



Orientación a Objetos I

Sesión Conceptual 01



- Objetivo de la sesión
- Activación de conceptos clave

- Conceptualización
- Ejercicios
- Quiz

- Cierre



Inicio



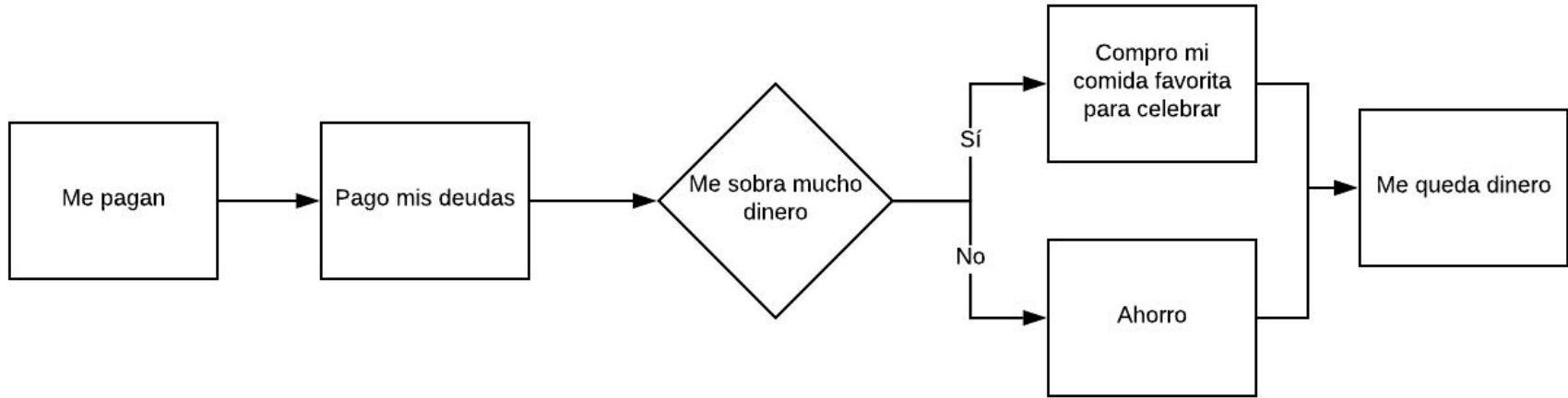
{desafío}
latam_

- Comprender el paradigma de la Programación Orientada a Objetos (POO).
- Comprender la estructura e instancia de una clase y sus atributos para aplicar la sobrecarga de métodos.

Objetivo

`/* Objetos, Clases y Sobrecarga */`

Una pincelada de historia



- Programación Lineal / Tipo espagueti
- Programación Estructurada
- Programación Orientada a Objetos

Objetos

- Los objetos pueden ser partes de un sistema o pueden ser un sistema
- Tienen atributos y comportamientos
- Tienen un estado



Las clases

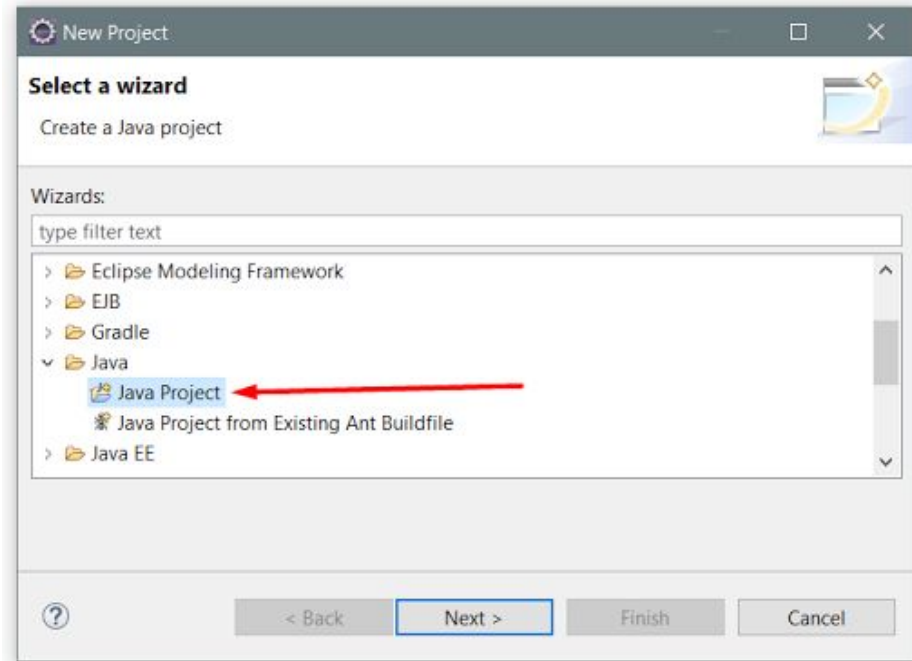
- Son algo así como plantillas para crear objetos de un software orientado a objetos.

