

**Lunes, 15 de Agosto de 2016**

### **Video 1. Presentación**



## Otros entornos de desarrollo

- JDK
- NetBeans
- BlueJ
- JBuilder
- JCreator
- JDeveloper



## Temario Parte 1

- Descarga e instalación de Eclipse
- Introducción a Java
- Estructuras principales del lenguaje
- Objetos y clases
- Herencia
- Clases internas e interfaces
- Programación de gráficos
- Eventos
- Componentes Swing
- Aplicaciones y Applets
- Tratamientos de errores (excepciones) y depuración
- Programación genérica
- Colecciones
- Programación multihilo (multithreading)



## Temario Parte 2

- Programación con archivos
- XML
- Programación para redes
- Programación para BBDD
- Programación cliente-servidor (objetos distribuidos)
- Swing avanzado
- AWT Avanzado
- JavaBeans
- Seguridad
- Programación internacional.
- Métodos nativos
- Anotaciones

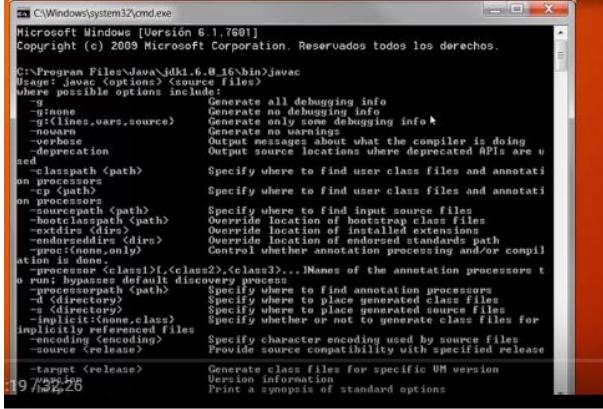
### **Video 4. Estructuras principales I**



## Tipos de programas Java

- Aplicaciones de consola.
- Aplicaciones de propósito general.
- Applets.

## Aplicaciones de consola

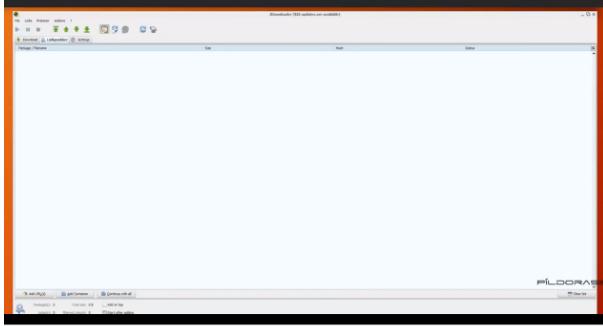


```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Program Files\Java\jdk1.6.0_16\bin>javac
Usage: javac <options> <source files>
where possible options include:
  -g:none           Generate all debugging info
  -g:{lines,vars,source} Generate no debugging info
  -g:lines          Generate only source debugging info
  -g:vars           Generate only variable debugging info
  -g:source         Generate only source code debugging info
  -verbose          Output messages about what the compiler is doing
  -deprecation      Output source locations where deprecated APIs are u
sed
  -classpath <path>  Specify where to find user class files and annotati
on processors
  -D <name>          Specify where to find user class files and annotati
on processors
  -processor <path>  Specify where to find input source files
  -sourcepath <path>  Override location of bootstrap class files
  -bootclasspath <path>  Override location of standard class files
  -endorseddirs <dirs>  Override location of endorsed standards path
  -proc:{none,only}   Control whether annotation processing and/or compil
ation is enabled
  -processor <class1>,<class2>...<classN>...Names of the annotation processors t
o run; bypasses default discovery process
  -processorpath <path>  Specify where to find annotation processors
  -s <directory>      Specify where to place generated class files
  -d <directory>      Specify where to place generated source files
  -implicit:{none,all}  Specify whether or not to generate class files for
input source and class files
  -encoding <encoding>  Specify character encoding used by source files
  -source <release>    Provide source compatibility with specified release
  -target <release>    Generate class files for specific VM version
  -version             Version information
  -help               Print a synopsis of standard options
19/10/2010 10:52:20
```

Eclipse tiene su consola incorporada.

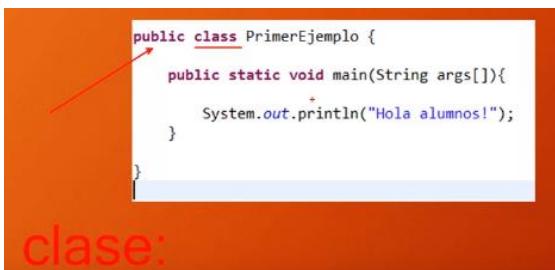
## Aplicaciones de propósito general



Este ejemplo es jdownloader (casi todos los sistemas en java).



**Applet:** Programas creados en java que se ejecutan dentro de un navegador, con un plugin



```
public class PrimerEjemplo {
    public static void main(String args[]){
        System.out.println("Hola alumnos!");
    }
}
```

clase:

Todo programa java tiene que estar dentro de alguna clase.

## Video 27. POO I



 **Programación orientada a procedimientos**

- Programación Orientada a procedimientos:
  - Algunos ejemplos de lenguajes: Fortran, Cobol, Basic etc.
  - **Ventajas:**
    - 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 de la vida real al código de programación.
- ¿Cuál es la naturaleza de un objeto de 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**



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

**Modularización:** Dividir un gran programa en diferentes partes que se conectan entre sí, para formar un todo.

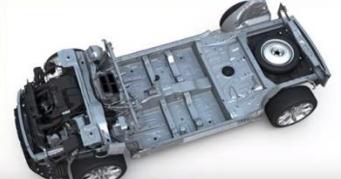
 **Vocabulario de la POO**

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

## Video 28. POO II

### Clase

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



### Objeto

- Ejemplar perteneciente a una clase



Clase → Objeto

### Objeto

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



Objeto →

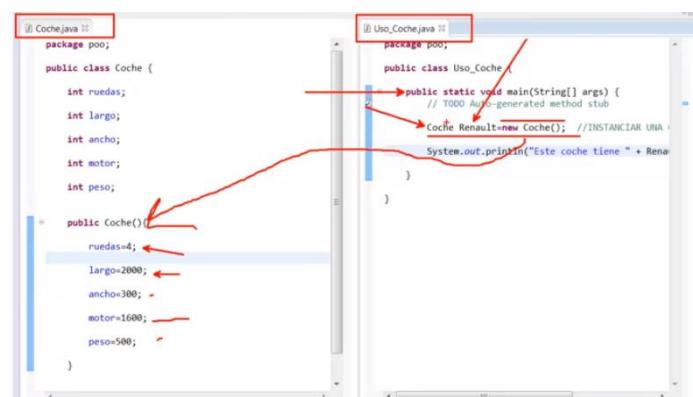
### Objeto.

