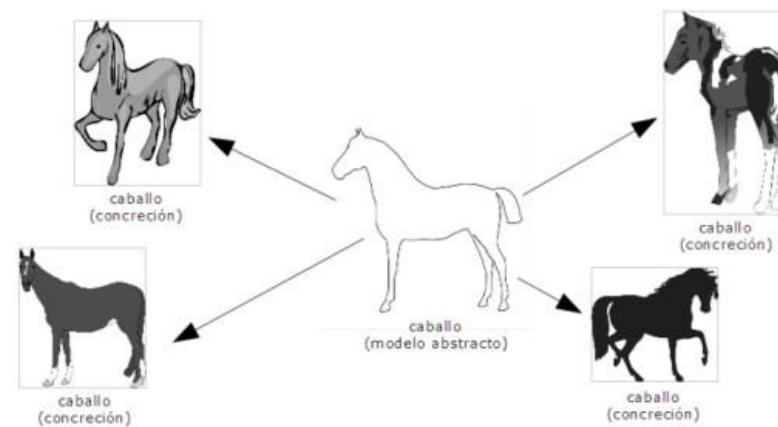
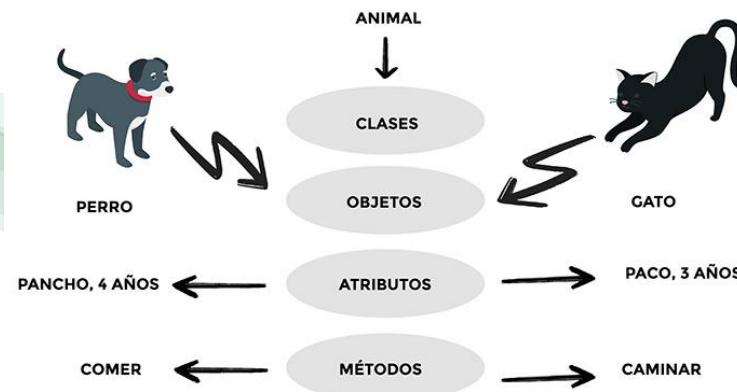
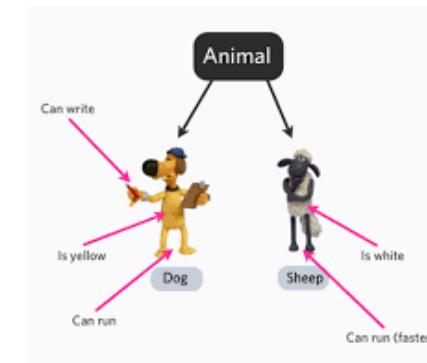
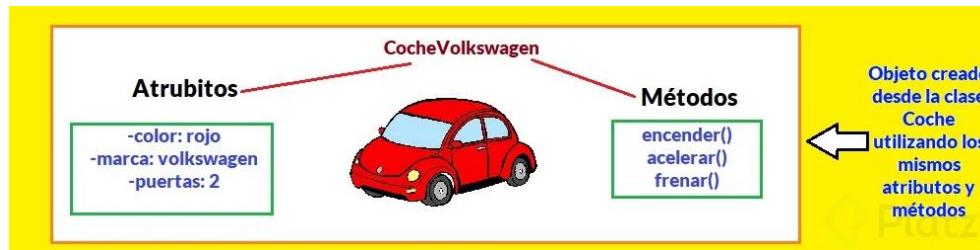
Clase: Es un modelo abstracto de un tipo de objeto. Define sus métodos y atributos.



