Contenido

[**Sección 14: Programación orientada a objetos: Clases y objetos** 4](#_Toc138842894)

[129. Introducción a las clases y objetos 4](#_Toc138842895)

[130. Creando nuestra primera clase 4](#_Toc138842896)

[131. Creando objetos con el operador new 5](#_Toc138842897)

[132. Agregando métodos 6](#_Toc138842898)

[133. Métodos con argumentos 7](#_Toc138842899)

[134. Métodos con argumentos, sobrecarga y encapsulamiento 8](#_Toc138842900)

[135. Principio de ocultación de los atributos 9](#_Toc138842901)

[136. Ocultación métodos get y set 11](#_Toc138842902)

[137. Implementando constructor 11](#_Toc138842903)

[138. Sobrecarga de constructores 12](#_Toc138842904)

[139. El método equals 14](#_Toc138842905)

[140. El método equals segunda parte 14](#_Toc138842906)

[141. El método toString 14](#_Toc138842907)

[Practica 1 14](#_Toc138842908)

[Practica 2 14](#_Toc138842909)

[142. Atributos y métodos estáticos de la clase 14](#_Toc138842910)

[143. Atributos y métodos estáticos 2° parte 14](#_Toc138842911)

[144. Atributos constantes final 15](#_Toc138842912)

[145. Atributos constantes final 2° parte (colores) 15](#_Toc138842913)

[146. Enumerador Color (enum) 15](#_Toc138842914)

[147. Implemetando enumerador con más atributos 17](#_Toc138842915)

[148. Utilizando enum con sentencias switch case 21](#_Toc138842916)

[149. Iterando enum con foreach 23](#_Toc138842917)

[Practica 3 24](#_Toc138842918)

[150. Diagrama de clases UML y relaciones de clases/objetos 24](#_Toc138842919)

[151. Creando las clases relacionadas 24](#_Toc138842920)

[152. Modificando clase con atributos de relaciones de objetos 24](#_Toc138842921)

[153. Soluciones a posibles errores NullPointerException 24](#_Toc138842922)

[154. Relacionando los demás objetos 24](#_Toc138842923)

[155. Añadiendo el método add para agregar elementos de un arreglo 24](#_Toc138842924)

[156. El método compareTo para ordenar objetos 24](#_Toc138842925)

[157. Descargar Código Fuente Recursos 24](#_Toc138842926)

[Tarea 20: Proyecto Orden de Compra 24](#_Toc138842927)

[**Sección 15: Programación orientada a objetos: Paquetes** 25](#_Toc138842928)

[158. Introducción a los paquetes 25](#_Toc138842929)

[159. Creando nuestros primeros package y sus clases 25](#_Toc138842930)

[160. Control de acceso mendiante modificadores 25](#_Toc138842931)

[161. Alcance de los modificadores de accesos 25](#_Toc138842932)

[162. Import estático 25](#_Toc138842933)

[163. Descargar Código Fuente 25](#_Toc138842934)

[**Sección 16: Programación orientada a objetos: Ejemplo Facturas** 25](#_Toc138842935)

[**Sección 17: Programación orientada a objetos: Sobrecarga de métodos** 25](#_Toc138842936)

[**Sección 18: Programación orientada a objetos: Herencia** 26](#_Toc138842937)

[178. Introducción a la herencia 26](#_Toc138842938)

[179. Creando la jerarquía de clases 26](#_Toc138842939)

[180. Sobre el modificador de acceso protected 26](#_Toc138842940)

[181. Herencia y el API de reflexión de java 26](#_Toc138842941)

[182. Llamando a los constructores en cadena 26](#_Toc138842942)

[183. La referencia super en constructor 26](#_Toc138842943)

[184. Creando instancias vía constructores con argumentos y control de tipos 26](#_Toc138842944)

[185. Sobre escritura de métodos en la herencia 26](#_Toc138842945)

[186. Sobre escritura método toString 26](#_Toc138842946)

[187. Debug mode 26](#_Toc138842947)

[188. Modificador final para evitar la herencia y la sobre-escritura de métodos 26](#_Toc138842948)

[189. Resumen con diagrama UML de Clases 26](#_Toc138842949)

[190. Descargar Código Fuente 26](#_Toc138842950)

[Tarea 21: Personas de una compañía 26](#_Toc138842951)

[Tarea 22: Almacén y verdulería 26](#_Toc138842952)

[Tarea 23: Un almacén de venta por mayor 26](#_Toc138842953)

[**Sección 19: Programación orientada a objetos: Herencia y Clases Abstractas** 27](#_Toc138842954)

[191. Introducción a las clases abstractas 27](#_Toc138842955)

