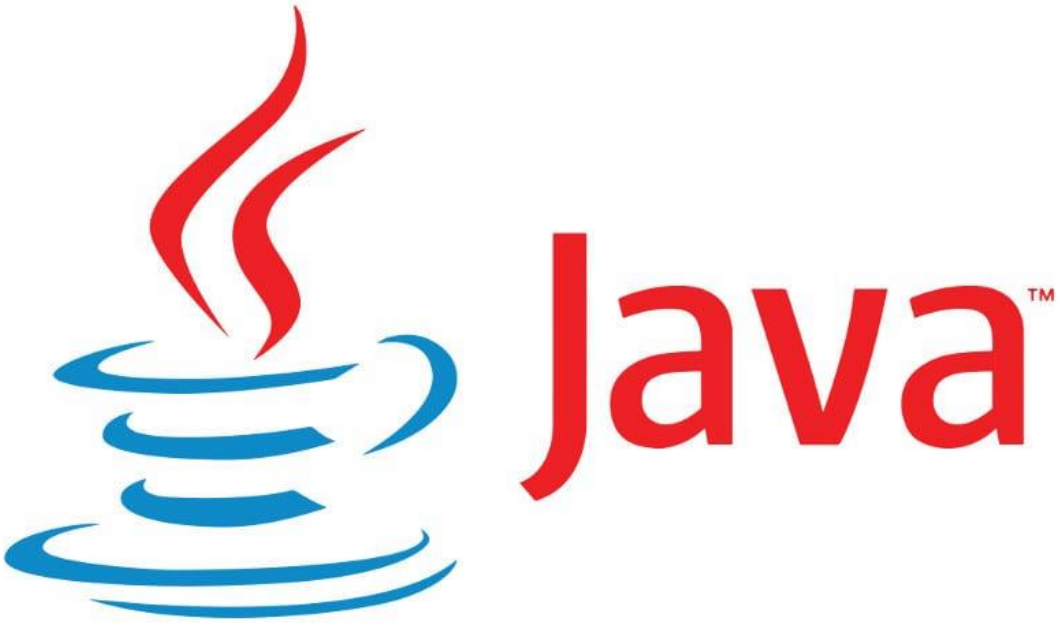
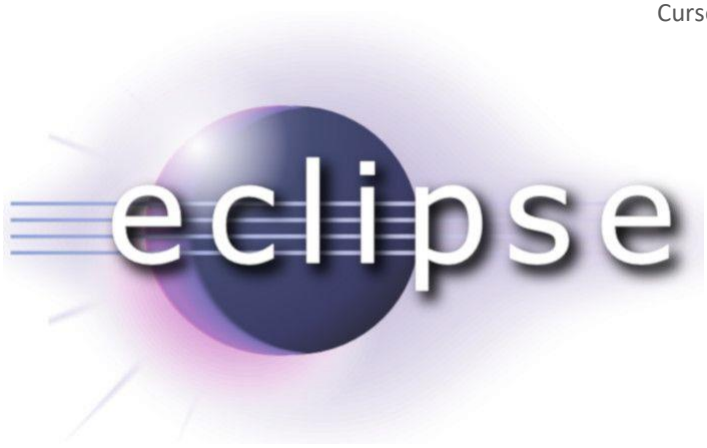


CURSO INICIACIÓN JAVA II

Tutoriales de pildorasinformaticas

Descripción breve

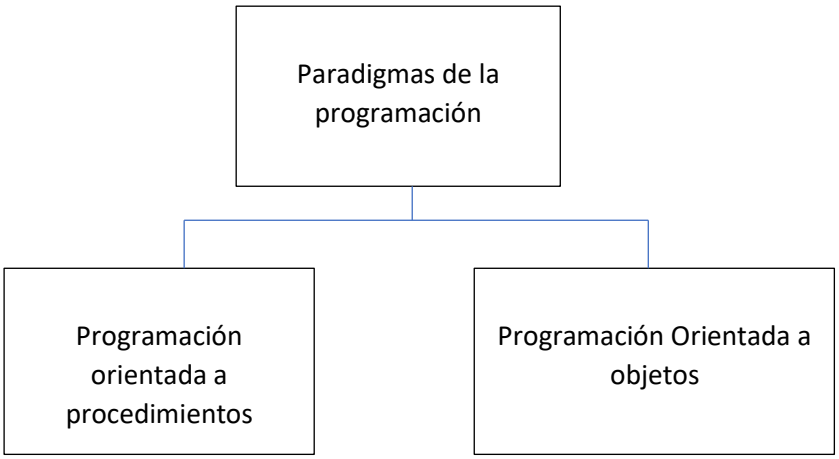
Curso introductorio de Java por pildorasinformaticas.



Pere Manel Verdugo Zamora
pereverdugo@gmail.com

POO I (Vídeo 27)

¿Qué es la programación orientada a objetos?



```
  10 LET X=0
  30 LET Y=X
  40 LET J=1
  45 LET I=X
  50 GO SUB 1000
  55 LET J=ABS (J-1)
  60 LET I=INT (6*RND+1)
  75 IF X=31 OR Y=31 THEN PRINT
"BRAVO ";J
  80 PRINT "JUGADOR ";J+1;"  :";
I:  BEEP 0.2,10+10*J
  85 PAUSE 4E4
  90 GO SUB 1000
 100 LET I=INT (4*RND)
 110 GO SUB I*(RND<.5)*50+150
 120 GO TO 50
 150 RETURN
 200 LET I=-I
 210 RETURN
 250 LET I=I+I
 260 RETURN
 300 IF NOT J THEN LET X=INT (30
*RND)
 310 IF J THEN LET Y=INT (30*RND
)
 320 RETURN
1000 IF NOT J THEN LET X=X+I*(X+
I<=31)
1005 LET Y=Y+J*I*(Y+I<=31)
1010 FOR I=0 TO 31
1020 PRINT AT 3,I;I AND I<=9; PR
INT AT 3,I;CHR$ (55+I) AND I>9
1030 PRINT INK 1;AT 4,I;CHR$ (CO
DE "■"-94*(I=X))
1040 PRINT INK 2;AT 5,I;CHR$ (CO
DE "■"-93*(I=Y))
1050 NEXT I
1060 RETURN
```

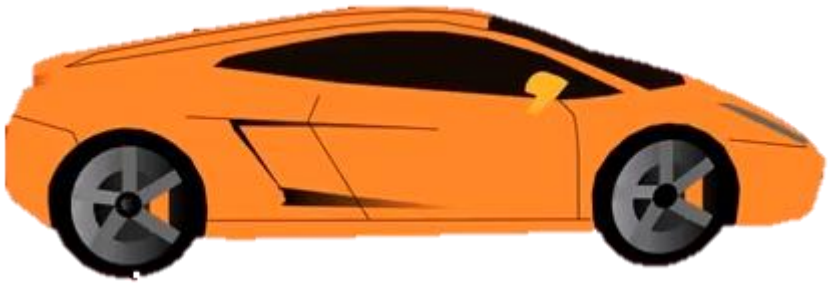

- Programación Orientada a procedimientos:
  - Algunos ejemplos de lenguajes: Fortram, Cobol, Basic, etc.
  - **Desventajas:**
    - Unidades de código muy grandes en aplicaciones complejas.
    - En aplicaciones complejas el código resultaba difícil de descifrar.
    - Poco reutilizable.

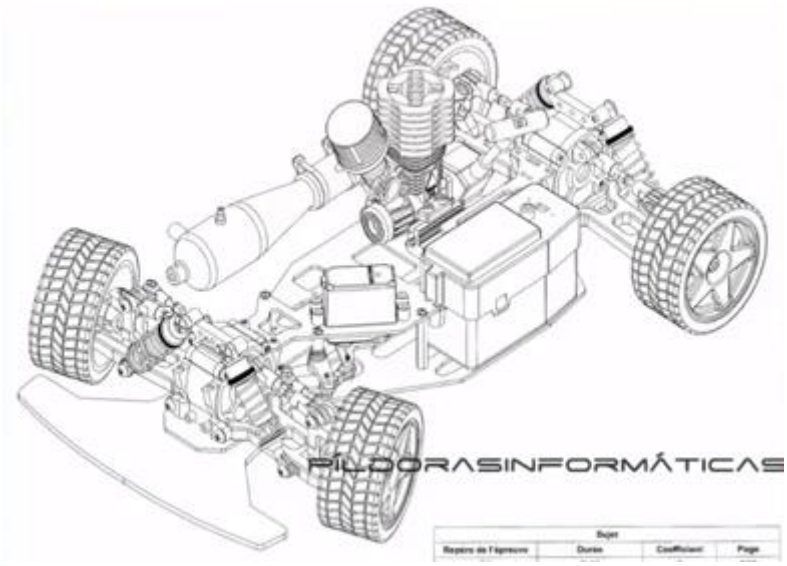
- Si existe fallo en alguna línea del código, es muy probable que el programa caiga.
- Aparición frecuente de código espagueti.
- Difícil de depurar por otros programadores en caso de necesidad o error.

Programación Orientada a objetos

- ¿En qué consiste?
  - Trasladar la naturaleza de los objetos a la vida real al código de programación.
- ¿Cuál es la naturaleza de un objeto en la vida real?
  - Los objetos tienen un estado, un comportamiento (¿Qué puede hacer?), y unas propiedades.
- Pongamos un ejemplo: El objeto coche.
  - ¿Cuál es el estado de un coche? Un coche puede estar parado, circulando, aparcado, etc.
  - ¿Qué propiedades tiene un coche? Un coche tiene un color, un peso, un tamaño, etc.
  - ¿Qué comportamiento tiene un coche? Un coche puede arrancar, frenar, acelerar, girar, etc.

- Programación Orientada a objetos:
  - Algunos ejemplos de lenguajes: C++, Java, Visual.NET, etc.
  - **Ventajas:**
    - Programas divididos en “trozos”, “partes”, “módulos” o “clases”. Modularización.
    - Muy reutilizables. Herencia.
    - Si existe fallo en alguna línea del código, el programa continuará con su funcionamiento. Tratamiento de Excepciones.
    - Encapsulamiento.





Algunos componente de un coche se pueden reutilizar en otros modelos de la misma similitud.

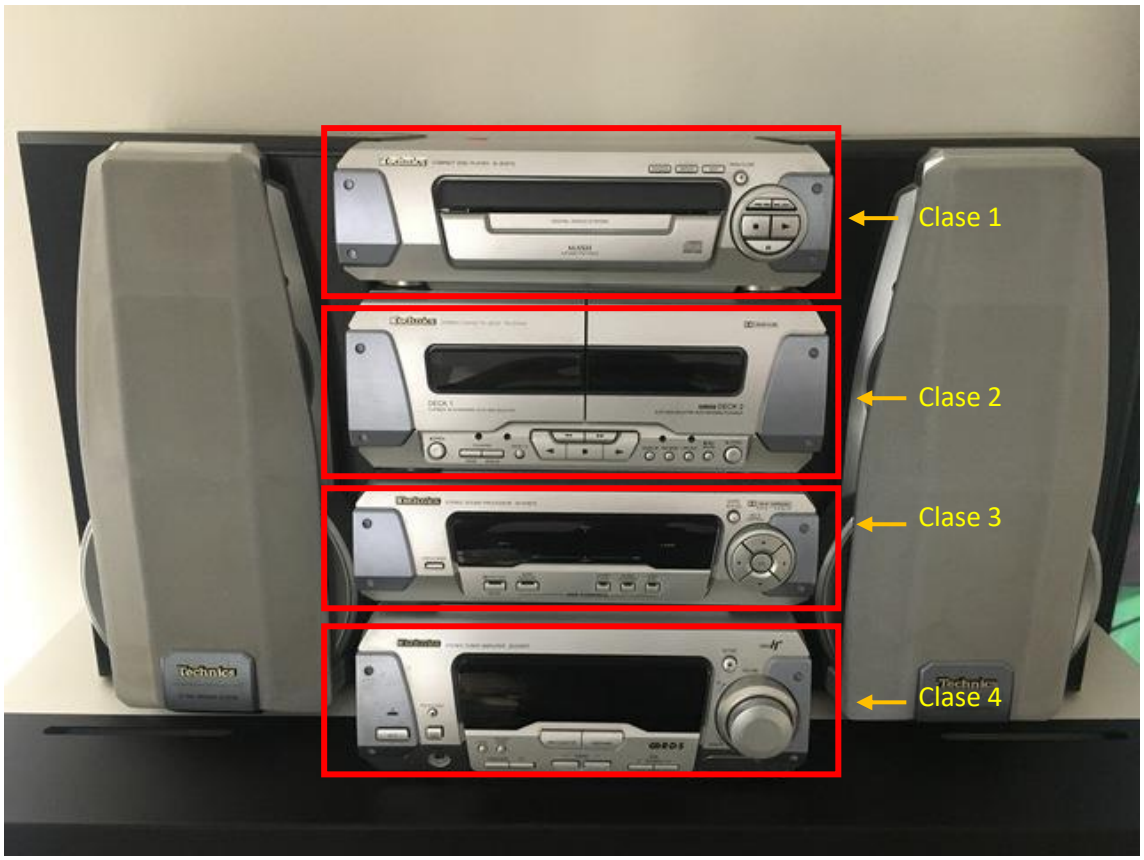
Esto se denomina herencia.

Algunos errores se pueden controlar y esto se llama tratamiento de excepciones.

Encapsulamiento: una parte del programa no tiene que saber como funciona otra parte de él.

P. O. Procedimientos.

Supongamos que en este equipo se me estropea el ecualizado, el problema es que al ser un equipo compacto lo tengo que llevar todo.



En este caso si se nos estropea el CD podemos desconectarlo del equipo y llevarlo a reparación y el resto del equipo está en casa, podremos escuchar discos, la radio, etc.

A este concepto en programación se les denomina Modularización.

La ventaja es que si en un módulo hay un error este módulo no funcione pero el resto del programa sí.

Un programa de Java es un grupo de clases unidas entre si de uno u otra forma para que el programa funciona como una sola unidad.

Vocabulario de la POO

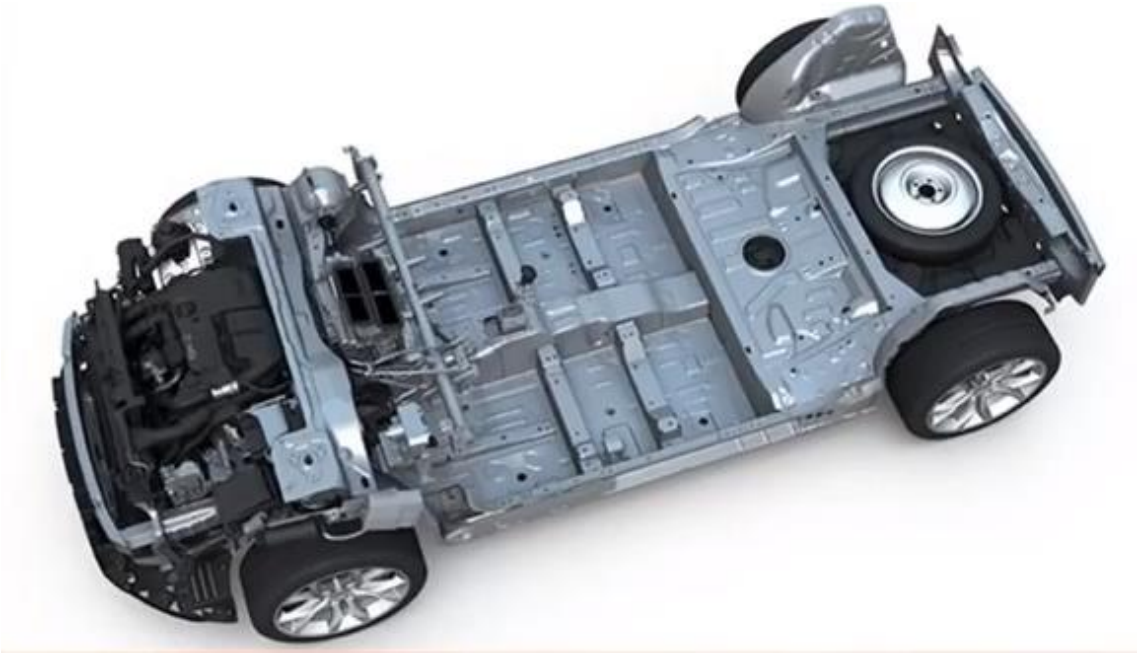
- Clase.
- Objeto.
- Ejemplar de clase. Instancia de clase. Ejemplarizar una clase. Instanciar una clase.
- Modularización.
- Encapsulamiento / encapsulación.
- Herencia.
- Polimorfismo


CURSO JAVA
27
POO
¿QUÉ ES LA PROGRAMACIÓN
ORIENTADA A OBJETOS?
eclipse
Java

POO II. (Vídeo 28)

Clase

- Modelo donde se redactan las características comunes de un grupo de objetos.



Ejemplo de plataforma que muchos modelos de vehículos tienen en común.

Estos dos coches comparten la misma plataforma.

La clase es la que define las características comunes a estos dos objetos.

Estos objetos se construyen a partir de estas clases.

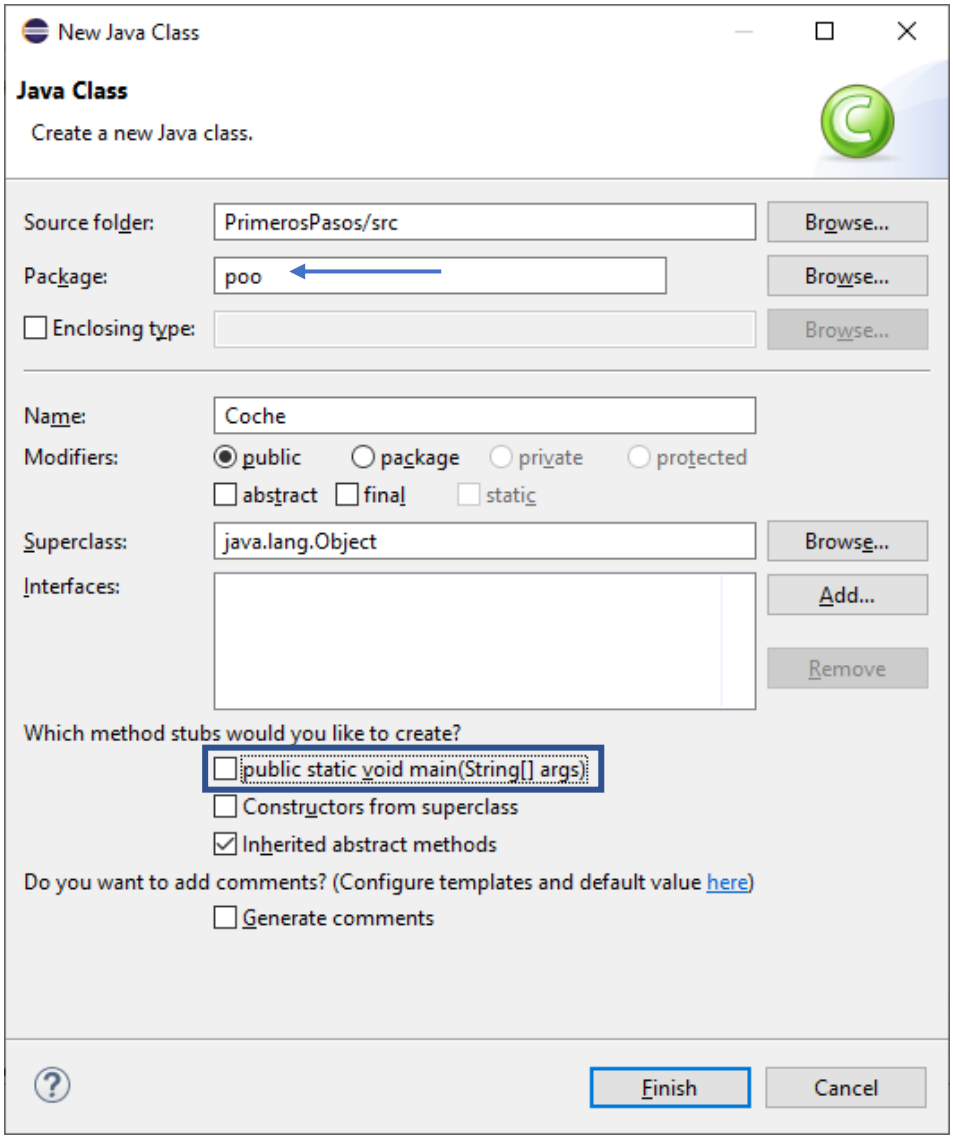
- Objeto:
  - Tiene propiedades (atributos)
    - Color
    - Peso
    - Alto
    - Largo
  - Tienen comportamiento (¿Qué es capaz de hacer?):
    - Arrancar
    - Frenar
    - Girar
    - Acelerar

- Objeto:
    - Accediendo a propiedades de objeto desde código (pseudocódigo):
        - Renault.color=”rojo”;
        - Renault.peso=1500;
        - Renault.ancho=2000;
        - Renault.alto=900;
    - Accediendo al comportamiento de objeto desde código (pseudocódigo):
        - Renault.arranca();
        - Renault.frena();
        - Renault.gira();
        - Renault.acelera();

Nomenglatura del punto: nombre.propiedad=valor;

Método: nombre.método();

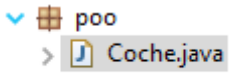
Vamos a Eclipse y creamos una nueva clase llamada Coche.