Accediendo a propiedades y comportamiento (pseudocódigo)

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

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

## Video 29. POO III



```
Coche.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;
    }
}
```

```
Uso_Coche.java
package poo;

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

```

Coche.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;
}

Uso_Coche.java
package poo;
public class Uso_Coche {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Coche Renault=new Coche(); //INSTANCIAR UNA COCHE
        System.out.println("Este coche tiene " + Renault);
    }
}

```

**Modularización:** Ya realizamos la modularización al separar en dos archivos: Coche.java y UsoCoche.java (main)

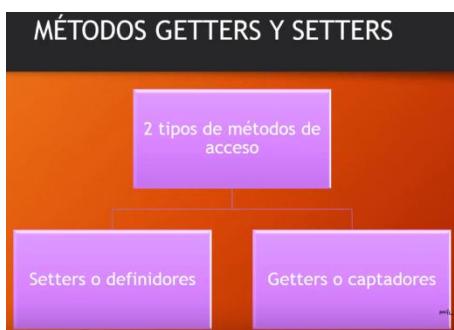
**\*\* Encapsulación:** Se realiza la encapsulación colocando el private en los atributos... Con el objetivo de que no se pueda modificar el valor de las propiedades desde fuera de la clase.

### Video 30. POO IV Getters y Setters

Tuesday, August 16, 2016

Vemos en este vídeo el uso de métodos getters y setters para acceder a las propiedades de los objetos.

La forma que tiene de comunicarse entre sí las clases es a través de los métodos (concepto relacionado a la encapsulación).



**Get:** Proporcionar el valor de esa propiedad.

Desde un getter obtenemos el valor de una propiedad.



Ejemplo externo:

```

46 //Métodos públicos
47
48 //métodos get de todos los atributos
49
50     public double getPrecioBase() {
51         return precioBase;
52     }
53
54     public String getColor() {
55         return color;
56     }
57
58     public char getConsumoEnergetico() {
59         return consumoEnergetico;
60     }
61
62     public double getPeso() {
63         return peso;
64     }

```

## Ejemplo del video:

```
Coche.java [ ] Uso_Coche.java [ ]
1 package poo_II_28;
2
3 public class Coche {
4     //Se realiza la encapsulación colocando el private en los atributos, encapsulamos esta variable.. propiedad
5     //Con esto no se puede modificar fuera de la clase
6     private int ruedas;
7     private int largo;
8     private int ancho;
9     private int motor;
10    private int peso;
11
12    //método constructor, método especial que se encarga de dar un estado inicial a nuestro objeto
13    public Coche(){
14        ruedas=4;
15        largo=2000;
16        ancho=300;
17        motor=1600;
18        peso=500;
19    }
20
21    //método de tipo getter
22    //modificador de acceso public permite utilizar este método en cualquier otra clase
23    public String dime_largo(){
24        return "El largo del coche es: " + largo;
25    }
26 }
```

```
Coche.java [ ] Uso_Coche.java [ ]
1 package PildorasInformaticas/src/poo_II_28;
2
3 public class Uso_Coche {
4
5     public static void main(String[] args) {
6         // Ya realizamos la modularización al separar en dos archivos: Coche.java y UsoCoche.java (main)
7         Coche renault=new Coche(); //instanciar una clase, ejemplar de clase
8         //renault.ruedas=3;
9
10        System.out.println(renault.dime_largo());
11        //System.out.println("Este coche tiene "+renault.ruedas+" ruedas");
12    }
13 }
```

Problems @ Javadoc Declaration Console Progress

<terminated> Uso\_Coche [Java Application] C:\Java\jre\bin\javaw.exe (16 ago. 2016 16:03:53)

El largo del coche es: 2000

La forma de acceder a las propiedades de los objetos (clases), es a través de sus métodos.

**Set:** Encargado de definir o establecer el valor de una propiedad.

Desde un setter modificamos el valor de una propiedad.

## 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

Ejemplo externo:

```
1 package password_1;
2
3 public class Password {
4     //Constantes, longitud por defecto
5     private final static int LONG_DEF = 8;
6
7     //Atributos
8     private int longitud;
9     private String contraseña;
10
11    //getters y setters, métodos públicos
12    public int getLongitud() {
13        return longitud;
14    }
15
16    public void setLongitud(int longitud) {
17        this.longitud = longitud;
18    }
19
20    public String getContraseña() {
21        return contraseña;
22    }
}
```

## Ejemplo del video:

```
Coche.java  Uso_Coche.java
1 package poo_IV_30;
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     private int peso_plataforma;
10
11    //propiedades que pueden variar, estas propiedades deberán encapsularse (private)
12    //para este ejemplo se dejó así
13    String color;
14    int peso_total;
15    boolean asientos_cuero, climatizador;
16
17    public Coche(){
18        ruedas=4;
19        largo=2000;
20        ancho=300;
21        motor=1600;
22        peso=500;
23        peso_plataforma=500;
24    }
25
26    //método tipo setter, encargado de modificar el valor de una propiedad
27    //void indica que el método no devuelve ningún valor
28    public void establece_color(){
29        color="azul";
30    }
31
32    //método tipo getter
33    public String dime_color(){
34        return "El color del coche es: " + color;
35    }
36 }
```

```
Coche.java  Uso_Coche.java
1 package PildorasInformaticas/src/poo_IV_30/Coche.java;
2
3 public class Uso_Coche {
4
5     public static void main(String[] args) {
6         Coche micoche = new Coche();
7         micoche.establece_color();
8         System.out.println(micoche.dime_color());
9     }
10
11 }
```

Problems @ Javadoc Declaration Console Program  
<terminated> Uso\_Coche(1) [Java Application] C:\Java\jre\bin\javaw.exe  
El color del coche es: azul

## Video 31. POO V Paso de parámetros

```
*Coche.java
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     private int peso_plataforma;
10
11    public String dime_largo(){ //GETTER
12        return "El largo del coche es " + largo;
13    }
14
15    public void establece_color(String color_coche){
16        color=color_coche;
17    }
18
19    public String dime_color(){
20        return "El color del coche es " + color;
21    }
22 }
```

```
*Uso_Coche.java
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 micoche=new Coche();
8         micoche.establece_color("amarillo");
9         System.out.println(micoche.dime_color());
10    }
11 }
```

## Ejemplo del video:

The screenshot shows an IDE interface with two tabs: "Coche.java" and "Uso\_Coche.java". The "Coche.java" tab is active, displaying the following code:

```
1 package poo_V_31;
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     private int peso_plataforma;
10
11    private String color;
12    private int peso_total;
13    private boolean asientos_cuero, climatizador;
14
15    public Coche(){
16        ruedas=4;
17        largo=2000;
18        ancho=300;
19        motor=1600;
20        peso=500;
21        peso_plataforma=500;
22    }
23
24    //get
25    public String dime_datos_generales(){
26        return "La plataforma del vehículo tiene: " + ruedas + " ruedas" +
27            ". Mide " + largo/1000 + " metros, con un ancho de " + ancho +
28            " cm y un peso de plataforma de " + peso_plataforma + " Kg.";
29    }
30
31    //set
32    //colocamos un parámetro o argumento en los paréntesis (zona de parámetros)
33    public void establece_color(String color_coche){
34        color=color_coche;
35    }
36
37    //get
38    public String dime_color(){
39        return "El color del coche es: " + color;
40    }
41
42    //set asientos cuero
43    //this hace referencia a la propia clase, estamos haciendo referencia a la variable de la clase
44    public void configura_asientos(String asientos_cuero){
45        if(asientos_cuero=="si"){
46            this.asientos_cuero=true;
47        }else{
48            this.asientos_cuero=false;
49        }
50    }
51    //get
52    public String dime_asientos(){
53        if(asientos_cuero==true){
54            return "El coche tiene asientos de cuero";
55        }else{
56            return "El coche tiene asientos de serie";
57        }
58    }
59 }
```

The screenshot shows the "Uso\_Coche.java" tab active, displaying the following code:

```
1 package poo_V_31;
2
3 public class Uso_Coche {
4
5    public static void main(String[] args) {
6        Coche micoche = new Coche();
7        micoche.establece_color("verde");
8        System.out.println(micoche.dime_datos_generales());
9        System.out.println(micoche.dime_color());
10
11        micoche.configura_asientos("si");
12        System.out.println(micoche.dime_asientos());
13    }
14 }
```

Below the code, the "Console" tab is open, showing the output of the program:

```
<terminated> Uso_Coche (2) [Java Application] C:\Java\jre\bin\javaw.exe (16 ago. 2016 18:04:36)
La plataforma del vehículo tiene: 4 ruedas. Mide 2 metros, con un ancho de 300 cm y un peso de plataforma de 500 Kg.
El color del coche es: verde
El coche tiene asientos de cuero
```

## Video 32. POO VI Construcción de Objetos

The screenshot shows a Java development environment with two tabs open: "Coche.java" and "Uso\_Coche.java". The "Coche.java" tab is active, displaying the following code:

```
1 package poo_VI_32;
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     private int peso_plataforma;
10
11    private String color;
12    private int peso_total;
13    private boolean asientos_cuero, climatizador;
14
15    public Coche(){
16        ruedas=4;
17        largo=2000;
18        ancho=300;
19        motor=1600;
20        peso=500;
21        peso_plataforma=500;
22    }
23
24    //get
25    public String dime_datos_generales(){
26        return "La plataforma del vehículo tiene: " + ruedas + " ruedas" +
27            ". Mide " + largo/1000 + " metros, con un ancho de " + ancho +
28            " cm y un peso de plataforma de " + peso_plataforma + " Kg.";
29    }
30
31    //set
32    //colocamos un parámetro o argumento en los paréntesis (zona de parámetros)
33    public void establece_color(String color_coche){
34        color=color_coche;
35    }
36
37    //get
38    public String dime_color(){
39        return "El color del coche es: " + color;
40    }
41
42    //set asientos_cuero
43    //this hace referencia a la propia clase, estamos haciendo referencia a la variable de la clase
44    public void configura_asientos(String asientos_cuero){
45        if(asientos_cuero.equalsIgnoreCase("si")){
46            this.asientos_cuero=true;
47        }else{
48            this.asientos_cuero=false;
49        }
50    }
51    //get
52    public String dime_asientos(){
53        if(asientos_cuero==true){
54            return "El coche tiene asientos de cuero";
55        }else{
56            return "El coche tiene asientos de serie";
57        }
58    }
59    //set climatizador
60    public void configura_climatizador(String climatizador){
61        if(climatizador.equalsIgnoreCase("si")){
62            this.climatizador=true;
63        }else{
64            this.climatizador=false;
65        }
66    }
67
68    //get climatizador
69    public String dime_climatizador(){
70        if(climatizador==true){
71            return "El coche incorpora climatizador";
72        }else{
73            return "El coche lleva aire acondicionado";
74        }
75    }
76    //setter y getter (no recomendada), se está estableciendo el valor de una variable,
77    //y estamos obteniendo un dato (return)
78    public String dime_peso_coche(){
79        //ámbito de esta variable es local, sólo se ve en este método
80        int peso_carroceria=500;
81        peso_total=peso_plataforma+peso_carroceria;
82        if(asientos_cuero==true){
83            peso_total=peso_total+50;
84        }
85        if(climatizador==true){
86            peso_total=peso_total+20;
87        }
88        return "El peso del coche es: " + peso_total;
89    }
90    //getter
91    public int precio_coche(){
92        int precio_final=10000;
93        if(asientos_cuero==true){
94            precio_final+=2000;
95        }
96        if(climatizador==true){
97            precio_final+=1500;
98        }
99        return precio_final;
100    }
101}
```

Coche.java    Uso\_Coche.java

```

1 package PildorasInformaticas/src/poo_VI_32/Coche.java
2
3 import javax.swing.*;
4
5 public class Uso_Coche {
6
7     public static void main(String[] args) {
8         Coche micoche = new Coche();
9         micoche.establece_color(JOptionPane.showInputDialog("Introduce color"));
10        System.out.println(micoche.dime_datos_generales());
11        System.out.println(micoche.dime_color());
12
13        micoche.configura_asientos(JOptionPane.showInputDialog("¿Tiene asientos de cuero?"));
14        System.out.println(micoche.dime_asientos());
15
16        micoche.configura_climatizador(JOptionPane.showInputDialog("¿Tiene climatizador?"));
17        System.out.println(micoche.dime_climatizador());
18
19        System.out.println(micoche.dime_peso_coche());
20
21        System.out.println("El precio final del coche es: " + micoche.precio_coche());
22    }
23 }
```

Problems    Javadoc    Declaration    Console    Progress

<terminated> Uso\_Coche [3] [Java Application] C:\Java\jre\bin\javaw.exe (17 ago. 2016 20:17:34)

La plataforma del vehículo tiene: 4 ruedas. Mide 2 metros, con un ancho de 300 cm y un peso de plataforma de 500 Kg.  
 El color del coche es: verde  
 El coche tiene asientos de cuero  
 El coche incorpora climatizador  
 El peso del coche es: 1070  
 El precio final del coche es: 13500