[192. Diseño de clases UML 27](#_Toc138842956)

[193. Creando clase padre abstracta ElementoForm 27](#_Toc138842957)

[194. Creando clases hijas concretas InputForm y TextareaForm 28](#_Toc138842958)

[195. Creando la clase concreta SelectForm 28](#_Toc138842959)

[196. Creando la clase ejemplo con el método main EjemploForm 28](#_Toc138842960)

[197. Dibujando elementos del formulario 28](#_Toc138842961)

[198. Clases Anónimas 28](#_Toc138842962)

[199. Creando la clase abstracta Validador 28](#_Toc138842963)

[con tres clases hijas concretas 28](#_Toc138842964)

[200. Creando la clase concreta LargoValidador para validar el min y máx de caracteres 28](#_Toc138842965)

[201. Creando la clase concreta EmailValidador 28](#_Toc138842966)

[202. Implementando la validación en la clase abstracta ElementoForm 28](#_Toc138842967)

[203. Validando los elementos del formulario 28](#_Toc138842968)

[204. Añadiendo el nombre del campo en los mensajes de errores 28](#_Toc138842969)

[205. Implementando interface para formatear mensajes de errores 28](#_Toc138842970)

[206. Descargar Código Fuente 28](#_Toc138842971)

[Tarea 24: Proyecto Mamíferos 28](#_Toc138842972)

[**Sección 20: Programación orientada a objetos: Interfaces** 29](#_Toc138842973)

[**Sección 21: Programación orientada a objetos: Ejemplo Interface Repositorio** 29](#_Toc138842974)

# **Sección 14: Programación orientada a objetos: Clases y objetos**

# 129. Introducción a las clases y objetos

En POO, una clase es una plantilla o molde para crear objetos. Define las propiedades y comportamientos que los objetos de esa clase tendrán. Las propiedades se conocen como atributos, mientras que los comportamientos se llaman métodos.

Un objeto es una instancia de una clase. Significa que es una entidad real creada a partir de la clase, con sus propias características y capacidades.

# 130. Creando nuestra primera clase

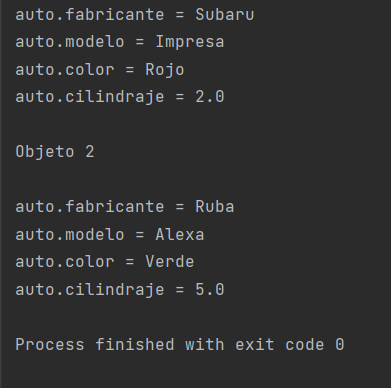
package \_130\_CreaClase;  
  
public class Automovil {  
  
 String fabricante;  
 String modelo;  
 String color;  
 double cilindraje;  
  
}

package \_130\_CreaClase;  
  
public class EjemploAutomovil {  
 public static void main(String[] args) {  
  
  
  
 }  
}

# 131. Creando objetos con el operador new

package \_130\_CreaClase;  
  
public class Automovil {  
  
 String fabricante;  
 String modelo;  
 String color;  
 double cilindraje;  
  
}

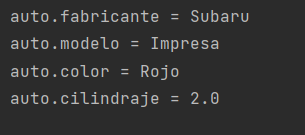
package \_130\_CreaClase;  
  
public class EjemploAutomovil {  
 public static void main(String[] args) {  
  
 Automovil auto = new Automovil(); //instancia de la clase  
  
 auto.fabricante = "Subaru";  
 auto.modelo = "Impresa";  
 auto.color = "Rojo";  
 auto.cilindraje = 2;  
  
 //imprime  
 System.*out*.println("auto.fabricante = " + auto.fabricante);  
 System.*out*.println("auto.modelo = " + auto.modelo);  
 System.*out*.println("auto.color = " + auto.color);  
 System.*out*.println("auto.cilindraje = " + auto.cilindraje);  
  
 //otro objeto  
  
 System.*out*.println("\nObjeto 2\n");  
  
 Automovil auto2 = new Automovil(); //instancia de la clase  
  
 auto2.fabricante = "Ruba";  
 auto2.modelo = "Alexa";  
 auto2.color = "Verde";  
 auto2.cilindraje = 5;  
  
 //imprime  
 System.*out*.println("auto.fabricante = " + auto2.fabricante);  
 System.*out*.println("auto.modelo = " + auto2.modelo);  
 System.*out*.println("auto.color = " + auto2.color);  
 System.*out*.println("auto.cilindraje = " + auto2.cilindraje);  
  