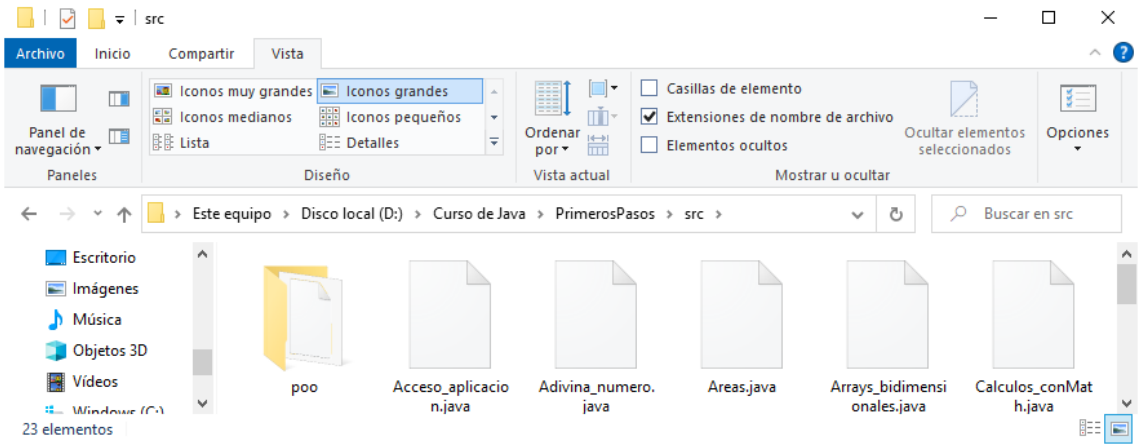
En el apartado Packege ponemos poo, como nombre de la clase Coche y dejamos desactivada la opción public static void main(String[] args).

```java
package poo;

public class Coche {

}

```

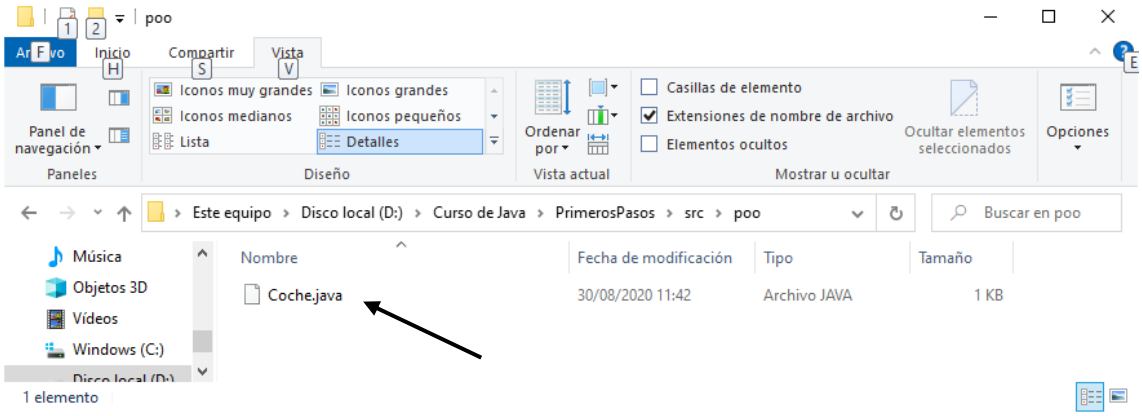
Este será la nueva clase, que está en el paquete poo.



En la carpeta donde guardamos los proyecto de Java, en la carpeta PrimerosPasos y en la carpeta src encontramos la carpeta poo.



Si entramos en ella:



Se encuentra nuestra clase.

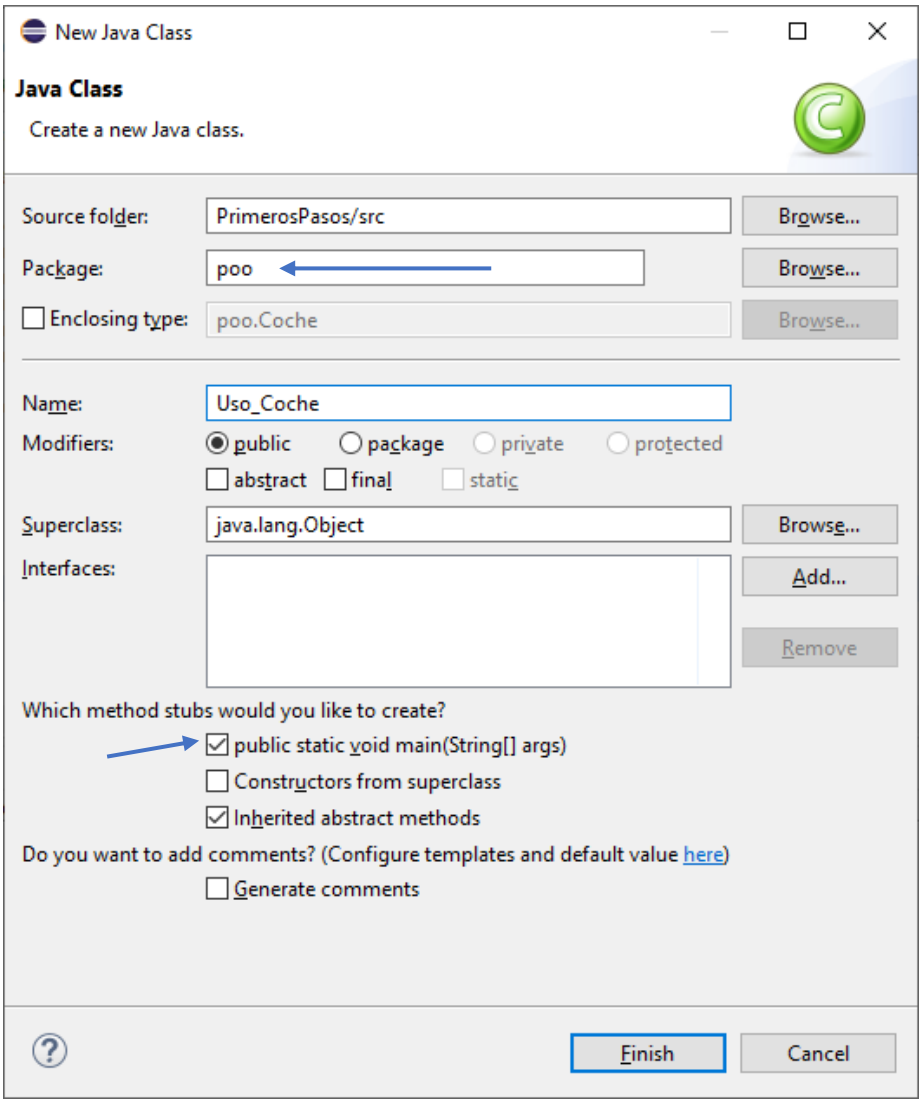
La opción de crear paquetes ni más ni menos consiste en organizar el código en Java.

Método constructor: es un método especial que se encarga de dar un estado inicial a nuestro objeto.

```java
package poo;

public class Coche {
    int ruedas;
    int largo;
    int ancho;
    int motor;
    int peso;

    public Coche() {
        ruedas=4;
        largo=2000;
        ancho=300;
        motor=1600;
        peso=500;
    }
}
```

Guardamos la clase.

Vamos a crear otra clase llamada Uso_Coche.



La guardaremos en el mismo paquete llamado poo y activaremos la casilla public static void main(String[] args).

Coche.java | Uso_Coche.java

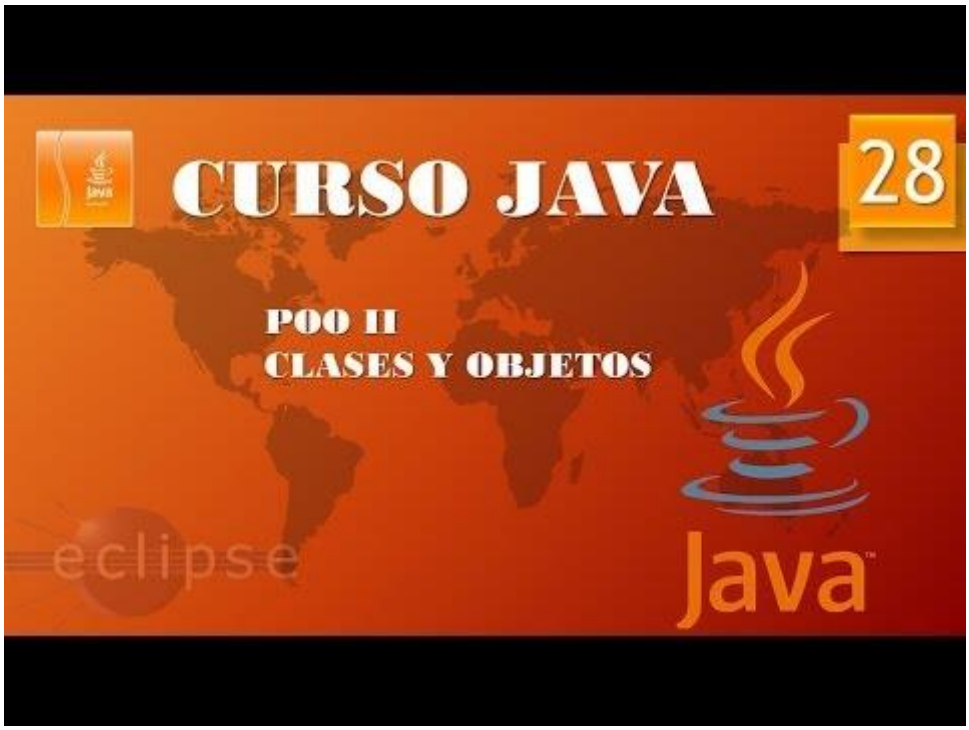
```java
package poo;

public class Uso_Coche {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Podrás observar que en Eclipse podemos tener abierta más de una pestaña.

```java
package poo;

public class Uso_Coche {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Coche Renault= new Coche(); //INSTANCIAR UNA CLASE

        System.out.println("Este coche tiene " + Renault.ruedas + " ruedas.");
    }

}
```

Si ejecutamos este será el resultado:

```
Este coche tiene 4 ruedas.
```

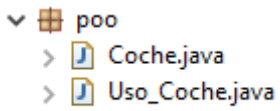

CURSO JAVA
28
POO II
CLASES Y OBJETOS
eclipse
Java

POO III. (Vídeo 29)

Modularización y encapsulación.

Modularización consiste en realizar un programa en trozos o partes.

Ya hemos hecho en programa que consta de dos partes:

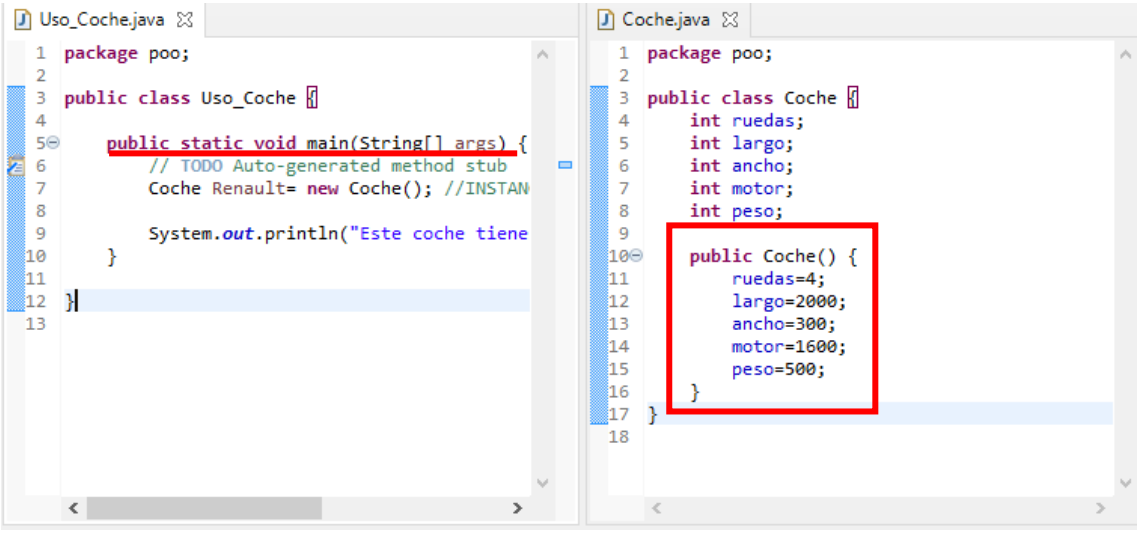


Si eliminamos la clase coche la clase Uso_Coche no funcionaría, diría que no reconoce este tipo de dato.

Cuando hay varias clases hay una que se considera principal y es a que tiene

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

Y es por donde empieza la ejecución del programa.



El operador new llama al constructor del objeto.

```java
package poo;

public class Coche {
    int ruedas;
    int largo;
    int ancho;
    int motor;
    int peso;

    public Coche() {
        ruedas=4;
        largo=2000;
        ancho=300;
        motor=1600;
        peso=500;
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Coche Renault= new Coche(); //INSTANCIAR UNA CLASE

        System.out.println("Este coche tiene " + Renault.ruedas + " ruedas.");
    }
}
```

También es posible en el mismo archivo poner la clase y a continuación el método public static void, de este modo el segundo archivo no sería necesario.

A la hora de modula hay dos alternativas, o bien lo pones todo en un única clase o creas varias clases.

Encapsulación: Hay cosas que solo se pueden hacer solo desde el módulo o la propia clase.

```java
package poo;

public class Uso_Coche {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Coche Renault= new Coche(); //INSTANCIAR UNA CLASE
        Renault.ruedas=3;
        System.out.println("Este coche tiene " + Renault.ruedas + " ruedas.");
    }

}
```

Es este ejemplo le estamos diciendo que el número de ruedas son 3, estamos accediendo a la clase Coches esto es debido al no haber encapsulamiento.

Lo correcto sería que las ruedas solo las pueda modificar desde su propia clase.

```java
package poo;

public class Coche {
    private int ruedas;
    int largo;
    int ancho;
    int motor;
    int peso;

    public Coche() {
        ruedas=4;
        largo=2000;
        ancho=300;
        motor=1600;
        peso=500;
    }

}
```

Al poner private en una variable la estamos encapsulando.

Si intentamos ejecutar la clase donde le decíamos que el coche tiene 3 ruedas este será el mensaje de error que observaremos:

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

	at poo.Uso_Coche.main(Uso_Coche.java:8)
```

```java
package poo;

public class Uso_Coche {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Coche Renault= new Coche(); //INSTANCIAR UNA CLASE
        Renault.ruedas=3;
        System.out.println("Este coche tiene " + Renault.ruedas + " ruedas.");
    }

}
```

La variable ruedas no es visible.

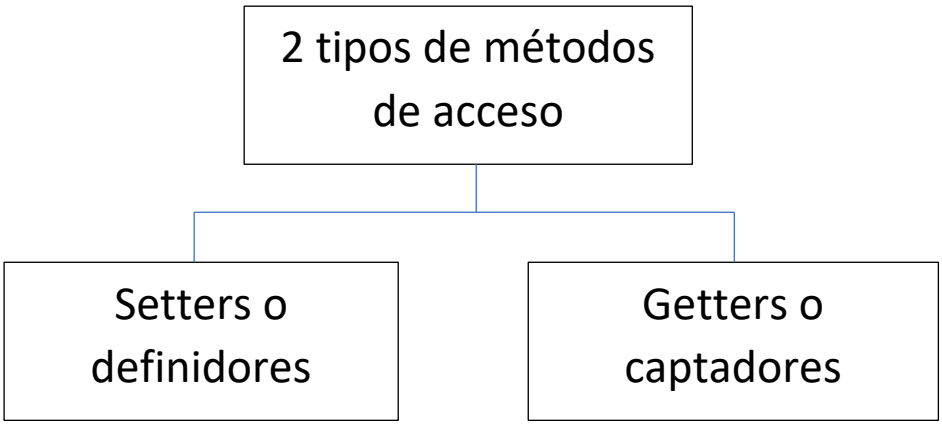
Los métodos van a permitir que nuestras clases interactúen entre sí formando una unidad para que un programa funcione.


CURSO JAVA
29
POO III
MODULARIZACIÓN
ENCAPSULACIÓN
eclipse
Java

POO IV Getters y Setters. (Vídeo 30)

¿Como modificar las propiedades de un objeto?

Método Getters y Setters



Con un Setters definimos la propiedad de un valor y con Getters obtenemos el valor de un objeto.

GETTERS

- Función: Devolver el valor de las propiedades a los objetos.
- Sintaxis: public dato_a_devolver nombre_método(){códio + return}

```
1  package poo;
2
3  public class Coche {
4      private int ruedas;
5      private int largo;
6      private int ancho;
7      private int motor;
8      private int peso;
9
10     public Coche() {
11         ruedas=4;
12         largo=2000;
13         ancho=300;
14         motor=1600;
15         peso=500;
16     }
17
18     public String dime_largo() {
19         return "El largo del coche es " + largo;
20     }
21
22 }
```

Hemos creado un método que nos retornará el valor de largo del coche junto con el texto, ya que está en la línea return.

```java
package poo;

public class Uso_Coche {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Coche Renault= new Coche(); //INSTANCIAR UNA CLASE

        System.out.println(Renault.dime_largo());
    }

}
```

Ahora ya podemos llamar al método getters dime_largo, ya que Renault es de la clase coche y tiene este método.

Si ejecutamos este será el resultado:

```
El largo del coche es 2000
```

SETTERS

- Función: Modificar el valor de las propiedades de los objetos.
- Sintaxis: public void nombre_método(){ código }
    - ¿Qué indica void? Indica que el método no devuelve ningún valor.

```java
package poo;

public class Coche {
    private int ruedas;
    private int largo;
    private int ancho;
    private int motor;
    private int peso_plataforma;
    private String color;
    int peso_total;
    boolean asientos_quero, climatizador;

    public Coche() {
        ruedas=4;
        largo=2000;
        ancho=300;
        motor=1600;
        peso_plataforma=500;
    }

    public String dime_largo() { //GETTER
        return "El largo del coche es " + largo;
    }

    public void establece_color() { //SETTER
        color="azul";
    }

    public String dime_color() {
        return "El color del coche es " + color;
    }

}
```

En la línea 9 hemos definido la variable color, pero no le hemos asignado ningún valor en el constructor.

En la línea 25 realizamos un Setter para asignarle a la variable color el valor azul.

En la línea 29 realizamos Getter para que nos retorne el color del coche.

```
1  package poo;
2
3  public class Uso_Coche {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          Coche Seat=new Coche();
8          System.out.println(Seat.dime_color());
9          Seat.establece_color();
10         System.out.println(Seat.dime_color());
11     }
12
13 }
```

En la clase principal en la línea 7 definimos un objeto llamado Seat de la clase Coche.

En la línea 8 llamaos al método dime_color() de tipo Getter para que nos muestre el color del coche, pero como la variable color no tiene ningún valor definido nos contesta con null.

En la línea 9 llamamos al método establece_color() de tipo Setter donde le asignamos a la variable color el valor azul.

En la línea 10 volvemos a llamar al método dime_color().

Cuando ejecutemos este será el resultado:

```
El color del coche es null
El color del coche es azul
```


CURSO JAVA
30
POO IV
ENCAPSULACIÓN
MODIFICACIÓN PROPIEDADES.
MÉTODOS SETTERS Y GETTERS
eclipse
Java

POO V. Paso de parámetros. (Vídeo 31)

Paso de parámetros

```
1  package poo;
2
3  public class Coche {
4      private int ruedas;
5      private int largo;
6      private int ancho;
7      private int motor;
8      private int peso_plataforma;
9      String color;
10     int peso_total;
11     boolean asientos_quero, climatizador;
12
13     public Coche() {
14         ruedas=4;
15         largo=2000;
16         ancho=300;
17         motor=1600;
18         peso_plataforma=500;
19     }
20
21     public String dime_largo() { //GETTER
22         return "El largo del coche es " + largo;
23     }
24
25     public void establece_color(String color_coche) { //SETTER
26         color=color_coche;
27     }
28
29     public String dime_color() {
30         return "El color del coche es " + color;
31     }
32
33 }
```

En la línea 25 en el método de setter le pasamos un parámetro, para poder cambia a más de un color.

```
1  package poo;
2
3  public class Uso_Coche {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          Coche Seat=new Coche();
8          Seat.establece_color("amarillo");
9          System.out.println(Seat.dime_color());
10         Seat.establece_color("verde");
11         System.out.println(Seat.dime_color());
12     }
13
14 }
```

Llamamos al método setter pero hay que pasarle un parámetro, primero le pasamos el color amarillo y luego verde, este será el resultado:

```
El color del coche es amarillo
El color del coche es verde
```

```java
package poo;

public class Coche {
    private int ruedas;
    private int largo;
    private int ancho;
    private int motor;
    private int peso_plataforma;
    private String color;
    private int peso_total;
    private boolean asientos_cuero, climatizador;

    public Coche() {
        ruedas=4;
        largo=2000;
        ancho=300;
        motor=1600;
        peso_plataforma=500;
    }

    public String dime_datos_generales() { //GETTER
        return "La plataforma del vehículo tiene " + ruedas  + " ruedas. Mide " +
            largo/1000 + " metros con un ancho " + ancho +
            " cm y un peso de plataforma de " + peso_plataforma + " Kg.";
    }

    public void establece_color(String color_coche) { //SETTER
        color=color_coche;
    }

    public String dime_color() {
        return "El color del coche es " + color;
    }

}
```

Vamos a la clase principal

```java
package poo;

public class Uso_Coche {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Coche Seat=new Coche();
        Seat.establece_color("amarillo");
        System.out.println(Seat.dime_color());
        Seat.establece_color("verde");
        System.out.println(Seat.dime_color());

        System.out.println(Seat.dime_datos_generales());
    }

}
```

Llamamos al método dime_datos_generales() para ver toda sus características, este será el resultado:

```
La plataforma del vehículo tiene 4 ruedas. Mide 2 metros con un ancho 300 cm y un peso de plataforma de 500 Kg.
```

```java
public void configura_asientos(String asientos_cuero) { //SETTER
    if(asientos_cuero=="si") {
        this.asientos_cuero=true;
    }else {
        this.asientos_cuero=false;
    }

}
public String dime_asientos() { //GETTER
    if(asientos_cuero==true) {
        return "El coche tiene asientos de cuero";
    }else {
        return "El coche no tiene asientosde de serie";
    }
}
```

Hemos creado los correspondientes métodos.

```
15          Seat.configura_asientos("si");
16          System.out.println(Seat.dime_asientos());
17
18          Seat.configura_asientos("no");
19          System.out.println(Seat.dime_asientos());
```

En la línea 15 le pasamos al parámetro configura_asientos “si”, en la línea 16 consultamos el parámetro dime_asientos().

