

Introducción

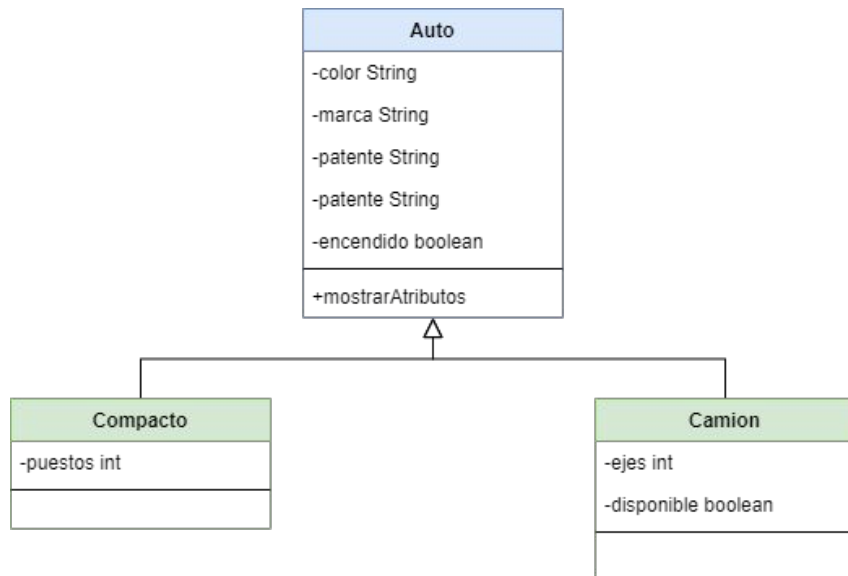
Como la palabra lo indica en los lenguajes Orientados a Objetos se pueden heredar características (Atributos y métodos) de otras clases.

La idea de la herencia es permitir la creación de nuevas clases basadas en clases existentes (Super Clases o Clases Padre).

Cuando heredas de una clase existente, puedes reutilizar métodos y campos de la clase principal. Además, también puede agregar nuevos métodos y campos en su clase actual.

Esto representa la relación ES-UN, que también se conoce como relación padre-hijo.

- **Un Compacto ES-UN Auto.**
- **Un Camión ES-UN Auto.**



Herencia en Java

Para poder heredar en Java se debe utilizar la palabra reservada **extends** después del identificador de la nueva clase seguido del identificador de la clase padre.

En java la Herencia es simple, lo que significa que solo podemos heredar de una sola clase, algunos lenguajes de programación permiten la herencia múltiple.

```
public class Compacto extends Auto {  
    private int puestos;  
  
    public int getPuestos() {  
        return puestos;  
    }  
  
    public void setPuestos(int puestos) {  
        this.puestos = puestos;  
    }  
}
```



```
public class Camion extends Auto {  
    private int ejes;  
    private boolean disponible;  
  
    public int getEjes() {  
        return ejes;  
    }  
  
    public void setEjes(int ejes) {  
        this.ejes = ejes;  
    }  
  
    public boolean isDisponible() {  
        return disponible;  
    }  
  
    public void setDisponible(boolean disponible) {  
        this.disponible = disponible;  
    }  
}
```




Constructores

Algo que pasa con los constructores es que no se heredan, así que al momento de instanciar un objeto de una clase hija se ejecuta el constructor de la clase padre, esto se hace de manera implícita si la clase padre posee el constructor simple o por defecto, de lo contrario debemos indicar qué constructor de la clase padre se debe invocar al construir un objeto de la clase hija.


Esto se hace con la sentencia **super()**, que invoca uno de los constructores de la superclase, el cual se escogerá en función de sus parámetros.

Si la super clase o clase padre no posee el constructor por defecto, ninguno de los hijos podrá instanciarse con un constructor de ese tipo (simple).

```
public Auto() {  
}  
  
public Auto(String color, String marca, String patente, boolean encendido) {  
    this.color = color;  
    this.marca = marca;  
    this.patente = patente;  
    this.encendido = encendido;  
}
```



```
public Camion() {  
    super();  
}  
  
public Camion(String color, String marca, String patente, boolean encendido, int ejes, boolean disponible) {  
    super(color, marca, patente, encendido);  
    this.ejes = ejes;  
    this.disponible = disponible;  
}
```



El orden en que se llaman los constructores es desde la clase que ocupa el nivel más alto en la jerarquía de herencia y se continúa de forma ordenada con el resto de las subclases.