  
 }  
}



# 132. Agregando métodos

package \_130\_CreaClase;  
  
public class Automovil {  
  
 String fabricante;  
 String modelo;  
 String color;  
 double cilindraje;  
  
 //metodos  
 public String detalle(){  
 //podria usar sout e imprimir pero no es buena practica  
 StringBuilder sb = new StringBuilder();  
 sb.append("\nauto.fabricante = " + this.fabricante);  
 sb.append("\nauto.modelo = " + this.modelo);  
 sb.append("\nauto.color = " + this.color);  
 sb.append("\nauto.cilindraje = " + this.cilindraje);  
 return sb.toString();  
 }  
  
}

package \_130\_CreaClase;  
  
public class EjemploAutomovil {  
 public static void main(String[] args) {  
  
 Automovil auto = new Automovil(); //instancia de la clase  
  
 auto.fabricante = "Subaru";  
 auto.modelo = "Impresa";  
 auto.color = "Rojo";  
 auto.cilindraje = 2;  
  
 System.*out*.println(auto.detalle());//imprime  
  
 }  
}

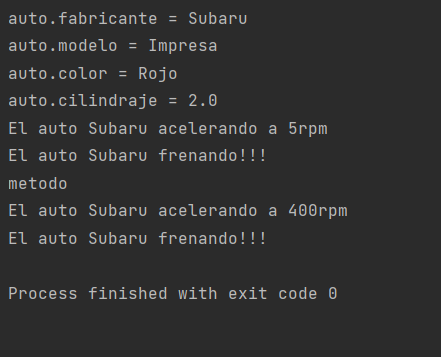


# 133. Métodos con argumentos

package \_130\_CreaClase;  
  
public class Automovil {  
  
 String fabricante;  
 String modelo;  
 String color;  
 double cilindraje;  
  
 //metodos  
 public String detalle(){  
 //podria usar sout e imprimir pero no es buena practica  
 StringBuilder sb = new StringBuilder();  
 sb.append("\nauto.fabricante = " + this.fabricante);  
 sb.append("\nauto.modelo = " + this.modelo);  
 sb.append("\nauto.color = " + this.color);  
 sb.append("\nauto.cilindraje = " + this.cilindraje);  
 return sb.toString();  
 }  
  
 public String acelerar(int rpm){  
 return "El auto " + fabricante + " acelerando a " +rpm +"rpm";  
 }  
  
 public String frenar(){  
 return "El auto " + fabricante + " frenando!!!";  
 }  
  
 public String acelerarFrenar(int rpm){  
 String acelerar = this.acelerar(rpm);  
 String frenar = this.frenar();  
  
 return acelerar + "\n" +frenar;  
 }

}

package \_130\_CreaClase;  
  
public class EjemploAutomovil {  
 public static void main(String[] args) {  
  
 Automovil auto = new Automovil(); //instancia de la clase  
  
 auto.fabricante = "Subaru";  
 auto.modelo = "Impresa";  
 auto.color = "Rojo";  
 auto.cilindraje = 2;  
  
 System.*out*.println(auto.detalle());//imprime  
  
 System.*out*.println(auto.acelerar(5));  
 System.*out*.println(auto.frenar());  
 System.*out*.println("metodo \n"+auto.acelerarFrenar(400));  
  
 }  
  
}



# 134. Métodos con argumentos, sobrecarga y encapsulamiento

package \_130\_CreaClase;  
  
public class Automovil {  
  
 String fabricante;  
 String modelo;  
 String color;  
 double cilindraje;  
 int capacidadEstanque = 40;  
  
 //metodos  
 public String detalle(){  
 StringBuilder sb = new StringBuilder();  
 sb.append("\nauto.fabricante = " + this.fabricante);  
 sb.append("\nauto.modelo = " + this.modelo);  
 sb.append("\nauto.color = " + this.color);  
 sb.append("\nauto.cilindraje = " + this.cilindraje);  
 return sb.toString();  
 }  
   
 //metodos  
 public float calcularConsumo(int km, float porcentajeBencina){  
 return km/(capacidadEstanque\*porcentajeBencina);  
 }  
  
 public float calcularConsumo(int km, int porcentajeBencina){  
 return km/(capacidadEstanque\*(porcentajeBencina/100f));  
 }  
  
}

package \_130\_CreaClase;  
  
public class EjemploAutomovil {  
 public static void main(String[] args) {  
  
 Automovil auto = new Automovil(); //instancia de la clase  
  
 auto.fabricante = "Subaru";  
 auto.modelo = "Impresa";  
 auto.color = "Rojo";  
 auto.cilindraje = 2;  
  