Auto
String marca String modelo String color int velocidadActual boolean velocidadMaxima
void encenderMotor() void aumentarVelocidad(int velocidad) void frenar() void apagarMotor()

Creando clases

Creando el proyecto

1



{desafío}
latam_

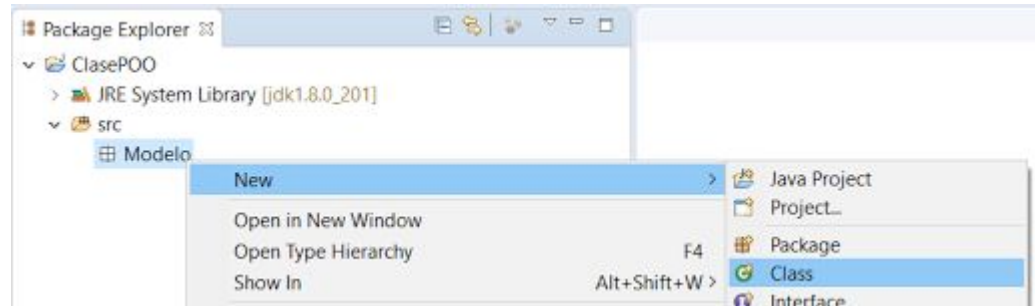
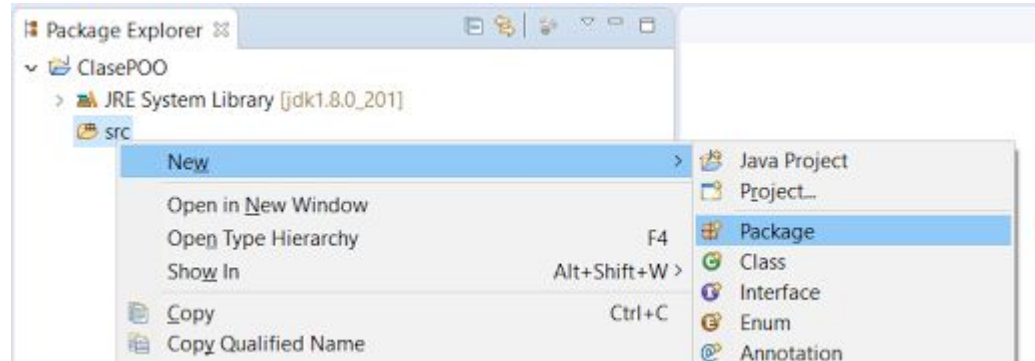
2



Project name:

Creando el paquete


- El paquete se debe llamar Modelo
New > Package
- Para crear la clase vamos a la opción:
New > Class



Creando la clase

New Java Class

Java Class

 This package name is discouraged. By convention, package names usually start with a lowercase letter

Source folder: ClasePOO/src Browse...

Package: Modelo Browse...

☐ Enclosing type: Browse...

Name: Auto

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...
Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

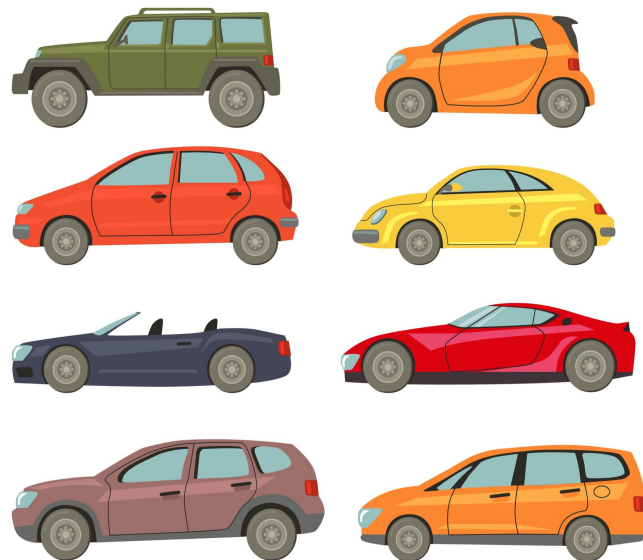
? Finish Cancel

Agregando los atributos

```
public class Auto{  
    String marca;  
    String modelo;  
    String color;  
    int velocidadActual;  
    boolean motorEncendido;  
}
```