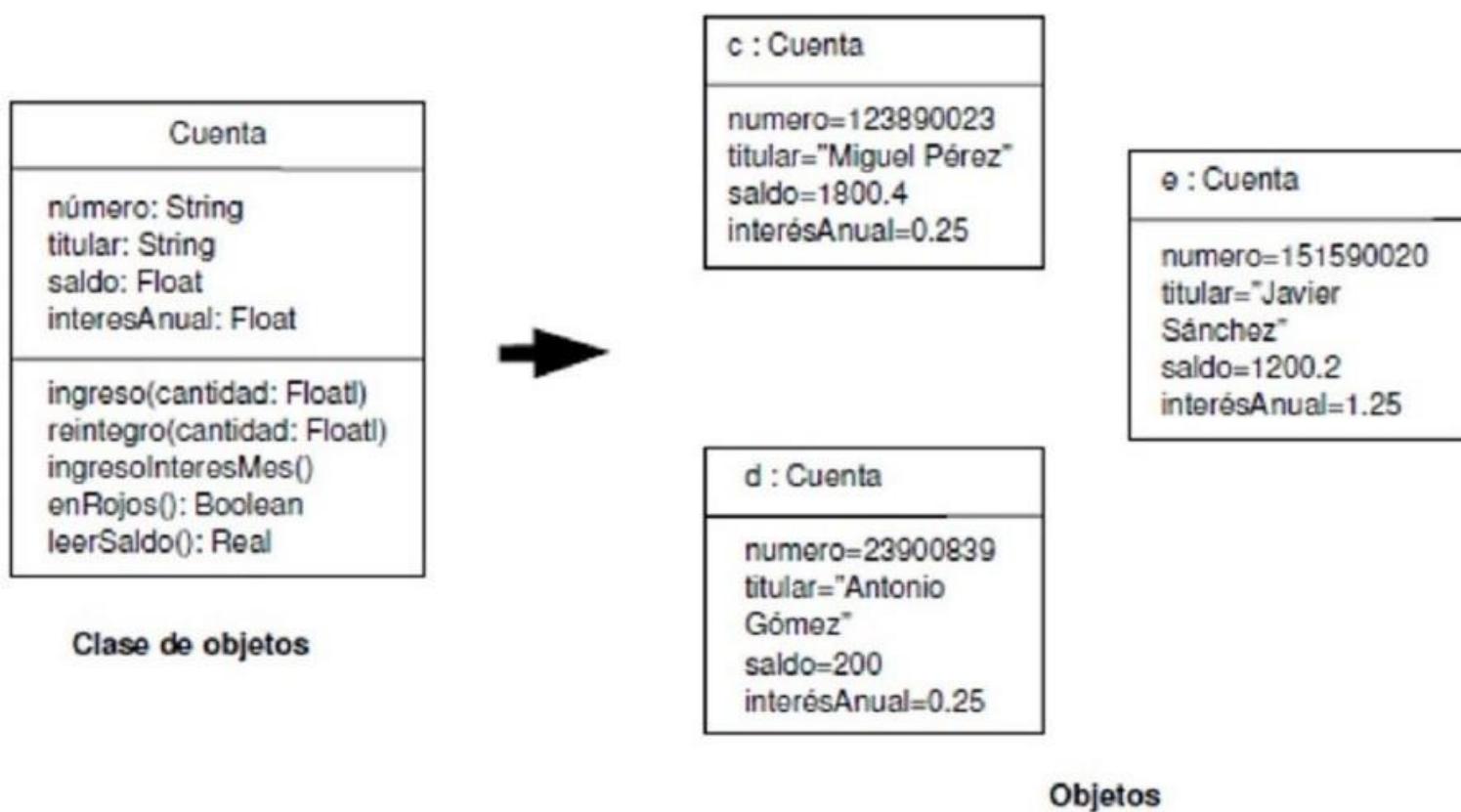
Unidad 3: Java y POO

Es importante distinguir entre los conceptos de clase y objeto:

Objeto: Es una instancia de una clase, es decir, la implementación con valores de un modelo abstracto.



Unidad 3: Java y POO



Unidad 3: Java y POO

Cuando finaliza el uso de un objeto, es frecuente la realización de ciertas tareas antes de su destrucción, principalmente la liberación de la memoria solicitada durante su ejecución.

En Java la **liberación de memoria** se realiza de **manera automática** por parte del **recolector de basura**, por tanto, la necesidad de este tipo de operaciones no existe en la mayor parte de los casos.



Unidad 3: Java y POO

Aplicaciones Orientadas a Objetos

En una aplicación orientada a objetos debe existir una clase que represente la propia aplicación. Este va a ser el **punto donde comienza la ejecución**.

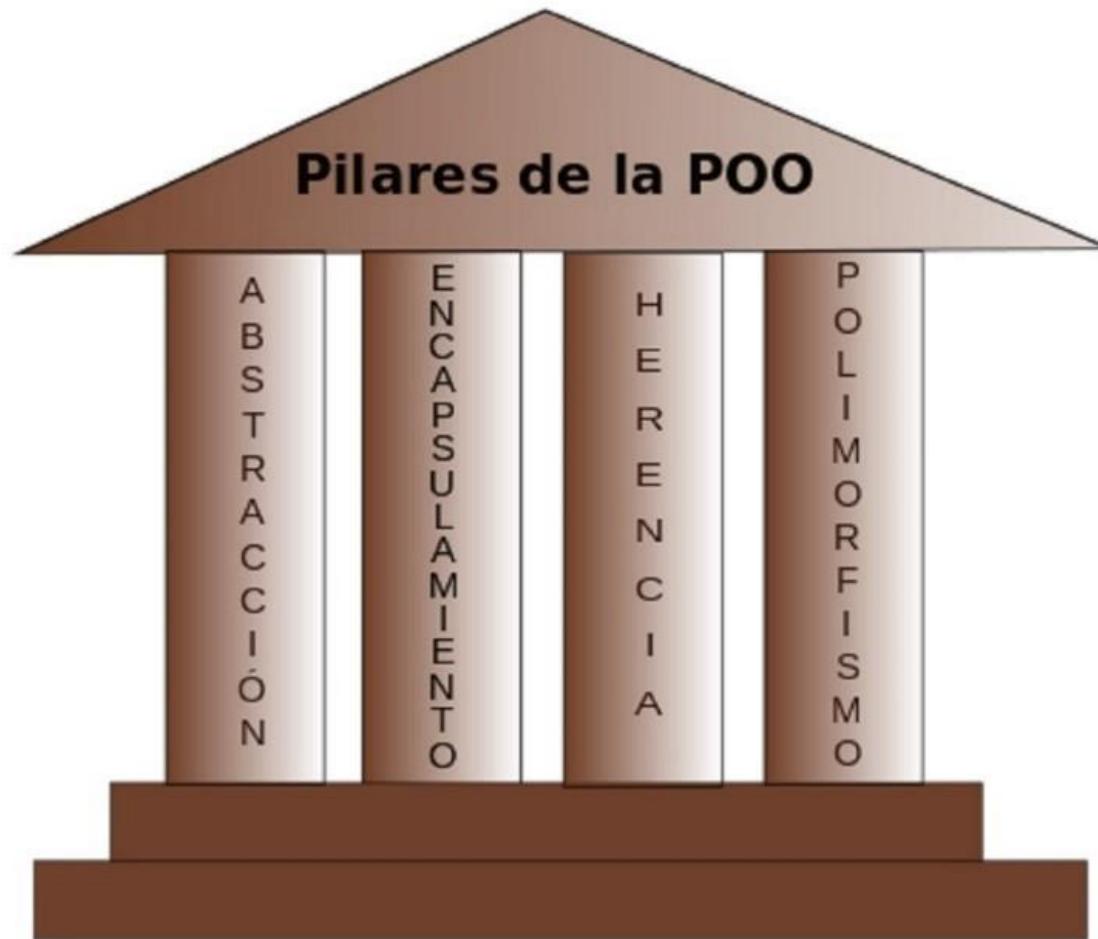
La **máquina virtual Java** se encarga de instanciar esta clase y llamar a una operación especial con nombre **main**. La existencia de esta operación especial es lo que caracteriza a la clase de aplicación.