 System.*out*.println(auto.detalle());//imprime  
  
 System.*out*.println(auto.calcularConsumo(300,0.75f));  
 System.*out*.println(auto.calcularConsumo(300,60));  
  
 }  
  
}

# 135. Principio de ocultación de los atributos

package \_130\_CreaClase;  
  
public class Automovil {  
  
 private String fabricante;  
 private String modelo;  
 private String color;  
 private double cilindraje;  
 private int capacidadEstanque = 40;  
  
 //metodos get and set  
  
 public String getFabricante() {  
 return fabricante;  
 }  
  
 public void setFabricante(String fabricante) {  
 this.fabricante = fabricante;  
 }  
  
 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 double getCilindraje() {  
 return cilindraje;  
 }  
  
 public void setCilindraje(double cilindraje) {  
 this.cilindraje = cilindraje;  
 }  
  
 public int getCapacidadEstanque() {  
 return capacidadEstanque;  
 }  
  
 public void setCapacidadEstanque(int capacidadEstanque) {  
 this.capacidadEstanque = capacidadEstanque;  
 }  
}

package \_130\_CreaClase;  
  
public class EjemploAutomovil {  
 public static void main(String[] args) {  
  
 Automovil auto = new Automovil(); //instancia de la clase  
  
 //asiga con set- asigna valor  
 auto.setFabricante("Subaru");  
 auto.setModelo("Impresa");  
 auto.setColor("Rojo");  
 auto.setCilindraje(2);  
  
 //imprime get- obtener valor  
 System.*out*.println("fabricante = " + auto.getFabricante());  
 System.*out*.println("modelo = " + auto.getModelo());  
 System.*out*.println("color = " + auto.getColor());  
 System.*out*.println("cilindraje = " + auto.getCilindraje());  
  
  
 }  
  
}

# 136. Ocultación métodos get y set

Es lo mismo de arriba solo que en el video le puso cualquier nombre y no get y set que yo le puse directo xd.

# 137. Implementando constructor

package \_130\_CreaClase;  
  
public class Automovil {  
  
 private String fabricante;  
 private String modelo;  
 private String color;  
 private double cilindraje;  
 private int capacidadEstanque = 40;  
  
 //constrcutor  
 public Automovil(String fabricante, String modelo, String color, double cilindraje, int capacidadEstanque) {  
 this.fabricante = fabricante;  
 this.modelo = modelo;  
 this.color = color;  
 this.cilindraje = cilindraje;  
 this.capacidadEstanque = capacidadEstanque;  
 }  
  
  
 //metodos get and set  
  
 public String getFabricante() {  
 return fabricante;  
 }  
  
 public void setFabricante(String fabricante) {  
 this.fabricante = fabricante;  
 }  
  
 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 double getCilindraje() {  
 return cilindraje;  
 }  
  
 public void setCilindraje(double cilindraje) {  
 this.cilindraje = cilindraje;  
 }  
  
 public int getCapacidadEstanque() {  
 return capacidadEstanque;  
 }  
  
 public void setCapacidadEstanque(int capacidadEstanque) {  
 this.capacidadEstanque = capacidadEstanque;  
 }  
}

package \_130\_CreaClase;  
  
public class EjemploAutomovil {  
 public static void main(String[] args) {  
  
 Automovil auto = new Automovil("Subaru","Impreza","Rojo",2,40);  
  
 //imprime get- obtener valor  
 System.*out*.println("fabricante = " + auto.getFabricante());  
 System.*out*.println("modelo = " + auto.getModelo());  
 System.*out*.println("color = " + auto.getColor());  
 System.*out*.println("cilindraje = " + auto.getCilindraje());  
  
 }  
}

# 138. Sobrecarga de constructores

package \_130\_CreaClase;  
  
public class Automovil {  
  
 private String fabricante;  
 private String modelo;  
 private String color;  
 private double cilindraje;  
 private int capacidadEstanque = 40;  
  
 //constrcutor  
 public Automovil(String fabricante, String modelo, String color, double cilindraje, int capacidadEstanque) {  
 this(fabricante, modelo);  
 this.color = color;  
 this.cilindraje = cilindraje;  
 this.capacidadEstanque = capacidadEstanque;  
 }  
  
 //sobre carga de constructores  
 public Automovil(String fabricante, String modelo) {  
 this.fabricante = fabricante;  
 this.modelo = modelo;  
 }  
  
  
 //metodos get and set  
  
 public String getFabricante() {  
 return fabricante;  
 }  
  
 public void setFabricante(String fabricante) {  
 this.fabricante = fabricante;  
 }  
  
