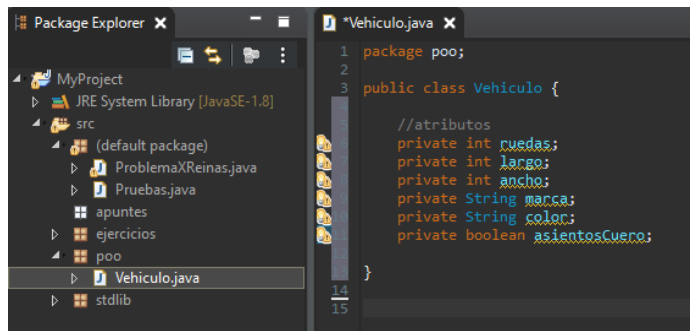


## ATRIBUTOS Y CONSTRUCTORES

Imaginemos que quiero crear un objeto con múltiples atributos, es decir, crear una variable que contenga datos de diferente tipo (*ints*, *booleans*, *Strings*, *doubles*, etc.), cuando hemos creado un *array*, de *ints* por ejemplo, estamos creando una variable que contiene varios datos, pero todos del mismo tipo (del tipo *int*), ahora propongo crear una variable (Objeto) que represente un vehículo, y que los datos que contengan sean; número de ruedas, el largo, el ancho, la marca, el color y si tiene asientos de cuero o no.

Para ello vamos a crear una clase llamada *Vehiculo*, y vamos a definir éstos atributos:



La clase se llamará como el tipo de Objeto que queremos crear, si queremos un Objeto que represente un Vehículo, llamaremos a la clase *Vehiculo*, ya que nos referiremos al Objeto por este nombre, si os fijáis, antes al crear una variable fuera de una función, poníamos *static* delante, a los atributos no hay que ponerles *static* delante, ya que, si no serían una variable en vez de un atributo, y les tenemos que poner *private*, explicaré esto mejor del *static/non-static* y del *private/public* en el siguiente PDF, por ahora quedémonos con que todo es *public* menos los atributos y que nada en objetos es del tipo *static*.

Para poder definir los atributos de nuestro vehículo, necesitamos crear un **constructor**, los constructores otorgan valores a los atributos:

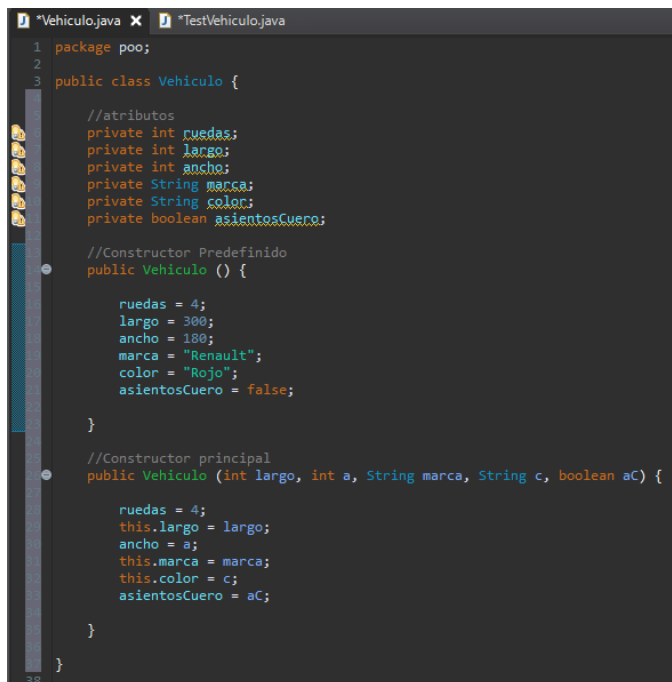
```
public class Vehiculo {  
    //atributos  
    private int ruedas;  
    private int largo;  
    private int ancho;  
    private String marca;  
    private String color;  
    private boolean asientosCuero;  
  
    //Constructor principal  
    public Vehiculo (int largo, int a, String marca, String c, boolean aC) {  
  
        ruedas = 4;  
        this.largo = largo;  
        ancho = a;  
        this.marca = marca;  
        this.color = c;  
        asientosCuero = aC;  
  
    }  
}
```