Las instancias

- Son una forma de representar una clase, "dándole vida" a esta en forma de objeto



Manipulando instancias

Los métodos

- Porciones de código que realizan una o más operaciones dentro del programa.
- En java, deben estar escritos dentro de una clase, tener un nombre único dentro de esa clase y deben tener la siguiente forma:

```
public void nombreDelMetodo(){  
  
}
```


Creando métodos

```
public class Auto{  
    ...  
    public void encenderMotor(){  
        motorEncendido = true;  
    }  
}
```

Creando métodos

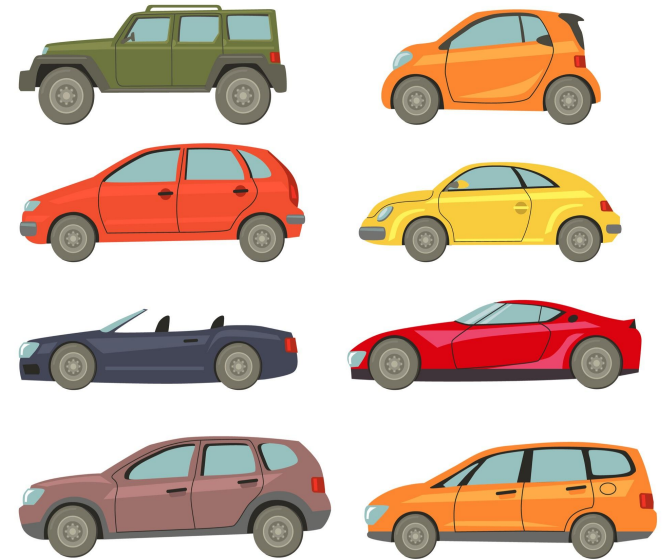
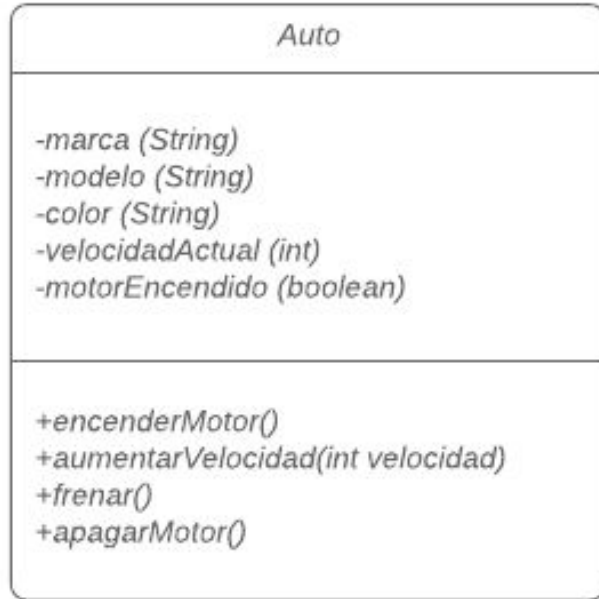
```
public class Auto{  
    ...  
    public void aumentarVelocidad(int velocidad){  
        velocidadActual = velocidadActual + velocidad;  
    }  
}
```

Creando métodos

```
public class Auto{  
    ...  
    public void frenar(){  
        while(velocidadActual > 0){  
            velocidadActual = velocidadActual-1;  
        };  
    }  
  
    public void apagarMotor(){  
        motorEncendido = false;  
        velocidadActual = 0;  
    }  
}
```

Los constructores

El método utilizado para instanciar clases



Los constructores estándar

Declaración

```
public Auto(){  
}
```

Llamada

```
Auto auto = new Auto();
```

Valores por defecto

Tipo de dato	Valor
int	0
boolean	false
String	null
Clases	null

Los constructores con parámetros

Declaración

```
public Auto(String marca, String modelo, int velocidadActual, boolean
motorEncendido){

    this.marca = marca;

    this.modelo = modelo;

    this.velocidadActual = velocidadActual

    this.motorEncendido = motorEncendido;

}
```

Llamada

```
Auto auto = new Auto("Kia", "Morning", 0, false);
```

Creando clase Main

New Java Class

Java Class

⚠ This package name is discouraged. By convention, package names usually start with a lowercase letter

Source folder: ClasePOO/src Browse...

Package: MainPackage Browse...

☐ Enclosing type: Browse...