```

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

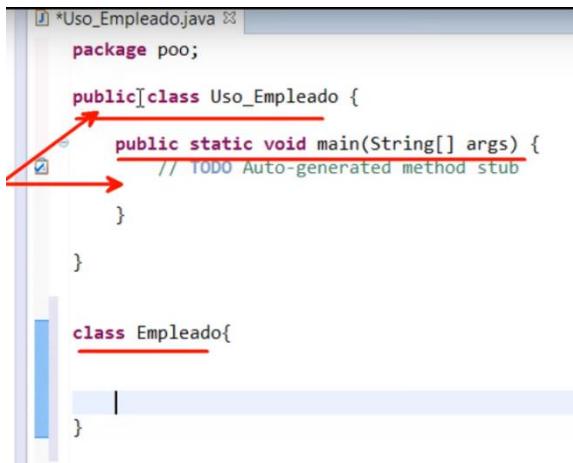
public String dime_asientos(){ //GETTER
    if(asientos_cuero==true){
        return "El coche tiene asientos de cuero";
    }else{
        return "El coche tiene asientos de serie";
    }
}
```

if(asientos\_cuero){

### Video 33. POO VII. Construcción objetos II, ficheros fuente, constructores

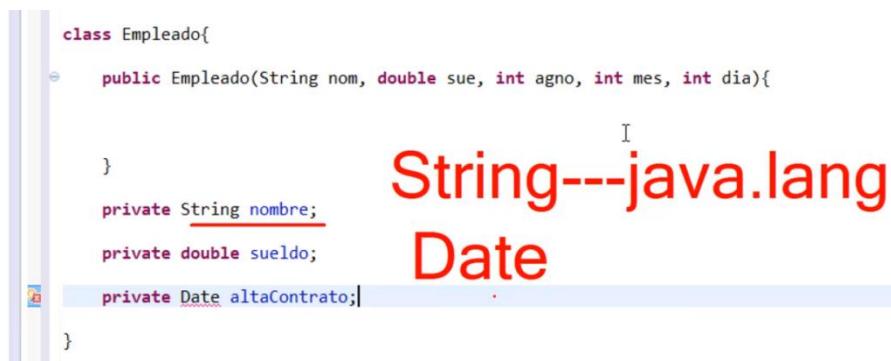


Programa en java para establecer las nóminas de empleados, características de Empleado son: Nombre, Sueldo, Fecha Alta. También un método para subir sueldo del empleado que se construyó.



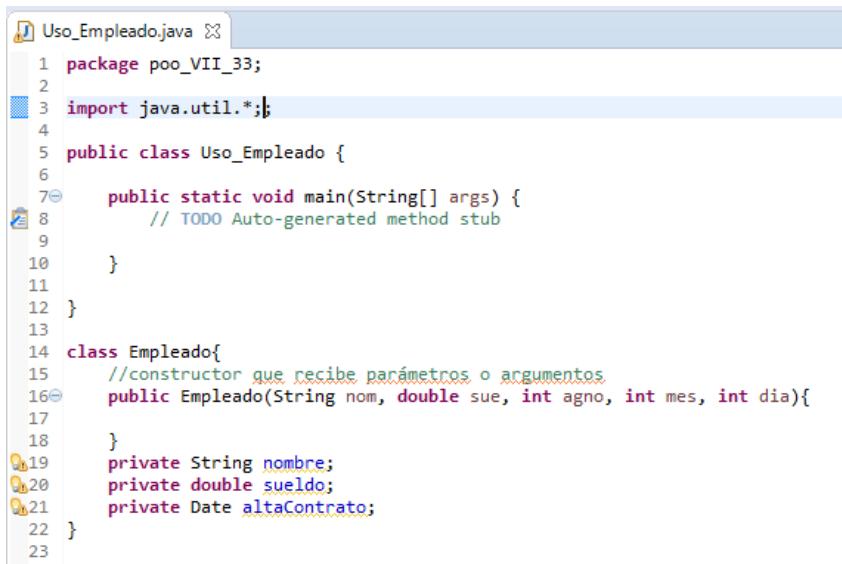
```
1 *Uso_Emppleado.java ✘ |  
2 package poo;  
3  
4 public class Uso_Emppleado {  
5     public static void main(String[] args) {  
6         // TODO Auto-generated method stub  
7     }  
8 }  
9  
10 class Empleado{  
11 }  
12 }
```

En caso de realizarse todo en un solo archivo (no recomendado), sólo una clase puede ser pública y el main en solo una clase.



```
1 class Empleado{  
2     public Empleado(String nom, double sue, int agno, int mes){  
3     }  
4     private String nombre;  
5     private double sueldo;  
6     private Date altaContrato;  
7 }
```

## String--java.lang Date