La clase de la aplicación debe ser pública y no tener ningún constructor o un constructor por defecto.

Al menos debe implementar el **método main**, con la siguiente declaración:

```
public static void main(String[] args)
```

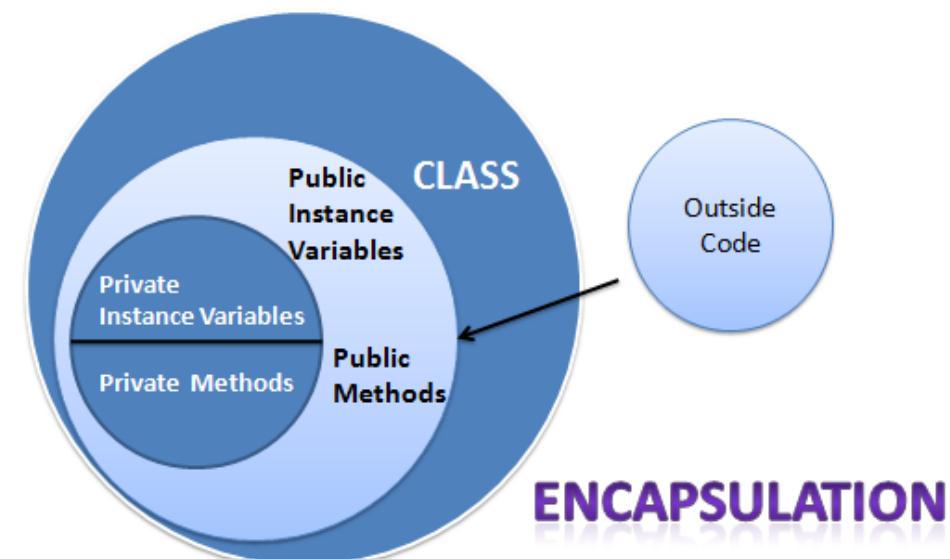
Unidad 3: Java y POO



Unidad 3: Java y POO

Encapsulamiento

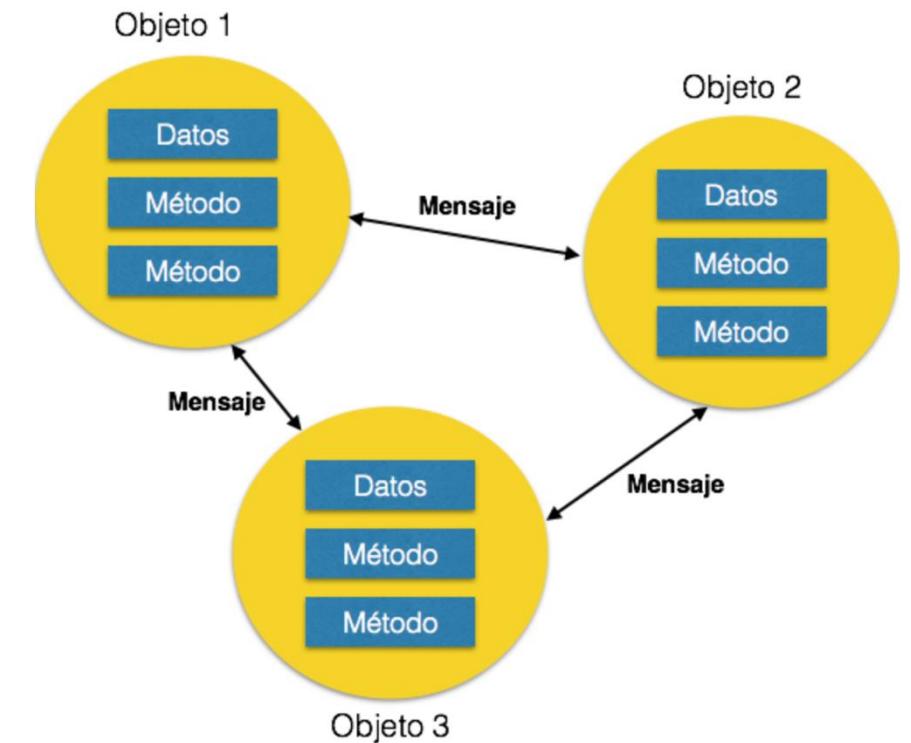
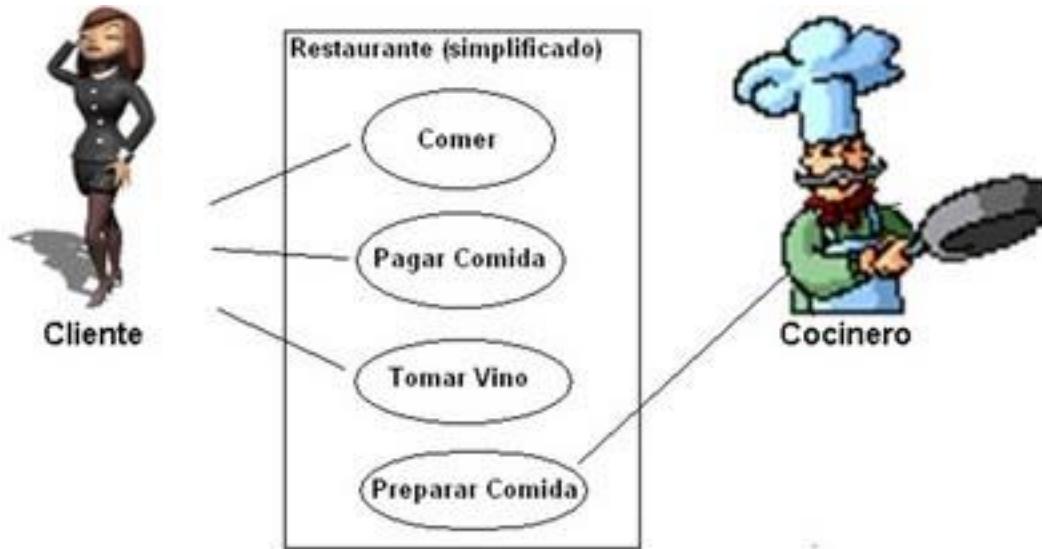
El objeto se encarga de **ocultar** sus datos al resto de objetos. La encapsulación permite una **seguridad mayor** en el acceso a los datos ya que este acceso depende directamente de cada objeto. Asimismo, permite **abstraer los detalles internos** de funcionamiento del objeto.