Name: MainClass

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☒ public static void main(String[] args) ←

☐ Constructors from superclass

☒ Inherited abstract methods

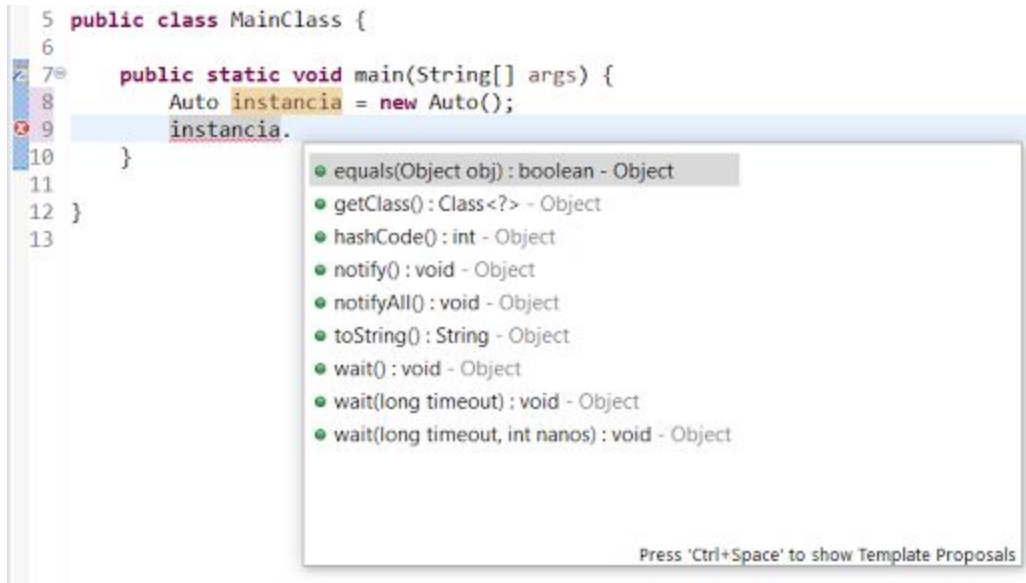
Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

? < Back Next > Finish Cancel


```
package MainPackage;  
import Modelo.Auto;  
  
public class MainClass {  
    public static void main(String[] args) {  
        Auto instancia = new Auto();  
        instancia.marca = "Chevrolet";  
    }  
}
```

Error de acceso



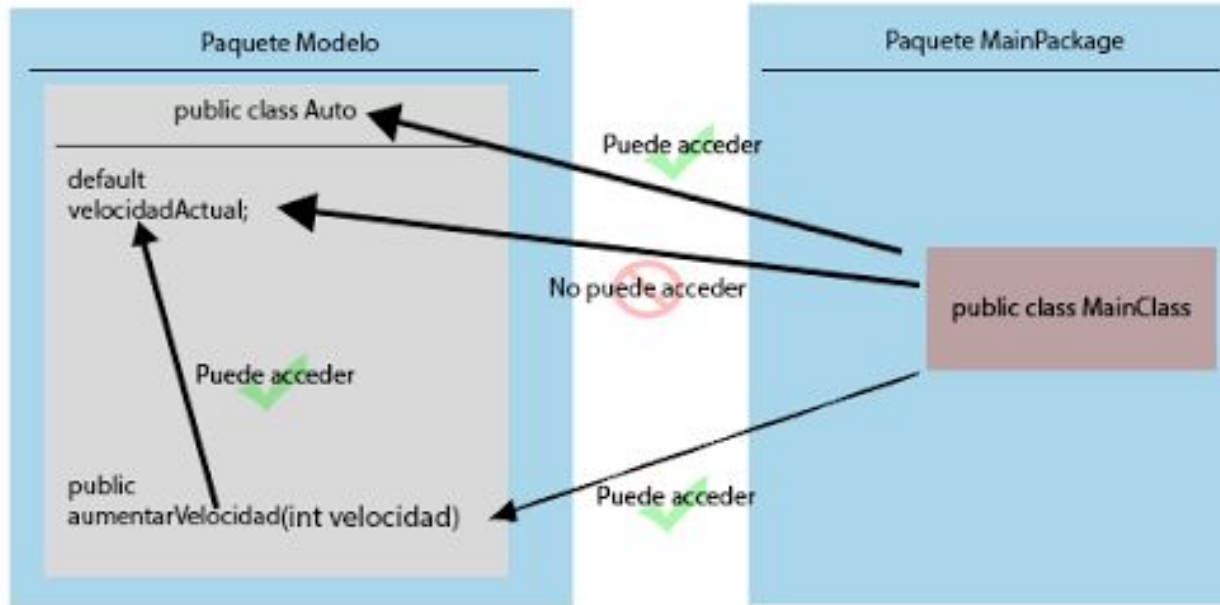
Encapsulamiento

Modificadores de acceso

- public
- private
- default

Desde	public	default	private
La misma clase	Si	Si	Si
Otra clase, mismo paquete	Si	Si	No
Clase de otro paquete	Si	No	No