 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 double getCilindraje() {  
 return cilindraje;  
 }  
  
 public void setCilindraje(double cilindraje) {  
 this.cilindraje = cilindraje;  
 }  
  
 public int getCapacidadEstanque() {  
 return capacidadEstanque;  
 }  
  
 public void setCapacidadEstanque(int capacidadEstanque) {  
 this.capacidadEstanque = capacidadEstanque;  
 }  
}

package \_130\_CreaClase;  
  
public class EjemploAutomovil {  
 public static void main(String[] args) {  
  
 Automovil auto = new Automovil("Subaru","Impreza","Rojo",2,40);  
  
 //imprime get- obtener valor  
 System.*out*.println("fabricante = " + auto.getFabricante());  
 System.*out*.println("modelo = " + auto.getModelo());  
 System.*out*.println("color = " + auto.getColor());  
 System.*out*.println("cilindraje = " + auto.getCilindraje());  
  
 System.*out*.println("\nSobrecarga de constructores \n");  
  
 Automovil auto2 = new Automovil("Chevrolet","aaaa");  
 System.*out*.println("fabricante = " + auto2.getFabricante());  
 System.*out*.println("modelo = " + auto2.getModelo());  
  
  
 }  
}

# 139. El método equals

Compara objetos y son diferentes asi tengan los mismos datos, porque cada objeto que se crea en memoria se guarda en memoria distinta indiferete si tienen los mismos datos

# 140. El método equals segunda parte

# 141. El método toString

package \_130\_CreaClase;  
  
public class Automovil {  
  
 private String fabricante;  
 private String modelo;  
 private String color;  
 private double cilindraje;  
 private int capacidadEstanque = 40;  
  
  
 //METODOS  
  
 //TOSTRING  
  
 @Override  
 public String toString() {  
 return "Automovil{" +  
 "fabricante='" + fabricante + '\'' +  
 ", modelo='" + modelo + '\'' +  
 ", color='" + color + '\'' +  
 ", cilindraje=" + cilindraje +  
 ", capacidadEstanque=" + capacidadEstanque +  
 '}';  
 }  
  
}

# Practica 1

# Practica 2

# 142. Atributos y métodos estáticos de la clase

Los atributos y métodos estáticos en una clase pertenecen a la clase en sí misma en lugar de pertenecer a las instancias individuales de la clase. Esto significa que se comparten entre todas las instancias de la clase y se pueden acceder sin necesidad de crear un objeto.

OK

# 143. Atributos y métodos estáticos 2° parte

OK

# 144. Atributos constantes final

OK

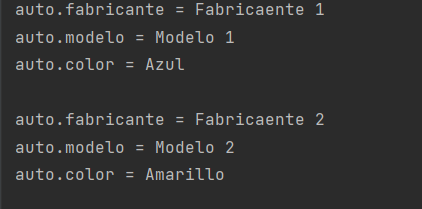
# 145. Atributos constantes final 2° parte (colores)

# 146. Enumerador Color (enum)

package \_146\_Enum;  
  
//AL CREAR DONDE DICES CLASS... ELEGIR ENUM  
public enum Color {  
  
 *ROJO*("Rojo"),  
 *AMARILLO*("Amarillo"),  
 *AZUL*("Azul"),  
 *NARANJO*("Naranjo");  
  
 private final String color;  
  
 Color(String color) {  
 this.color = color;  
 }  
  
 public String getColor() {  
 return color;  
 }  
  
}//ENUM

package \_146\_Enum;  
  
public class Automovil {  
  
 private String fabricante;  
 private String modelo;  
 private Color color; //Color es de la clase Enum  
  
  
 public Automovil(){  
  
 }  
  
 public Automovil(String fabricante, String modelo, Color color) {  
 this.fabricante = fabricante;  
 this.modelo = modelo;  
 this.color = color;  
 }  
  
  
 public String getFabricante() {  
 return fabricante;  
 }  
  
 public void setFabricante(String fabricante) {  
 this.fabricante = fabricante;  
 }  
  
 public String getModelo() {  
 return modelo;  
 }  
  
 public void setModelo(String modelo) {  
 this.modelo = modelo;  
 }  
  
 public Color getColor() {  
 return color;  
 }  
  
 public void setColor(Color color) {  
 this.color = color;  
 }  
  
  
 public String detalle() {  
 StringBuilder sb = new StringBuilder();  
 sb.append("\nauto.fabricante = " + this.fabricante);  
 sb.append("\nauto.modelo = " + this.modelo);  
 sb.append("\nauto.color = " + this.color.getColor());//como estaba asiciona el getColor paar hacer uso del contenido de minusculas  
 return sb.toString();  
 }  
  