Para crear un constructor pondremos → *public NOMBRE\_DE\_LA\_CLASE (PARAMETROS)*

Es decir, pondremos *public* seguido del nombre de la clase (en nuestro caso *Vehiculo*) y luego los parámetros que le queremos pasar, yo he decidido que todos los *Vehiculos* van a tener 4 ruedas, por lo que solo le paso el largo, el ancho, la marca, el color y si tiene asiento de cuero o no. Ahora queda asignarle los atributos, viendo la primera asignación (*ruedas = 4*) no hay nada raro, ahora fijemos en los 2 siguientes, *largo* y *ancho*, a uno le he puesto *this*. delante y al otro

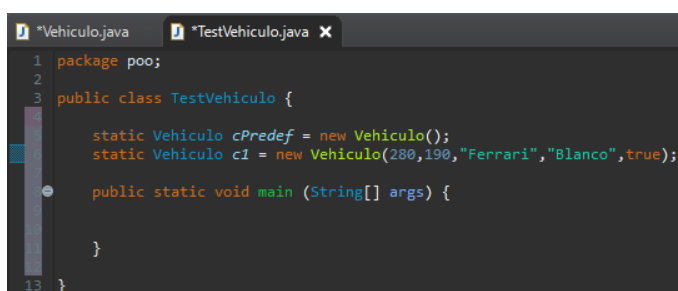
no, pero ¿Por qué?, si nos fijamos el parámetro *largo* tiene el mismo nombre que el atributo *largo*, mientras que para el parámetro que indica el ancho del vehículo "*a*" no ocurre lo mismo, tienen diferente nombre, por lo que, **si tenemos un parámetro que se llama igual que un atributo, diferenciaremos uno de otro poniendo *this*. delante del ATRIBUTO**. Es decir, el *this* nos diferencia cuál es el atributo, y cuál el parámetro, veamos los siguientes 2 atributos, *marca* y *color*, en estos casos he puesto *this* delante a pesar de que en el segundo caso el parámetro tiene distinto nombre que el atributo, esto se debe a que no es necesario poner el *this* solo cuando los dos se llaman igual, puedes ponerlo siempre que quieras, y bueno, el último atributo (*asientosCuero*) no tiene nada de especial.

Ahora supongamos que quiero crear un constructor que nos cree un *Vehiculo* predefinido, que le de unas características generales:



```
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 String marca;
10    private String color;
11    private boolean asientosCuero;
12
13    //Constructor Predefinido
14    public Vehiculo () {
15
16        ruedas = 4;
17        largo = 300;
18        ancho = 180;
19        marca = "Renault";
20        color = "Rojo";
21        asientosCuero = false;
22    }
23
24    //Constructor principal
25    public Vehiculo (int largo, int a, String marca, String c, boolean aC) {
26
27        ruedas = 4;
28        this.largo = largo;
29        ancho = a;
30        this.marca = marca;
31        this.color = c;
32        asientosCuero = aC;
33    }
34
35 }
```

Ahora tengo dos constructores, uno es el **predefinido**, y otro el **principal**, vamos a crear dos vehículos en la *class TestVehiculos*:



```
1 package poo;
2
3 public class TestVehiculo {
4
5     static Vehiculo cPredef = new Vehiculo();
6     static Vehiculo c1 = new Vehiculo(280,190,\"Ferrari\", \"Blanco\", true);
7
8     public static void main (String[] args) {
9
10    }
11
12 }
```

Para crear un objeto pondremos:

**TIPO\_DE\_OBJETO NOMBRE = new TIPO\_DE\_OBJETO(PARÁMETROS);**

El *static* dependerá de si está dentro de una función o no, como el mío está fuera, se lo he puesto, entonces, para referirnos al constructor predefinido no le pasamos parámetros, como hemos puesto en nuestra clase *Vehiculo*, mientras que para referirnos al constructor principal tendremos que pasarle (*int,int,String,String,boolean*), al igual que ocurría en lo que vimos en el

PDF de programación 1 acerca de “Funciones y Procedimientos”, diferenciamos constructores en base a los parámetros que le pasamos.