Modificadores de acceso

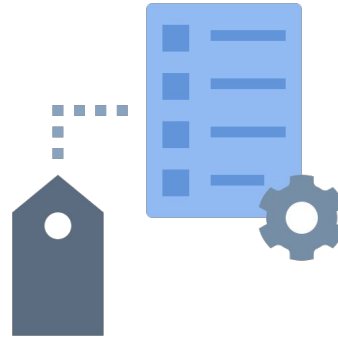


Atributos privados

```
public class Auto{  
    private String marca;  
    private String modelo;  
    private String color;  
    private int velocidadActual;  
    private boolean motorEncendido;  
}
```

El encapsulamiento

- Atributos trabajados a través de métodos



Ventajas del encapsulamiento

- Integridad de la información
- Reglas de negocio
- Estabilidad



Usando atributos sin encapsulamiento

```
public class Auto{  
    public String marca;  
    public String modelo;  
    public String color;  
    public int velocidadActual;  
    public boolean motorEncendido;  
}
```

Usando atributos sin encapsulamiento

```
Auto instancia = new Auto();  
System.out.println("Auto creado");  
instancia.motorEncendido = true;  
System.out.println("Motor Encendido: "+instancia.motorEncendido);  
instancia.velocidadActual = 50;  
System.out.println("Velocidad: "+instancia.velocidadActual);  
instancia.motorEncendido = false;  
System.out.println("Motor Encendido: "+instancia.motorEncendido);
```

Usando atributos sin encapsulamiento

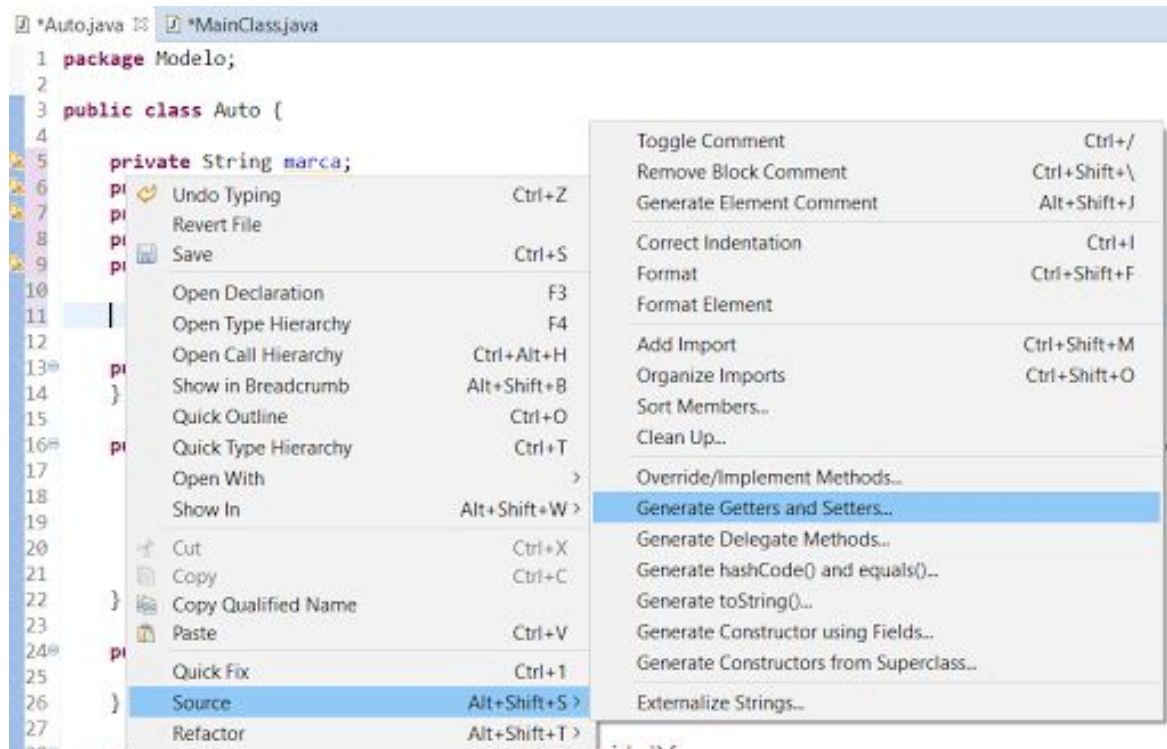
```
Auto instancia = new Auto();  
System.out.println("Auto creado");  
instancia.motorEncendido = true;  
System.out.println("Motor Encendido: "+instancia.motorEncendido);  
instancia.velocidadActual = 50;  
System.out.println("Velocidad: "+instancia.velocidadActual);  
if(instancia.velocidadActual == 0){  
    instancia.motorEncendido = false;  
}  
System.out.println("Motor Encendido: "+instancia.motorEncendido);
```