Unidad 3: Java y POO

Intercambio de mensajes

Los objetos se comunican entre sí mediante mensajes de invocación a métodos:

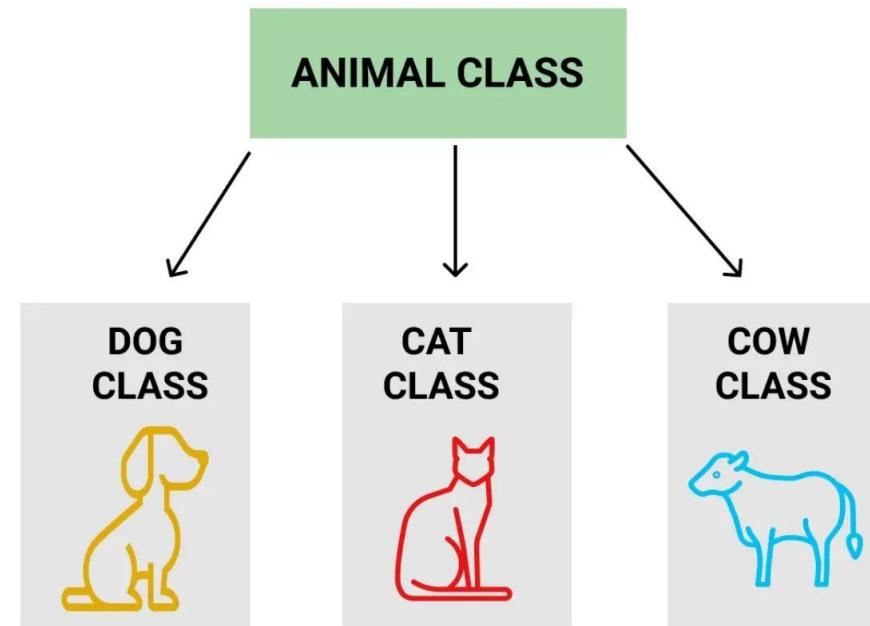


Herencia

Unidad 3: Java y POO

Herencia

Es el concepto que define la **adopción de todas las características** de una clase por parte de otra clase que es definida como **descendiente** o heredera de la primera.



Unidad 3: Java y POO

La herencia es un mecanismo de la OOP que permite construir una clase incorporando de manera implícita todas las características de una clase previamente existente. Las razones que justifican su necesidad son variadas:

Modelado de la realidad. Son frecuentes las relaciones de especialización/generalización entre las entidades del mundo real, por tanto, es lógico que dispongamos de un mecanismo similar entre las clases de objetos

Evitar redundancias. Toda la funcionalidad que aporta una clase de objetos es adoptada de manera inmediata por la clase que hereda, por tanto, evitamos la repetición de código entre clases semejantes

Facilitar la reutilización. Una clase no tiene por qué limitarse a recibir una serie de características de otra clase por herencia de forma pasiva. También disponen de cierto margen de adaptación de estas características

Soporte al polimorfismo

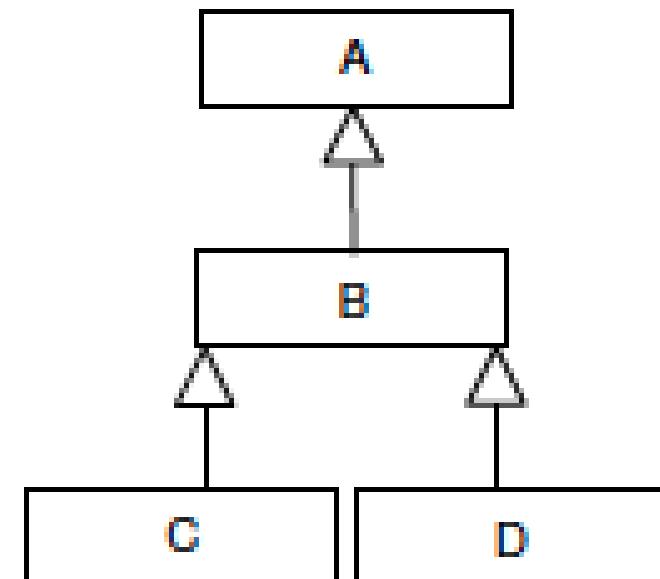
Herencia

Unidad 3: Java y POO

Sea una clase A. Si una segunda clase B hereda de A entonces decimos:

A es un **ascendiente o superclase** de B. Si la herencia entre A y B es directa decimos además que A es la clase padre de B

B es un **descendiente o subclase** de A. Si la herencia entre A y B es directa decimos además que B es una clase hija de A