}//CLASS

package \_146\_Enum;  
  
  
public class Principal {  
  
 public static void main(String[] args) {  
  
 Automovil auto1 = new Automovil();  
  
 auto1.setModelo("Modelo 1");  
 auto1.setFabricante("Fabricaente 1");  
 auto1.setColor(Color.*AZUL*);  
  
 Automovil auto2 = new Automovil();  
  
 auto2.setModelo("Modelo 2");  
 auto2.setFabricante("Fabricaente 2");  
 auto2.setColor(Color.*AMARILLO*);  
  
 System.*out*.println(auto1.detalle());  
 System.*out*.println(auto2.detalle());  
  
 }//MAIN  
}//CLASS



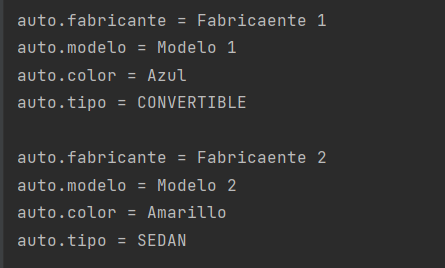
# 147. Implemetando enumerador con más atributos

package \_147\_EnumConMasAributos;  
  
public class Automovil {  
  
 private String fabricante;  
 private String modelo;  
 private Color color; //Color es de la clase Enum  
  
 private TipoAutomovil tipo;  
  
 public Automovil(){  
  
 }  
  
 public Automovil(String fabricante, String modelo, Color color) {  
 this.fabricante = fabricante;  
 this.modelo = modelo;  
 this.color = color;  
 }  
  
  
 public String getFabricante() {  
 return fabricante;  
 }  
  
 public void setFabricante(String fabricante) {  
 this.fabricante = fabricante;  
 }  
  
 public String getModelo() {  
 return modelo;  
 }  
  
 public void setModelo(String modelo) {  
 this.modelo = modelo;  
 }  
  
 public Color getColor() {  
 return color;  
 }  
  
 public void setColor(Color color) {  
 this.color = color;  
 }  
  
  
 public TipoAutomovil getTipo() {  
 return tipo;  
 }  
  
 public void setTipo(TipoAutomovil tipo) {  
 this.tipo = tipo;  
 }  
  
 public String detalle() {  
 StringBuilder sb = new StringBuilder();  
 sb.append("\nauto.fabricante = " + this.fabricante);  
 sb.append("\nauto.modelo = " + this.modelo);  
 sb.append("\nauto.color = " + this.color.getColor());//como estaba asiciona el getColor paar hacer uso del contenido de minusculas  
 sb.append("\nauto.tipo = " + this.getTipo());  
 return sb.toString();  
 }  
  
}//CLASS

package \_147\_EnumConMasAributos;  
  
//AL CREAR DONDE DICES CLASS... ELEGIR ENUM  
public enum Color {  
  
 *ROJO*("Rojo"),  
 *AMARILLO*("Amarillo"),  
 *AZUL*("Azul"),  
 *NARANJO*("Naranjo");  
  
 private final String color;  
  
 Color(String color) {  
 this.color = color;  
 }  
  
 public String getColor() {  
 return color;  
 }  
  
}//ENUM

package \_147\_EnumConMasAributos;  
  
public enum TipoAutomovil {  
  
 *SEDAN*("Sedan", "Auto mediano", 4),  
 *STATION*("Station", "Auto familiar", 5),  
 *PICKUP*("Pickup", "Camioneta de carga", 2),  
 *CONVERTIBLE*("Convertible", "Auto descapotable", 2),  
 *FURGON*("Furgon", "Vehículo de carga", 4);  
  
 private final String nombre;  
 private final int numeroPuertas;  
 private final String descipcion;  
  
  
 TipoAutomovil(String nombre, String descipcion , int numeroPuertas) {  
 this.nombre = nombre;  
 this.numeroPuertas = numeroPuertas;  
 this.descipcion = descipcion;  
 }  
  
 public String getNombre() {  
 return nombre;  
 }  
  
 public int getNumeroPuertas() {  
 return numeroPuertas;  
 }  
  
 public String getDescipcion() {  
 return descipcion;  
 }  
  
}//ENUM

package \_147\_EnumConMasAributos;  
  
  
public class Principal {  
  
 public static void main(String[] args) {  
  
 Automovil auto1 = new Automovil();  
  
 auto1.setModelo("Modelo 1");  
 auto1.setFabricante("Fabricaente 1");  
 auto1.setColor(Color.*AZUL*);  
 auto1.setTipo(TipoAutomovil.*CONVERTIBLE*);  
  