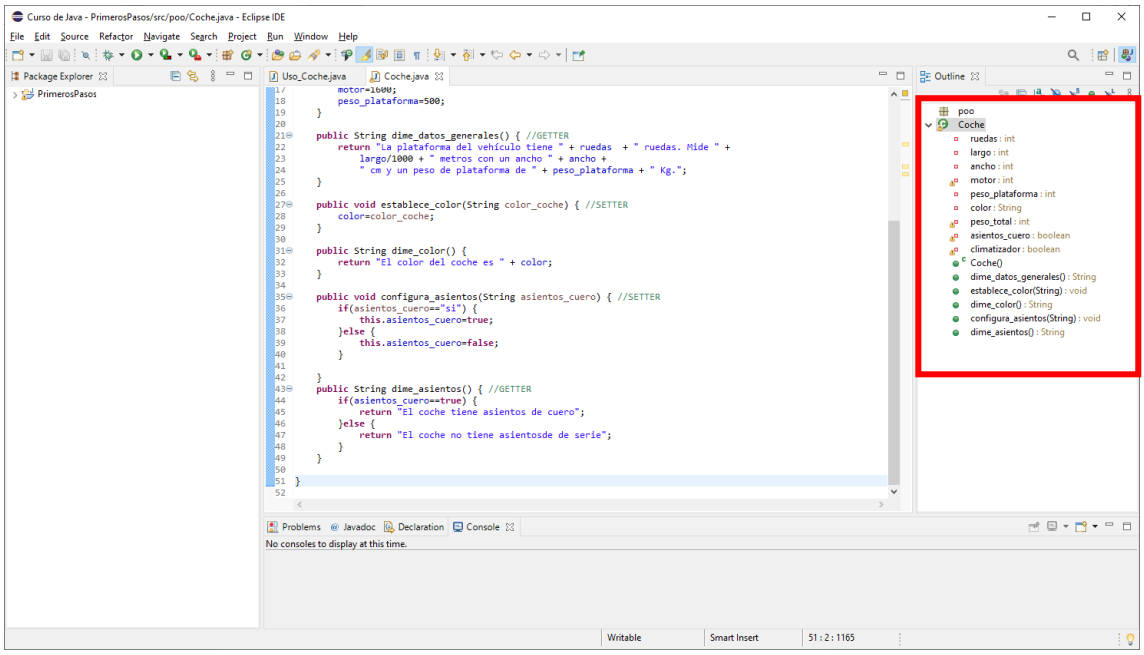
En la línea 18 le pasamos al parámetro configura_asientos “no”, en la línea 19 consultamos el parámetro dime_asientos().

Este será el resultado:

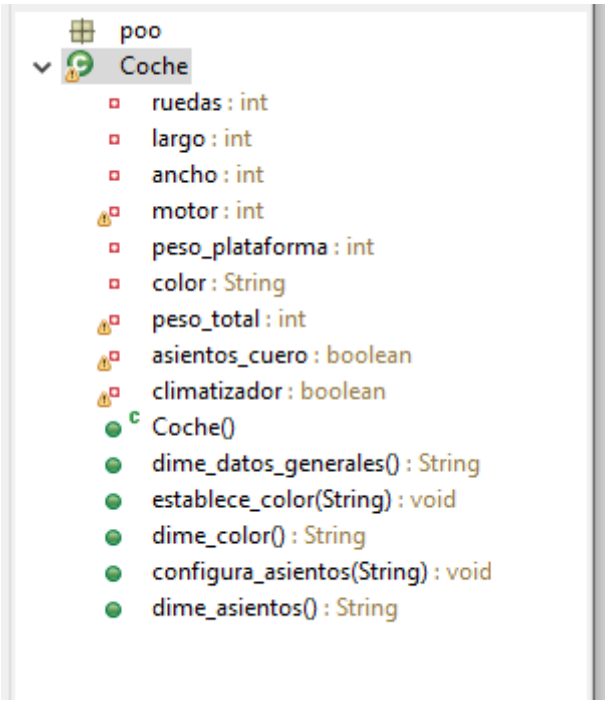
```
El coche tiene asientos de cuero
El coche no tiene asientosde de serie
```


CURSO JAVA
31
POO V
ENCAPSULACIÓN
MÉTODOS SETTERS Y GETTERS II
PASO DE PARÁMETROS
eclipse
Java

POO VI. Construcción objetos (Vídeo 32)



Cada vez los programas se hacen más largos, nos podremos desplazar por la barra de herramientas, hay un panel llamado Outline que nos ayudará a desplazarnos por todo el programa y ver información de los métodos y variables.



Vamos a continuar con el proyecto anterior.

```
52      public void configura_climatizador(String climatizador){ //SETTER
53          if(climatizador =="si") {
54              this.climatizador=true;
55          }else {
56              this.climatizador=false;
57          }
58      }
59
60      public String dime_climatizador() { //GETTER
61          if(climatizador==true) {
62              return "El coche incorpora climatizador";
63          }else {
64              return "El coche llevas aire acondicionado";
65          }
66      }
```

En la clase Coche vamos a crear dos métodos más, para controlar si el coche lleva climatizador o no.

El siguiente método hace de SETTER y GETTER.

```
68      public String dime_peso_coche() { // SETTER Y GETTER
69          int peso_carroceria=500;
70          peso_total=peso_plataforma + peso_carroceria;
71          if(asientos_cuero==true) {
72              peso_total=peso_total + 50;
73          }
74          if(climatizador==true) {
75              peso_total=peso_total + 20;
76          }
77          return "El peso del coche es " + peso_total;
78      }
```

Este método que hace Setter y Getter no es muy frecuente.

No dirá el peso del coche según los tipos de accesorios que lleva.

```
80      public int percio_coche() {
81          int precio_final=10000;
82          if(asientos_cuero==true) {
83              precio_final+=2000 ;
84          }
85          if(climatizador==true) {
86              precio_final+=1500 ;
87          }
88          return  precio_final;
89      }
```

Este método Setter devuelve el precio final del coche según los accesorios que lleva.

Ahora nos vamos a la clase principal Uso_Coche.

```
21          Seat.configura_climatizador("si");
22          Seat.configura_asientos("si");
23
24          System.out.println(Seat.dime_peso_coche()+ " Kgrs.");
25          System.out.println("El precio del coche es de " + Seat.percio_coche() + " €.");
```

Llamamos a los métodos diciéndole que tenemos asientos de cuero y climatizador, este será el resultado:

```
El peso del coche es 1070 Kgrs.
El precio del coche es de 13500 €.
```

```
21          Seat.configura_climatizador("no");
22          Seat.configura_asientos("no");
23
24          System.out.println(Seat.dime_peso_coche()+ " Kgrs.");
25          System.out.println("El precio del coche es de " + Seat.percio_coche() + " €.");
```

Ahora le decimos que no tenemos climatizador solo aire acondicionado y los asientos no son de piel, este será el resultado:

```
El peso del coche es 1000 Kgrs.
El precio del coche es de 10000 €.
```

Resumen del proyecto.

Código de la clase Coches, donde está definida la clase Coches, el constructor y los métodos.

```
package poo;

public class Coche {
	private int ruedas;
	private int largo;
	private int ancho;
	private int motor;
	private int peso_plataforma;
	private String color;
	private int peso_total;
	private boolean asientos_cuero, climatizador;


	public Coche() {
		ruedas=4;
		largo=2000;
		ancho=300;
		motor=1600;
		peso_plataforma=500;
	}

	public String dime_datos_generales() { //GETTER
		return "----------------------------------------\n"
				+ "La plataforma del vehículo tiene " + ruedas  + "
ruedas.\nMide " +
			largo/1000 + " metros. \nCon un ancho " + ancho +
			" cm. \ny un peso de plataforma de " + peso_plataforma + "
Kg.\n"
					+ "----------------------------------------";
	}

	public void establece_color(String color_coche) { //SETTER
		color=color_coche;
	}

	public String dime_color() {
		return "El color del coche es " + color;
	}

	public void configura_asientos(String asientos_cuero) { //SETTER
		if(asientos_cuero=="si") {
```

```
                this.asientos_cuero=true;
            }else {
                this.asientos_cuero=false;
            }

        }
        public String dime_asientos() { //GETTER
            if(asientos_cuero==true) {
                return "El coche tiene asientos de cuero";
            }else {
                return "El coche no tiene asientosde de serie";
            }
        }

        public void configura_climatizador(String climatizador){ //SETTER
            if(climatizador =="si") {
                this.climatizador=true;
            }else {
                this.climatizador=false;
            }
        }

        public String dime_climatizador() { //GETTER
            if(climatizador==true) {
                return "El coche incorpora climatizador";
            }else {
                return "El coche llevas aire acondicionado";
            }
        }

        public String dime_peso_coche() { // SETTER Y GETTER
            int peso_carroceria=500;
            peso_total=peso_plataforma + peso_carroceria;
            if(asientos_cuero==true) {
                peso_total=peso_total + 50;
            }
            if(climatizador==true) {
                peso_total=peso_total + 20;
            }
            return "El peso del coche es " + peso_total;
        }

        public int percio_coche() {
            int precio_final=10000;
            if(asientos_cuero==true) {
                precio_final+=2000 ;
            }
            if(climatizador==true) {
                precio_final+=1500 ;
            }
            return  precio_final;
        }


}
```

El código de la clase Uso_Coche que es la clase principal.

```
package poo;

public class Uso_Coche {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Coche Seat=new Coche();
		Seat.establece_color("verde");
		System.out.println(Seat.dime_color());
		System.out.println(Seat.dime_datos_generales());
		Seat.configura_climatizador("si");
		Seat.configura_asientos("no");
		System.out.println(Seat.dime_asientos());
		System.out.println(Seat.dime_peso_coche()+ " Kgrs.");
		System.out.println("El precio del coche es de " +
Seat.percio_coche() + " €.");

	}
}
```

Si ejecutamos este será el resultado:

```
El color del coche es verde
----------------------------------------
La plataforma del vehículo tiene 4 ruedas.
Mide 2 metros.
Con un ancho 300 cm.
y un peso de plataforma de 500 Kg.
----------------------------------------
El coche no tiene asientosde de serie
El coche no tiene asientosde de serie
El peso del coche es 1020 Kgrs.
El precio del coche es de 11500 €.
```

Ahora para finalizar queremos que el color del coche, si tiene climatizador y si tiene asientos de piel nos lo pregunte en una ventana de diálogo y según la respuesta introducida por teclado nos de la información.

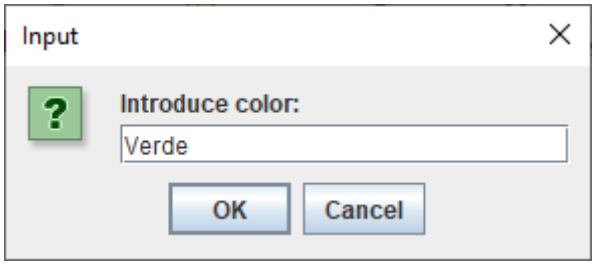
```
34      public String dime_color() {
35          return "El color del coche es " + color;
36      }
37
38      public void configura_asientos(String asientos_cuero) { //SETTER
39          if(asientos_cuero.equalsIgnoreCase("si")) {
40              this.asientos_cuero=true;
41          }else {
42              this.asientos_cuero=false;
43          }
44
45      }
46      public String dime_asientos() { //GETTER
47          if(asientos_cuero==true) {
48              return "El coche tiene asientos de cuero";
49          }else {
50              return "El coche tiene asientos de de serie";
51          }
52      }
53
54      public void configura_climatizador(String climatizador){ //SETTER
55          if(climatizador.equalsIgnoreCase("si")) {
56              this.climatizador=true;
57          }else {
58              this.climatizador=false;
59          }
60      }
```

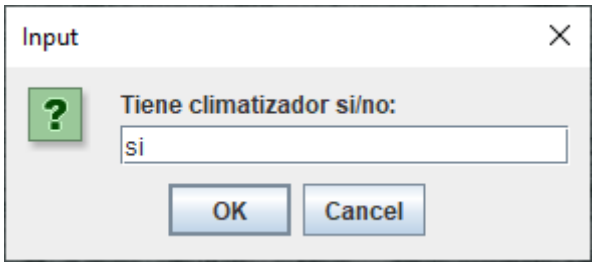
En la clase coche cambiaremos la comparación.

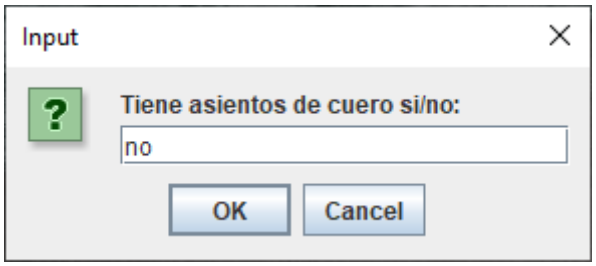
```
1  package poo;
2
3  import javax.swing.*;
4
5  public class Uso_Coche {
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9          Coche Seat=new Coche();
10
11
12
13         Seat.establece_color(JOptionPane.showInputDialog("Introduce color: "));
14         System.out.println(Seat.dime_color());
15         System.out.println(Seat.dime_datos_generales());
16
17         Seat.configura_climatizador(JOptionPane.showInputDialog("Tiene climatizador si/no: "));
18
19         Seat.configura_asientos(JOptionPane.showInputDialog("Tiene asientos de cuero si/no: "));
20
21         System.out.println(Seat.dime_climatizador());
22         System.out.println(Seat.dime_asientos());
23         System.out.println(Seat.dime_peso_coche()+ " Kgrs.");
24         System.out.println("El precio del coche es de " + Seat.percio_coche() + " €.");
25
26     }
```

El parámetro del método lo introduciremos por una ventana de diálogo.

Vamos a ejecutar:







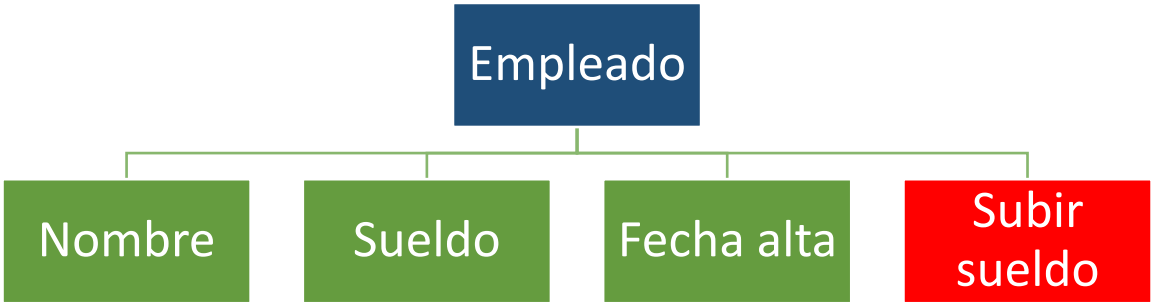

```
El color del coche es Verde
----------------------------------------
La plataforma del vehículo tiene 4 ruedas.
Mide 2 metros.
Con un ancho 300 cm.
y un peso de plataforma de 500 Kg.
----------------------------------------
El coche incorpora climatizador
El coche tiene asientos de de serie
El peso del coche es 1020 Kgrs.
El precio del coche es de 11500 €.
```

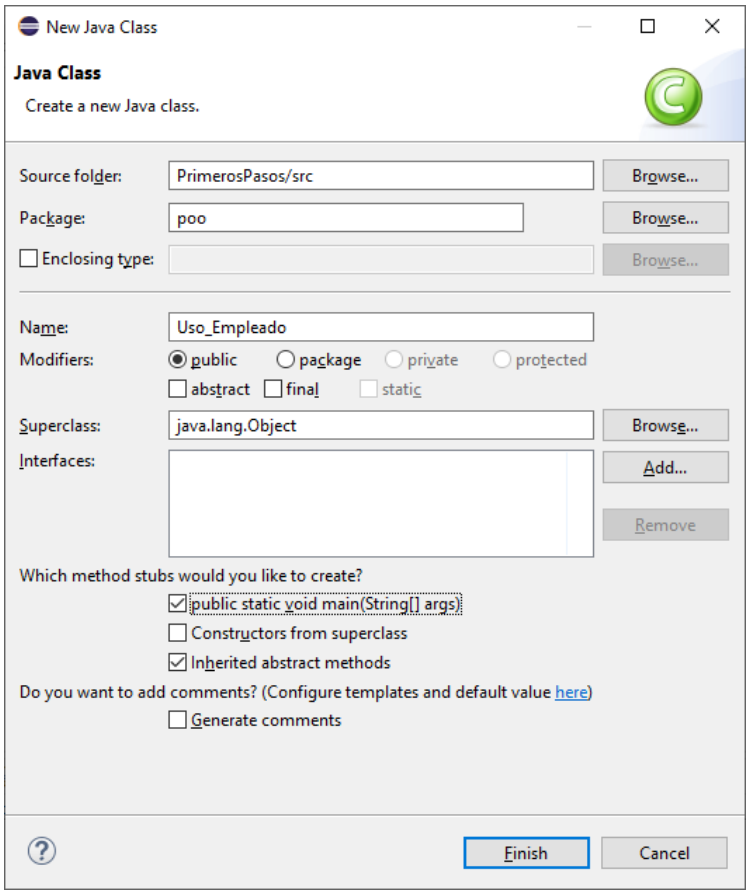

CURSO JAVA
32
POO VI
CONSTRUCCIÓN DE OBJETOS
eclipse
Java

POO VII. Construcción objetos II. (Vídeo 33)

Ficheros fuentes. Constructores.

Objeto empleado.