Como vemos en nuestros constructores se le asigna el valor 4 a ruedas en todos los casos, así que, en vez de hacerlo en los constructores, podemos hacer que éste sea el valor predefinido:

```
3 public class Vehiculo {
4
5     //atributos
6     private int ruedas = 4;
7     private int largo;
8     private int ancho;
9     private String marca;
10    private String color;
11    private boolean asientosCuero;
12
13    //Constructor Predefinido
14    public Vehiculo () {
15
16        largo = 300;
17        ancho = 180;
18        marca = "Renault";
19        color = "Rojo";
20        asientosCuero = false;
21
22    }
23
24    //Constructor principal
25    public Vehiculo (int largo, int a, String marca, String c, boolean aC) {
26
27        this.largo = largo;
28        ancho = a;
29        this.marca = marca;
30        this.color = c;
31        asientosCuero = aC;
32
33    }
34
35 }
```

He borrado esas sentencias de los constructores, y ahora en la parte de atributos (línea 6) he puesto que ruedas es igual a 4, de ésta forma decimos que el valor de las ruedas será siempre 4, a menos que un constructor o un *setter* (algo que aún no hemos visto) lo modifique posteriormente, vamos a hacer unos cuantos cambios, quiero que nuestros valores predefinidos sean los que están en ese constructor, y que, si llamamos al principal, se cambien todos los valores:

```
public class Vehiculo {
    //atributos
    private int ruedas = 4;
    private int largo = 300;
    private int ancho = 180;
    private String marca = "Renault";
    private String color = "Rojo";
    private boolean asientosCuero = false;
    //Constructor Predefinido
    public Vehiculo () {
    }
    //Constructor principal
    public Vehiculo (int ruedas, int largo, int a, String marca, String c, boolean aC) {
        this.ruedas = ruedas;
        this.largo = largo;
        ancho = a;
        this.marca = marca;
        this.color = c;
        asientosCuero = aC;
    }
}
```

```
public class TestVehiculo {
    static Vehiculo cPredef = new Vehiculo();
    static Vehiculo c1 = new Vehiculo(4,280,190,"Ferrari","Blanco",true);
    public static void main (String[] args) {
    }
}
```

Nuestras clases quedan así, como veis el constructor predefinido no hace nada, solo sirve para que se le pueda invocar, y dar los valores predefinidos a nuestro objeto *cPredef* de nuestra clase *TestVehiculo*, y ahora el constructor principal recibe tantos parámetros como atributos,

aunque se pueda hacer esto puede resultar un poco lío, y no son malas prácticas de programación, ya que hacer esto es más óptimo, pero puede liar un poco, así que yo os recomiendo lo siguiente:

-Siempre que nos refiramos a un **atributo** poner *this.* delante.

-En los constructores llamar a los parámetros igual que los atributos.

-Solamente si somos principiantes, asignar todos los valores a los atributos dentro de los constructores, esto lo digo para que no se os olvide asignar parámetros, ya que, si no, tendrán el valor predeterminado que les asigna java, que os recordaré que son:

*int = 0 - double = 0 - boolean = false - char = " - Objetos (String, Double) = null*

Explicaré esto del *null* en la parte de "¿Cómo funciona la memoria del ordenador?" del final de este PDF.

