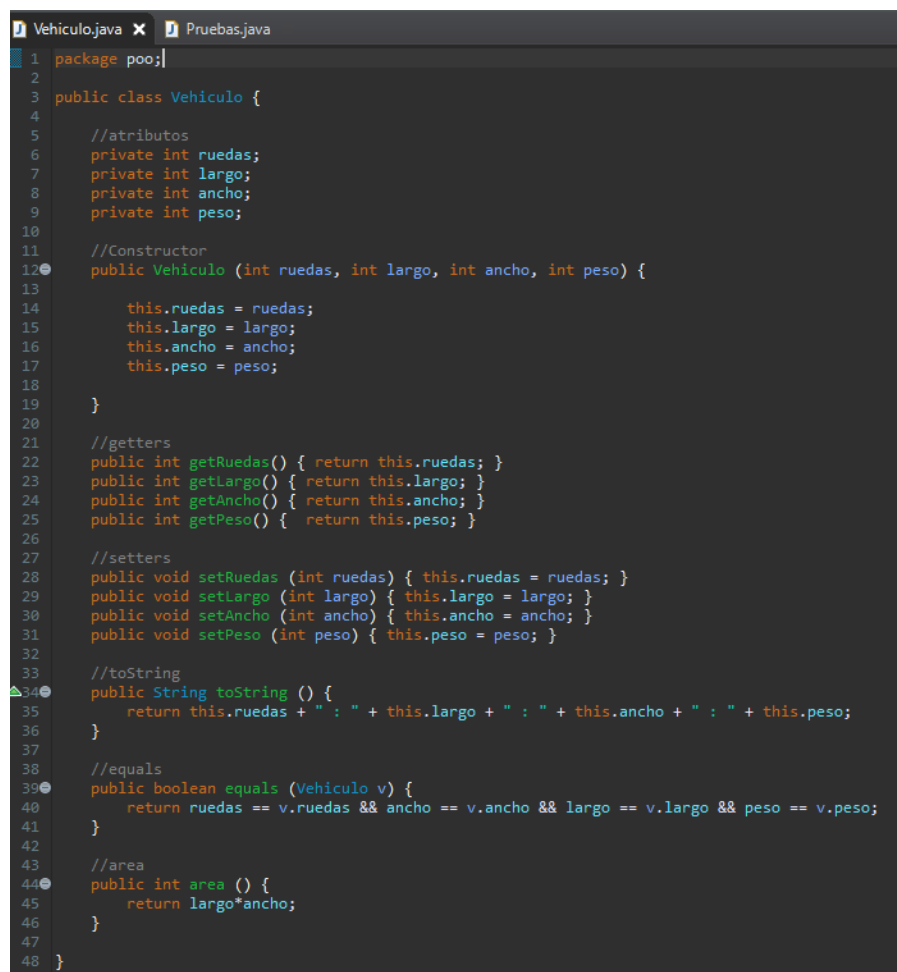


Herencia

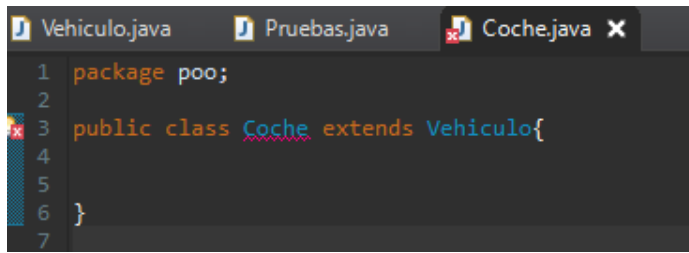
Imaginemos qué queremos crear una clase orientada a objetos que sea la extensión de otra clase, es decir, que hereda de ella, cuando una clase B, hereda de una clase A, la clase B va a tener todos los atributos, constructores y métodos de la clase A, aunque si estos tienen el modificador *private* o *final* pueden ocurrir otras cosas, para saber más sobre esto mirar el PDF sobre “Modificadores y Visibilidad”, ahora bien, ¿Cuándo queremos hacer que una clase sea la extensión de otra?, principalmente cuando ésta clase A tiene cosas (atributos, constructores o métodos) que no queremos tener que volver a hacer, esto es más fácil de ver con un ejemplo, supongamos que tenemos ésta clase:



```
1 package poo;
2
3 public class Vehiculo {
4
5     //atributos
6     private int ruedas;
7     private int largo;
8     private int ancho;
9     private int peso;
10
11     //Constructor
12     public Vehiculo (int ruedas, int largo, int ancho, int peso) {
13
14         this.ruedas = ruedas;
15         this.largo = largo;
16         this.ancho = ancho;
17         this.peso = peso;
18     }
19
20
21     //getters
22     public int getRuedas() { return this.ruedas; }
23     public int getLargo() { return this.largo; }
24     public int getAncho() { return this.ancho; }
25     public int getPeso() { return this.peso; }
26
27     //setters
28     public void setRuedas (int ruedas) { this.ruedas = ruedas; }
29     public void setLargo (int largo) { this.largo = largo; }
30     public void setAncho (int ancho) { this.ancho = ancho; }
31     public void setPeso (int peso) { this.peso = peso; }
32
33     //toString
34     public String toString () {
35         return this.ruedas + " : " + this.largo + " : " + this.ancho + " : " + this.peso;
36     }
37
38     //equals
39     public boolean equals (Vehiculo v) {
40         return ruedas == v.ruedas && ancho == v.ancho && largo == v.largo && peso == v.peso;
41     }
42
43     //area
44     public int area () {
45         return largo*ancho;
46     }
47
48 }
```