```
1 package poo_VII_33;  
2  
3 import java.util.*;  
4  
5 public class Uso_Emppleado {  
6  
7     public static void main(String[] args) {  
8         // TODO Auto-generated method stub  
9     }  
10 }  
11  
12 class Empleado{  
13     //constructor que recibe parámetros o argumentos  
14     public Empleado(String nom, double sue, int agno, int mes){  
15     }  
16     private String nombre;  
17     private double sueldo;  
18     private Date altaContrato;  
19 }
```

The screenshot shows the Java API documentation for the `Date` class. The URL is [docs.oracle.com/javase/8/docs/api/](https://docs.oracle.com/javase/8/docs/api/). The page title is "Date (Java Platform SE 8)". The navigation bar includes links for Overview, Package, Class (which is selected), Use, Tree, Deprecated, Index, and Help. Below the navigation bar, there are links for Prev Class, Next Class, Frames, and No Frames. The summary section indicates that the class has nested fields (`compact1`, `compact2`, `compact3`), methods (`get`, `set`, etc.), and constructors (`getConstructors`). The class implements `Serializable`, `Cloneable`, and `Comparable<Date>`. It also has subclasses: `Date`, `Time`, and `Timestamp`. The class extends `java.lang.Object` and implements `java.util.Date`. A note states: "The class Date represents a specific instant in time, with millisecond precision."

## Video 34. POO VIII. Construcción objetos III, ficheros fuente, constructores

Utilizamos el constructor `GregorianCalendar(int year, int month, int dayOfMonth)`:

The screenshot shows the Java API documentation for the `GregorianCalendar` class. The URL is <https://docs.oracle.com/javase/8/docs/api/java/util/GregorianCalendar.html>. The page title is "Constructor Summary". The navigation bar includes links for Constructors (selected) and Constructor and Description. The constructor `GregorianCalendar()` constructs a default `GregorianCalendar` using the current time in the default time zone with the default `FORMAT` locale. The constructor `GregorianCalendar(int year, int month, int dayOfMonth)` constructs a `GregorianCalendar` with the given date set in the default time zone with the default locale. The constructor `GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute)` constructs a `GregorianCalendar` with the given date and time set for the default time zone with the default locale. The constructor `GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute, int second)` constructs a `GregorianCalendar` with the given date and time set for the default time zone with the default locale. The constructor `GregorianCalendar(Locale aLocale)` constructs a `GregorianCalendar` based on the current time in the default time zone with the given locale. The constructor `GregorianCalendar(TimeZone zone)` constructs a `GregorianCalendar` based on the current time in the given time zone with the default `FORMAT` locale. The constructor `GregorianCalendar(TimeZone zone, Locale aLocale)` constructs a `GregorianCalendar` based on the current time in the given time zone with the given locale.

Se hace uso del método `getTime`, que no se encuentra en la clase `GregorianCalendar`:

GregorianCalendar (Java Platform SE 6)

**OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP**

**PREV CLASS NEXT CLASS FRAMES NO FRAMES**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3  
java.util

**Class GregorianCalendar**

java.lang.Object  
  java.util.Calendar  
    java.util.GregorianCalendar

**All Implemented Interfaces:**  
Serializable, Cloneable, Comparable<Calendar>

---

public class **GregorianCalendar**  
extends Calendar

Pero que esta heredando de la clase `java.util.Calendar`, que a su vez hereda de la clase `java.lang.Object`, esto quiere decir que tiene por herencia los métodos y propiedades de la clase `Object` y de la clase `Calendar` mas los suyos propios.

**Herencia:** Característica de la POO, mediante la cual se construye una clase que herede de otra, se hereda todas las propiedades y todos los métodos de la clase padre.

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

**java.util**

**Class GregorianCalendar**

java.lang.Object  
  java.util.Calendar  
    java.util.GregorianCalendar

**All Implemented Interfaces:**  
Serializable, Cloneable, Comparable<Calendar>

---

public class **GregorianCalendar**  
extends Calendar

Herencia

getTime()

`getTime()` hereda de `calendar`, que es tipo `Date`

Calendar (Java Platform SE 6)

**OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP**

**PREV CLASS NEXT CLASS FRAMES NO FRAMES**

Returns the minimum value for the given calendar field of this Calendar instance.

**Date**

**getTime()**  
Returns a Date object representing this Calendar's time value (millisecond offset from the Epoch).

---

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

}

private String nombre;

private double sueldo;

private Date altaContrato;

```

6
7      //metodo de acceso public
8  public static void main(String[] args) {
9      // TODO Auto-generated method stub
10 }
11 }
12 class Empleado{
13     //constructor que recibe parámetros o argumentos
14     public Empleado(String nom, double sue, int agno, int mes, int dia){
15         nombre=nom;
16         sueldo=sue;
17         GregorianCalendar calendario=new GregorianCalendar(agno, mes-1, dia);
18         altaContrato=calendario.getTime();
19     }
20     //getter
21     public String dameNombre(){
22         return nombre;
23     }
24     //getter
25     public double dameSueldo(){
26         return sueldo;
27     }
28     //getter
29     public Date dameFechaContrato(){
30         return altaContrato;
31     }
32     //setter, para aumentar el sueldo
33     public void subeSueldo(double porcentaje){
34         double aumento=sueldo*porcentaje/100;
35         sueldo+=aumento;
36     }
37     private String nombre;
38     private double sueldo;
39     private Date altaContrato;
40 }
41

```

### Video 35. POO IX. Construcción objetos IV

```

public class Uso_Emppleado {

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

        Empleado empleado1=new Empleado("Paco Gómez", 85000, 1990, 12, 17);
    }
}

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);
    }
}

```

```
Uso_Emppleado.java 33
1 package poo_IIX_35;
2
3 import java.util.*;
4
5 public class Uso_Emppleado {
6
7     //metodo de acceso public
8     public static void main(String[] args) {
9         Empleado empleado1=new Empleado("Ariel", 85000, 1990, 12, 17);
10        Empleado empleado2=new Empleado("Ana", 95000, 1980, 3, 11);
11        Empleado empleado3=new Empleado("Oswaldo", 55000, 2012, 7, 3);
12        empleado1.subeSueldo(5);
13        empleado2.subeSueldo(10);
14        empleado3.subeSueldo(5);
15        System.out.println("Nombre: "+empleado1.dameNombre()+" Sueldo: "+empleado1.dameSueldo()+
16                            " Fecha de alta: "+empleado1.dameFechaContrato());
17        System.out.println("Nombre: "+empleado2.dameNombre()+" Sueldo: "+empleado2.dameSueldo()+
18                            " Fecha de alta: "+empleado2.dameFechaContrato());
19        System.out.println("Nombre: "+empleado3.dameNombre()+" Sueldo: "+empleado3.dameSueldo()+
20                            " Fecha de alta: "+empleado3.dameFechaContrato());
21    }
22 }
```

Problems @ Javadoc Declaration Console

<terminated> Uso\_Emppleado [Java Application] C:\Program Files\Java\jre1.8.0\_91\bin\javaw.exe (19 ago. 2016 17:59:17)

Nombre: Ariel Sueldo: 89250.0 Fecha de alta: Mon Dec 17 00:00:00 BOT 1990  
Nombre: Ana Sueldo: 104500.0 Fecha de alta: Tue Mar 11 00:00:00 BOT 1980  
Nombre: Oswaldo Sueldo: 57750.0 Fecha de alta: Tue Jul 03 00:00:00 BOT 2012