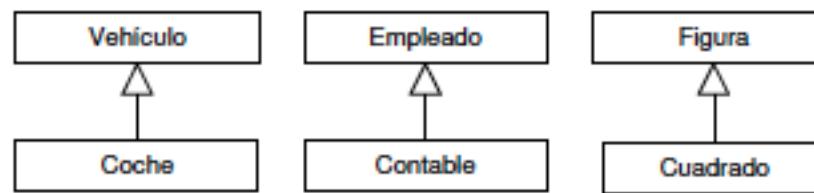


Unidad 3: Java y POO

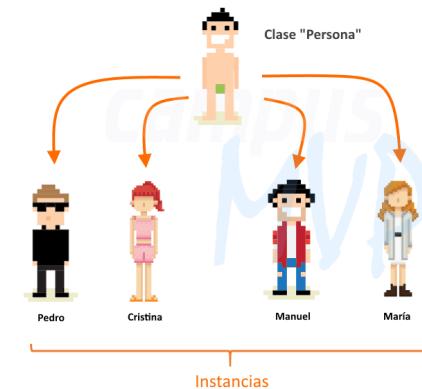
Todas las clases heredan automáticamente de una superclase universal. En Java esta superclase se denomina **Object**.

Existen diferentes situaciones en las que puede aplicarse herencia:

Especialización

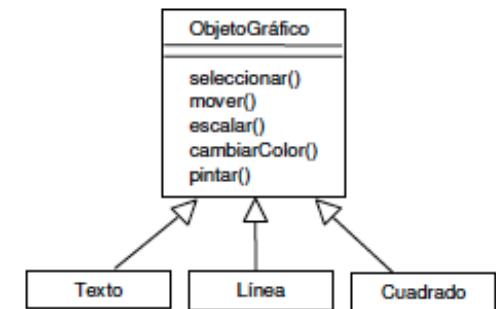
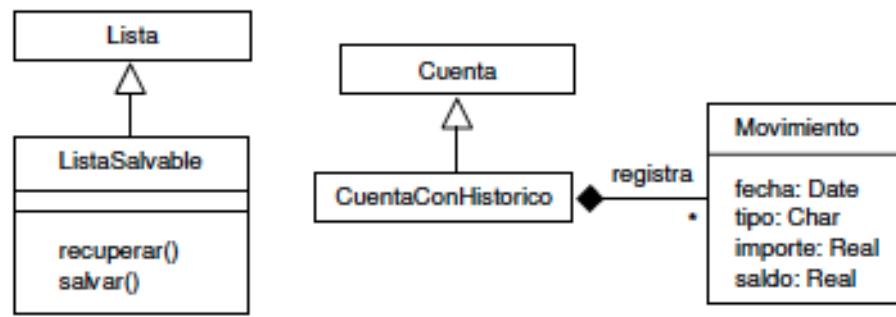


Extensión



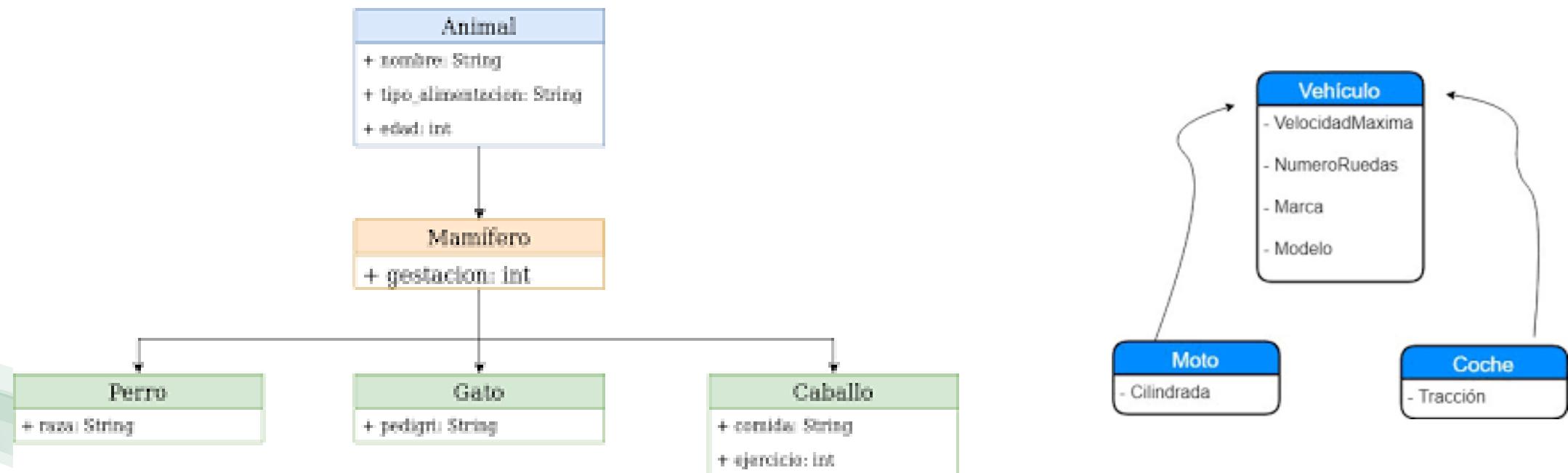
Especificación

Construcción



Unidad 3: Java y POO

Ejemplos de Herencia



Unidad 3: Java y POO

Hemos visto anteriormente como los distintos niveles de protección limitan el acceso a los miembros de la clase desde el exterior. ¿Pero cómo afectan estos niveles de protección a los miembros heredados?