Instancia y uso de los métodos y atributos

Instanciamos los objetos con los constructores disponibles

```
// le damos valores a los atributos del compacto 2
compacto2.setColor("Azul");
compacto2.setMarca("Mercedes Benz");
compacto2.setPatente("COM-89655");
compacto2.setPuestos(8);

// le damos valores a los atributos del camion 2
camion1.setColor("Negro");
camion1.setMarca("Tata");
camion1.setPatente("ARG-32169");
camion1.setEjes(16);
camion1.setDisponible(true);

// mostramos en consola el mensaje
System.out.println("La patente del Compacto1 es :" + compacto1.getPatente());
System.out.println("La patente del Camion es :" + camion1.getPatente());

// mostramos sus atributos
System.out.println(compacto1.mostrarDatos());
System.out.println(compacto2.mostrarDatos());

System.out.println(camion1.mostrarDatos());
System.out.println(camion2.mostrarDatos());
```

```
// creamos o instanciamos los objetos con sus constructores simples
Auto auto1 = new Auto();
Camion camion1 = new Camion();
Compacto compacto1 = new Compacto();

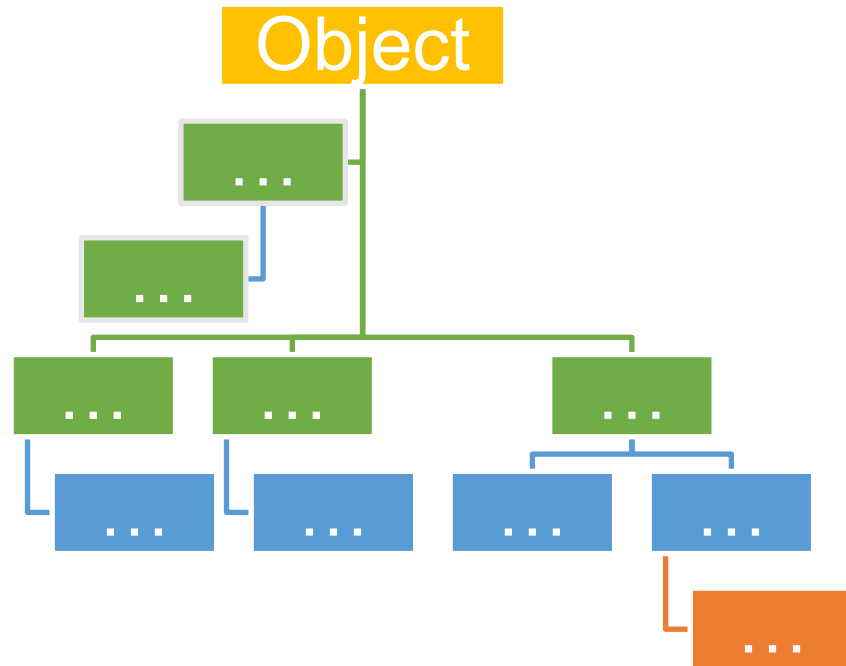
// creamos o instanciamos los objetos con sus constructores con atributos
// iniciales
Auto auto2 = new Auto("Plateado", "Audi", "ZBG-999", true);
Compacto compacto2 = new Compacto("Azul", "Ford", "ANZ-963", true, 3);
Camion camion2 = new Camion("Verde", "Mercedes Benz", "CAM-7896", false, 16, true);
```

Métodos heredados de la clase Auto

Métodos propios de la clase

Jerarquía de Herencia en Java

Java no escapa en si de la herencia, lo que representa una gran ventaja a la hora de utilizar métodos y atributos propios de la API de JAVA ya que todas las clases heredan por defecto de la super clase Object.



final

Una clase puede poseer la palabra clave **final** si se desea eliminar la posibilidad de herencia en dicha clase.

Imaginemos que la Patente no solo es un dato alfanumérico sino que posee asociado al numero un atributo que indique que se encuentre activo, esto nos llevaría a crear una clase para solo la patente y usarla en la clase Auto para tener un código mas ordenado y reutilizar mas adelante este objeto.

Además podemos decirle a Java a través de **final** que ninguna otra clase puede heredar de esta Patente, esto tiene sentido ya que ¿que objeto de la vida real tiene la relación ES-UNA patente?

Aunque nosotros al programar sabemos que no deberíamos nunca heredar de estas clases, es importante indicarlo ya que en una organización o empresa no somos los únicos programadores y con esto evitamos posibles errores.

```
public final class Patente {  
    private String numero;  
    private boolean activa;  
  
    public Patente(String numero, boolean activa) {  
        super();  
        this.numero = numero;  
        this.activa = activa;  
    }  
  
    public String getNumero() {  
        return numero;  
    }  
  
    public void setNumero(String numero) {  
        this.numero = numero;  
    }  
  
    public boolean isActive() {  
        return activa;  
    }  
  
    public void setActiva(boolean activa) {  
        this.activa = activa;  
    }  
}
```