Dados éstos consejos:

```
Vehiculo.java x TestVehiculo.java
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 String marca;
10    private String color;
11    private boolean asientosCuero;
12
13    //Constructor Predefinido
14    public Vehiculo () {
15
16        this.ruedas = 4;
17        this.largo = 300;
18        this.ancho = 180;
19        this.marca = "Renault";
20        this.color = "Rojo";
21        this.asientosCuero = false;
22
23    }
24
25    //Constructor principal
26    public Vehiculo (int ruedas, int largo, int ancho, String marca, String color, boolean asientosCuero) {
27
28        this.ruedas = ruedas;
29        this.largo = largo;
30        this.ancho = ancho;
31        this.marca = marca;
32        this.color = color;
33        this.asientosCuero = asientosCuero;
34
35    }
36
37 }
38
```

```
Vehiculo.java x TestVehiculo.java x
1 package poo;
2
3 public class TestVehiculo {
4
5     static Vehiculo cPredef = new Vehiculo();
6     static Vehiculo c1 = new Vehiculo(4,280,190,"Ferrari","Blanco",true);
7     static Vehiculo c2 = new Vehiculo(4,310,150,"Citroen","Azul",false);
8     static Vehiculo c3 = new Vehiculo(4,275,165,"Toyota","Gris",false);
9     static Vehiculo c1Copia = new Vehiculo(4,280,190,"Ferrari","Blanco",true);
10
11    public static void main (String[] args) {
12
13    }
14
15
16 }
```

Nuestras dos clases quedarán así, hasta el final del PDF, así que tomadlas de referencia.

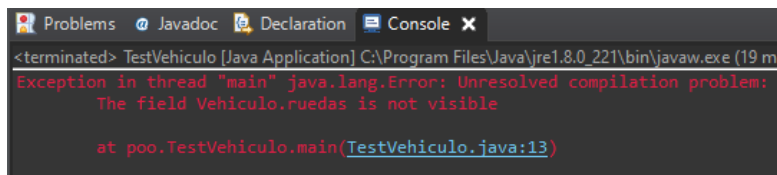
Explicado atributos y constructores, pasemos a los métodos.

## MÉTODOS

Pongamos que quiero escribir en pantalla cuantas ruedas tiene c1, pues pondría lo siguiente:

```
public class TestVehiculo {  
  
    static Vehiculo cPredef = new Vehiculo();  
    static Vehiculo c1 = new Vehiculo(4,280,190,"Ferrari","Blanco",true);  
    static Vehiculo c2 = new Vehiculo(4,310,150,"Citroen","Azul",false);  
    static Vehiculo c3 = new Vehiculo(4,275,165,"Toyota","Gris",false);  
    static Vehiculo c1Copia = new Vehiculo(4,280,190,"Ferrari","Blanco",true);  
  
    public static void main (String[] args) {  
  
        System.out.println(c1.ruedas);  
  
    }  
}
```

>>>



The screenshot shows an IDE window with a tab labeled 'Console'. The console output displays a Java compilation error: 'Exception in thread "main" java.lang.Error: Unresolved compilation problem: The field Vehiculo.ruedas is not visible'. The error points to the line 'System.out.println(c1.ruedas);' in the file 'TestVehiculo.java' at line 13.

Me dice: *The field Vehiculo.ruedas is not visible*, con esto se refiere a que el atributo *ruedas* no es visible desde ésta clase, esto se debe a que *ruedas* es del tipo *private*, os dejo un resumen referente a atributos:

-*public*: Todas las clases pueden verla y modificarla.

-*friendly* (no poner nada): Todas las clases del mismo paquete pueden verla y modificarla.

-*protected*: Todas pueden verla, pero solo la que la contiene puede modificarla.

-*private*: Solo la clase que la contiene puede verla y modificarla.

Nuestro atributo *ruedas* es del tipo *private*, por lo que no podemos hacer nada con *ruedas* desde otra clase.

Para poder recibir su valor tenemos lo que se llaman *getters* o visualizadores, los *getters* son *métodos* con la siguiente forma (en la clase *Vehiculo.java*):

```
//getters  
public int getRuedas() {  
  
    return this.ruedas;  
  
}
```

Recordad no poner *static*, ya que esto no es Programación Orientada a Objetos (POO), ya no estamos en programación imperativa, algunas veces si hará falta poner *static*, pero eso es para funciones/procedimientos, **los métodos no llevan *static***.

Hemos puesto *public* para que pueda usar este *método* todas las clases, *int* porque tiene que devolvernos un valor entero (el número de ruedas), de nombre recomiendo poner *getAtributo*, cambiando *atributo* por el que toque, en este caso, *ruedas*, no recibe parámetros, y lo único que nos devuelve es el valor del atributo *ruedas*.