Aplicando el encapsulamiento

Creando Accesadores y Mutadores

```
private String marca;  
private String modelo;  
private String color;  
private int velocidadActual;  
private boolean motorEncendido;  
  
public void setMarca(String marca){  
    this.marca = marca;  
}  
  
public String getMarca(){  
    return this.marca;  
}
```

Forma rápida de crear el código en eclipse



Forma rápida de crear el código en eclipse



Corrigiendo el Main

```
//Creamos la instancia
Auto instancia = new Auto();
//Encendemos el motor y aceleramos hasta 100
instancia.setMotorEncendido(true);
instancia.setVelocidadActual(100);
//Vamos a imprimir el estado de nuestra instancia:
System.out.println("Auto: ");
System.out.println("Velocidad: "+instancia.getVelocidadActual());
System.out.println("Motor Encendido: "+instancia.getMotorEncendido());
```


Usando el método apagarMotor()

```
public void apagarMotor(){
    if(this.velocidadActual == 0){
        this.setMotorEncendido(false);
    }
}
```

```
public static void main(String[] args) {
    //Creamos la instancia
    Auto instancia = new Auto();
    //Encendemos el motor y aceleramos hasta 100
    instancia.setMotorEncendido(true);
    instancia.setVelocidadActual(100);
    //Intentamos apagar el motor
    instancia.apagarMotor();
    //Vamos a imprimir el estado de nuestra instancia:
    System.out.println("Auto: ");
    System.out.println("Velocidad: "+instancia.getVelocidadActual());
    System.out.println("Motor Encendido: "+instancia.getMotorEncendido());
}
```

Usando el método frenar()

```
public static void main(String[] args) {  
    //Creamos la instancia  
    Auto instancia = new Auto();  
    System.out.println("Auto creado");  
    //Encendemos el motor y aceleramos hasta 100  
    instancia.encenderMotor();  
    System.out.println("Motor Encendido: "+instancia.getMotorEncendido());  
    instancia.aumentarVelocidad(100);  
    System.out.println("Velocidad: "+instancia.getVelocidadActual());  
    //Frenamos hasta detenernos  
    instancia.frenar();  
    //Intentamos apagar el motor  
    instancia.apagarMotor();  
    //Vamos a imprimir el estado de nuestra instancia:  
    System.out.println("Auto: ");  
    System.out.println("Velocidad: "+instancia.getVelocidadActual());  
    System.out.println("Motor Encendido: "+instancia.getMotorEncendido());  
}
```