```
Uso_Emppleado.java ✘ |
```

```
1 package poo_IX_35;
2
3 import java.util.*;
4
5 public class Uso_Emppleado {
6
7     //metodo de acceso public
8     public static void main(String[] args) {
9         /*Empleado empleado1=new Empleado("Ariel", 85000, 1990, 12, 17);
10        Empleado empleado2=new Empleado("Ana", 95000, 1980, 3, 11);
11        Empleado empleado3=new Empleado("Oswaldo", 55000, 2012, 7, 3);
12        empleado1.subeSueldo(5);
13        empleado2.subeSueldo(10);
14        empleado3.subeSueldo(5);
15        System.out.println(
16            "Nombre: "+empleado1.dameNombre()+" Sueldo: "+empleado1.dameSueldo()+
17            " Fecha de alta: "+empleado1.dameFechaContrato());
18        System.out.println("Nombre: "+empleado2.dameNombre()+" Sueldo: "+empleado2.dameSueldo()+
19            " Fecha de alta: "+empleado2.dameFechaContrato());
20        System.out.println("Nombre: "+empleado3.dameNombre()+" Sueldo: "+empleado3.dameSueldo()+
21            " Fecha de alta: "+empleado3.dameFechaContrato());*/
22        Empleado[] misEmpleados=new Empleado[3];
23        misEmpleados[0]=new Empleado("Ariel", 85000, 1990, 12, 17);
24        misEmpleados[1]=new Empleado("Ana", 95000, 1980, 3, 11);
25        misEmpleados[2]=new Empleado("Oswaldo", 55000, 2012, 7, 3);
26
27        /*for(int i=0; i<misEmpleados.length; i++){
28            misEmpleados[i].subeSueldo(5);
29        }*/
30        for(Empleado e: misEmpleados){
31            e.subeSueldo(5);
32        }
33
34        /*for(int i=0; i<misEmpleados.length; i++){
35            System.out.println("Nombre: "+misEmpleados[i].dameNombre()+" Sueldo: "+
36                misEmpleados[i].dameSueldo()+" Fecha de alta: "+misEmpleados[i].dameFechaContrato());
37        }*/
38        for(Empleado e: misEmpleados){
39            System.out.println("Nombre: "+e.dameNombre()+" Sueldo: "+
40                e.dameSueldo()+" Fecha de alta: "+e.dameFechaContrato());
41        }
42    }
43 }
44 class Empleado{
45     //constructor que recibe parámetros o argumentos
46     public Empleado(String nom, double sue, int agno, int mes, int dia){
47         nombre=nom;
48         sueldo=sue;
49         GregorianCalendar calendario=new GregorianCalendar(agno, mes-1, dia);
50         altaContrato=calendario.getTime();
51     }
52     //getter
53     public String dameNombre(){
54         return nombre;
55     }
56     //getter
57     public double dameSueldo(){
58         return sueldo;
59     }
60     //getter
61     public Date dameFechaContrato(){
62         return altaContrato;
63     }
64     //setter, para aumentar el sueldo
65     public void subeSueldo(double porcentaje){
66         double aumento=sueldo*porcentaje/100;
67         sueldo+=aumento;
68     }
69     private String nombre;
70     private double sueldo;
71     private Date altaContrato;
72 }
```

## Vídeo 36. Constantes Uso final

Método tipo setter por que establece un valor en un campo.

Método tipo getter siempre devuelve un dato.

final lo convierte en constante

The screenshot shows an IDE interface with the following details:

- Pruebas.java:** The code defines a class `Pruebas` with a static main method. It creates two `Empleados` objects, `trabajador1` and `trabajador2`, and prints their names and sections using getters.
- Empleados Class:** Contains private fields `nombre` and `seccion`. It has a constructor, a `cambia_seccion` setter, and a `cambia_nombre` setter. A `devuelveDatos` getter returns a string combining name and section.
- Console Output:** Shows the execution of the program and the resulting output:

```
<terminated> Pruebas [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (21 ago. 2016 10:23:51)
El nombre es: Ariel, y la seccion es: Sistemas
El nombre es: Oswaldo, y la seccion es: Administracion
```

## Vídeo 37. Uso Static

La palabra static se puede aplicar tanto a variables, constantes y métodos.

The slide has the following content:

- Title:** Campos static
- Code Example:** Shows a class definition with three fields: `private final String nombre;`, `private String seccion;`, and `private int Id;`. A red arrow points from the `Id` field to the `private static int Id=1;` declaration in the code below.
- Execution Output:** Displays two lines of Java code:

```
Empleados trabajador1=new Empleados("Paco",1);
Empleados trabajador2=new Empleados("Ana",2);
```
- Annotations:** Several `private final String nombre;` and `private String seccion;` annotations are scattered across the slide, likely indicating static variable usages.

Cada uno de estos objetos (trabajador1 y trabajador2) tiene una copia de cada variable declarada, trabajador 1 tiene una copia de la variable nombre, de la variable sección, lo mismo ocurre con trabajador2.

Esto implica que como cada objeto recibe su copia, estas copias son diferentes, por ejemplo, en copia de nombre de trabajador1 se almaceno Ariel y en copia de nombre de trabajador se almaceno Oswaldo.

Con static se comparte la variable id, static implica que no tengan los objetos cada uno su copia de id y que comparten la variable id.

Tenemos una variable estática id que pertenece a la clase Empleados y que es única para cada objeto perteneciente a la clase Empleados, mientras que el resto de variables cada objeto tiene su propia copia, id es compartida por todos los objetos pertenecientes a la clase Empleados.

The screenshot shows the Eclipse IDE interface. The top part displays the code for `Pruebas.java`. The bottom part shows the `Console` view with the application's output.

```
1 package static_37;
2 public class Pruebas {
3     public static void main(String[] args) {
4         Empleados trabajador1 = new Empleados("Ariel");
5         Empleados trabajador2 = new Empleados("Oswaldo");
6         Empleados trabajador3 = new Empleados("Teodoro");
7         trabajador1.cambia_seccion("Sistemas");
8         System.out.println(trabajador1.devuelveDatos());
9         System.out.println(trabajador2.devuelveDatos());
10        System.out.println(trabajador3.devuelveDatos());
11    }
12 }
13 class Empleados{
14     private final String nombre;
15     private String seccion;
16     private int id;
17     private static int idSiguiente=1;//eclipse pone en cursiva lo que está con static
18     //constructor
19     public Empleados(String nom){
20         nombre=nom;
21         seccion="Administracion";
22         id=idSiguiente;
23         idSiguiente++;
24     }
25     //set
26     public void cambia_seccion(String seccion){
27         this.seccion=seccion;
28     }
29     //get
30     public String devuelveDatos(){
31         return "El nombre es: "+nombre+", la seccion es: "+seccion+", y el ID es igual a: "+id;
32     }
33 }
```

<terminated> Pruebas (1) [Java Application] C:\Program Files\Java\jre1.8.0\_91\bin\javaw.exe (21 ago. 2016 20:49:24)

```
El nombre es: Ariel, la seccion es: Sistemas, y el ID es igual a: 1
El nombre es: Oswaldo, la seccion es: Administracion, y el ID es igual a: 2
El nombre es: Teodoro, la seccion es: Administracion, y el ID es igual a: 3
```

### Vídeo 38. Métodos Static

Al igual que con las variables y constantes static, ocurre lo mismo con los métodos static, el método pertenece a la clase, no pertenece a ningún objeto.

El método main no actúa sobre ningún objeto.

El método static no puede acceder a las variables, a no ser que la variable o constante que acceda sea también static. En este caso `idSiguiente` es static.