Miembros públicos. Son accesibles desde los descendientes, y se heredan como públicos

Miembros privados. No son accesibles desde los descendientes

Miembros con acceso a nivel de paquete. Son accesibles desde los descendientes siempre y cuando pertenezcan al mismo paquete que el ascendiente. Se heredan con el mismo nivel de protección

Un nuevo nivel de protección es el de **miembros protegidos** (protected). Un miembro protegido es accesible únicamente desde los descendientes. Además, un miembro protegido mantiene en las subclases el nivel de acceso protegido

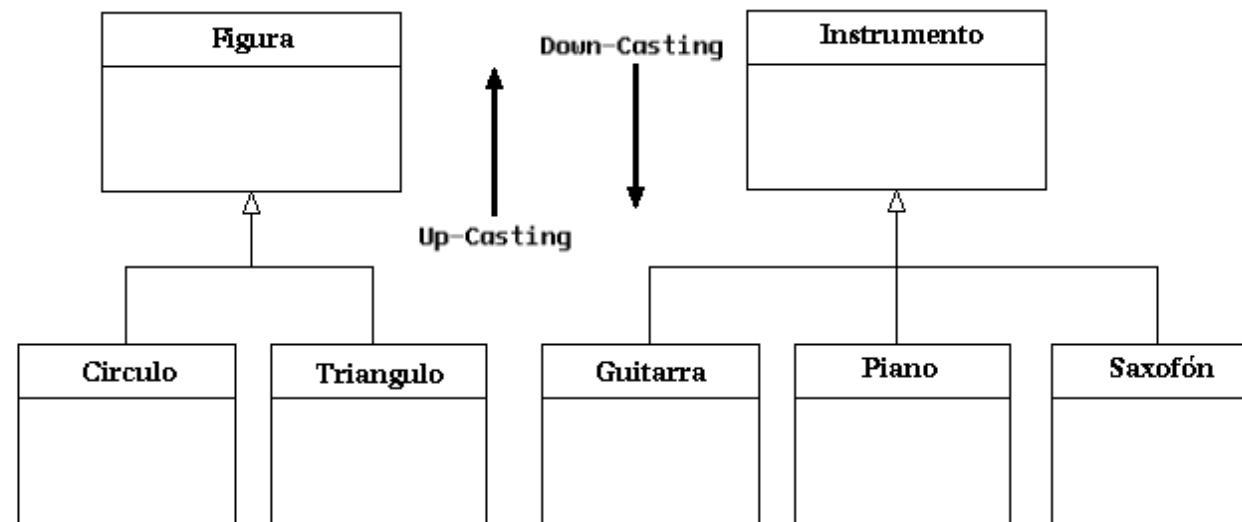
Polimorfismo

Unidad 3: Java y POO

Polimorfismo

Son dos mecanismos relacionados que otorgan a la OOP una gran potencia frente a otros paradigmas de programación. Únicamente tiene sentido por la **existencia de la herencia**.

El polimorfismo (o **upcasting**) consiste en la posibilidad de que una referencia a objetos de una clase pueda conectarse también con objetos de descendientes de ésta



Clases en Java

Unidad 3: Java y POO

Clases en Java

Una **clase** es una agrupación de variables y de métodos que operan sobre esos datos. A estos datos y funciones pertenecientes a una clase se les denomina **atributos y métodos**.

La **programación orientada a objetos** se basa en la programación de clases.

Un **programa** se construye a partir de un conjunto de clases.

Unidad 3: Java y POO

Clases en Java

- La **implementación de una clase** Java debe ir en un fichero en formato texto con la extensión **.java** y **nombre idéntico a la clase** implementada.
- La clase MiClase debe ir en un fichero: MiClase.java
- La **declaración de una clase** Java se realiza mediante la keyword: **class** seguida de su nombre.
- La keyword siempre va precedida por un **modificador de acceso**: public, protected, private o **default** (nada).

```
class MiClase{ //Cabecera  
    //Variables y constantes  
    final static int MICONSTANTE = 10;  
    public int miVariable = 0;  
  
    public MiClase(){  
        //Constructor  
        miVariable = 3;  
    }  
  
    //Métodos de la clase  
    public int getValor(){  
        return miVariable;  
    }  
  
    public void setValor(int valor){  
        miVariable = valor;  
    }  
  
    public void imprimeValor(){  
        System.out.println(miVariable);  
    }  
}
```

Unidad 3: Java y POO

Clases en Java

La **implementación de la clase** irá contenida en un bloque {} justo después de la declaración de esta.

Declaración de una clase:

```
modificador_acceso class nombre_clase {  
    // Cuerpo de la clase  
}
```

Ejemplo:

```
public class MiClase{  
}
```

Atributos y métodos

Unidad 3: Java y POO

Atributos y métodos

La **implementación de una clase** consiste en una serie de:

- Atributos.
- Métodos.

Declaración de un **atributo**:

modificador_acceso tipo nombre [= valor_inicial];

Ejemplo:

```
private boolean sw = true;  
private int i;
```

Unidad 3: Java y POO

Atributos y métodos

Declaración de un método:

```
modificador_acceso tipo_retorno nombre([tipo parametro,...]) {}
```

La implementación del método irá contenida en un bloque { } justo después de la declaración.

Ejemplo:

```
public int suma(int param1, int param2) {  
    return param1 + param2;  
}
```

Constructores

Unidad 3: Java y POO

Constructores

Existe un tipo de método especial en Java llamado **constructor**.

Sirve para la construcción (instanciación) de objetos (instancias) a partir de esa clase.

En su implementación se suele dar valores a los atributos para ese objeto.

Su declaración es **idéntica** a la de los métodos convencionales con dos salvedades:

- No tienen tipo de retorno.
- Se tienen que llamar exactamente igual que la clase.