Quiz

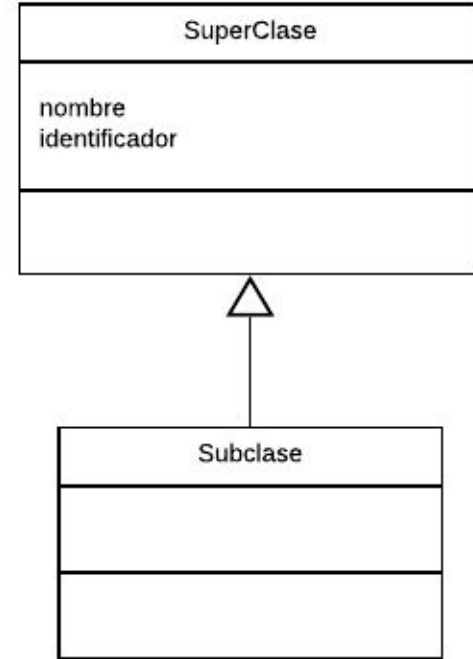
{desafío}
latam_

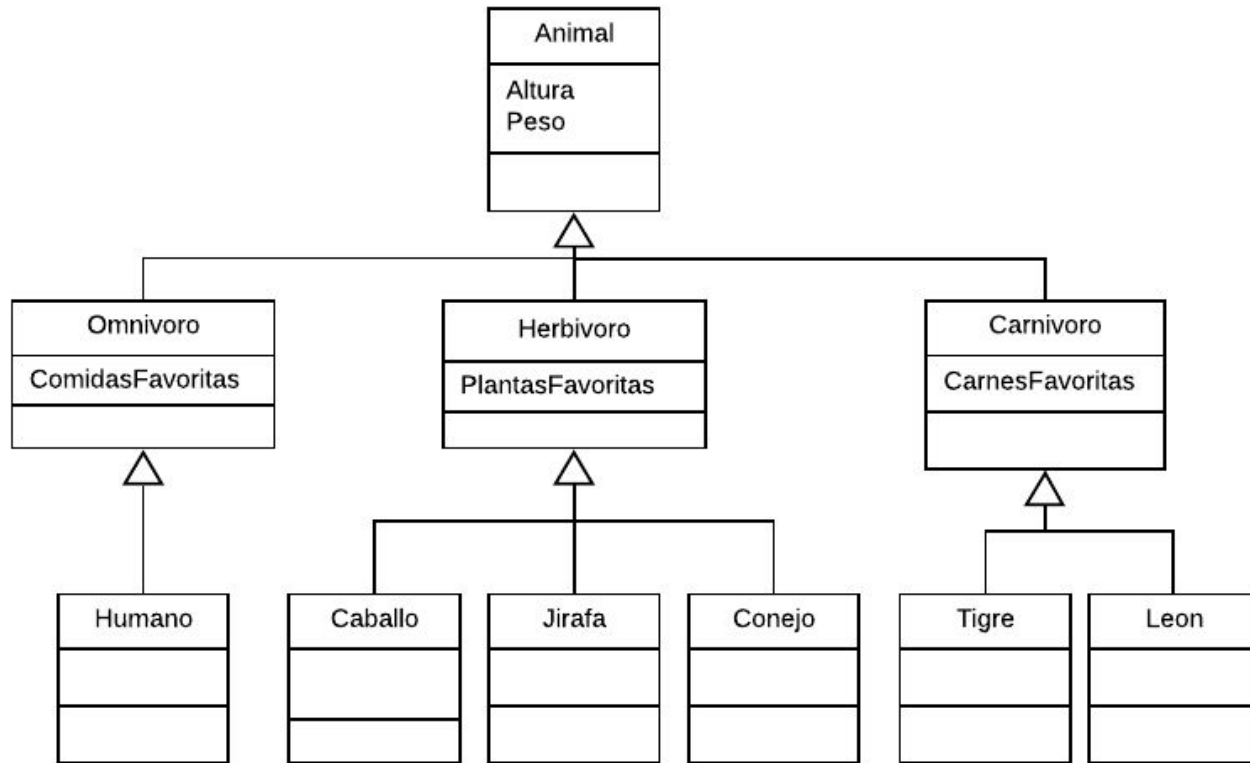


/* Herencia */

La herencia

- Heredar atributos y métodos desde una superclase hacia una o más subclases.
- Permite reutilizar atributos y métodos
- Sirve para categorizar las clases de un programa





La superclase Object

```
5 public class MainClass {  
6  
7     public static void main(String[] args) {  
8         Auto instancia = new Auto();  
9         instancia.  
10    }  
11  
12 }  
13
```

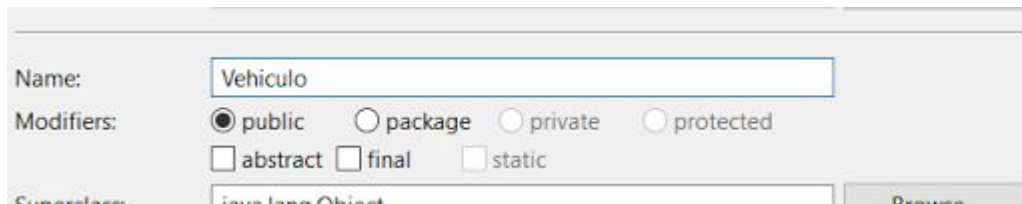
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