```
Pruebas.java
```

```
    this.seccion=seccion;

}

public String devuelveDatos(){ //getter
    return "El nombre es: " + nombre + " la sección es " + seccion + " y el Id=" +Id;

}

public static String dameIdSiguiente(){
    return "El IdSiguiente es: " + IdSiguiente;
}

private final String nombre; ←
private String seccion; ←
private int Id; ←
```

## Métodos static

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\_método

### Vídeo 39. Sobre carga de constructores

El constructor debe llevar el mismo nombre de la clase, con eso se diferencia de un método no constructor, además un constructor no lleva void ni debe devolver ningún tipo de datos.

Los métodos constructores deben tener diferentes números de argumentos.

Para dar a los objetos diferentes estados iniciales.

```

1 package sobrecarga_constructores_39;
2 import java.util.*;
3 public class Uso_Emppleado {
4     public static void main(String[] args) {
5         Empleado[] misEmpleados=new Empleado[4];
6         misEmpleados[0]=new Empleado("Ariel", 85000, 1990, 12, 17);
7         misEmpleados[1]=new Empleado("Ana", 95000, 1980, 3, 11);
8         misEmpleados[2]=new Empleado("Oswaldo", 55000, 2012, 7, 3);
9         misEmpleados[3]=new Empleado("Rodrigo");
10        for(Empleado e: misEmpleados){
11            e.subeSueldo(5);
12        }
13        for(Empleado e: misEmpleados){
14            System.out.println("Nombre: "+e.dameNombre()+" Sueldo: "+
15                e.dameSueldo()+" Fecha de alta: "+e.dameFechaContrato());
16        }
17    }
18 }
19 class Empleado{
20     //constructor que recibe parámetros o argumentos
21     public Empleado(String nom, double sue, int agno, int mes, int dia){
22         nombre=nom;
23         sueldo=sue;
24         GregorianCalendar calendario=new GregorianCalendar(agno, mes-1, dia);
25         altaContrato=calendario.getTime();
26     }
27     //si solo se conoce nombre
28     public Empleado(String nom){
29         //this está llamando al otro constructor de la clase y pasarle los parámetros correspondientes
30         this(nom, 30000, 2000, 01, 01);
31     }
32     //getter
33     public String dameNombre(){
34         return nombre;
35     }
36     //getter
37     public double dameSueldo(){
38         return sueldo;
39     }
40     //getter
41     public Date dameFechaContrato(){
42         return altaContrato;
43     }
44     //setter, para aumentar el sueldo
45     public void subeSueldo(double porcentaje){
46         double aumento=sueldo*porcentaje/100;
47         sueldo+=aumento;
48     }
49     private String nombre;
50     private double sueldo;
51     private Date altaContrato;
52 }

```

Problems Javadoc Declaration Console

```

<terminated> Uso_Emppleado (1) [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (22 ago. 2016 14:58:34)
Nombre: Ariel Sueldo: 89250.0 Fecha de alta: Mon Dec 17 00:00:00 BOT 1990
Nombre: Ana Sueldo: 99750.0 Fecha de alta: Tue Mar 11 00:00:00 BOT 1980
Nombre: Oswaldo Sueldo: 57750.0 Fecha de alta: Tue Jul 03 00:00:00 BOT 2012
Nombre: Rodrigo Sueldo: 31500.0 Fecha de alta: Sat Jan 01 00:00:00 BOT 2000

```

## Vídeo 40. Herencia I

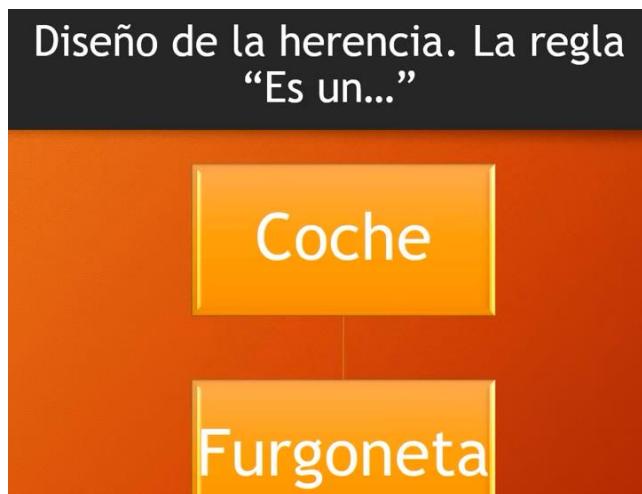


Java no admite la herencia múltiple, o sea no puede heredar de dos o más clases, sólo de una clase puede heredar. Para solventar esto en java se utiliza las interfaces y otras técnicas de programación.

```
Furgoneta.java
```

```
1 package herencia_I_40;
2
3 //clase Coche se convierte en superclase (clase padre)
4 //la clase furgoneta se convierte en subclase (clase hijo)
5 public class Furgoneta extends Coche{
6     private int capacidad_carga;
7     private int plazas_extra;
8     public Furgoneta(int plazas_extra, int capacidad_carga){
9         //llama al constructor de la clase Coche (padre), con el objetivo de dar un
10        //esta inicial
11        super();
12        this.capacidad_carga=capacidad_carga;
13        this.plazas_extra=plazas_extra;
14    }
15 }
```

#### Vídeo 41. Herencia II



¿Una furgoneta es un coche?

Una furgoneta no es del todo un coche, eso quiere decir que nuestras clases no están bien diseñadas, por lo menos la herencia de las clases no está bien diseñada.

Un diseño correcto:



¿Un coche es un vehículo? Si

¿Una furgoneta es un vehículo? Si

Nos permitirá implementar otras clases en el futuro, si es necesario (camión y moto).