Como veis tiene 4 atributos, un constructor, sus *setters* y *getters*, el método *toString* y el *equals* y aparte un método que te devuelve el área que ocupa el vehículo, digamos que ésta clase se encarga de las especificaciones que puede tener un Vehiculo (coche, moto, furgoneta, camión...), ahora queremos crear una clase que represente un Coche, para ello no vamos a volver a poner estos 4 atributos, el constructor y sus métodos, es mejor reciclar éste código.

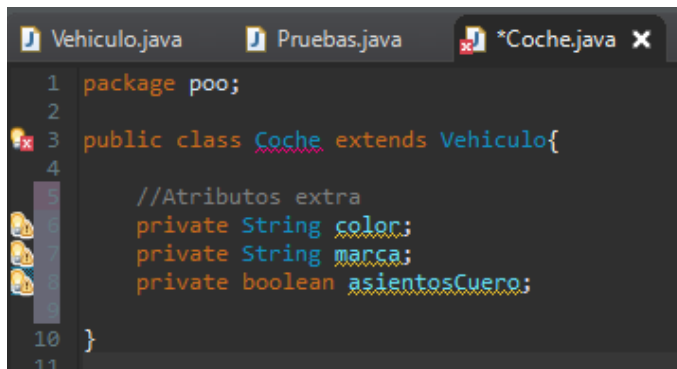
Para hacer que la clase Coche.java herede de Vehiculo.java vamos a hacer lo siguiente:



```
1 package poo;
2
3 public class Coche extends Vehiculo{
4
5
6 }
7
```

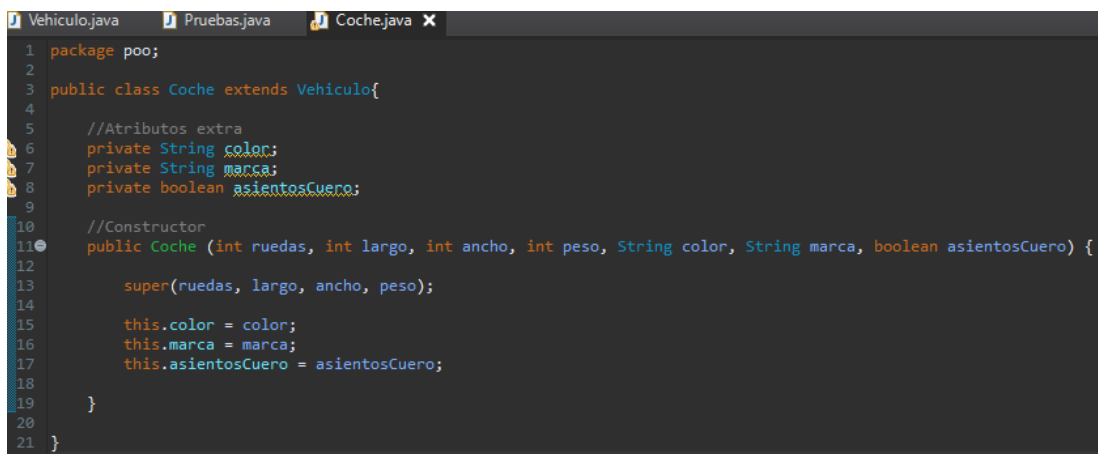
(El error que me da se debe a que tengo que hacer un constructor en la clase B, sí la clase A tenía un constructor).

Solamente tenemos que poner ClaseHijo **extends** ClasePadre, de ésta forma todos los atributos, constructores y métodos de la clase padre, ahora también pertenecen a la clase hijo, vamos a poner unos atributos extra, que la clase padre no tenía:



```
1 package poo;
2
3 public class Coche extends Vehiculo{
4
5     //Atributos extra
6     private String color;
7     private String marca;
8     private boolean asientosCuero;
9
10 }
11
```

Algo importante es que la clase Coche no puede modificar directamente los atributos de la clase Vehiculo, es decir, aunque un Coche tenga el atributo *alto*, *ancho*, *peso* y *ruedas*, no puede modificarlos directamente, esto se debe a que éstos atributos tienen el modificador *private*, y **aunque Coche hereda los atributos de Vehiculo, no puede modificarlos**, vamos a crear el constructor:



```
1 package poo;
2
3 public class Coche extends Vehiculo{
4
5     //Atributos extra
6     private String color;
7     private String marca;
8     private boolean asientosCuero;
9
10     //Constructor
11     public Coche (int ruedas, int largo, int ancho, int peso, String color, String marca, boolean asientosCuero) {
12
13         super(ruedas, largo, ancho, peso);
14
15         this.color = color;
16         this.marca = marca;
17         this.asientosCuero = asientosCuero;
18     }
19 }
20
21
```

Este constructor no solo recibe los parámetros necesarios para la Coche, es decir, no solo recibe el *color*, la *marca* y *asientosCuero*, también recibe los parámetros necesarios para el constructor que queramos de la clase padre.