Press 'Ctrl+Space' to show Template Proposals

Aplicando el herencia

La superclase Vehiculo

Creamos la clase



Name: Vehiculo

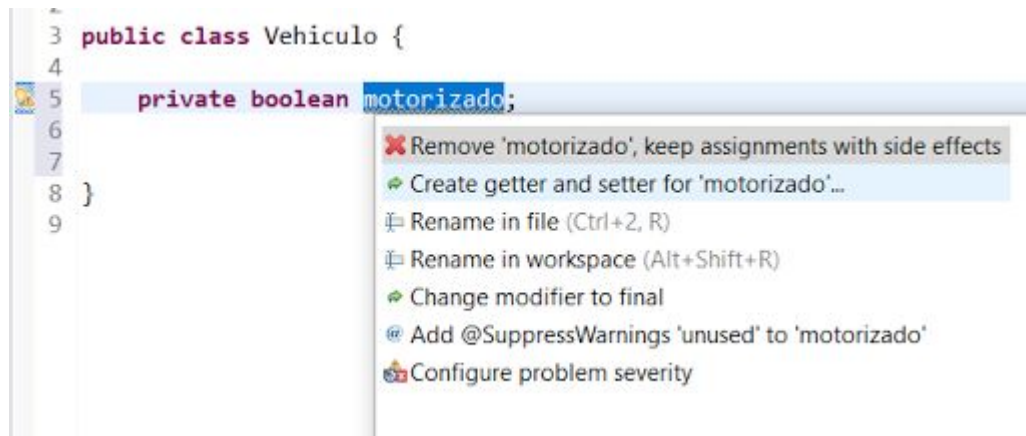
Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Creamos en la clase Vehículo, el atributo:

boolean motorizado

```
3 public class Vehiculo {  
4  
5     private boolean motorizado;  
6  
7  
8 }  
9
```



- Remove 'motorizado', keep assignments with side effects
- Create getter and setter for 'motorizado'...
- Rename in file (Ctrl+2, R)
- Rename in workspace (Alt+Shift+R)
- Change modifier to final
- Add @SuppressWarnings 'unused' to 'motorizado'
- Configure problem severity

Aplicando encapsulamiento

```
private boolean  motorizado;  
  
public boolean  isMotorizado() {  
    return motorizado;  
}  
  
public void setMotorizado(boolean  motorizado) {  
    this.motorizado = motorizado;  
}
```

Aplicando herencia

```
1 package Modelo;  
2  
3 public class Auto extends Vehiculo {  
4  
5     private String marca;
```

Probando la herencia

```
public static void main(String[] args) {  
    //Creamos la instancia  
    Auto instancia = new Auto();  
    instancia.setMotorizado(true);  
    System.out.println("Auto creado");  
    //Encendemos el motor y aceleramos hasta 100  
    instancia.encenderMotor();  
    System.out.println("Motor Encendido: "+instancia.getMotorEncendido());  
    instancia.aumentarVelocidad(100);  
    System.out.println("Velocidad: "+instancia.getVelocidadActual());  
    //Frenamos hasta detenernos  
    instancia.frenar();  
    //Intentamos apagar el motor  
    instancia.apagarMotor();  
    //Vamos a imprimir el estado de nuestra instancia:  
    System.out.println("Auto: ");  
    System.out.println("Es motorizado ? : "+instancia.isMotorizado());  
    System.out.println("Velocidad: "+instancia.getVelocidadActual());  
    System.out.println("Motor Encendido: "+instancia.getMotorEncendido());  
}
```

Modificador de acceso protected

- protected

Desde	protected
La misma clase	Si
Una subclase que hereda de esta	Si
Una subclase que no hereda de esta	No
Otra clase, mismo paquete	Si
Clase de otro paquete	No

Probando protected

55 public Auto(){
56 this.motorizado = true;
57 }
58
59 public Auto(){
60 this.motorizado = true;
61 this.motorEncendido = false;
62 this.motorEncendido = true;
63 }

The field Vehiculo.motorizado is not visible
3 quick fixes available:
• Change visibility of 'motorizado' to 'protected'
• Create field 'motorizado' in type 'Auto'
• Change to 'motorEncendido'

package Modelo;

public class Vehiculo {

protected boolean motorizado;

public boolean isMotorizado() {
return motorizado;
}

public void setMotorizado(boolean motorizado) {
this.motorizado = motorizado;
}

}

55 public Auto(){
56 this.motorizado = true;
57 }
58
59 public Auto(){
60 this.motorizado = true;
61 this.motorEncendido = false;
62 this.motorEncendido = true;
63 }

boolean
Modelo.Vehiculo.motorizado

{desafío}
latam_



Quiz

{desafío}
latam_





Cierre

{desafío}
latam_



**¿Existe algún concepto que no
hayas comprendido?**

**Volvamos a revisar los conceptos que más te
hayan costado antes de seguir adelante**

Reflexionemos



*Academia de
talentos digitales*

www.desafiolatam.com



/DesafioLatam



/DesafioLatam



/DesafioLatam



/DesafioLatam