 Automovil auto2 = new Automovil();  
  
 auto2.setModelo("Modelo 2");  
 auto2.setFabricante("Fabricaente 2");  
 auto2.setColor(Color.*AMARILLO*);  
 auto2.setTipo(TipoAutomovil.*SEDAN*);  
  
 System.*out*.println(auto1.detalle());  
 System.*out*.println(auto2.detalle());  
  
 }//MAIN  
}//CLASS

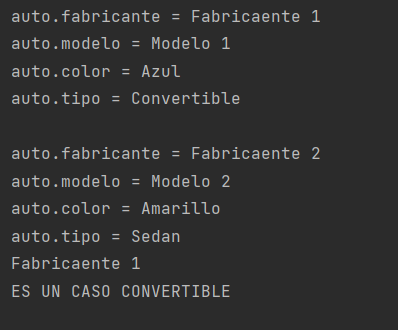


# 148. Utilizando enum con sentencias switch case

package \_148\_EnumWithSwitchCase;  
  
  
public class Principal {  
  
 public static void main(String[] args) {  
  
 Automovil auto1 = new Automovil();  
  
 auto1.setModelo("Modelo 1");  
 auto1.setFabricante("Fabricaente 1");  
 auto1.setColor(Color.*AZUL*);  
 auto1.setTipo(TipoAutomovil.*CONVERTIBLE*);  
  
 Automovil auto2 = new Automovil();  
  
 auto2.setModelo("Modelo 2");  
 auto2.setFabricante("Fabricaente 2");  
 auto2.setColor(Color.*AMARILLO*);  
 auto2.setTipo(TipoAutomovil.*SEDAN*);  
  
 System.*out*.println(auto1.detalle());  
 System.*out*.println(auto2.detalle());  
  
 /////////////////////////////////////////  
 TipoAutomovil tipoAuto1 = auto1.getTipo();  
 System.*out*.println(auto1.getFabricante());  
  
 switch (tipoAuto1){  
 case *SEDAN* :  
 System.*out*.println("ES UN CASO SEDAN");  
 break;  
 case *FURGON*:  
 System.*out*.println("ES UN CASO FURGON");  
 break;  
 case *CONVERTIBLE*:  
 System.*out*.println("ES UN CASO CONVERTIBLE");  
 break;  
 }//SWITCH  
  
 }//MAIN  
}//CLASS

package \_148\_EnumWithSwitchCase;  
  
public class Automovil {  
  
 private String fabricante;  
 private String modelo;  
 private Color color; //Color es de la clase Enum  
  
 private TipoAutomovil tipo;  
  
 public Automovil(){  
  
 }  
  
 public Automovil(String fabricante, String modelo, Color color) {  
 this.fabricante = fabricante;  
 this.modelo = modelo;  
 this.color = color;  
 }  
  
  
 public String getFabricante() {  
 return fabricante;  
 }  
  
 public void setFabricante(String fabricante) {  
 this.fabricante = fabricante;  
 }  
  
 public String getModelo() {  
 return modelo;  
 }  
  
 public void setModelo(String modelo) {  
 this.modelo = modelo;  
 }  
  
 public Color getColor() {  
 return color;  
 }  
  
 public void setColor(Color color) {  
 this.color = color;  
 }  
  
  
 public TipoAutomovil getTipo() {  
 return tipo;  
 }  
  
 public void setTipo(TipoAutomovil tipo) {  
 this.tipo = tipo;  
 }  
  
 public String detalle() {  
 StringBuilder sb = new StringBuilder();  
 sb.append("\nauto.fabricante = " + this.fabricante);  
 sb.append("\nauto.modelo = " + this.modelo);  
 sb.append("\nauto.color = " + this.color.getColor());//como estaba asiciona el getColor paar hacer uso del contenido de minusculas  
 sb.append("\nauto.tipo = " + this.getTipo().getNombre());  
 return sb.toString();  
 }  
  