Vamos a crear una nueva clase llamada Uso_Empleado dentro del paquete poo, activando public static void main(String[] args).

```java
package poo;

import java.util.*;

public class Uso_Empleado {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}

class Empleado{

	public Empleado(String nom, double sue, int agno, int mes, int dia) {


	}

	private String nombre;
	private double sueldo;
	private Date altaContrato;

}
```

Java POO VIII. Construcción objetos III. (Vídeo 34)

```
package poo;

import java.util.*;

public class Uso_Empleado {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}

class Empleado{

	public Empleado(String nom, double sue, int agno, int mes, int dia) {

		nombre=nom;
		sueldo=sue;
		GregorianCalendar calendario= new GregorianCalendar(agno, mes-1,
dia);
		altaContrato=calendario.getTime();
	}

	public String dameNombre() { //getter
		return nombre;
	}

	public double dameSueldo() //getter
	{
		return sueldo;
	}

	public Date dameFechaContrato() { //getter
		return altaContrato;
	}

	public void subeSueldo(double porcentaje) { //setter
		double aumento=sueldo*porcentaje/100;
		sueldo+=aumento;
	}

	private String nombre;
	private double sueldo;
	private Date altaContrato;

}
```


CURSO JAVA
34
POO VIII
CONSTRUCCIÓN DE OBJETOS III
FICHEROS FUENTE. CONSTRUCTORES
eclipse
Java

POO IX. Construcción objetos IV. (Vídeo 35)

```
3  import java.util.*;
4
5  public class Uso_Empleado {
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9
10         Empleado empleado1=new Empleado("Paco Gómez", 85000, 1990, 12, 17);
11         Empleado empleado2=new Empleado("Ana López", 95000, 1995, 6, 2);
12         Empleado empleado3=new Empleado("Maria Martín", 105000, 2002, 3, 15);
13
14         empleado1.subeSueldo(5);
15         empleado2.subeSueldo(5);
16         empleado3.subeSueldo(5);
17
18         System.out.println("Nombre: " + empleado1.dameNombre() + " sueldo: " + empleado1.dameSueldo()
19                 + "Fecha de alta: "+ empleado1.dameFechaContrato());
20
21         System.out.println("Nombre: " + empleado2.dameNombre() + " sueldo: " + empleado2.dameSueldo()
22         + "Fecha de alta: "+ empleado2.dameFechaContrato());
23
24         System.out.println("Nombre: " + empleado3.dameNombre() + " sueldo: " + empleado3.dameSueldo()
25         + "Fecha de alta: "+ empleado3.dameFechaContrato());
26
27     }
28
29 }
```

En las líneas 10, 11 y 12 definimos tres variables de tipo Empleado pasándole los parámetros Nombre, sueldo, año, mes y día.
En las líneas 14,15 y 16 incrementamos el sueldo un 5%,
Desde la línea 18 hasta 25 imprimimos por consola el estado de los empleados.
Este será el resultado:

```
Nombre: Paco Gómez sueldo: 89250.0Fecha de alta: Mon Dec 17 00:00:00 CET 1990
Nombre: Ana López sueldo: 99750.0Fecha de alta: Fri Jun 02 00:00:00 CEST 1995
Nombre: Maria Martín sueldo: 110250.0Fecha de alta: Fri Mar 15 00:00:00 CET 2002
```

Ahora vamos a modificar el código para simplificarlo utilizando arrays y bucles for.

```
3  import java.util.*;
4
5  public class Uso_Empleado {
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9
10     Empleado[] misEmpleados= new Empleado[3];
11     misEmpleados[0]=new Empleado("Paco Gómez", 85000, 1990, 12, 17);
12     misEmpleados[1]=new Empleado("Ana López", 95000,1995 ,6 ,2 );
13     misEmpleados[2]=new Empleado("María Martín",105000 ,2002 ,3 , 15);
14
15     for(int i=0; i<3; i++) {
16         misEmpleados[i].subeSueldo(5);
17     }
18
19     for(int i=0; i<3; i++) {
20         System.out.println("Nombre: " + misEmpleados[i].dameNombre() +
21                 " Sueldo: " + misEmpleados[i].dameSueldo() +
22             " Fecha de alta :" + misEmpleados[i].dameFechaContrato());
23     }
24
25     }
26
27 }
```

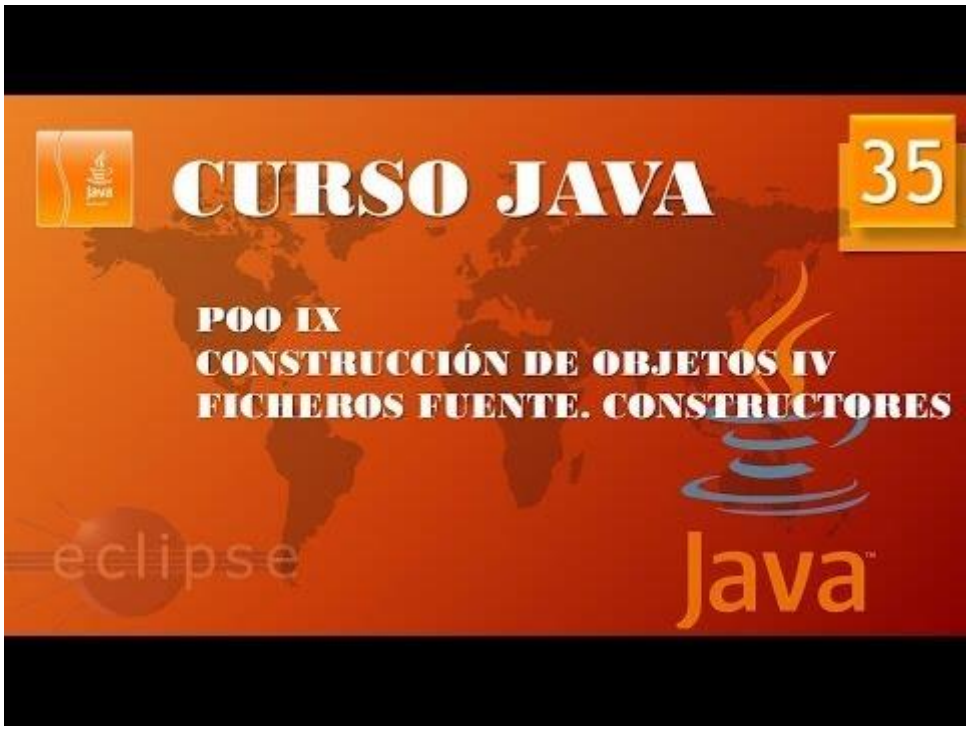
Ahora vamos ha realizar un bucle for mejorado.

```
3  import java.util.*;
4
5  public class Uso_Empleado {
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9
10     Empleado[] misEmpleados= new Empleado[3];
11     misEmpleados[0]=new Empleado("Paco Gómez", 85000, 1990, 12, 17);
12     misEmpleados[1]=new Empleado("Ana López", 95000,1995 ,6 ,2 );
13     misEmpleados[2]=new Empleado("María Martín",105000 ,2002 ,3 , 15);
14
15     for(Empleado i: misEmpleados) {
16         i.subeSueldo(5);
17     }
18
19     for(Empleado i: misEmpleados) {
20         System.out.println("Nombre: " + i.dameNombre() +
21                 " Sueldo: " + i.dameSueldo() +
22             " Fecha de alta :" + i.dameFechaContrato());
23     }
24
25     }
26
27 }
```

Como podrás observar el código se simplifica.

Este será el resultado:

```
Nombre: Paco Gómez Sueldo: 89250.0 Fecha de alta :Mon Dec 17 00:00:00 CET 1990
Nombre: Ana López Sueldo: 99750.0 Fecha de alta :Fri Jun 02 00:00:00 CEST 1995
Nombre: María Martín Sueldo: 110250.0 Fecha de alta :Fri Mar 15 00:00:00 CET 2002
```



Antes de seguir con el siguiente video veo interesante hacer una practica con los siguientes conceptos.

Vamos a crear una clase llamada Empleados con los siguientes campos:

Nombre, Departamento, Categoría, Sueldo

Vamos a crear una array de tipo empleado para almacenar a 10 empleados que tienen que ser de los departamentos Administración, Dirección, Logística, Producción.

Realiza los pasos necesarios par que el sueldo de los que trabajan en Administración suba un 10% y los de Logística un 5%.

```
package poo2;

public class Sueldos {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Empleat[] misEmpleats = new Empleat[10];

		misEmpleats[0]=new Empleat("Paco Martínez", "Administración",
"Oficial", 25000);
		misEmpleats[1]=new Empleat("María Perez", "Dirección",
"Secretaria", 32000);
		misEmpleats[2]=new Empleat("Joaquín Vico", "Logística", "Peon",
15000);
		misEmpleats[3]=new Empleat("Luis Coll", "Administración",
"Contable", 42000);
		misEmpleats[4]=new Empleat("Jorge Cesar", "Logística",
"Encargado", 25000);
		misEmpleats[5]=new Empleat("Piedad Fernández", "Producción",
"Operaria",12000);
```

```
            misEmpleats[6]=new Empleat("Pere Manel", "Administración", "Aux.
Informáica", 18000);
            misEmpleats[7]=new Empleat("Carlos Pons", "Logistica", "Chofer",
22000);
            misEmpleats[8]=new Empleat("Jaime Guitierres", "Dirección",
"Director General", 55000);
            misEmpleats[9]=new Empleat("Cayetano Soriano", "Administración",
"Conserge", 13500);

            for(Empleat e: misEmpleats) {
                  if(e.dameDepartamento1()=="Administración") {
                        e.subeSueldo1(10);
                  }
                  if(e.dameDepartamento1()=="Logística") {
                        e.subeSueldo1(5);
                  }

            }

            for(Empleat e: misEmpleats) {

                  System.out.println("El empleado " + e.dameNombre1() + "
Del departamento: " +
                              e.dameDepartamento1() + " Con categoría de "
+ e.dameCategoria1() +
                              " Tendrá un sueldo de " + e.dameSueldo1());
            }


      }
      static class Empleat{
            public Empleat(String no, String de, String ca, double su) {
                  nombre=no;
                  departamento=de;
                  categoria=ca;
                  sueldo=su;
            }

      private String nombre;
      private String departamento;
      private String categoria;
      private double sueldo;

      public String dameNombre1() {
            return nombre;
      }

      public String dameDepartamento1() {
            return departamento;
      }

      public String dameCategoria1() {
            return categoria;
      }

      public double dameSueldo1() {
            return sueldo;
      }
```

```
	public void subeSueldo1(double porcentaje) {
		double aumentosubida=sueldo*porcentaje/100;
		sueldo+=aumentosubida;
	}
}
}
```

Este será el resultado:

```
El empleado Paco Martínez Del departamento: Administración Con categoría de Oficial Tendrá un sueldo de 27500.0
El empleado María Perez Del departamento: Dirección Con categoría de Secretaria Tendrá un sueldo de 32000.0
El empleado Joaquín Vico Del departamento: Logística Con categoría de Peon Tendrá un sueldo de 15750.0
El empleado Luis Coll Del departamento: Administración Con categoría de Contable Tendrá un sueldo de 46200.0
El empleado Jorge Cesar Del departamento: Logística Con categoría de Encargado Tendrá un sueldo de 26250.0
El empleado Piedad Fernández Del departamento: Producción Con categoría de Operaria Tendrá un sueldo de 12000.0
El empleado Pere Manel Del departamento: Administración Con categoría de Aux. Informáica Tendrá un sueldo de 19800.0
El empleado Carlos Pons Del departamento: Logistica Con categoría de Chofer Tendrá un sueldo de 22000.0
El empleado Jaime Guitierres Del departamento: Dirección Con categoría de Director General Tendrá un sueldo de 55000.0
El empleado Cayetano Soriano Del departamento: Administración Con categoría de Conserge Tendrá un sueldo de 14850.0
```

Constantes Uso final (Vídeo 36)

Constantes uso “Final”

Vamos a crear un clase llamada Pruebas.

```java
package poo;

public class Prueba {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Empleados trabajador1  = new Empleados("Paco");
        Empleados trabajador2  = new Empleados("Ana");

        System.out.println(trabajador1.devuelveDatos());
        System.out.println(trabajador2.devuelveDatos());
    }

}

class Empleados{

    public Empleados(String nom) {

        nombre=nom;
        seccion="Administración";

    }

    public void cambiaSeccion(String seccion) { // Setter
        this.seccion=seccion;
    }

    public String devuelveDatos() {

        return "El nombre es: " + nombre + " y la sección es " + seccion ;
    }

    private String nombre;
    private String seccion;

}
```

Al ejecutar este será el resultado:

```
El nombre es: Paco y la sección es Administración
El nombre es: Ana y la sección es Administración
```

```java
package poo;

public class Prueba {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Empleados trabajador1  = new Empleados("Paco");
        Empleados trabajador2  = new Empleados("Ana");

        trabajador1.cambiaSeccion("RRHH");

        System.out.println(trabajador1.devuelveDatos());
        System.out.println(trabajador2.devuelveDatos());
    }

}

class Empleados{

    public Empleados(String nom) {

        nombre=nom;
        seccion="Administración";

    }

    public void cambiaSeccion(String seccion) { // Setter
        this.seccion=seccion;
    }

    public String devuelveDatos() {

        return "El nombre es: " + nombre + " y la sección es " + seccion ;
    }

    private String nombre;
    private String seccion;

}
```

Este será el resultado:

```
El nombre es: Paco y la sección es RRHH
El nombre es: Ana y la sección es Administración
```

Paco ha cambiado de sección.

Ahora vamos a realizar un método para cambiar el nombre del empleado, como podréis comprender en nombre de usuario no se tendría que cambiar.

```java
package poo;

public class Prueba {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Empleados trabajador1  = new Empleados("Paco");
        Empleados trabajador2  = new Empleados("Ana");

        trabajador1.cambiaSeccion("RRHH");

        trabajador1.cambiarNombre("Maria");

        System.out.println(trabajador1.devuelveDatos());
        System.out.println(trabajador2.devuelveDatos());
    }

}

class Empleados{

    public Empleados(String nom) {

        nombre=nom;
        seccion="Administración";

    }
    public void cambiarNombre(String nombre) {
        this.nombre=nombre;
    }

    public void cambiaSeccion(String seccion) { // Setter
        this.seccion=seccion;
    }

    public String devuelveDatos() {

        return "El nombre es: " + nombre + " y la sección es " + seccion ; // Getter
    }

    private String nombre;
    private String seccion;

}
```

Para ello en la línea 41 vamos a agregar el parámetro final.

```java
package poo;

public class Prueba {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Empleados trabajador1  = new Empleados("Paco");
        Empleados trabajador2  = new Empleados("Ana");

        trabajador1.cambiaSeccion("RRHH");

        trabajador1.cambiarNombre("Maria");

        System.out.println(trabajador1.devuelveDatos());
        System.out.println(trabajador2.devuelveDatos());
    }

}

class Empleados{

    public Empleados(String nom) {

        nombre=nom;
        seccion="Administración";

    }
    public void cambiarNombre(String nombre) {
        this.nombre=nombre;
    }

    public void cambiaSeccion(String seccion) { // Setter
        this.seccion=seccion;
    }

    public String devuelveDatos() {

        return "El nombre es: " + nombre + " y la sección es " + seccion ; // Getter
    }

    private final String nombre;
    private String seccion;

}
```

Ahora en la línea 29 nos avisa de un error, nos dice que no podemos cambiar el nombre, de este modo podemos evitar errores que podamos realizar cambiando el valor de una variable que no sería correcto.

Para poder ejecutar el programa sería necesario eliminar lo que está seleccionado.


CURSO JAVA
36
CONSTANTES. USO "FINAL"
eclipse
Java

Uso static. (Vídeo 37)

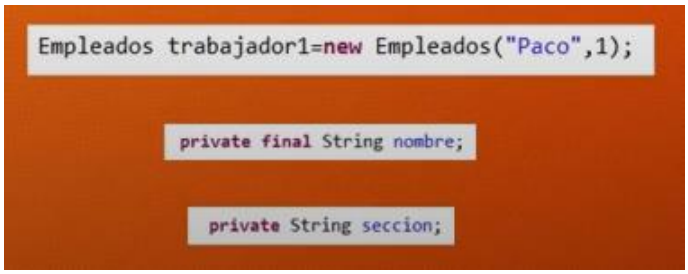
Uso de la palabra static.

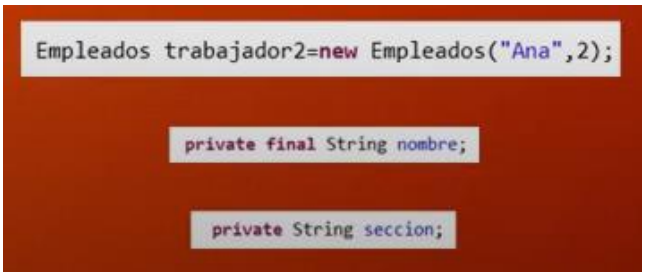
Campos static

```
private static int Id=1;
```

```
private final String nombre;

private String seccion;

private int Id;
```

```
Empleados trabajador1=new Empleados("Paco",1);

Empleados trabajador2=new Empleados("Ana",2);
```





```
package poo;

public class Prueba {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Empleados trabajador1  = new Empleados("Paco");
		Empleados trabajador2  = new Empleados("Ana");
		Empleados trabajador3 = new Empleados("Antonio");

		trabajador1.cambiaSeccion("RRHH");


		System.out.println(trabajador1.devuelveDatos());

		Empleados.Id++; <-----------

		System.out.println(trabajador2.devuelveDatos());

		Empleados.Id++; <-----------

		System.out.println(trabajador3.devuelveDatos());
	}

}

class Empleados{

	public Empleados(String nom) {
```

```
            nombre=nom;
            seccion="Administración";
            Id=1;
      }


      public void cambiaSeccion(String seccion) { // Setter
            this.seccion=seccion;
      }

      public String devuelveDatos() {

            return "El nombre es: " + nombre + " y la sección es " + seccion
+ " Id: " + Id ; // Getter
      }

      private final String nombre;
      private String seccion;
      public static int Id;

}
```

Este será el resultado:

```
El nombre es: Paco y la sección es RRHH Id: 1
El nombre es: Ana y la sección es Administración Id: 2
El nombre es: Antonio y la sección es Administración Id: 3
```

```
package poo;

public class Prueba {

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            Empleados trabajador1  = new Empleados("Paco");
            Empleados trabajador2  = new Empleados("Ana");
            Empleados trabajador3 = new Empleados("Antonio");
            Empleados trabajador4 = new Empleados("María");

            trabajador1.cambiaSeccion("RRHH");

            System.out.println(trabajador1.devuelveDatos());

            System.out.println(trabajador2.devuelveDatos());

            System.out.println(trabajador3.devuelveDatos());

            System.out.println(trabajador4.devuelveDatos());
      }

}

class Empleados{

      public Empleados(String nom) {

            nombre=nom;
            seccion="Administración";
            Id =IdSiguiente;
```

```
            IdSiguiente++;
      }


      public void cambiaSeccion(String seccion) { // Setter
            this.seccion=seccion;
      }

      public String devuelveDatos() {

            return "El nombre es: " + nombre + " y la sección es " + seccion
+ " Id: " + Id ; // Getter
      }

      private final String nombre;
      private String seccion;
      public int Id;
      private static int IdSiguiente=1;

}
```

Para que el Id sea automático declaramos una variable private static int IdSigueinte.
Mantenemos public int id;
En el constructor le decimos que id=IdSiguiente; y a continuación IdSiguiente ++; conador.
Ahora puedes observar que en el recuadro azul no hace falta hacer ningún incremento.
Ahora si la variable int Id la pasas a private podrás observar que también funciona.

Aprovechando la clase sueldos que realizamos con anterioridad haz que cada trabajador tenga su Id.

```
package poo2;

public class Sueldos {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Empleat[] misEmpleats = new Empleat[10];

		misEmpleats[0]=new Empleat("Paco Martínez", "Administración",
"Oficial", 25000);
		misEmpleats[1]=new Empleat("María Perez", "Dirección",
"Secretaria", 32000);
		misEmpleats[2]=new Empleat("Joaquín Vico", "Logística", "Peon",
15000);
		misEmpleats[3]=new Empleat("Luis Coll", "Administración",
"Contable", 42000);
		misEmpleats[4]=new Empleat("Jorge Cesar", "Logística",
"Encargado", 25000);
		misEmpleats[5]=new Empleat("Piedad Fernández", "Producción",
"Operaria",12000);
		misEmpleats[6]=new Empleat("Pere Manel", "Administración", "Aux.
Informáica", 18000);
		misEmpleats[7]=new Empleat("Carlos Pons", "Logistica", "Chofer",
22000);
		misEmpleats[8]=new Empleat("Jaime Guitierres", "Dirección",
"Director General", 55000);
		misEmpleats[9]=new Empleat("Cayetano Soriano", "Administración",
"Conserge", 13500);

		for(Empleat e: misEmpleats) {
			if(e.dameDepartamento1()=="Administración") {
				e.subeSueldo1(10);
			}
			if(e.dameDepartamento1()=="Logística") {
				e.subeSueldo1(5);
			}

		}

		for(Empleat e: misEmpleats) {

			System.out.println("Id: " + e.dameId()+ ": " + "El
empleado " + e.dameNombre1() + " Del departamento: " +
					e.dameDepartamento1() + " \n     Con
categoría de " + e.dameCategoria1() +
					" Tendrá un sueldo de " +
e.dameSueldo1()+"\n-----------------------------------------");

	}
	static class Empleat{
		public Empleat(String no, String de, String ca, double su) {
			nombre=no;
			departamento=de;
			categoria=ca;
			sueldo=su;
			Id=IdSiguiente;
```

```
                IdSiguiente++;
        }

    private String nombre;
    private String departamento;
    private String categoria;
    private double sueldo;
    private int Id;
    private static int IdSiguiente=1;

    public String dameNombre1() {
        return nombre;
    }

    public String dameDepartamento1() {
        return departamento;
    }

    public String dameCategoria1() {
        return categoria;
    }

    public double dameSueldo1() {
        return sueldo;
    }

    public int dameId() {
        return Id;
    }

    public void subeSueldo1(double porcentaje) {
        double aumentosubida=sueldo*porcentaje/100;
        sueldo+=aumentosubida;
    }
}
}
```

Este será el resultado:

```
Id: 1: El empleado Paco Martínez Del departamento: Administración
     Con categoría de Oficial Tendrá un sueldo de 27500.0
-----------------------------------------------
Id: 2: El empleado María Perez Del departamento: Dirección
     Con categoría de Secretaria Tendrá un sueldo de 32000.0
-----------------------------------------------
Id: 3: El empleado Joaquín Vico Del departamento: Logística
     Con categoría de Peon Tendrá un sueldo de 15750.0
-----------------------------------------------
Id: 4: El empleado Luis Coll Del departamento: Administración
     Con categoría de Contable Tendrá un sueldo de 46200.0
-----------------------------------------------
Id: 5: El empleado Jorge Cesar Del departamento: Logística
     Con categoría de Encargado Tendrá un sueldo de 26250.0
-----------------------------------------------
Id: 6: El empleado Piedad Fernández Del departamento: Producción
     Con categoría de Operaria Tendrá un sueldo de 12000.0
-----------------------------------------------
Id: 7: El empleado Pere Manel Del departamento: Administración
     Con categoría de Aux. Informáica Tendrá un sueldo de 19800.0
-----------------------------------------------
Id: 8: El empleado Carlos Pons Del departamento: Logistica
     Con categoría de Chofer Tendrá un sueldo de 22000.0
-----------------------------------------------
Id: 9: El empleado Jaime Guitierres Del departamento: Dirección
     Con categoría de Director General Tendrá un sueldo de 55000.0
-----------------------------------------------
Id: 10: El empleado Cayetano Soriano Del departamento: Administración
     Con categoría de Conserge Tendrá un sueldo de 14850.0
-----------------------------------------------
```

Método static (Vídeo 38)

```
package poo;

public class Prueba {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Empleados trabajador1  = new Empleados("Paco");
		Empleados trabajador2  = new Empleados("Ana");
		Empleados trabajador3 = new Empleados("Antonio");
		Empleados trabajador4 = new Empleados("María");

		trabajador1.cambiaSeccion("RRHH");

		System.out.println(trabajador1.devuelveDatos());

		System.out.println(trabajador2.devuelveDatos());

		System.out.println(trabajador3.devuelveDatos());

		System.out.println(trabajador4.devuelveDatos());

		System.out.println(Empleados.dameIdSiguiente());
	}

}

class Empleados{

	public Empleados(String nom) {

		nombre=nom;
		seccion="Administración";
		Id =IdSiguiente;
		IdSiguiente++;
	}

	public void cambiaSeccion(String seccion) { // Setter
		this.seccion=seccion;
	}

	public String devuelveDatos() {

		return "El nombre es: " + nombre + " y la sección es " + seccion
+ " Id: " + Id ; // Getter
	}

	public static String dameIdSiguiente() {

		return "El IdSiguiente es: " + IdSiguiente;
	}

	private final String nombre;
	private String seccion;
	private int Id;
	private static int IdSiguiente=1;

}
```

Creamos un método que pertenecerá a la clase Empleados, cuando la llamemos:

```
System.out.println(Empleados.dameIdSiguiente());
```

Este será el resultado:

```
El nombre es: Paco y la sección es RRHH Id: 1
El nombre es: Ana y la sección es Administración Id: 2
El nombre es: Antonio y la sección es Administración Id: 3
El nombre es: María y la sección es Administración Id: 4
El IdSiguiente es: 5
```

Los métodos estáticos son aquellos que tienen que ir acompañados de la clase en ejemplo lo tenemos con la clase Math.