Para hacer referencia al constructor de la clase padre, ponemos **super(PARÁMETROS)**, *super* hace referencia a “superclase”, y diferenciaremos el constructor que queramos utilizar en base a los parámetros, en el ejemplo que he puesto solo hay un constructor, por lo que no tiene que diferenciar entre constructores, pero a veces nos encontraremos con que la clase padre tiene varios constructores.

Después de llamar a la clase padre utilizando *super* para que nos asigne los atributos de la superclase, asignamos los de nuestra clase hijo, es decir, *color*, *marca* y *asientosCuero*.

Vamos a hacer un test:

```
Vehiculo v = new Vehiculo(4,250,170,1500);
Coche c = new Coche(4,250,170,1500,"Rojo","Toyota",false);

System.out.println(v.getPeso());
System.out.println(c.getLargo());
```

>>>

Problems
<terminated>
1500
250

Se puede apreciar que *c*, que es del tipo **Coche**, puede utilizar el método *getLargo()* de la clase Vehiculo, en cambio, vamos a ver que pasa si queremos ver los datos tanto del Vehiculo como del Coche:

```
Vehiculo v = new Vehiculo(4,250,170,1500);
Coche c = new Coche(4,280,180,1700,"Rojo","Toyota",false);

System.out.println(v);
System.out.println(c);
```

>>>

<terminated> PruebasPoo [Ja
4 : 250 : 170 : 1500
4 : 280 : 180 : 1700

(He cambiado los datos de *c* para evitar confusiones)

Los datos del vehiculo se muestran correctamente, pero faltan los datos de esos atributos extra del Coche *c*, es decir, no nos muestra el color, la marca ni los asientos de cuero.

Vamos a crear los métodos *getters*, *setters* y a **sobrescribir** el método *toString*.

```
//setters y getters
public String getColor () { return this.color; }
public String getMarca () { return this.marca; }
public boolean getAsientosCuero () { return this.asientosCuero; }

public void setColor (String color) { this.color = color; }
public void setMarca (String marca) { this.marca = marca; }
public void setAsientosCuero (boolean asientosCuero) { this.asientosCuero = asientosCuero; }

//toString @Override
public String toString () {
    return super.toString() + " : " + color + " : " + marca + " : " + asientosCuero;
}
}
```

class Coche.java

Los setters y getters no tienen nada de especial, pero vamos a ver el método *toString()*, si os fijáis no tengo que escribir otra vez `this.ruedas + " : " + this.largo + " : " + this.ancho + " : " + this.peso;`, esto ya lo hace el método *toString()* de la superclase, por lo que para referirnos a ésta, y que el programa sepa que nos referimos al método *toString()* de la clase Vehiculo y no el de la clase Coche, ponemos **super.** delante: `super.toString()`. Ahora al ejecutar el

programa ya estaría solucionado >>>

```
4 : 250 : 170 : 1500
4 : 280 : 180 : 1700 : Rojo : Toyota : false
```

POLIMORFISMO

Aunque tenga un nombre de parecer complicado en realidad es bien sencillo, en vez de usar el típico ejemplo del Vehículo y el Coche prefiero utilizar uno que es más fácil de entender, imaginemos que tenemos una clase Persona y otra clase Estudiante, cada una con sus atributos, constructores y métodos, pero la clase Estudiante hereda de la clase Persona, y ¿Por qué no al revés?, vamos a utilizar la frase “es un...” para entender esto, ¿Un Estudiante es una Persona?, si para cualquier caso, pero, ¿Una Persona es un Estudiante?, no tiene por qué serlo, es decir, todos los atributos que tiene una Persona también los tiene un Estudiante (nombre, altura, edad, peso...), pero los atributos de un Estudiante no los tiene una persona (numeroMatricula, curso...), ésta frase se utiliza mucho para saber si una clase puede heredar o no de otra, y ¿Qué tiene que ver esto con el polimorfismo?, como su nombre indica, el polimorfismo es cambiar de forma un dato, por ejemplo, podemos convertir de *int* a *double* tranquilamente poniendo:

int entero = (int)5.75;

En cambio, para hacer esto:

```
String str = (String)5;    >>> Cannot cast from int to String
```

Nos dará un error, no se puede convertir de *int* a *String* directamente, existen ciertos métodos que nos permiten hacer esto en específico, pero no me voy a meter, entonces bien, ¿Puedo convertir a un Estudiante en una Persona?, sí que puedo, ya que un Estudiante **SÍ** es una Persona, por lo que ésta sentencia estaría bien:

Persona p1 = (Persona)estudiante12;

En cambio, como una Persona no es un Estudiante, no se puede realizar a la inversa:

~~Estudiante e1 = (Estudiante)persona12;~~

El polimorfismo solo es eso, transformar un tipo de objeto en otro siempre que se pueda y se necesite.