Os dejo por aquí todos los *getters* y algunos ejemplos:

```
//getters
public int getRuedas() {
    return this.ruedas;
}

public int getLargo() {
    return this.largo;
}

public int getAncho() {
    return this.ancho;
}
```

```
public String getMarca() {
    return this.marca;
}

public String getColor() {
    return this.color;
}

public boolean getAsientosCuero() {
    return this.asientosCuero;
}
```

```
public class TestVehiculo {

    static Vehiculo cPredef = new Vehiculo();
    static Vehiculo c1 = new Vehiculo(4,280,190,"Ferrari","Blanco",true);
    static Vehiculo c2 = new Vehiculo(4,310,150,"Citroen","Azul",false);
    static Vehiculo c3 = new Vehiculo(4,275,165,"Toyota","Gris",false);
    static Vehiculo c1Copia = new Vehiculo(4,280,190,"Ferrari","Blanco",true);

    public static void main (String[] args) {

        System.out.println(c1.getRuedas());
        System.out.println(c2.getAsientosCuero());
        System.out.println(c3.getColor());

        int areaCPredef = cPredef.getLargo() * cPredef.getAncho();
        System.out.println(areaCPredef);

    }

}
```

>>>

```
4
false
Gris
54000
```

Como veis los tres primeros *prints* nos muestran los datos perfectamente, ahora, si queremos usar los atributos para sumarlos, multiplicarlos, o lo que sea, vamos a necesitar llamar también a estos métodos *getters*, que no os confunda el nombre de *visualizador*, estos atributos no dan problemas solo a la hora de mostrarlos en pantalla con un *print*, el hecho de que sean de tipo *private* nos prohíbe usar *nombreAtributo.ruedas* directamente, tenemos que crear un método *getter* obligatoriamente para poder usar este atributo.

Veamos que ocurre si tratamos de modificar *ruedas*:

```
public static void main (String[] args) {

    c1.ruedas = 3;
    System.out.println(c1.getRuedas());

}
```

>>>

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The field Vehiculo.ruedas is not visible

at poo.TestVehiculo.main(TestVehiculo.java:13)
```

Ahí lo teneís, el mismo error, supongo que ya sabéis el motivo, pero lo diré aun así, los atributos *private* solo se pueden modificar desde la misma clase, por lo que poner *c1.ruedas = 3* es imposible, tenemos que crear lo que se llama un método *setter*, os pongo todos ya creados:

```
//setters
public void setRuedas (int ruedas) {
    this.ruedas = ruedas;
}

public void setLargo (int largo) {
    this.largo = largo;
}

public void setAncho (int ancho) {
    this.ancho = ancho;
}
```

```
public void setMarca (String marca) {
    this.marca = marca;
}

public void setColor (String color) {
    this.color = color;
}

public void setAsientosCuero (boolean asientosCuero) {
    this.asientosCuero = asientosCuero;
}
```

```
public class TestVehiculo {
    static Vehiculo cPredef = new Vehiculo();
    static Vehiculo c1 = new Vehiculo(4,280,190,"Ferrari","Blanco",true);
    static Vehiculo c2 = new Vehiculo(4,310,150,"Citroen","Azul",false);
    static Vehiculo c3 = new Vehiculo(4,275,165,"Toyota","Gris",false);
    static Vehiculo c1Copia = new Vehiculo(4,280,190,"Ferrari","Blanco",true);

    public static void main (String[] args) {
        c1.setRuedas(3);
        System.out.println(c1.getRuedas());
    }
}
```

>>>

```
<termi
3
```

Y ahora ya puedo cambiar el valor de *ruedas*, usando *c1.setRuedas(3)*.

Veamos que pasa si imprimo un objeto directamente:

```
System.out.println(c1);
```

>>>

```
poo.Vehiculo@15db9742
```

Lo que nos imprime es la dirección de memoria, eso de *@15db9742*, acompañada del paquete y la clase donde se crea nuestro Objeto, veamos como podemos hacer para solucionar esto, y que nos escriba el Objeto tal y como queremos:

```
//toString
public String toString () {
    return this.ruedas + " : " + this.largo + " : " + this.ancho + " : " + this.marca + " : " + this.color + " : " + this.asientosCuero;
}
```

Tenemos que crear un método llamado *toString*, dentro de la clase *Vehiculo*, y ahora:

```
System.out.println(c1);
```

>>>

```
4 : 280 : 190 : Ferrari : Blanco : true
```

Lo que hace el programa si tratamos de imprimir un objeto es:

-Va a la clase del tipo de objeto (como nuestro objeto es del tipo *Vehiculo*, va a ésta clase).

-Busca el método con ésta forma *public String toString () { //Lo que sea }*

-Si lo encuentra, nos devuelve lo que ella en el *return* de ese método, sino, nos devolverá la dirección de memoria.

Posteriormente os explicaré lo que ocurrirá en la clase *String.java*, y como funcionan las direcciones de memoria.

## NORMA DE LA UPM

Si un método se llama *equals* → tiene que comparar **TODOS** los atributos.

Supongamos que quiero saber si 2 vehículos son iguales, voy a crear un método que los compare:

```
//equals
public boolean equals (Vehiculo v) {
    return ruedas == v.ruedas && ancho == v.ancho && largo == v.largo && marca.equals(v.marca) && color.equals(v.color) && asientosCuero == v.asientosCuero;
}
```

Tenemos que comparar **TODOS** los atributos, lo remarco porque si no os quitan puntos.

Fijaros que los atributos que son *Strings* (*marca* y *color*) no los comparo directamente, uso el método *equals*, que no es el de ésta clase, es el de la clase *String*, pero esto lo explicaré mejor al final de éste PDF.

También démonos cuenta que le paso como parámetro otro Objeto del tipo *Vehiculo*, vamos a ver un ejemplo del *equals* en acción:

```
System.out.println(c1.equals(c2));
System.out.println(c1.equals(c1Copia)); >>> false
true
```

Vemos que funciona, vamos a hacer otro simple método que nos devuelva el área que ocupa el vehículo en el asfalto, por ejemplo:

```
//area
public int area () {
    return largo*ancho;
}
```

```
System.out.println(c1.area());
System.out.println(c3.area()); >>> 53200
45375
```

He de mencionar que los métodos deberían estar mucho mejor documentados, pero esto es solo para que sea más didáctico.

Ya hemos acabado con la parte de como crear Objetos y manejarlos, pero todavía queda explicar como funciona la memoria del ordenador, así que...



## ¿Cómo funciona la memoria del ordenador?

Al tratar de mostrar por pantalla un Objeto sin haber creado el método *toString* pasaba esto:

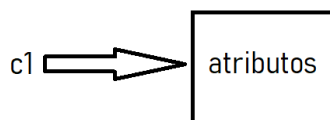
(He borrado el método *toString* de la clase *Vehiculo.java*)

```
System.out.println(c1); >>> poo.Vehiculo@15db9742
```

Esto se debe a que cuando ponemos:

```
static Vehiculo c1 = new Vehiculo(4,280,190,"Ferrari","Blanco",true);
```

Ese **new** está creando un espacio de memoria en el ordenador, al cuál apunta *c1*:



Si ponemos algo como: *static Vehiculo v = null*;



Tendremos una variable apuntando a la nada, sin ningún espacio de memoria reservado.

Por lo que cuando ponemos:

```
static Vehiculo cPredef = new Vehiculo();
static Vehiculo c1 = new Vehiculo(4,280,190,"Ferrari","Blanco",true);
static Vehiculo c2 = new Vehiculo(4,310,150,"Citroen","Azul",false);
static Vehiculo c3 = new Vehiculo(4,275,165,"Toyota","Gris",false);
static Vehiculo c1Copia = new Vehiculo(4,280,190,"Ferrari","Blanco",true);

public static void main (String[] args) {

    System.out.println(c1);
    System.out.println(c2);
    System.out.println(c1Copia);

    System.out.println(c1 == c2);
    System.out.println(c1 == c1Copia);