Ejemplo: Math.sqrt(9);

Método main: Todo programa Java empieza por el método main.

El método main no devuelve ningún dato, por eso se le agrega el modificador void.

- No actúan sobre objetos.
- No acceden a campos de ejemplar (variables / constantes declaradas en la clase), a menos que estas sean también static.
- Para llamarlos se utiliza el nombre_clase.nombre_metodo.

Sobrecarga de constructores. (Vídeo 39)

Vamos a modificar un proyecto para ver la sobrecarga de constructores.

```
package poo;

import java.util.*;

public class Uso_Empleado {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

	Empleado[] misEmpleados= new Empleado[4];
	misEmpleados[0]=new Empleado("Paco Gómez", 85000, 1990, 12, 17);
	misEmpleados[1]=new Empleado("Ana López", 95000,1995 ,6 ,2 );
	misEmpleados[2]=new Empleado("María Martín",105000 ,2002 ,3 , 15);
	misEmpleados[3]=new Empleado("Pere Manel");

	for(Empleado i: misEmpleados) {
		i.subeSueldo(5);
	}

	for(Empleado i: misEmpleados) {
		System.out.println("Nombre: " + i.dameNombre() +
				" Sueldo: " + i.dameSueldo() +
			" Fecha de alta :" + i.dameFechaContrato());
	}

	}

}

class Empleado{

	public Empleado(String nom, double sue, int agno, int mes, int dia) {

		nombre=nom;
		sueldo=sue;
		GregorianCalendar calendario= new GregorianCalendar(agno, mes-1,
dia);
		altaContrato=calendario.getTime();
	}

	public Empleado(String nom) {
		nombre=nom;
	}

	public String dameNombre() { //getter
		return nombre;
	}

	public double dameSueldo() //getter
	{
		return sueldo;
	}

	public Date dameFechaContrato() { //getter
		return altaContrato;
```

```
	}

	public void subeSueldo(double porcentaje) { //setter
		double aumento=sueldo*porcentaje/100;
		sueldo+=aumento;
	}

	private String nombre;
	private double sueldo;
	private Date altaContrato;

}
```

Con los dos recuadros vemos como hay dos constructores, estos tienen el mismo nombre, lo que les diferencian en el número de parámetros que hay que pasarles, según los parámetros que pasemos Java sabrá si es un constructor u otro.

Cuando agregamos al empleado 4 como solo le pasamos el parámetro del nombre y ya sabe que tiene que ser el segundo constructor.

Cuando ejecutemos el proyecto este será el resultado:

```
Nombre: Paco Gómez Sueldo: 89250.0 Fecha de alta :Mon Dec 17 00:00:00 CET 1990
Nombre: Ana López Sueldo: 99750.0 Fecha de alta :Fri Jun 02 00:00:00 CEST 1995
Nombre: María Martín Sueldo: 110250.0 Fecha de alta :Fri Mar 15 00:00:00 CET 2002
Nombre: Pere Manel Sueldo: 0.0 Fecha de alta :null
```

Como al último empleado le hemos pasado un solo parámetro vemos como está el nombre, pero el resto de información como sueldo y fecha están vacíos.

Si utilizando el segundo constructor que solo nos pide el nombre a los demás datos ponga unos valores por defecto, realizaríamos lo siguiente:

```
32    public Empleado(String nom, double sue, int agno, int mes, int dia) {
33
34        nombre=nom;
35        sueldo=sue;
36        GregorianCalendar calendario= new GregorianCalendar(agno, mes-1, dia);
37        altaContrato=calendario.getTime();
38    }
39
40    public Empleado(String nom) {
41        this(nom, 30000, 2000, 01, 01);
42    }
```

Le estamos diciendo que al pasarle solo el parámetro del nombre los demás valores como 30000 de sueldo, 2000 el año, 01 el mes y 01 el día los pase al primer constructor, de este modo los valores por defecto no tenemos que introducirlos.

Este sería el resultado:

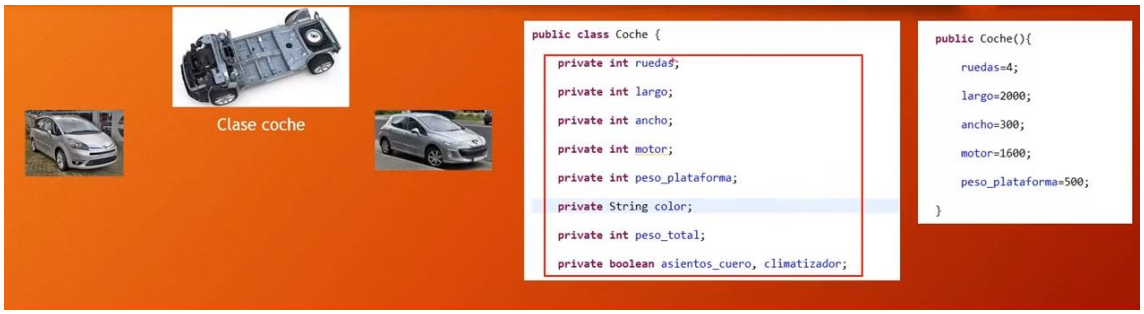
```
Nombre: Paco Gómez Sueldo: 89250.0 Fecha de alta :Mon Dec 17 00:00:00 CET 1990
Nombre: Ana López Sueldo: 99750.0 Fecha de alta :Fri Jun 02 00:00:00 CEST 1995
Nombre: María Martín Sueldo: 110250.0 Fecha de alta :Fri Mar 15 00:00:00 CET 2002
Nombre: Pere Manel Sueldo: 31500.0 Fecha de alta :Sat Jan 01 00:00:00 CET 2000
```

Sale 31500 porque le estamos incrementando un 5% a todos los sueldos.

Se puede crear una clase sin constructor, lo que se llama constructor por defecto.

Herencia I. (Vídeo 40)

Reutilización de código



Vamos a crear una nueva clase llamada furgoneta in el método main.

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

Furgoneta hereda de la clase Coche.

Java no admite herencia múltiple.

```
1  package poo;
2
3  public class Furgoneta extends Coche {
4
5      private int capacidad_carga;
6      private int plazas_extras;
7
8      public Furgoneta(int plazas_extra, int capacidad_carga) {
9
10         super(); // Llamar al constructor de la clase padre
11
12         this.capacidad_carga = capacidad_carga;
13         this.plazas_extras = plazas_extra;
14     }
15 }
```


CURSO JAVA
40
LA HERENCIA I
eclipse
Java

Herencia II (Vídeo 41)

Para cambiar el nombre a una clase la seleccionaremos con el botón derecho del ratón y del menú seleccionaremos Refactor…y de este Rename…

El Eclipese ya se encargará de camibiar el nombre en las clases que llamen a esta clase.

```
1  package poo;
2
3  public class Uso_Vehiculo {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7
8          Coche micoche1=new Coche();
9
10         micoche1.establece_color("Rojo");
11
12         Furgoneta mifurgoneta1=new Furgoneta(7, 580);
13
14         mifurgoneta1.establece_color("azul");
15
16         mifurgoneta1.configura_asientos("si");
17
18         mifurgoneta1.configura_climatizador("si");
19
20         System.out.println(micoche1.dime_datos_generales() + "\nColor: " +
21                 micoche1.dime_color());
22
23         System.out.println(mifurgoneta1.dime_datos_generales() + "\n" +
24                 mifurgoneta1.dimeDatosFurgoneta()+ "\n" +
25                 mifurgoneta1.dime_color());
26
27     }
28 }
```

En la línea 8 creamos un objeto llamado micoche1 de tipo Coche().

En la línea 10 le asignamos color con el método .estable_color(Pasándole un color como parámetro).

En la línea 12 creamos un objeto llamado mifurgoneta1 de tipo Furgoneta(pasándole dos parámetros, el número de plazas y el máximo de carga).

En la línea 14 asignamos un color al objeto mifurgoneta1 heredado de Coche().

En la línea 16 le decimos que tiene asientos de cuero, heredado de Coche().

En la línea 18 le decimos que tiene climatizador, heredado de Coche().

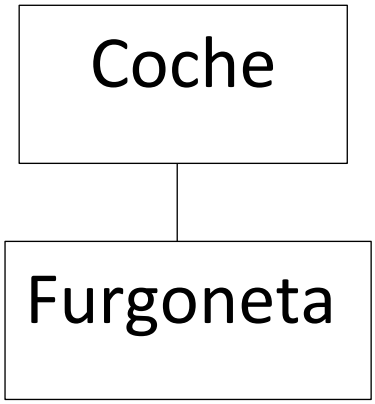
En la línea 20 imprimimos los datos generales del objeto micoche1.

En la línea 23 imprimimos los datos generales del objeto mifurgoneta1 una parte heredara de Coche() y otros propios Furgoneta().

Este será el resultado:

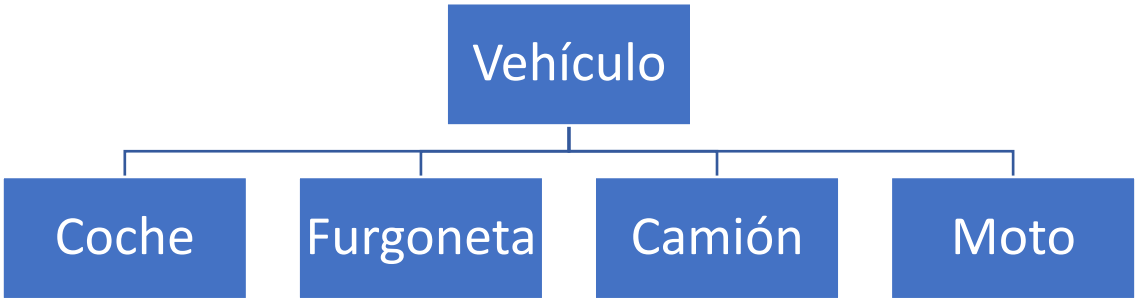
```
----------------------------------------
La plataforma del vehículo tiene 4 ruedas.
Mide 2 metros.
Con un ancho 300 cm.
y un peso de plataforma de 500 Kg.
----------------------------------------
Color: El color del coche es Rojo
----------------------------------------
La plataforma del vehículo tiene 4 ruedas.
Mide 2 metros.
Con un ancho 300 cm.
y un peso de plataforma de 500 Kg.
----------------------------------------
La capacidad de carga es: 580 y las plazas son 7
El color del coche es azul
```

Diseño de la herencia. La regla “Es un…”



La furgoneta está por un nivel abajo porque hereda de la clase Coche.

Lo más correcto sería:



La pregunta es

- ¿Un coche es un vehículo? Si
- ¿Una furgoneta es un vehículo? Si
- ¿Un camión es un vehículo? Si
- ¿Una moto es un vehículo? Si

Herencia III. Diseñando la herencia. (Vídeo 42)

Diseñador de herencia. Clase Empleado.

```
public Empleado(String nom, double sue, int agno, int mes, int dia){
    nombre=nom;
    sueldo =sue;
    GregorianCalendar calendario=new GregorianCalendar(agno, mes-1,dia);
    altaContrato=calendario.getTime();
    ++IdSiguiente;
    Id=IdSiguiente;

}
```

```
public Empleado(String nom){
    this(nom, 30000, 2000,01,01);
}

public String dameNombre(){ //getter
    return nombre + " Id: " + Id;
}
public double dameSueldo(){ //getter
    return sueldo;
}
public Date dameFechaContrato(){ //getter
    return altaContrato;
}
public void subeSueldo(double porcentaje){ //setter
    double aumento=sueldo*porcentaje/100;
    sueldo+=aumento;
}
```


¿Y si queremos ser Jefes?

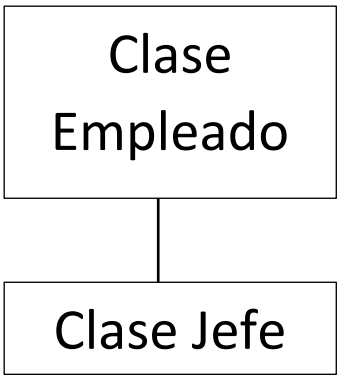
¿Y si estos Jefes reciben además del sueldo, un incentivo?

¿Me sirve la clase Empleado para crear Jefes?

Recuerda: Regala “Es un…”

¿Un Jefe es un empleado? (siempre)

Un empleado es un Jefe? (siempre)



La clase Jefe es una subclase de Empleado.

Abrimos la clase Uso_Empleado y en la parte final escribimos:

```
66      class Jefatura extends Empleado{
67          public Jefatura(String nom, double sue, int agno, int mes, int dia) {
68              super(nom, sue, agno, mes, dia);
69          }
70
71          public void estableceIncentivos(double b) {
72              incentivo=b;
73          }
74
75          public double dameSueldo() {
76              double sueldoJefe=dameSueldo();
77              return sueldoJefe + incentivo;
78
79          }
80          private double incentivo;
81      }
82
83  }
```

En la línea 66 creamos una clase llamada Jefatura que hereda de Empleado.

Esta tendrá los mismos parámetros que la clase Empleado.

En la línea 80 definimos otro campo que es solo para la clase Jefatura.

En la línea 71 hacemos un método que asigne que le pasaremos por parámetros el incentivo.

```
75          public double dameSueldo() {
76              double sueldoJefe=dameSueldo();
77              return sueldoJefe + incentivo;
78
79          }
```

Creamos un métodos que nos retornará el sueldo + los incentivos del jefe, este tiene el mismo nombre que el método utilizado para empleados.

Observamos en la parte izquierda un triangulito verde, esto nos indica que este método sobrescribirá al método del mismo nombre que tiene Empelado.

En la línea 76 queremos asignar a la variable sueldoJefe el sueldo que nos devuelve el método para Empleados:

```
48      public double dameSueldo() //getter
49      {
50          return sueldo;
51      }
```

Y en la línea 77 le retornamos el sueldoJefe + incentivos.

Pero nos encontramos que en el método dameSueldo() encontramos en la primera línea la siguiente instrucción double sueldoJefe=dameSueldo(); para diferenciar que este valor viene de la clase Empleado que es la clase super escribiremos:

```
75          public double dameSueldo() {
76              double sueldoJefe=super.dameSueldo();
77              return sueldoJefe + incentivo;
78
79          }
```

Con Super le estamos diciendo que no llame dameSuedo de la clase Jefatura sino dameSueldo de la clase Empleado.

Polimorfismo y enlazado dinámico. (Vídeo 43)

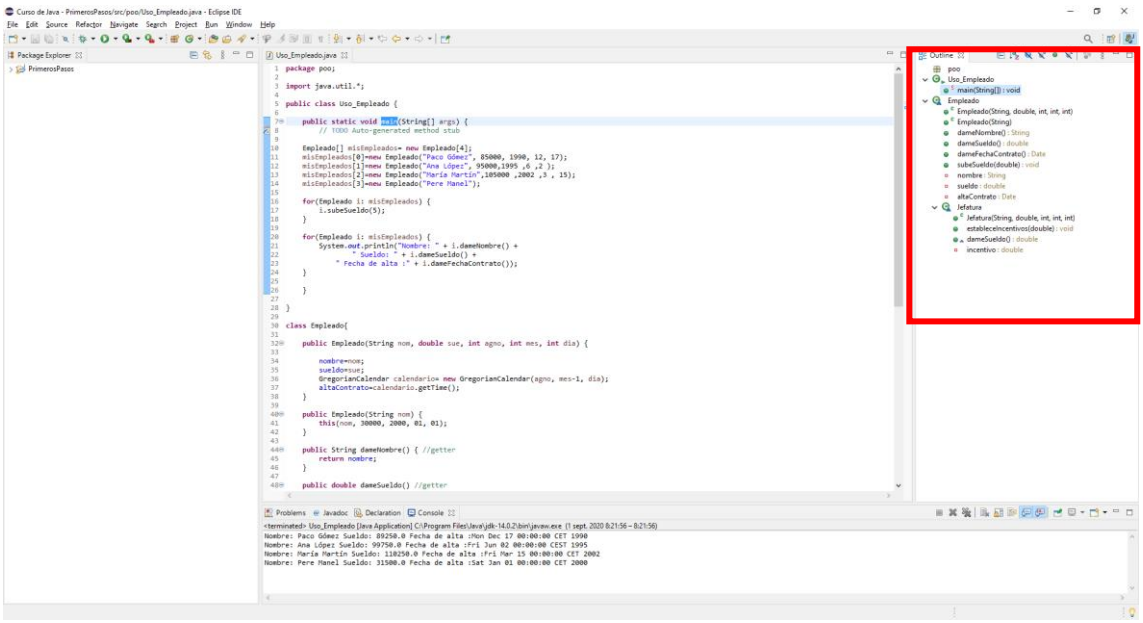
Polimorfismo. Principio de sustitución.

¿Qué demonios es eso del polimorfismo y el principio de sustitución?

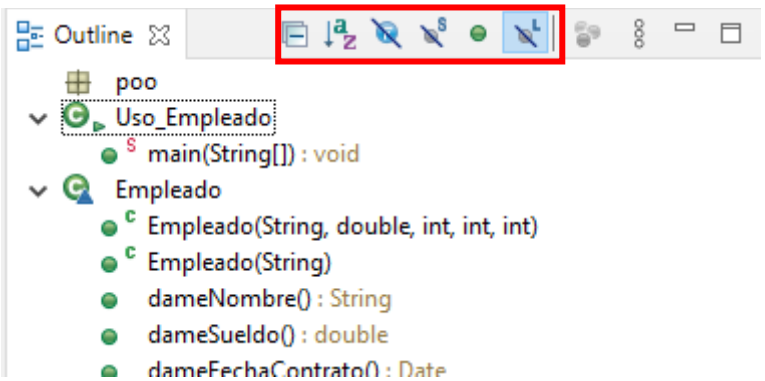
Principio de sustitución: Se puede utilizar un objeto de la subclase siempre que el programa espere un objeto de la superclase.

O lo que es lo mismo: un objeto se puede comportar de diferente forma despendiendo del contexto. Las variables objeto son polimórficas.

Un comentario sobre Eclipse:



En este ventana nos permitirá desplazarlos a lo largo de toda la clase.



Podemos activar o desactivar parte de la información en un momento determinado si la clase en muy grande.

```
package poo;

import java.util.*;

public class Uso_Empleado {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 3, 25);
		jefe_RRHH.estableceIncentivo(2570);

		Empleado[] misEmpleados= new Empleado[6];
		misEmpleados[0]=new Empleado("Paco Gómez", 85000, 1990, 12, 17);
		misEmpleados[1]=new Empleado("Ana López", 95000,1995 ,6 ,2 );
		misEmpleados[2]=new Empleado("María Martín",105000 ,2002 ,3 , 15);
		misEmpleados[3]=new Empleado("Antonio", 47500, 2009, 11, 9);
		misEmpleados[4]=jefe_RRHH; //Polimorfismo en acción. Principio de
		 sustitución
		misEmpleados[5]=new Jefatura("María", 95000, 1999, 5, 26);

		for(Empleado i: misEmpleados) {
			i.subeSueldo(5);
		}

		for(Empleado i: misEmpleados) {
			System.out.println("Nombre: " + i.dameNombre() +
					" Sueldo: " + i.dameSueldo() +
				" Fecha de alta :" + i.dameFechaContrato());
		}

		}

}

class Empleado{

	public Empleado(String nom, double sue, int agno, int mes, int dia) {

			nombre=nom;
			sueldo=sue;
			GregorianCalendar calendario= new GregorianCalendar(agno, mes-1,
dia);
			altaContrato=calendario.getTime();
			++IdSiguiente;
			Id=IdSiguiente;
		}

		public Empleado(String nom) {
			nombre=nom;
		}

		public String dameNombre() { //getter
			return nombre + " Id: " + Id;
		}

		public double dameSueldo() //getter
		{
			return sueldo;
```

2

3

4

```
	}

	public Date dameFechaContrato() { //getter
		return altaContrato;
	}

	public void subeSueldo(double porcentaje) { //setter
		double aumento=sueldo*porcentaje/100;
		sueldo+=aumento;
	}

	private String nombre;
	private double sueldo;
	private Date altaContrato;
	private static int IdSiguiente;
	private int Id;

}
```

```
class Jefatura extends Empleado{
	public Jefatura(String nom, double sue, int agno, int mes, int dia) {
		super(nom, sue, agno, mes, dia);
	}

	public void estableceIncentivo(double b) {
		incentivo=b;
	}

	public double dameSueldo() {
		double sueldoJefe=super.dameSueldo();
		return sueldoJefe + incentivo;
	}


	private double incentivo;
}
```

1

Vamos a comentar el programa.