Unidad 3: Java y POO

Constructores

Declaración de un constructor:

```
modificador_acceso nombre([tipo parametro,...]) {}
```

Ejemplo:

```
public MiClase(int valor, String nombre) {  
    this.valor = valor;  
    this.nombre = nombre;  
}
```

Si nuestra clase no tiene constructores, el compilador **añade por defecto un constructor sin parámetros**

Convenciones en Java

Unidad 3: Java y POO

Convenciones en Java

El **nombre de las clases** comenzará con mayúsculas:

MiClase, String, Circulo, Cuenta, CuentaCorriente,.....

El **nombre de los atributos** comenzará con minúsculas:

contador, switch, i, segundoContador,.....

El **nombre de los métodos** comenzará con minúsculas (a excepción de los constructores):

ingresar, miMetodo, sumar,.....

Unidad 3: Java y POO

Convenciones en Java

```
public class Cuenta {  
    private String numero;  
    private double saldo;  
  
    public Cuenta(String numero, double saldo) {  
        this.numero = numero;  
        this.saldo = saldo;  
    }  
  
    public String getNumero() {  
        return numero;  
    }  
  
    public void setNumero(String numero) {  
        this.numero = numero;  
    }  
  
    public double getSaldo() {  
        return saldo;  
    }  
  
    public void setSaldo(double saldo) {  
        this.saldo = saldo;  
    }  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
  
    public void retirar(double valor) {  
        this.saldo -= valor;  
    }  
}
```

Objetos

Unidad 3: Java y POO

Objetos

Los objetos en Java no son más que **variables de tipo complejo**, frente a las de tipo primitivo.

El **tipo de un objeto** es la clase de la que se ha instanciado.

La declaración de un objeto es idéntica a la declaración de una variable de tipo primitivo:

tipo identificador;

- Ejemplo:

```
Cuenta miCuenta;
```

Unidad 3: Java y POO

Objetos

El **valor por defecto** de un objeto sin inicializar es: **null**

La **inicialización de un objeto** si que es algo distinta a la inicialización de las variables de tipo primitivo:

- Se utiliza el operador **new**.
- Se llama a un **constructor de la clase** de la que queremos instanciar.

Es decir:

```
tipo identificador = new tipo([parametro,...]);
```

```
Cuenta miCuenta = new Cuenta( numero: "ES88019282983839632" , saldo: 1535.25);
```

Unidad 3: Java y POO

Objetos

El trabajo con un objeto consiste en acceder:

- A sus atributos
- y a sus métodos

En ambos casos utilizaremos el operador . (punto).

Acceso a un atributo:

objeto.atributo

miCuenta.saldo = 2345.76;

Unidad 3: Java y POO

Objetos

Acceso a un método (lo que en Orientación a Objetos se denominaba **mensaje**):

 objeto.metodo([parametro,...])

```
  miCuenta.setSaldo(2345.76);
```

La posibilidad de acceso a un atributo o a un método de un objeto **dependerá del modificador de acceso** que exista en su definición.

Unidad 3: Java y POO

Objetos

```
public class Cuenta {  
    private String numero;  
    private double saldo;  
  
    public Cuenta() {}  
  
    public Cuenta(String numero, double saldo) {  
        this.numero = numero;  
        this.saldo = saldo;  
    }  
  
    public String getNumero() {  
        return numero;  
    }  
  
    public void setNumero(String numero) {  
        this.numero = numero;  
    }  
  
    public double getSaldo() {  
        return saldo;  
    }  
  
    public void setSaldo(double saldo) {  
        this.saldo = saldo;  
    }  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
  
    public void retirar(double valor) {  
        this.saldo -= valor;  
    }  
}
```

Unidad 3: Java y POO

Objetos

```
public class Principal {  
    public static void main(String[] args) {  
        Cuenta miCuenta1 = new Cuenta( numero: "ES88019282983839632", saldo: 1535.25);  
        Cuenta miCuenta2 = new Cuenta();  
        System.out.println("El saldo de la cuenta 1 es "+miCuenta1.getSaldo());  
        System.out.println("El saldo de la cuenta 2 es "+miCuenta2.getSaldo());  
        miCuenta1.retirar( valor: 342.87);  
        miCuenta2.depositar( valor: 2345.76);  
        System.out.println("El saldo de la cuenta 1 es "+miCuenta1.getSaldo());  
        System.out.println("El saldo de la cuenta 2 es "+miCuenta2.getSaldo());  
    }  
}
```

```
C:\Users\Public\Pictures\My Pictures  
El saldo de la cuenta 1 es 1535.25  
El saldo de la cuenta 2 es 0.0  
El saldo de la cuenta 1 es 1192.38  
El saldo de la cuenta 2 es 2345.76
```

```
Process finished with exit code 0
```

Unidad 3: Java y POO

Objetos

Existe un método especial en Java llamado **main**:

```
public static void main(String[] args)
```

Es el método donde **comienza la ejecución** de un programa Java.

Las clases representaban entidades que participaban en la resolución de un problema. ¿En qué entidad tiene sentido incluir el método main?

En ninguna. Por eso crearemos siempre una **clase aparte**, que solo tenga el método main.