}
```

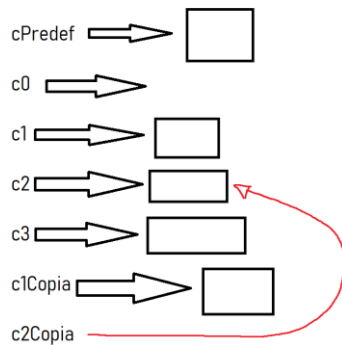
```
poo.Vehiculo@15db9742
poo.Vehiculo@6d06d69c
poo.Vehiculo@7852e922
false
false
```

Como veis, no importa que los atributos que contienen sean los mismos, ambos dan *false*.

Por eso creamos el método *equals*, para que se puedan comparar dos objetos, ya que sino, estamos comparando dos direcciones de memoria, no dos Objetos y sus atributos, por todo esto los *Strings* son objetos, ya que si os fijais, cuando creamos un Objeto y un dato *primitivo* lo único que los diferencia es si llevan mayúscula o no (*int* y *String*, *double* y *Double*), y por eso los primitivos pueden ser comparados directamente, mientras que los *no-primitivos* son objetos, y necesitan de un método *equals* que compare todos sus datos *primitivos*, por lo mismo que cuando comparamos 2 *Strings* no ponemos directamente *==*, y tenemos que usar el método *equals*, ya que si no, estaríamos comparando sus direcciones de memoria, en vez de sus atributos *primitivos*, ahora espero que tenga más sentido eso que vimos en el PDF de programación 1 “Strings”, y por qué existía el *equals*, el porque los *Strings* eran Objetos, pero

esto no es todo, vamos a ver un pequeño ejemplo más, de como funcionan las direcciones de memoria:

```
static Vehiculo cPredef = new Vehiculo();
static Vehiculo c0 = null;
static Vehiculo c1 = new Vehiculo(4,280,190,"Ferrari","Blanco",true);
static Vehiculo c2 = new Vehiculo(4,310,150,"Citroen","Azul",false);
static Vehiculo c3 = new Vehiculo(4,275,165,"Toyota","Gris",false);
static Vehiculo c1Copia = new Vehiculo(4,280,190,"Ferrari","Blanco",true);
static Vehiculo c2Copia = c2;
```



Cuando creamos estos Objetos de ésta forma, como veis solo creamos direcciones de memoria en aquellos que tienen un *new*, *c0* no apunta a nada, y *c2Copia* apunta a *c2*, por lo que:

```
System.out.println(c2 + " " + c2Copia);
System.out.println(c1 == c0);
System.out.println(c1 == c1Copia);
System.out.println(c2 == c2Copia);
```

>>>

```
poo.Vehiculo@15db9742  poo.Vehiculo@15db9742
false
false
true
```

*c2* y *c2Copia* comparten dirección de memoria, por lo que al comparar ambas nos da *true*.

Y lo más importante, si cambiamos el atributo de uno, se lo cambiamos al otro:

```
System.out.println(c2.getRuedas() + " " + c2Copia.getRuedas());
c2.setRuedas(3);
System.out.println(c2.getRuedas() + " " + c2Copia.getRuedas());
```

>>>

```
4 4
3 3
```

Por esto mismo funciona el “Paso de Valor por Referencia”, por que una función hace una copia de una variable, y si ponemos parámetro *a* = variable *b*, tanto *a* como *b* tienen la misma dirección de memoria siempre y cuando *a* y *b* sean Objetos, y los *arrays* son Objetos, no estoy muy seguro de como funciona la clase *String* en éste aspecto, ya que en vez de crear un *array* de *chars* para formar una palabra, cada *char* es un Objeto, lo cuál sin estudiar su funcionamiento me hace imposible saber porque a pesar de que un *String* es un objeto, no funciona el paso de valor por referencia, al igual que si ponemos:

```
String str1 = "Hola ";
String str2 = "Hola ";
System.out.println(str1 == str2);
```

>>> true

Aunque he de mencionar, que si ponemos:

```
String str1 = new String("Hola ");
String str2 = new String("Hola ");
System.out.println(str1 == str2);
```

>>> false

(Ésta es otra forma de crear *Strings*), como veis, tiene un funcionamiento muy raro.