En el apartado 1 creamos una nueva clase llamada Jefatura que hereda de Empleado (extends).

A la clase Jefatura hay que pasarle 5 parámetros, igual que la de empleados.

Utilizamos “super” para que estos valores sean asignados a la clase empleado.

Creamos un método Setter llamado estableIncentivos(double b) para poder pasar a un objeto Jefatura los incentivos.

Creamos un método Getter llamado dameSueldo(), para que nos retorne el sueldo más los incentivos.

Utilizamos super para diferencias a que métodos nos estamos refiriendo en este caso nos referimos a la clase padre llamada Empleados y al final nos retorna el sueldoJefe + incentivo.

Definimos en el constructor la variable incentivo de tipo double.

```
Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 3, 25);
jefe_RRHH.estableceIncentivo(2570);
```

2

De la clase Jefatura definimos un objeto llamado jefe_RRHH a los que le pasamos 5 parámetros.

Utilizamos el método estableceIncentivo(2570); para asignárselo al objeto jefe_RRHH.

```
      misEmpleados[4]=jefe_RRHH; //Polimorfismo en acción. Principio de
sustitución
      misEmpleados[5]=new Jefatura("María", 95000, 1999, 5, 26);
```

3

En la array misEmpleado[3] =almacenamos la información dela variable jefe_RRHH de tipo Jefatura, a este procedimiento se denomina Polimorfismo en acción.

En la array misEmpleados[4] =almacenamos la información de la clase Jefatura, como tienen los mismos parámetros esto funciona correctamente.

```
for(Empleado i: misEmpleados) {
      System.out.println("Nombre: " + i.dameNombre() +
                  " Sueldo: " + i.dameSueldo() +
            " Fecha de alta :" + i.dameFechaContrato());
}
```

4

En el bucle for la variable i hace referencia a Empleado, pero si os acordáis hay dos métodos con el mismo nombre dameSueldo(), pues el compilador de Java sabe distinguir cuando es Empleado o Jefatura, están señalados con recuadro verde.

Este será el resultado:

```
Nombre: Paco Gómez Id: 2 Sueldo: 89250.0 Fecha de alta :Mon Dec 17 00:00:00 CET 1990
Nombre: Ana López Id: 3 Sueldo: 99750.0 Fecha de alta :Fri Jun 02 00:00:00 CEST 1995
Nombre: María Martín Id: 4 Sueldo: 110250.0 Fecha de alta :Fri Mar 15 00:00:00 CET 2002
Nombre: Antonio Id: 5 Sueldo: 49875.0 Fecha de alta :Mon Nov 09 00:00:00 CET 2009
Nombre: Juan Id: 1 Sueldo: 60320.0 Fecha de alta :Sat Mar 25 00:00:00 CET 2006
Nombre: María Id: 6 Sueldo: 99750.0 Fecha de alta :Wed May 26 00:00:00 CEST 1999
```

Podréis observar que el orden el primero es está en su posición, esto es debido a que el bloque 2 no está en la posición correcta, según flecha.

```
Nombre: Paco Gómez Id: 1 Sueldo: 89250.0 Fecha de alta :Mon Dec 17 00:00:00 CET 1990
Nombre: Ana López Id: 2 Sueldo: 99750.0 Fecha de alta :Fri Jun 02 00:00:00 CEST 1995
Nombre: María Martín Id: 3 Sueldo: 110250.0 Fecha de alta :Fri Mar 15 00:00:00 CET 2002
Nombre: Antonio Id: 4 Sueldo: 49875.0 Fecha de alta :Mon Nov 09 00:00:00 CET 2009
Nombre: Juan Id: 5 Sueldo: 60320.0 Fecha de alta :Sat Mar 25 00:00:00 CET 2006
Nombre: María Id: 6 Sueldo: 99750.0 Fecha de alta :Wed May 26 00:00:00 CEST 1999
```


CURSO JAVA
43
LA HERENCIA IV
POLIMORFISMO (Bonita palabra)
ENLAZADO DINÁMICO
eclipse
Java

Ejercicios POO

1. Crear una clase Libro que contenga los siguiente atributos:

- ISBN

- Titulo

- Autor

- Numero de paginas

Crear sus respectivos metodos get y set correspondientes para cada atributo.

Crear el método toString() para mostrar la informacion relativa al libro con el siguiente formato:

"El libro su_titulo con ISBN su_ISBN creado por el autor su_autor tiene num_paginas páginas"

En el fichero main, crear 2 objetos Libro, los valores que se quieran, y mostrarlos por pantalla.

Por último, indicar cual de los 2 tiene más páginas.

La clase Libro:

```
package Ejercicio_poo_ddr_1;

public class Libro {

	//Atributos
	private int ISBN;
	private String titulo;
	private String autor;
	private int numPaginas;

	//Constructores
	public Libro(int pISBN, String pTitulo, String pAutor, int pNumPaginas
) {
		ISBN = pISBN;
		titulo=pTitulo;
		autor=pAutor;
		numPaginas=pNumPaginas;

	}

	//Métodos

	public int getISBN() {
		return ISBN;
	}

	public void setISBN(int ISBN) {
		this.ISBN=ISBN;
	}

	public String getTitulo() {
		return titulo;
```

```
	}

	public void setTitulo(String titulo) {
		this.titulo=titulo;
	}

	public String getAutor() {
		return autor;
	}
	public void setAutor(String autor) {
		this.autor=autor;
	}

	public int getNumPaginas() {
		return numPaginas;
	}

	public void setNumPaginas(int numPaginas) {
		this.numPaginas=numPaginas;
	}

	public String toString() {
		return "El libro " + titulo + " con ISBN " + ISBN + " creado por
el autor " +
				autor + " Tiene " + numPaginas + " Páginas.";
	}
}
```

La clase Ejercicio_POO_DDR_1

```
package Ejercicio_poo_ddr_1;

public class Ejercicio_POO_DDR_1 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Libro libro1=new Libro(23849495, "Tutotial de Java", "Juan
Fernández", 150);
		Libro libro2=new Libro(98364646, "Tutotial de Python", "Pere
Manel", 185);

		System.out.println(libro1.toString());
		System.out.println(libro2.toString());

		libro1.setNumPaginas(300);
		libro2.setNumPaginas(400);

		if(libro1.getNumPaginas()>libro2.getNumPaginas()) {
			System.out.println("El libro " + libro1.getTitulo() + "
tiene más páginas. " + libro1.getNumPaginas());
		}else {
			System.out.println("El libro " + libro2.getTitulo() + "
tiene más páginas. " + libro2.getNumPaginas());
		}

	}

}
```

- Ejercicio:

Vamos a realizar una clase llamada Raices, donde representaremos los valores de una ecuación de 2º grado.

Tendremos los 3 coeficientes como atributos, llamémosles a, b y c.

Hay que insertar estos 3 valores para construir el objeto.

Las operaciones que se podrán hacer son las siguientes:

- obtenerRaices(): imprime las 2 posibles soluciones

- obtenerRaiz(): imprime única raíz, que será cuando solo tenga una solución posible.

- getDiscriminante(): devuelve el valor del discriminante (double), el discriminante tiene la siguiente formula, b^2-4ac

- tieneRaices(): devuelve un booleano indicando si tiene dos soluciones, para que esto ocurra, el discriminante debe ser mayor o igual que 0.

-tieneRaiz(): devuelve un booleano indicando si tiene una única solución, para que esto ocurra, el discriminante debe ser igual que 0.

-calcular(): mostrara por consola las posibles soluciones que tiene nuestra ecuación, en caso de no existir solución, mostrarlo también.

Formula ecuación 2º grado: $\frac{-b\pm\sqrt{b^2}-4ac}{2a}$

Solo varia el signo delante de

Archivo Raices conde contiene las clases.

```
package ejercicio_poo_ddr_2;

public class Raices {
	private double a;
	private double b;
	private double c;

	public Raices(double a, double b, double c) {
		this.a=a;
		this.b=b;
		this.c=c;
	}

	private void  obtenerRaices(){
		double x1=(-b+Math.sqrt(getDiscriminante()))/(2*a);
		double x2=(-b-Math.sqrt(getDiscriminante()))/(2*a);
		System.out.println("Solución X1: ");
		System.out.println(x1);
		System.out.println("Solución X2: ");
		System.out.println(x2);
	}
```

```
	private void  obtenerRaiz(){

		double x=(-b)/(2*a);
		System.out.println("Unica solución: ");
		System.out.println(x);
	}

	private double getDiscriminante(){
		return Math.pow(b, 2)-(4*a*c);

	}

	private boolean tieneRaices() {
		return  getDiscriminante()>0;
	}

	private boolean tieneRaiz() {
		return getDiscriminante()==0;
	}

	public void calcular() {
		if(tieneRaices()) {
			obtenerRaices();
		}else if (tieneRaiz()) {
			obtenerRaiz();
		}else {
			System.out.println("No tiene soluciones.");
		}
	}
}
```

Archivo RaicesSolucion

```
package ejercicio_poo_ddr_2;

public class RaicesSolucion {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Raices ecuacion=new Raices(1,4,4);
		ecuacion.calcular();
	}

}
```

Ejercicio:

Queremos representar con programación orientada a objetos, un aula con estudiantes y un profesor.

Tanto de los estudiantes como de los profesores necesitamos saber su nombre, edad y sexo. De los estudiantes, queremos saber también su calificación actual (entre 0 y 10) y del profesor que materia da.

Las materias disponibles son matemáticas, filosofía y física.

Los estudiantes tendrán un 50% de hacer novillos, por lo que si hacen novillos no van a clase pero aunque no vayan quedara registrado en el aula (como que cada uno tiene su sitio).

El profesor tiene un 20% de no encontrarse disponible (reuniones, baja, etc.)

Las dos operaciones anteriores deben llamarse igual en Estudiante y Profesor (polimorfismo).

El aula debe tener un identificador numérico, el número máximo de estudiantes y para que esta destinada (matemáticas, filosofía o física). Piensa que más atributos necesita.

Un aula para que se pueda dar clase necesita que el profesor esté disponible, que el profesor de la materia correspondiente en el aula correspondiente (un profesor de filosofía no puede dar en un aula de matemáticas) y que haya más del 50% de alumnos.

El objetivo es crear un aula de alumnos y un profesor y determinar si puede darse clase, teniendo en cuenta las condiciones antes dichas.

Si se puede dar clase mostrar cuantos alumnos y alumnas (por separado) están aprobados de momento (imaginad que les están entregando las notas).

NOTA: Los datos pueden ser aleatorios (nombres, edad, calificaciones, etc.) siempre y cuando tengan sentido (edad no puede ser 80 en un estudiante o calificación ser 12).

Antes de empezar un comentario sobre Eclipse si pulsamos Alt + May + S, seguido de R podremos generar las Getter/Setter automáticamente.

```
package ejercicio_poo_ddr_03;

//Clase Persona
public abstract class Persona {

    /*Atributos*/
    private String nombre;
    private char sexo;
    private int edad;
    private boolean asistencia;

    /*Contantes*/
    private final String[] NOMBRES_CHICOS={"Pepe", "Fernando", "Alberto",
"Nacho", "Eustaquio"};
    private final String[] NOMBRES_CHICAS={"Alicia", "Laura", "Clotilde",
"Pepa", "Elena"};
    private final int CHICO=0;
    private final int CHICA=1;

    /*Constructores*/
    public Persona(){

        //entre 0 y 1
        int determinar_sexo=MetodosSueltos.generaNumeroAleatorio(0,1);

        //Si es 0 es un chico
        if(determinar_sexo==CHICO){
            nombre=NOMBRES_CHICOS[MetodosSueltos.generaNumeroAleatorio(0,4)];
            sexo='H';
        }else{
            nombre=NOMBRES_CHICAS[MetodosSueltos.generaNumeroAleatorio(0,4)];
            sexo='M';
```

```
        }

        //Indicamos la disponibilidad
        disponibilidad();

    }

    /*Metodos*/

    /**
     * Devuelve el nombre
     * @return
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * Modifica el nombre
     * @param nombre
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Devuelve el sexo de la persona
     * @return
     */
    public char getSexo() {
        return sexo;
    }

    /**
     * Modifica el sexo de la persona
     * @param sexo
     */
    public void setSexo(char sexo) {
        this.sexo = sexo;
    }

    /**
```

```
package ejercicio_poo_ddr_03;

public class Aula {

    /*Atributos*/
    private int id;
    private Profesor profesor;
    private Alumno[] alumnos;
    private String materia;

    /*Constantes*/
    private final int MAX_ALUMNOS=20;
```

```
    /*Constructores*/
    public Aula(){

        id=1;

        profesor=new Profesor();
        alumnos= new Alumno[MAX_ALUMNOS];
        creaAlumnos();

materia=Constantes.MATERIAS[MetodosSueltos.generaNumeroAleatorio(0,2)];

    }

    /*Metodos*/

    /**
     * Crea los alumnos para el aula
     */
    private void creaAlumnos(){

        for(int i=0;i<alumnos.length;i++){
            alumnos[i]=new Alumno();
        }

    }


    /**
     * Indica si la asistencia de los alumnos es mayor del 50%
     * @return
     */
    private boolean asistenciaAlumnos(){

        int cuentaAsistencias=0;

        //contamos las asistencias
        for(int i=0;i<alumnos.length;i++){

            if(alumnos[i].isAsistencia()){
                cuentaAsistencias++;
            }

        }

        //Muestro la asistencia total
        System.out.println("Hay "+cuentaAsistencias+" alumnos");

        return cuentaAsistencias>=((int)(alumnos.length/2));

    }

    /**
     * Indicamos si se puede dar clase
     * @return
```

```
     */
    public boolean darClase(){

        //Indicamos las condiciones para que se pueda dar la clase

        if(!profesor.isAsistencia()){
            System.out.println("El profesor no esta, no se puede dar clase");
            return false;
        }else if(!profesor.getMateria().equals(materia)){
            System.out.println("La materia del profesor y del aula no es la
misma, no se puede dar clase");
            return false;
        }else if (!asistenciaAlumnos()){
            System.out.println("La asistencia no es suficiente, no se puede
dar clase");
            return false;
        }

        System.out.println("Se puede dar clase");
        return true;

    }

    /**
     * Indicamos las notas de los alumnos aprobados, chicos y chicas
     */
    public void notas(){

        int chicosApro=0;
        int chicasApro=0;

         for(int i=0;i<alumnos.length;i++){

           //Comprobamos si el alumno esta aprobado
           if(alumnos[i].getNota()>=5){
               //Segun el sexo, aumentara uno o otro
               if(alumnos[i].getSexo()=='H'){
                   chicosApro++;
               }else{
                   chicasApro++;
               }

               System.out.println(alumnos[i].toString());

           }

        }

        System.out.println("Hay "+chicosApro+" chicos y "+chicasApro+" chicas
aprobados/as");

    }

}
```

```
package ejercicio_poo_ddr_03;
```

```
//Clase profesor que hereda de la clase Persona
public class Profesor extends Persona{

    /*Atributos*/
    private String materia;

    /*Constructores*/
    public Profesor(){
       super(); //Llama al constructor padre

       super.setEdad(MetodosSueltos.generaNumeroAleatorio(25,50)); //llama al
metodo padre


materia=Constantes.MATERIAS[MetodosSueltos.generaNumeroAleatorio(0,2)];
    }

    /*Metodos*/

    /**
     * Devuelve la materia del profesor
     * @return
     */
    public String getMateria() {
        return materia;
    }

    /**
     * Modifica la materia del profesor
     * @param materia
     */
    public void setMateria(String materia) {
        this.materia = materia;
    }

    /**
     * Calcula la disponibilidad del profesor(20%)
     */
    @Override
    public void disponibilidad() {

        int prob=MetodosSueltos.generaNumeroAleatorio(0, 100);

        if(prob<20){
            super.setAsistencia(false);
        }else{
            super.setAsistencia(true);
        }

    }

}
```

```
package ejercicio_poo_ddr_03;
```

```
//Clase Alumno, hereda de la clase Persona
public class Alumno extends Persona{

    /*Atributos*/
    private int nota;

    /*Constructor*/
    public Alumno(){
        super();

        nota=MetodosSueltos.generaNumeroAleatorio(0,10);

        super.setEdad(MetodosSueltos.generaNumeroAleatorio(12,15));

    }

    /*Metodos*/

    /**
     * Devuelve la nota
     * @return nota del alumno
     */

    public int getNota() {
        return nota;
    }

    /**
     * Modifica la nota del alumno
     * @param nota
     */
    public void setNota(int nota) {
        this.nota = nota;
    }

    /**
     * Indica si el alumno esta disponible (50%)
     */
    @Override
    public void disponibilidad() {

        int prob=MetodosSueltos.generaNumeroAleatorio(0, 100);

        if(prob<50){
            super.setAsistencia(false);
        }else{
            super.setAsistencia(true);
        }

    }

    /**
     * Muestra la informacion del alumno
     * @return informacion
     */
```

```
    public String toString(){

        return "Nombre: "+super.getNombre()+" ,sexo: "+super.getSexo()+" ,
nota: "+nota;

    }
}
```

```
package ejercicio_poo_ddr_03;

//Clase constantes
public class Constantes {

    public static final String[] MATERIAS={"Matematicas", "Filosofia",
"Fisica"};

}
```

```
package ejercicio_poo_ddr_03;

public class MetodosSueltos {

    /**
     * Genera un numero aleatorio entre dos numeros.
     * Entre el minimo y el maximo
     * @param minimo
     * @param maximo
     * @return numero entre minimo y maximo
     */
    public static int generaNumeroAleatorio(int minimo, int maximo){

        int num=(int)Math.floor(Math.random()*(minimo-
(maximo+1))+(maximo+1));
        return num;
    }

}
```

```
package ejercicio_poo_ddr_03;

public class Ejercicio_POO_DDR_03 {

    public static void main(String[] args) {

        //Creamos el objeto
        Aula aula=new Aula();

        //Indicamos si se puede dar la clase
        if(aula.darClase()){
            aula.notas();
        }

    }

}
```

Casting de objetos. Clases y métodos final. (Vídeo 44)

```
11
12      Empleado[] misEmpleados= new Empleado[6];
13      misEmpleados[0]=new Empleado("Paco Gómez", 85000, 1990, 12, 17);
14      misEmpleados[1]=new Empleado("Ana López", 95000,1995 ,6 ,2 );
15      misEmpleados[2]=new Empleado("María Martín",105000 ,2002 ,3 , 15);
16      misEmpleados[3]=new Empleado("Antonio", 47500, 2009, 11, 9);
17      Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 3, 25);
18      jefe_RRHH.estableceIncentivo(2570);
19      misEmpleados[4]=jefe_RRHH; //Polimorfismo en acción. Principio de sustitución
20      misEmpleados[5]=new Jefatura("María", 95000, 1999, 5, 26);
21
22      Jefatura jefa_finanzas=(Jefatura)misEmpleados[5];
23      jefa_finanzas.estableceIncentivo(55000);
24
```

En la línea 20 creamos un objeto de tipo Jefatura que es María como tiene los mismos parámetros que el objeto misEmpleados se lo asignamos a misEmpleados[5].

Si queremos establecer a misEmpleados[5] el estableceIncentivo(), no nos lo va a reconocer porque está hecho para el objeto Jefatura.

En la línea 22 le decimos que Jefatura jefa_finanzas = (Jefatura)misEmpleados[5], es un modo de reconvertir un objeto a otro tipo de objeto.

Con lo cual jefa_finanzas que es de tipo Jefatura ya admite el estableceIncentivo().