Unidad 3: Java y POO

Objetos

Las llamadas a métodos se pueden **encadenar**:

```
String s1 = "abc";
char c = s1.toUpperCase().charAt(0);
System.out.println("El carácter es: "+c);
```

Equivaldría a:

```
String s1 = "abc";
String s2 = s1.toUpperCase();
char c = s2.charAt(0);
System.out.println("El carácter es: "+c);
```

Ejercicios

Unidad 3: Java y POO

Ejercicios

Identificar si hay algo mal en este código:

```
public class CDRom {  
    private boolean grabable = false;  
  
    public boolean isGrabable() {  
        return grabable;  
    }  
  
    public void setGrabable(boolean grabable) {  
        this.grabable = grabable;  
    }  
  
    public void escucharCD(){  
        System.out.println("Escuchando CD");  
    }  
  
    public void grabarCD(){  
        System.out.println("Grabando CD");  
    }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        miCD.escucharCD();  
        if (miCD.isGrabable()){  
            miCD.grabarCD();  
        }  
    }  
}
```

Unidad 3: Java y POO

Ejercicios

Estaba mal. No habíamos creado el objeto miCD :

```
public class CDRom {  
    private boolean grabable = false;  
  
    public boolean isGrabable() {  
        return grabable;  
    }  
  
    public void setGrabable(boolean grabable) {  
        this.grabable = grabable;  
    }  
  
    public void escucharCD(){  
        System.out.println("Escuchando CD");  
    }  
  
    public void grabarCD(){  
        System.out.println("Grabando CD");  
    }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        CDRom miCD = new CDRom();  
        miCD.escucharCD();  
        if (miCD.isGrabable()){  
            miCD.grabarCD();  
        }  
    }  
}
```

Unidad 3: Java y POO

Ejercicios

Identificar si hay algo mal en este código:

```
public class ReproductorDVD {  
    private boolean grabable=false;  
  
    public boolean isGrabable() {  
        return grabable;  
    }  
  
    public void setGrabable(boolean grabable) {  
        this.grabable = grabable;  
    }  
  
    public void grabarDVD(){  
        System.out.println("Grabando DVD");  
    }  
}  
  
public class Test2 {  
    public static void main(String[] args) {  
        ReproductorDVD dvd = new ReproductorDVD();  
        dvd.verDVD();  
        if (dvd.isGrabable())  
            dvd.grabarDVD();  
    }  
}
```

Unidad 3: Java y POO

Ejercicios

Estaba mal. Se estaba llamando a un método inexistente :

```
public class ReproductorDVD {  
    private boolean grabable=false;  
  
    public boolean isGrabable() {  
        return grabable;  
    }  
  
    public void setGrabable(boolean grabable) {  
        this.grabable = grabable;  
    }  
  
    public void grabarDVD(){  
        System.out.println("Grabando DVD");  
    }  
  
    public void verDVD(){  
        System.out.println("Viendo DVD");  
    }  
}  
  
public class Test2 {  
    public static void main(String[] args) {  
        ReproductorDVD dvd = new ReproductorDVD();  
        dvd.verDVD();  
        if (dvd.isGrabable())  
            dvd.grabarDVD();  
    }  
}
```

Unidad 3: Java y POO

Ejercicios

Identificar si hay algo mal en este código , suponiendo que **la clase Rectángulo existe**

```
public class Test3 {  
    public static void main(String[] args) {  
        Rectangulo miRect;  
        miRect.setAncho(40);  
        miRect.setAlto(50);  
        System.out.println("El área del rectángulo es "+miRect.getArea());  
    }  
}
```

Unidad 3: Java y POO

Ejercicios

Estaba mal. El objeto miRect **no está inicializado**, por tanto, vale null

```
public class Test3 {  
    public static void main(String[] args) {  
        Rectangulo miRect = new Rectangulo();  
        miRect.setAncho(40);  
        miRect.setAlto(50);  
        System.out.println("El área del rectángulo es "+miRect.getArea());  
    }  
}
```

Unidad 3: Java y POO

Ejercicios

Definir una clase que represente a un coche. Se debe incluir:

- modelo
- color
- pintura metalizada o no
- matrícula
- año de fabricación
- seguro a terceros o a todo riesgo

Unidad 3: Java y POO

Ejercicios

Solución

```
public class Coche {  
    private String modelo;  
    private String color;  
    private boolean esMetalizado;  
    private String matricula;  
    private int anhoFabricacion;  
    private boolean seguroTodoRiesgo;  
}
```

Unidad 3: Java y POO

Ejercicios

Ahora vamos a implementar los setters y getters de los atributos.

Unidad 3: Java y POO

Ejercicios

```
public String getModelo() {
    return modelo;
}

public void setModelo(String modelo) {
    this.modelo = modelo;
}

public String getColor() {
    return color;
}

public void setColor(String color) {
    this.color = color;
}

public boolean isEsMetalizado() {
    return esMetalizado;
}

public void setEsMetalizado(boolean esMetalizado) {
    this.esMetalizado = esMetalizado;
}

public String getMatricula() {
    return matricula;
}

public void setMatricula(String matricula) {
    this.matricula = matricula;
}

public int getAnhoFabricacion() {
    return anhoFabricacion;
}

public void setAnhoFabricacion(int anhoFabricacion) {
    this.anhoFabricacion = anhoFabricacion;
}

public boolean isSeguroTodoRiesgo() {
    return seguroTodoRiesgo;
}

public void setSeguroTodoRiesgo(boolean seguroTodoRiesgo) {
    this.seguroTodoRiesgo = seguroTodoRiesgo;
}
```

Unidad 3: Java y POO

Ejercicios

Ahora se desea imprimir el modelo y el color de un coche dado. Escribir un método que imprimirá el modelo y el color.

Unidad 3: Java y POO

Ejercicios

Ahora se desea imprimir el modelo y el color de un coche dado. Escribir un método que imprimirá el modelo y el color.

```
public void imprimeCoche() {  
    System.out.println("Modelo: " + modelo + " - Color: " + color);  
}
```

Unidad 3: Java y POO

Ejercicios

Escribir un programa que tenga una instancia de mi coche que es un Rolls Royce dorado. El programa debe imprimir un mensaje que diga de qué modelo y de qué color es mi coche.

Unidad 3: Java y POO

Ejercicios

```
public class Test4 {  
    public static void main(String[] args) {  
        Coche miCoche = new Coche();  
        miCoche.setModelo("Rolls Royce");  
        miCoche.setColor("dorado");  
        miCoche.imprimeCoche();  
    }  
}
```