

Práctica #06 – Polimorfismo.

Laboratorio de Programación Orientada a Objetos.

Alumno: Jeremy Uriel Rossell Segura.

Matrícula: 2173396.

Grupo: 036.

Horario: VIERNES 7:00-9:00.

Materia: Laboratorio de Programación Orientada a objetos.

Docente: Jorge Alberto Islas Pineda.

Compilación.

El programa compiló adecuadamente.

The image shows two side-by-side screenshots of the IntelliJ IDEA IDE's run console. Both consoles are titled '2173396_practica06' and show the output of a Java application named 'Main'. The top console shows the execution of the program, which prints out details for three vehicle types: Auto Híbrido, Auto Eléctrico, and Auto Combustión. The bottom console shows the decompiled code for the Main class, specifically the implementation of the 'tipo()' method for each vehicle type. The output in both consoles ends with 'Process finished with exit code 0'.

```
2173396_practica06 - kynZai
Sat, 22 Nov 10:31 24 °C
Current File Version control Run Main
...
Auto Híbrido
200.0
Cargando vehículo modo eléctrico...
Batería: 5000.0
Cargando vehículo modo gasolina...
Gas: 20.0
-----
Auto Eléctrico
200.0
Cargando vehículo modo eléctrico...
Batería: 7000.0
-----
Auto Combustión
200.0
Cargando vehículo con gasolina...
Combustible: 45.0
-----
Auto Volador
200.0
Cargando vehículo volador con diesel...
Tanque: 450.0
-----
Tipo de vehículo: AutoHíbrido
Métodos disponibles:
- public java.lang.String AutoHíbrido.tipo()

2173396_practica06 > src > Main
Sat, 22 Nov 10:31 23 °C
Current File Version control Run Main
...
Tipo de vehículo: AutoElectrico
Métodos disponibles:
- public java.lang.String AutoElectrico.tipo()
- public void AutoElectrico.cargar(double)
- public final double transporte.Vehiculo.setVelocidad(double)
- public double transporte.Vehiculo.getVelocidad()
- public void transporte.Vehiculo.acelerar()
- public void transporte.Vehiculo.acelerar(double)

-----
Tipo de vehículo: AutoCombustion
Métodos disponibles:
- public java.lang.String AutoCombustion.tipo()
- public void AutoCombustion.cargarGas(double)
- public final double transporte.Vehiculo.setVelocidad(double)
- public double transporte.Vehiculo.getVelocidad()
- public void transporte.Vehiculo.acelerar()
- public void transporte.Vehiculo.acelerar(double)

-----
Tipo de vehículo: AutoVolador
Métodos disponibles:
- public java.lang.String AutoVolador.tipo()
- public void AutoVolador.cargarDiesel(double)
- public final double transporte.Vehiculo.setVelocidad(double)
- public double transporte.Vehiculo.getVelocidad()
- public void transporte.Vehiculo.acelerar()
- public void transporte.Vehiculo.acelerar(double)

Process finished with exit code 0
```

Depuración.

El programa se depuró adecuadamente.

Sat, 22 Nov 10:31 23 °C

Current File Version control

Project Debug Main

Threads & Variables

Frames Threads

Debug stopped

Variables Console

Metodos disponibles:

- public java.lang.String AutoElectrico.tipo()
- public void AutoElectrico.cargar(double)
- public final double transporte.Vehiculo.setVelocidad(double)
- public double transporte.Vehiculo.getVelocidad()
- public void transporte.Vehiculo.acelerar()
- public void transporte.Vehiculo.acelerar(double)

Tipo de vehiculo: AutoCombustion

Metodos disponibles:

- public java.lang.String AutoCombustion.tipo()
- public void AutoCombustion.cargarGas(double)
- public final double transporte.Vehiculo.setVelocidad(double)
- public double transporte.Vehiculo.getVelocidad()
- public void transporte.Vehiculo.acelerar()
- public void transporte.Vehiculo.acelerar(double)

Tipo de vehiculo: AutoVolador

Metodos disponibles:

- public java.lang.String AutoVolador.tipo()
- public void AutoVolador.cargarDiesel(double)
- public final double transporte.Vehiculo.setVelocidad(double)
- public double transporte.Vehiculo.getVelocidad()
- public void transporte.Vehiculo.acelerar()
- public void transporte.Vehiculo.acelerar(double)

Disconnected from the target VM, address: '127.0.0.1:36917', transport: 'socket'

Process finished with exit code 0

Memory Overhead

Name	Hits	Time (ms)
Instrumentacion	0	0

2173396_practica06 > src > Main

62:2 LF UTF-8 4 spaces

This screenshot shows a Java IDE's debugger interface. The main window displays a list of methods for three vehicle classes: AutoElectrico, AutoCombustion, and AutoVolador. The 'Variables' tab is selected, showing the available methods for each class. The 'Memory' tab is also visible, showing a single entry for 'Instrumentacion' with 0 hits and 0 time. The status bar at the bottom indicates the file path '2173396_practica06 > src > Main', encoding 'UTF-8', and a code style preference of '4 spaces'.

Código.

El código del programa.

The screenshot shows a Java code editor with nine tabs open, displaying the source code for a program involving vehicle classes and interfaces.