Si ejecutamos este será el resultado:

```
Nombre: Paco Gómez Id: 1 Sueldo: 89250.0 Fecha de alta :Mon Dec 17 00:00:00 CET 1990
Nombre: Ana López Id: 2 Sueldo: 99750.0 Fecha de alta :Fri Jun 02 00:00:00 CEST 1995
Nombre: María Martín Id: 3 Sueldo: 110250.0 Fecha de alta :Fri Mar 15 00:00:00 CET 2002
Nombre: Antonio Id: 4 Sueldo: 49875.0 Fecha de alta :Mon Nov 09 00:00:00 CET 2009
Nombre: Juan Id: 5 Sueldo: 60320.0 Fecha de alta :Sat Mar 25 00:00:00 CET 2006
Nombre: María Id: 6 Sueldo: 154750.0 Fecha de alta :Wed May 26 00:00:00 CEST 1999
```

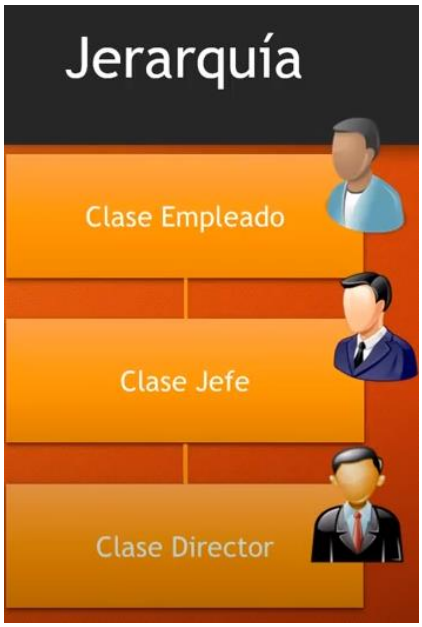
```
25      Jefatura jefe_compras=(Jefatura) misEmpleados[1];
```

Así como Jefatura hereda los comportamientos de Empleado, Empleado no hereda los comportamientos de Jefatura por este motivo el ejemplo que se muestro no es correcto, a la hora de ejecutar el programa el compilador nos mostrará el siguiente error.

```
Exception in thread "main" java.lang.ClassCastException: class poo.Empleado cannot be cast to class poo.Jefatura
        at poo.Uso_Empleado.main(Uso_Empleado.java:25)
```

Método Final

Supongamos que creamos una nueva clase llamada Director que su vez hereda de la clase Jefe.



En este ejemplo no queremos crear ninguna clase que herede de la clase jefe.


```
101 class Director extends Jefatura{
102
103     public Director (String nom, double sue, int agno, int mes, int dia) {
104         super(nom, sue, agno, mes, dia);
105     }
106 }
```

Hemos creado una clase Director que hereda de Jefatura.

```
83 final class Jefatura extends Empleado{
84     public Jefatura(String nom, double sue, int agno, int mes, int dia) {
85         super(nom, sue, agno, mes, dia);
86     }
```

En la clase Jefatura hemos agregado la instrucción final al principio, como podrás observar si ahora miramos en la clase director observaremos los siguientes errores.

```
101 class Director extends Jefatura{
102
103     public Director (String nom, double sue, int agno, int mes, int dia) {
104         super(nom, sue, agno, mes, dia);
105     }
106 }
```

Con la instrucción final detenemos las herencias desde la clase Jefatura.

El método dameSueldo de la clase Empleados le agregamos la instrucción final.

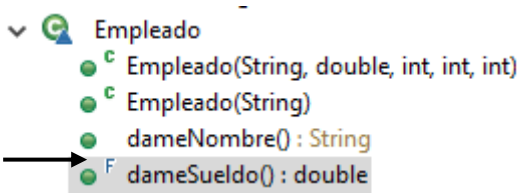
```
61     public final double dameSueldo() //getter
62     {
63         return sueldo;
64     }
```

Le estamos diciendo que todas las clases que hereden de Empleado no podrán utilizar el método dameSueldo, si observamos el método dameSueldo de Jefatura observaremos que nos está indicando un error.

```
92     public double dameSueldo() {
93         double sueldoJefe=super.dameSueldo();
94         return sueldoJefe + incentivo;
95     }
96
97
98     private double incentivo;
99 }
```

Para solucionar el problema cambiamos el nombre del método dameSuelo de la clase Jefatura.

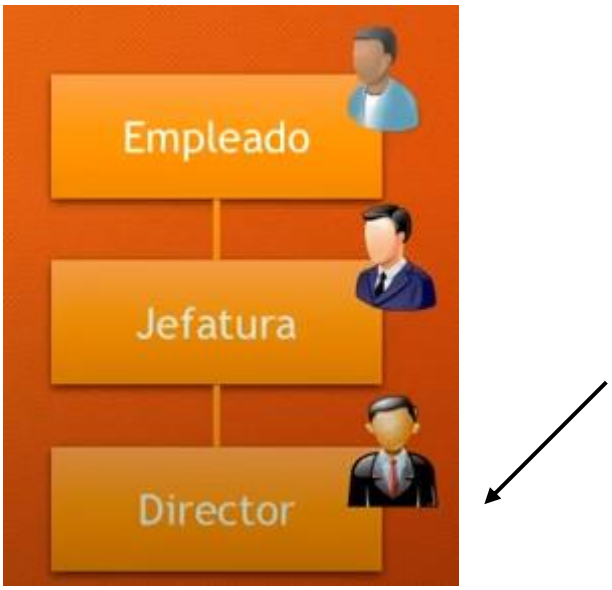
```
92     public double dameSueldo2() {
93         double sueldoJefe=super.dameSueldo();
94         return sueldoJefe + incentivo;
95     }
96
97
98     private double incentivo;
99 }
```



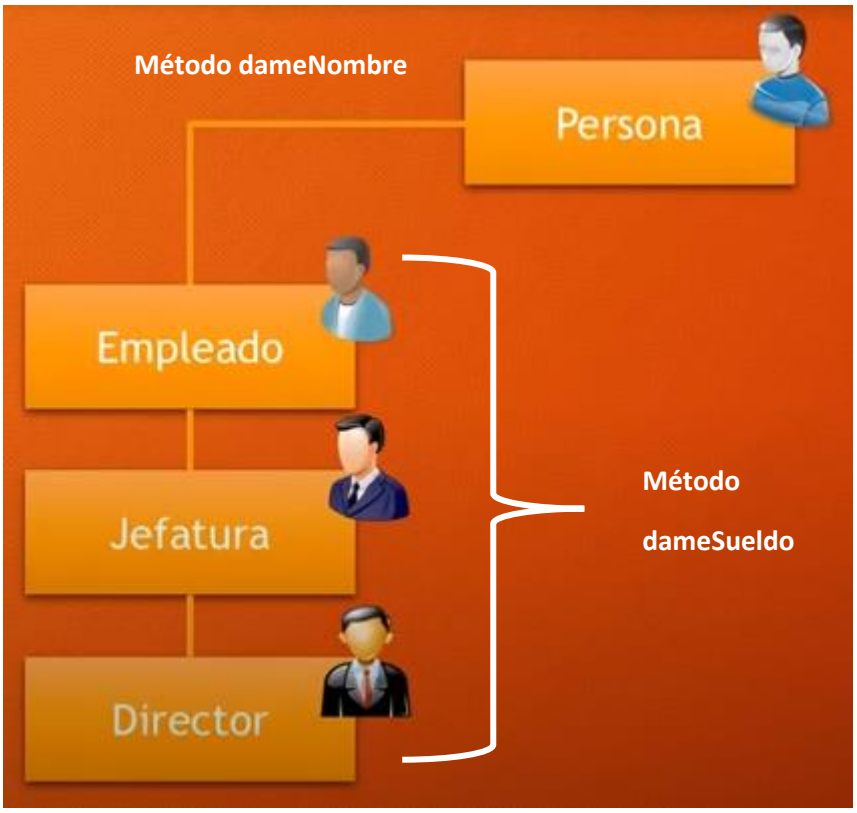
En la ventana de estructuras nos muestra que le hemos aplicado la instrucción final.


CURSO JAVA
44
LA HERENCIA V
REFUNDICIÓN DE OBJETOS (CASTING)
CLASES Y MÉTODOS FINAL
eclipse
Java

Clases Abstractas I (Video 45)

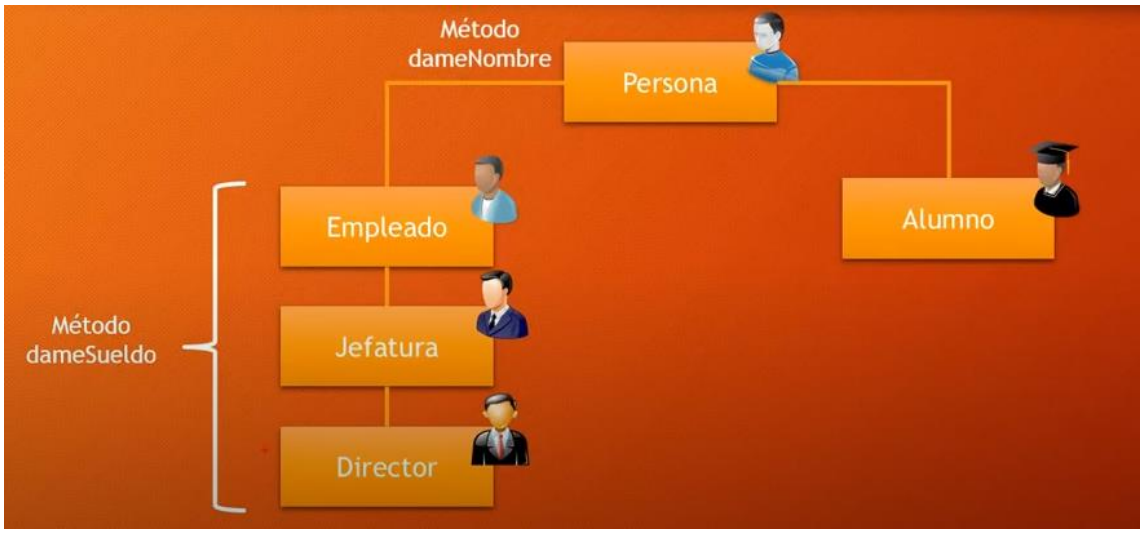


La clase Director es la más potente ya que es la que hereda de Jefatura y de Empleado, además de tener sus métodos propios.



El método dameSueldo de la clase empleado lo puede tener la clase Persona, no ya que todas las personas no tienen sueldo.

El método dameNombre de la clase empleado sería más correcto pasarlo a la clase persona ya que todas las personas tienen nombre y de igual modo Empleado heredaría este este método, así como Jefatura y Director.



Si creamos una nueva clase llamada Alumno esta puede heredar de Persona, pero no de Empleado, Jefatura o Director.

Si creamos un método dameDescripción en la clase Persona este lo heredaría tanto Alumno como Empleado, Jefatura o Director.

Método abstracto:

public abstract String dameDescripción();

No lleva llave de apertura ni cierre. Si declaras un método abstracto también estas obligado a declarar la clase como abstracta.

¿Cómo se declara una clase abstracta? abstract class Persona{……} con su constructor y sus métodos.

Todas las clases que están heredando sobre una clase abstracta está obligados a sobrescribir todos los métodos.

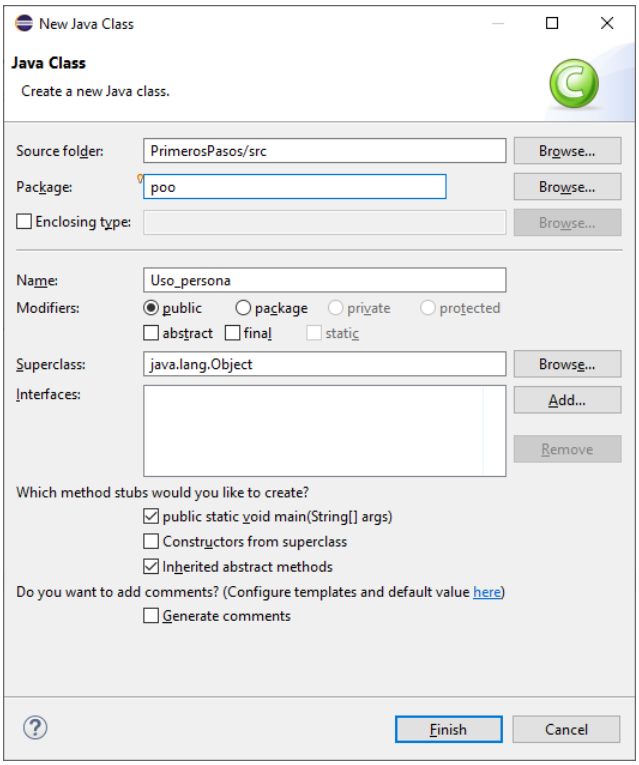
public String dameDatos{….}

El motivo de crear un método abstracto te obliga a crear este método en todas las clase que hereden, es ni más ni menos es crear un patrón de diseño.


CURSO JAVA
45
LA HERENCIA VI
CLASES ABSTRACTAS
eclipse
Java

Clases Abstractas II (Vídeo 46)

Vamos a crear una nueva clase llamada uso_Persona tiene que estar en el paquete poo.



copiaremos la clase Empleado para reutilizar el código.

```
package poo;

import java.util.Date;
import java.util.GregorianCalendar;

public class Uso_persona {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Persona[] lasPersonas=new Persona[2];

		lasPersonas[0]=new Empleado2("Luis Conde", 50000, 2009, 02, 25);
		lasPersonas[1]=new Alumno("Ana López", "Biológicas");

		for(Persona p: lasPersonas) {
			System.out.println(p.dameNombre()+ ", "+
p.dameDescripcion());
		}
	}

}
```

```
abstract class Persona{


	public Persona(String nom) {
		nombre=nom;
	}
```

```
        public String dameNombre() {
                return nombre;
        }

        public abstract String dameDescripcion();

        private String nombre;

}
class Empleado2 extends Persona{

        public Empleado2(String nom, double sue, int agno, int mes, int dia) {
                super(nom);
                sueldo=sue;
                GregorianCalendar calendario= new GregorianCalendar(agno, mes-1,
dia);
                altaContrato=calendario.getTime();
                ++IdSiguiente;
                Id=IdSiguiente;
        }

        public String dameDescripcion() {
                return "Este empleado  tiene un Id= " +Id + " con un seueldo de
" + sueldo;
        }

        public double dameSueldo() //getter
        {
                return sueldo;
        }

        public Date dameFechaContrato() { //getter
                return altaContrato;
        }

        public void subeSueldo(double porcentaje) { //setter
                double aumento=sueldo*porcentaje/100;
                sueldo+=aumento;
        }

        private double sueldo;
        private Date altaContrato;
        private static int IdSiguiente;
        private int Id;

}
class Alumno extends Persona{
        public Alumno(String nom, String car) {
                super(nom);
                carrera=car;
        }

        public String dameDescripcion() {
                return "Este alumno está estudiando la carrera de " + carrera;
        }

        private String carrera;
}
```

Este será el resultado:

```
Luis Conde, Este empleado  tiene un Id= 1 con un seueldo de 50000.0
Ana López, Este alumno está estudiando la carrera de Biológicas
```

Una vez copiada la clase Empleado le decimos Empleado2, creamos la case Persona y la clase Alumno.

La clase Persona la definimos como abstract y definimos un método abstracto llamado dameDescripcion() esto obligará a las clases que heredan crear su propio método dameDescripcion().

Tanto la clase Empleado2 como la clase Alumno tendrá obligatoriamente crear su propio método dameDescripcion().

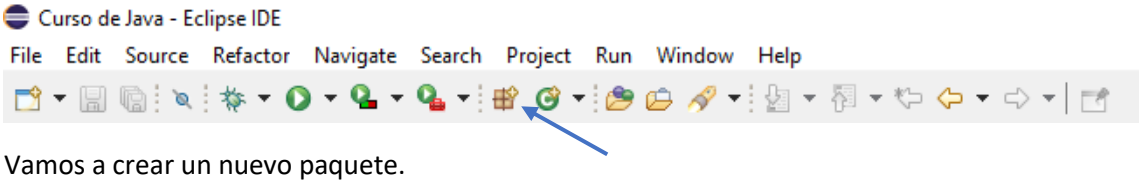
Creamos un método llamado dameNombre (), que si podrán utilizar todas las clases que hereda.

Así cada método se adecua a cada clase.

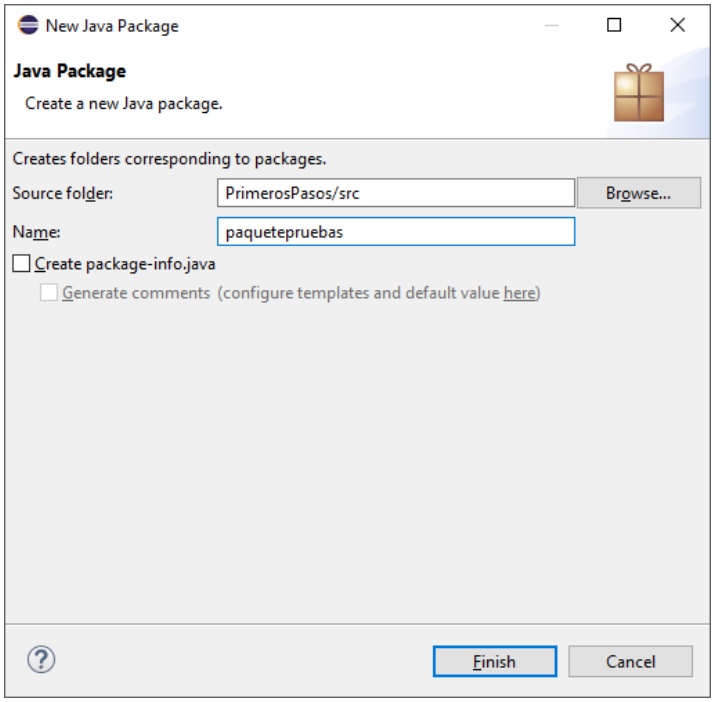
Modificadores de acceso. Clase Object. (Vídeo 47)

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
Public	Sí	Sí	Sí	Sí
Protected	Sí	Sí	Sí	No
Private	Sí	No	No	No
Por defecto	Sí	Sí	No	No

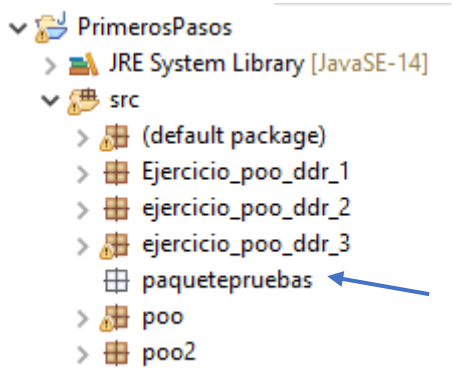
Ejecutaremos Eclipse para ver unos ejemplos de esto modificadores.



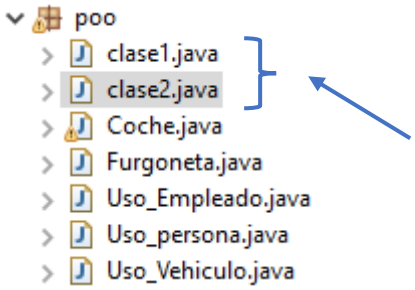
Vamos a crear un nuevo paquete.



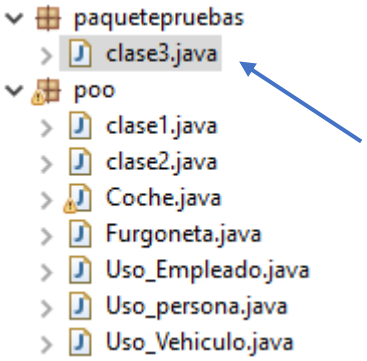
Le llamaremos paquetepruebas.



Vamos a crear una clase1 sin método main y otra llamada clase2 con método main en el paquete poo.



Vamos a crear la clase3 sin método main en el paquete paquetepruebas.

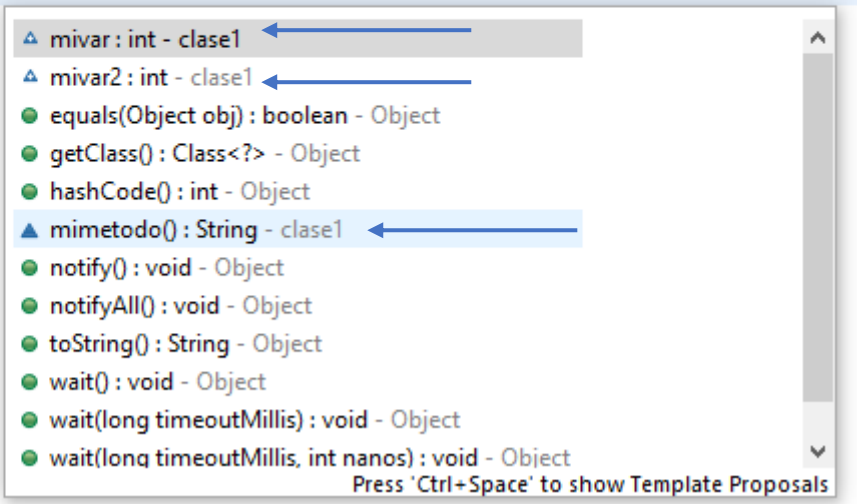


En la clase1 escribiremos el correspondiente código:

```
1  package poo;
2
3  public class clase1 {
4      int mivar=5;
5
6      int mivar2=7;
7
8      String mimetodo() {
9          return "El valor de mivar2 es: " + mivar2;
10     }
11 }
```

Como podrás observar tanto a las variables como al método no le hemos puesto ningún modificador.

```
1  package poo;
2
3  public class clase2 {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          clase1 miobj=new clase1();
8          miobj.
9      }
10
11 }
12
```



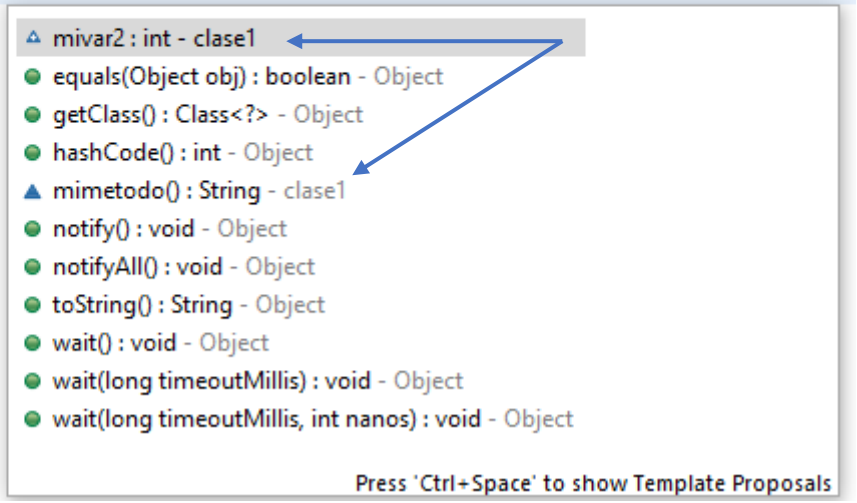
Desde la clase2 cuando uilizamos miobj de la clase1 al poner el punto vemos que tenemos acceso tanto a las variables como al método.