}//CLASS

EL RESTO SON LAS MISMAS



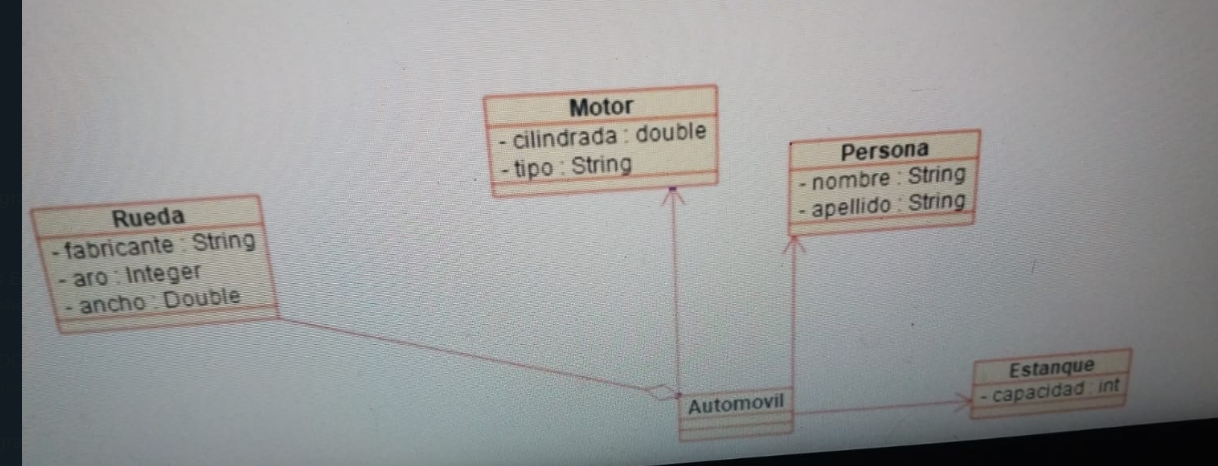
# 149. Iterando enum con foreach

//FOREACH  
TipoAutomovil[] tipos = TipoAutomovil.*values*();  
  
for (TipoAutomovil ta: tipos){  
 System.*out*.print(ta +"->" +ta.name()+"\n"  
 +"->" +ta.getNombre()+"\n"  
 +"->" +ta.getDescipcion()+"\n"  
 +"->" +ta.getNumeroPuertas());  
}

Solo agregue eso, obvio hay que acomodar como imprimir

# Practica 3

# 150. Diagrama de clases UML y relaciones de clases/objetos



# 151. Creando las clases relacionadas

OK

# 152. Modificando clase con atributos de relaciones de objetos

# 

# 153. Soluciones a posibles errores NullPointerException

# 154. Relacionando los demás objetos

# 155. Añadiendo el método add para agregar elementos de un arreglo

# 156. El método compareTo para ordenar objetos

# 157. Descargar Código Fuente Recursos

# Tarea 20: Proyecto Orden de Compra

# **Sección 15: Programación orientada a objetos: Paquetes**

# 158. Introducción a los paquetes

# 159. Creando nuestros primeros package y sus clases

# 160. Control de acceso mendiante modificadores

# 161. Alcance de los modificadores de accesos

# 162. Import estático

# 163. Descargar Código Fuente

# **Sección 16: Programación orientada a objetos: Ejemplo Facturas**

# **Sección 17: Programación orientada a objetos: Sobrecarga de métodos**

# **Sección 18: Programación orientada a objetos: Herencia**

# 178. Introducción a la herencia

# 179. Creando la jerarquía de clases

# 180. Sobre el modificador de acceso protected

# 181. Herencia y el API de reflexión de java

# 182. Llamando a los constructores en cadena

# 183. La referencia super en constructor

# 184. Creando instancias vía constructores con argumentos y control de tipos

# 185. Sobre escritura de métodos en la herencia

# 186. Sobre escritura método toString

# 187. Debug mode

# 188. Modificador final para evitar la herencia y la sobre-escritura de métodos

# 189. Resumen con diagrama UML de Clases

# 190. Descargar Código Fuente

# Tarea 21: Personas de una compañía

# Tarea 22: Almacén y verdulería

# Tarea 23: Un almacén de venta por mayor

# **Sección 19: Programación orientada a objetos: Herencia y Clases Abstractas**

# 191. Introducción a las clases abstractas

# 192. Diseño de clases UML

# 193. Creando clase padre abstracta ElementoForm

# 194. Creando clases hijas concretas InputForm y TextareaForm

# 195. Creando la clase concreta SelectForm

# 196. Creando la clase ejemplo con el método main EjemploForm

# 197. Dibujando elementos del formulario

# 198. Clases Anónimas

# 199. Creando la clase abstracta Validador

# con tres clases hijas concretas

# 200. Creando la clase concreta LargoValidador para validar el min y máx de caracteres

# 201. Creando la clase concreta EmailValidador

# 202. Implementando la validación en la clase abstracta ElementoForm

# 203. Validando los elementos del formulario

# 204. Añadiendo el nombre del campo en los mensajes de errores

# 205. Implementando interface para formatear mensajes de errores

# 206. Descargar Código Fuente

# Tarea 24: Proyecto Mamíferos

# **Sección 20: Programación orientada a objetos: Interfaces**

# **Sección 21: Programación orientada a objetos: Ejemplo Interface Repositorio**