- Main.java**:

```
6  public class Main {  
7      public static void main(String[] args) {  
8          Toyota.tipo();  
9          Toyota.setVelocidad(200.00);  
10         Toyota.getVelocidad();  
11         Toyota.acelerar();  
12         Toyota.cargarGas(litrosGas: 45);  
13         System.out.println("-----\n");  
14  
15         // Volador  
16         AutoVolador Boeing = new AutoVolador(marca: "boeing", modelo: "ct-70", velocidad: 200.00);  
17         Boeing.tipo();  
18         Boeing.setVelocidad(200.00);  
19         Boeing.getVelocidad();  
20         Boeing.acelerar();  
21         Boeing.cargarDiesel(litrosDiesel: 450);  
22         System.out.println("-----\n");  
23  
24         List<Vehiculo> flota = Arrays.asList(BYD, Tesla, Toyota, Boeing);  
25         for(Vehiculo v : flota) {  
26             ClaseVehiculo = v.getClass();  
27             System.out.println("\n" + ClaseVehiculo.getSimpleName());  
28             Method[] metodos = ClaseVehiculo.getMethods();  
29             System.out.println("Metodos disponibles: ");  
30  
31             for(Method metodo : metodos) {  
32                 if (!metodo.getDeclaringClass().equals(Object.class)) {  
33                     System.out.println("- " + metodo.toGenericString());  
34                 }  
35             }  
36         }  
37     }  
38 }
```
- Vehiculo.java**:

```
1  package transporte;  
2  
3  public abstract class Vehiculo {  
4      private double velocidad; 3 usages  
5      protected String marca; 1 usage  
6      protected String modelo; 1 usage  
7  
8      public Vehiculo() { no usages  
9  }  
10  
11     public Vehiculo(String marca, String modelo) { no usages  
12         this.marca = marca;  
13         this.modelo = modelo;  
14         setVelocidad(velocidad);  
15     }  
16  
17     public double getVelocidad() { 6 usages  
18         return this.velocidad;  
19     }  
20  
21     public final double setVelocidad(double v) { 7 usages  
22         if (v >= 0) {  
23             this.velocidad = v;  
24         }  
25         return v;  
26     }  
27  
28     public void acelerar() { 4 usages  
29         setVelocidad(getVelocidad() + 10);  
30     }  
31 }
```
- AutoCombustion.java**:

```
1  import tipoDeMotor.ICombustion;  
2  import transporte.Vehiculo;  
3  
4  public class AutoCombustion extends Vehiculo implements ICombustion { 2 usages  
5  
6      public AutoCombustion(String marca, String modelo, double velocidad) { super(marca, modelo, velocidad);  
7  }  
8  
9      @Override 2 usages  
10     public void cargarGas(double litrosGas) {  
11         System.out.println("Cargando vehículo con gasolina...");  
12         System.out.println("Combustible: " + litrosGas);  
13     }  
14  
15     @Override 4 usages  
16     public String tipo() { return "Auto Combustión"; }  
17 }
```
- AutoVolador.java**:

```
1  import tipoDeMotor.IVolador;  
2  import transporte.Vehiculo;  
3  
4  public class AutoVolador extends Vehiculo implements IVolador {  
5  
6      public AutoVolador(String marca, String modelo, double velocidad) { super(marca, modelo, velocidad);  
7  }  
8  
9      @Override  
10     public void cargarDiesel(double litrosDiesel) {  
11         System.out.println("Cargando vehículo volador con diesel...");  
12         System.out.println("Tanque: " + litrosDiesel);  
13     }  
14  
15     @Override  
16     public String tipo() { return "Auto Volador"; }  
17 }
```
- AutoElectrico.java**:

```
1  import tipoDeMotor.IElectrico;  
2  import transporte.Vehiculo;  
3  
4  public class AutoElectrico extends Vehiculo implements IElectrico { 2 usages  
5  
6      public AutoElectrico(String marca, String modelo, double velocidad) { super(marca, modelo, velocidad);  
7  }  
8  
9      @Override 2 usages  
10     public void cargar(double KWh) {  
11         System.out.println("Cargando vehículo modo eléctrico...");  
12         System.out.println("Batería: " + KWh);  
13     }  
14  
15     @Override 4 usages  
16     public String tipo() { return "Auto Eléctrico"; }  
17 }
```
- AutoHibrido.java**:

```
1  import tipoDeMotor.ICombustion;  
2  import tipoDeMotor.IElectrico;  
3  import transporte.Vehiculo;  
4  
5  public class AutoHibrido extends Vehiculo implements IElectrico, ICombustion { 2 usages  
6  
7      public AutoHibrido(String marca, String modelo, double velocidad) { super(marca, modelo, velocidad);  
8  }  
9  
10     @Override 2 usages  
11     public void cargar(double KWh) {  
12         System.out.println("Cargando vehículo modo eléctrico...");  
13         System.out.println("Batería: " + KWh);  
14     }  
15  
16     @Override 2 usages  
17     public void cargarGas(double litrosGas) {  
18         System.out.println("Cargando vehículo modo gasolina...");  
19     }  
20 }
```
- IVolador.java**:

```
1  package tipoDeMotor;  
2  
3  public interface IVolador { 2 usages 1 implementation  
4      void cargarDiesel(double litrosDiesel); 1 usage  
5  }
```
- IElectrico.java**:

```
1  package tipoDeMotor;  
2  
3  public interface IElectrico { 4 usages 2 implementations  
4      void cargar(double KWh); 2 usages 2 implementations  
5  }
```
- ICombustion.java**:

```
1  package tipoDeMotor;  
2  
3  public interface ICombustion { 4 usages 2 implementations  
4      void cargarGas(double litrosGas); 2 usages 2 implementations  
5  }
```

Archivos.

Los archivos del programa.