Ahora vamos a la clase1 y ponemos una variable como private.

```
1  package poo;
2
3  public class clase1 {
4      private int mivar=5;
5
6      int mivar2=7;
7
8      String mimetodo() {
9          return "El valor de mivar2 es: " + mivar2;
10     }
11 }
```

Ahora si accedemos a la clase2 veremos que al valor de esta variable no tenemos acceso.

```
1  package poo;
2
3  public class clase2 {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          clase1 miobj=new clase1();
8          miobj.
9      }
10
11 }
12
```



La variable mivar1 está encapsulada.

Ahora borremos el private de la variable clase1.

Desde la clase3 vamos a escribir el correspondiente código:

```
1  package paquetepruebas;
2
3  import poo.clase1;
4
5  public class clase3 extends clase1 {
6
7
8  }
```

Al encontrarse la clase3 en otro paquete que la clase2 hemos de importar la clase.

Cuando declaremos una clase3 de tipo clase1 la heredaremos sin ningún problema.

```
1  package poo;
2
3  import paquetepruebas.clase3;
4
5  public class clase2 {
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9
10         clase1 miobj=new clase1();
11
12         clase3 miobj2=new clase3();
13     }
14
15 }
```

Lo mismo pasa en la clase2 si queremos definir un objeto clase3, primero importaremos la clase3.

Después podremos definir un objeto llamado miobj2 de tipo clase3.

```
1  package poo;
2
3  import paquetepruebas.clase3;
4
5  public class clase2 {
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9
10         clase1 miobj=new clase1();
11
12         clase3 miobj2=new clase3();
13
14         miobj2.
15     }
16
17 }
18
```

```
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeoutMillis) : void - Object
wait(long timeoutMillis, int nanos) : void - Object

Press 'Ctrl+Space' to show Template Proposals
```

No tenemos acceso ni a las variables y métodos, el motivo es porque estoy utilizando el modificador por defecto:

```
1  package poo;
2
3  public class clase1 {
4      int mivar=5;
5
6      int mivar2=7;
7
8      String mimetodo() {
9          return "El valor de mivar2 es: " + mivar2;
10     }
11 }
```

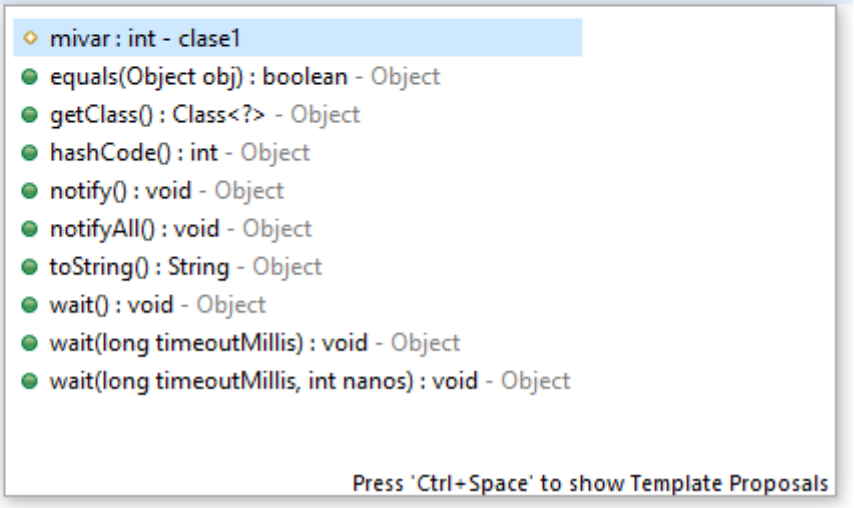
Tanto en las variables como en el método y se recordamos no permite acceder a sus datos si es una subclase que se encuentra en otro paquete.

Si le ponemos el modificador Protected podremos acceder a la subclase.

```java
package poo;

public class clase1 {

    protected int mivar=5;

    int mivar2=7;

    String mimetodo() {
        return "El valor de mivar2 es: " + mivar2;
    }
}
```

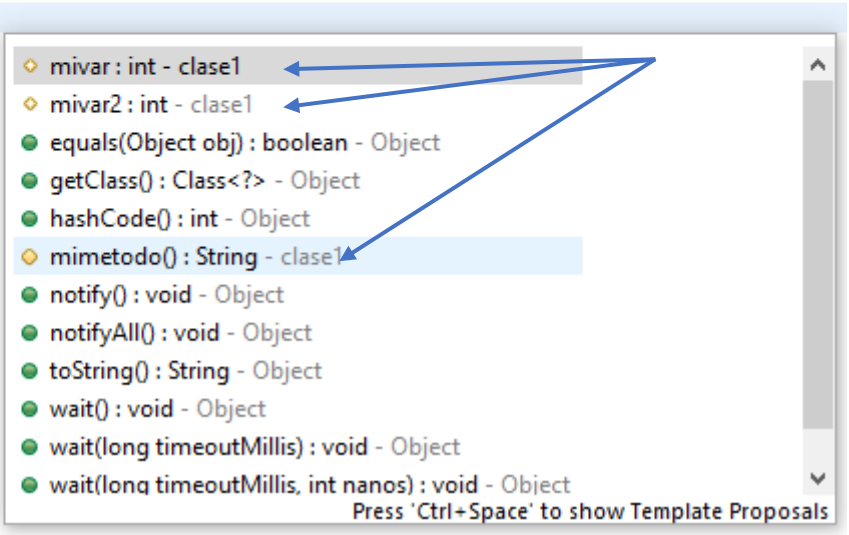
Accedemos a la clase2.

```java
package poo;

import paquetepruebas.clase3;

public class clase2 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        clase1 miobj=new clase1();

        clase3 miobj2=new clase3();

        miobj2.
    }

}
```



Ya accedemos a la variable mivar, ahora pondremos protected tanto a la variable mivar2 como al método.

```
1  package poo;
2
3  import paquetepruebas.clase3;
4
5  public class clase2 {
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9
10         clase1 miobj=new clase1();
11
12         clase3 miobj2=new clase3();
13
14         miobj2.
15     }
16
17 }
18
```



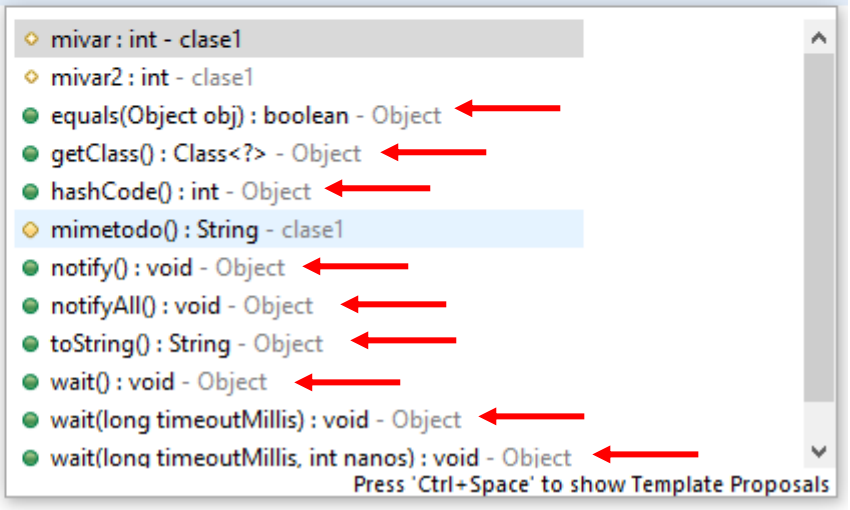
El modificador protected no es ni más ni menos un poco de protección un poco de encapsulación a nuestras variables y métodos per en más permisivo que el private.

Clase object: todas las clase tienen como jerarquía la clase object.

```
1  package poo;
2
3  import paquetepruebas.clase3;
4
5  public class clase2 extends Object{
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9
10         clase1 miobj=new clase1();
11
12         clase3 miobj2=new clase3();
13
14         miobj2.
15     }
16
17 }
```

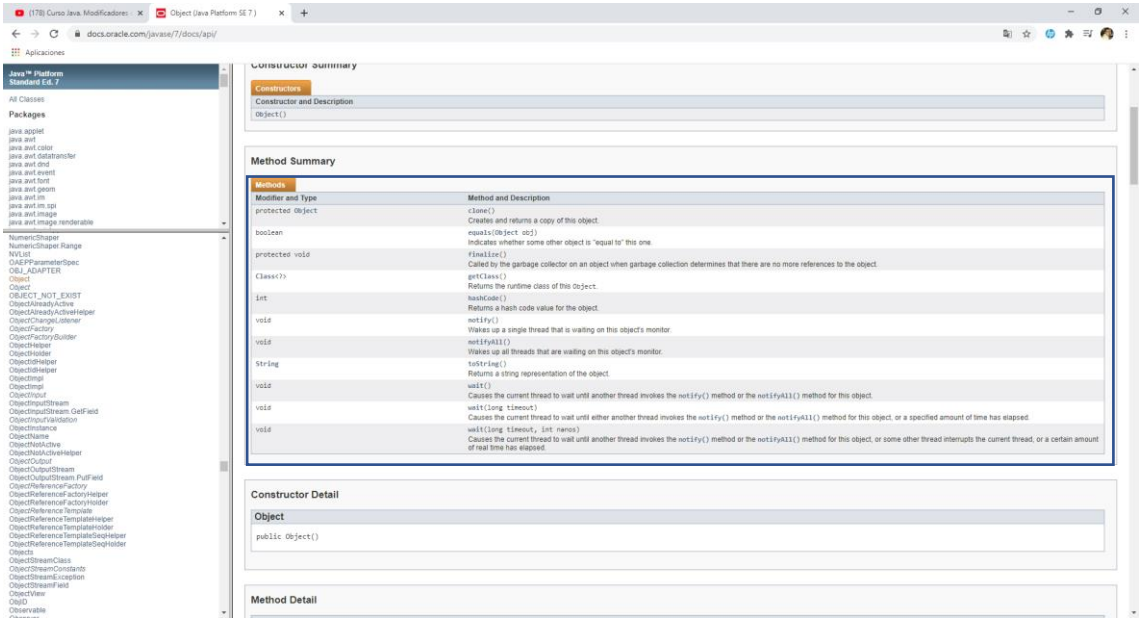
Aunque no la especifiques estás heredando la extensión Object no hace falta especificarlo, por defecto ya lo asume.

```java
package poo;

import paquetepruebas.clase3;

public class clase2 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        clase1 miobj=new clase1();

        clase3 miobj2=new clase3();

        miobj2.
    }

}
```



Y por ese motivo tiene acceso a las instrucciones Object.

Si vamos a la API de Java y buscamos Object.



Method Summary

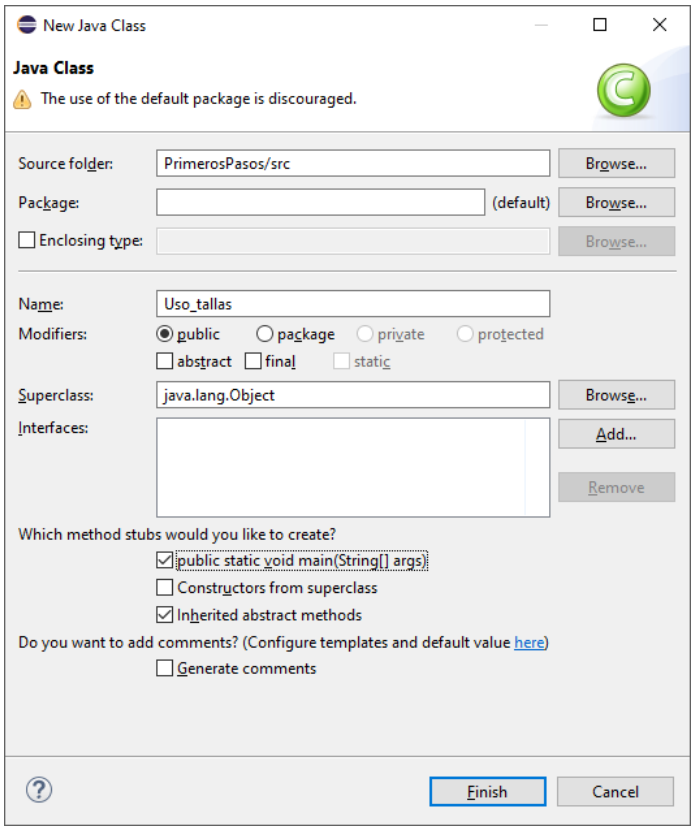
Methods

Modifier and Type	Method and Description
`protected Object`	`clone()` Creates and returns a copy of this object.
`boolean`	`equals(Object obj)` Indicates whether some other object is "equal to" this one.
`protected void`	`finalize()` Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
`Class<?>`	`getClass()` Returns the runtime class of this `Object`.
`int`	`hashCode()` Returns a hash code value for the object.
`void`	`notify()` Wakes up a single thread that is waiting on this object's monitor.
`void`	`notifyAll()` Wakes up all threads that are waiting on this object's monitor.
`String`	`toString()` Returns a string representation of the object.
`void`	`wait()` Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.
`void`	`wait(long timeout)` Causes the current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.
`void`	`wait(long timeout, int nanos)` Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.


CURSO JAVA
47
MODIFICADORES DE ACCESO
LA CLASE OBJECT (SUPERCLASE CÓSMICA)
eclipse
Java

Tipos enumerados. (Vídeo 48)

Vamos a crear una nueva clase en el paquete por defecto llamado Uso_Tallas.




```
2  public class Uso_tallas {
3
4      enum Talla {MINI, MEDIANO, GRANDE, MUY_GRANDE};
5
6      public static void main(String[] args) {
7          // TODO Auto-generated method stub
8          String talla;
9          Talla s=Talla.
10
11     }
12
13 }
14
```

GRANDE : Uso_tallas.Talla - Uso_tallas.Talla
MEDIANO : Uso_tallas.Talla - Uso_tallas.Talla
MINI : Uso_tallas.Talla - Uso_tallas.Talla
MUY_GRANDE : Uso_tallas.Talla - Uso_tallas.Talla
valueOf(String arg0) : Talla - Talla
class : Class<Uso_tallas.Talla>
values() : Talla[] - Talla
valueOf(Class<T> enumType, String name) : T - Enum

Press 'Ctrl+Space' to show Template Proposals

Si queremos que una variable solo asuma determinados valores lo haremos con el método enum, lo tenemos que definir fuera de la clase principal.

Cuando definimos un objeto de tipo Talla al marcar el punto ya nos muestra los valores que le tenemos que asignar.

```
1
2  public class Uso_tallas {
3
4      enum Talla {MINI, MEDIANO, GRANDE, MUY_GRANDE};
5
6      public static void main(String[] args) {
7          // TODO Auto-generated method stub
8          String talla;
9          Talla s=Talla.MINI;
10         Talla m=Talla.MEDIANO;
11         Talla l=Talla.GRANDE;
12         Talla xl=Talla.MUY_GRANDE;
13
14         Talla  xxl=talla.ENORME;
15
16     }
17
18 }
```

Una vez que definamos un objeto de tipo Talla si le asignamos un valor que no definimos en la línea 4 este nos daré error.

```
1  import java.util.*;
2  public class Uso_tallas {
3
4
5      enum Talla{
6
7          MINI("S"), MEDIANO("M"),GRANDE("L"), MUY_GRANDE("XL");
8
9          private Talla(String abreviatura) {
10             this.abreviatura=abreviatura;
11         }
12
13         public String dameAbreviatura() {
14             return abreviatura;
15         }
16
17         private String abreviatura;
18     }
19
20     public static void main(String[] args) {
21         // TODO Auto-generated method stub
22
23         Scanner entrada=new Scanner(System.in);
24         System.out.println("Escribe una talla: MINI, MEDIANO, GRANDE, MUY_GRANDE");
25         String entrada_datos=entrada.next().toUpperCase();
26
27         Talla la_talla=Enum.valueOf(Talla.class, entrada_datos);
28
29         System.out.println("Talla= "+ la_talla);
30         System.out.println("Abrebiatura= "+ la_talla.dameAbreviatura());
31     }
32
33 }
```

En la línea 7 dentro de la clase Uso_tallas creamos una variable que solo puede adminitr MINI, MEDIANO, GRANDE y MUY_GRANDE, que acompañamos con su abreviatura.

En la línea 9 al constructor definimos la variable abreviatura que es el parámetro a pasar.

En la línea 13 hacemos el correspondiente método para que retorne el valor abreviatura.

Dentro del main en la línea 23definimos una variable de entrada llamada entrada.

En la línea 24 que imprima un mensaje.

En la línea 25 espera que introduzcas un dato por teclado.

En la línea 27 La variable la_talla de tipo Talla:

```
Talla la_talla=Enum.valueOf(Talla.class, entrada_datos);
```

Devuelve la constante de enumeración de tipo de enumeración especificado. El nombre debe coincidir exactamente con un identificador utilizado para declarar una constante de enumeración en este tipo.

La línea 29 y 30 imprimen en consola.

Este será el resultado:

```
Escribe una talla: MINI, MEDIANO, GRANDE, MUY_GRANDE
mediano
Talla= MEDIANO
Abrebiatura= M
```

Contenido

POO I (Vídeo 27) 1
POO II. (Vídeo 28) 6
POO III. (Vídeo 29) 12
POO IV Getters y Setters. (Vídeo 30) 16
POO V. Paso de parámetros. (Vídeo 31) 20
POO VI. Construcción objetos (Vídeo 32) 24
POO VII. Construcción objetos II. (Vídeo 33) 32
Java POO VIII. Construcción objetos III. (Vídeo 34) 34
POO IX. Construcción objetos IV. (Vídeo 35) 36
Constantes Uso final (Vídeo 36) 41
Uso static. (Vídeo 37) 46
Método static (Vídeo 38) 51
Sobrecarga de constructores. (Vídeo 39) 53
Herencia I. (Vídeo 40) 56
Herencia II (Vídeo 41) 59
Herencia III. Diseñando la herencia. (Vídeo 42) 62
Polimorfismo y enlazado dinámico. (Vídeo 43) 65
Ejercicios POO 70
1. Crear una clase Libro que contenga los siguiente atributos: 70
- ISBN 70
- Titulo 70
- Autor 70
- Numero de paginas 70
Crear sus respectivos metodos get y set correspondientes para cada atributo 70
Crear el método toString() para mostrar la informacion relativa al libro con el siguiente formato: 70
"El libro su_titulo con ISBN su_ISBN creado por el autor su_autor tiene num_paginas páginas" 70
En el fichero main, crear 2 objetos Libro, los valores que se quieran, y mostrarlos por pantalla. 70
Por último, indicar cual de los 2 tiene más páginas. 70
La clase Libro: 70
La clase Ejercicio_POO_DDR_1 71
- Ejercicio: 72
Vamos a realizar una clase llamada Raices, donde representaremos los valores de una ecuación de 2º grado. 72

Tendremos los 3 coeficientes como atributos, llamémosles a, b y c. 72

Hay que insertar estos 3 valores para construir el objeto. 72

Las operaciones que se podrán hacer son las siguientes: 72

- obtenerRaices(): imprime las 2 posibles soluciones 72

- obtenerRaiz(): imprime única raíz, que será cuando solo tenga una solución posible. 72

- getDiscriminante(): devuelve el valor del discriminante (double), el discriminante tiene la siguiente formula, b^2-4ac 72

- tieneRaices(): devuelve un booleano indicando si tiene dos soluciones, para que esto ocurra, el discriminante debe ser mayor o igual que 0. 72

-tieneRaiz(): devuelve un booleano indicando si tiene una única solución, para que esto ocurra, el discriminante debe ser igual que 0. 72

-calcular(): mostrara por consola las posibles soluciones que tiene nuestra ecuación, en caso de no existir solución, mostrarlo también. 72

Formula ecuación 2º grado: $-b \pm b2 - 4ac2a$ 72

Solo varia el signo delante de 72

Casting de objetos. Clases y métodos final. (Vídeo 44) 81

Clases Abstractas I (Video 45) 85

Clases Abstractas II (Vídeo 46) 88

Modificadores de acceso. Clase Object. (Vídeo 47) 91

Tipos enumerados. (Vídeo 48) 101