```
Furgoneta.java Uso_Vehiculo.java Coche.java
1 package herencia_II_41;
2
3 //clase Coche se convierte en superclase (clase padre)
4 //la clase furgoneta se convierte en subclase (clase hijo)
5 public class Furgoneta extends Coche{
6     private int capacidad_carga;
7     private int plazas_extra;
8     public Furgoneta(int plazas_extra, int capacidad_carga){
9         //llama al constructor de la clase Coche (padre), con el objetivo de dar un
10        //esta inicial
11        super();
12        this.capacidad_carga=capacidad_carga;
13        this.plazas_extra=plazas_extra;
14    }
15    //get
16    public String dime_datos_furgoneta(){
17        return ("La capacidad de carga es: "+capacidad_carga+", y las plazas son: "+plazas_extra);
18    }
19 }
```

```
Furgoneta.java Uso_Vehiculo.java Coche.java
1 package herencia_II_41;
2
3 public class Uso_Vehiculo {
4
5     public static void main(String[] args) {
6         Coche micoche1=new Coche();
7         micoche1.establece_color("rojo");
8         Furgoneta mifurgoneta1=new Furgoneta(7, 580);
9         mifurgoneta1.establece_color("azul");
10        mifurgoneta1.configura_asientos("si");
11        mifurgoneta1.configura_climatizador("si");
12        System.out.println(micoche1.dime_datos_generales()+" "+micoche1.dime_color());
13        System.out.println(mifurgoneta1.dime_datos_generales()+" "+mifurgoneta1.dime_datos_furgoneta());
14    }
15 }
16
```

## Vídeo 42. Herencia III Diseñando la herencia

```
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 crear Jefes?

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

¿Me sirve la clase Empleado para crear Jefes?



## Diseñando la herencia. Clase Jefatura

Recuerda: Regla “Es un...”

¿Un Jefe **es un** Empleado? (siempre)

¿Un Empleado **es un** Jefe? (siempre)



¿Un empleado es siempre un jefe? No

¿Un jefe es siempre un empleado? Si

Entonces la clase Jefe es derivada de la clase Empleado



## Diseñando la herencia. Clase Jefatura

Recuerda: Regla “Es un...”

¿Un Jefe **es un** Empleado? (siempre)

¿Un Empleado **es un** Jefe? (siempre)

Superclase

Subclase

Clase Empleado

Clase Jefe

Polimorfismo

```

        super(nom, sue, agno, mes, dia);

    }

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

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

    private double incentivo;

```

~~dameSueldo~~  
dameSueldo

**Triangulo verde invertido:** Este método está sobrescribiendo o reemplazando para la clase jefatura el método que hereda de la clase empleado.

### Vídeo 43. Herencia IV. Polimorfismo y enlazado dinámico



## 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 dependiendo del contexto. Las variables objeto son polimórficas.

Empleado emp1

```
Empleado empleado1=new Empleado();
```

Empleado1 (nombre de la variable objeto), dependiendo del contexto donde se utilice, se puede comportar de una forma u otra, puede adoptar una forma u otra y de ahí viene el concepto de polimorfismo.

Podemos almacenar tanto empleados como objetos de la subclase Jefatura, como Jefe\_RRHH que ya está construido e instanciado.

```
misEmpleados[0]=new Empleado( Ana , 31500, 2000, 07, 07 );
misEmpleados[1]=new Empleado("Carlos",50000, 1995, 06, 15);
misEmpleados[2]=new Empleado("Paco",25000, 2005, 09, 25);
misEmpleados[3]=new Empleado("Antonio",47500, 2009, 11, 09);
misEmpleados[4]=jefe_RRHH; //Polimorfismo en acción. Principio de sustitución
misEmpleados[5]=new Jefatura("María",95000, 1999,05,26);

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

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

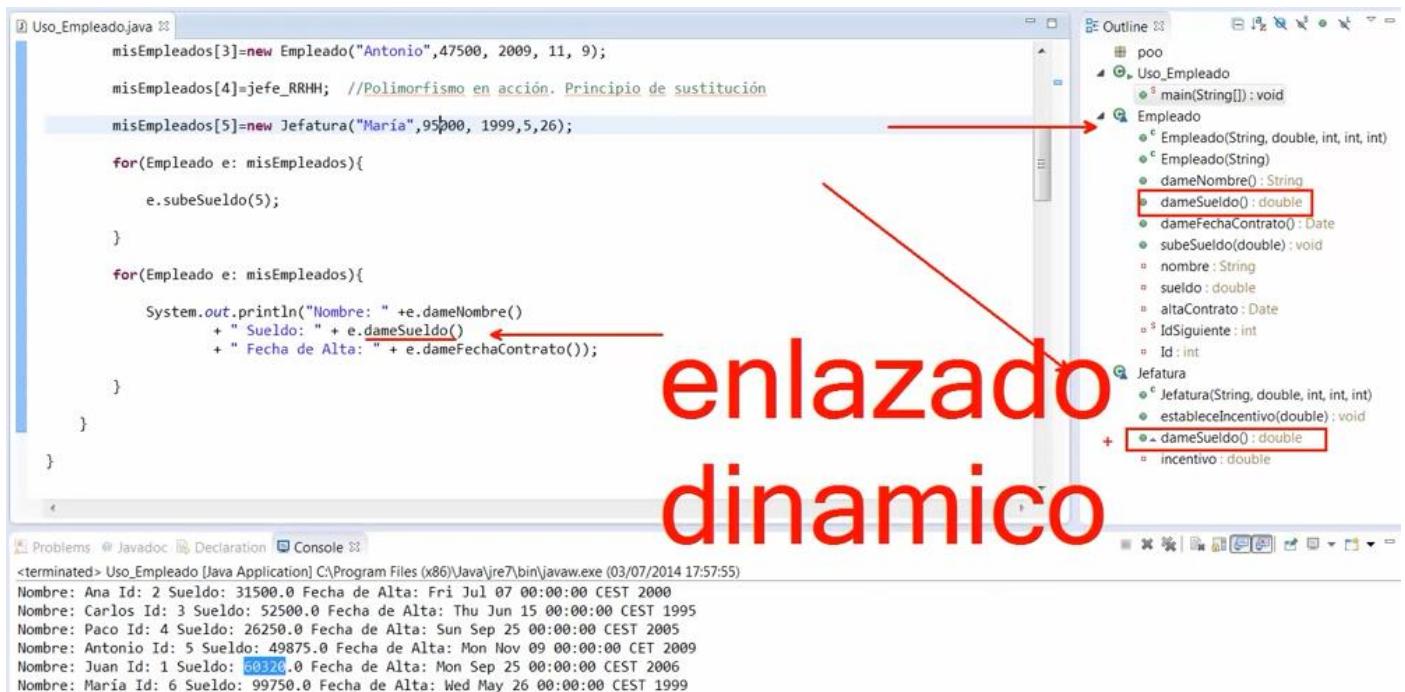
## Empleado e

Esta variable e es de tipo empleado y se usa para recorrer y se utiliza para recorrer el array misEmpleados.

Al recorrer con el bucle foreach, llama en algunos casos el método dameSueldo() de empleado y en otro caso el método dameSueldo() de jefatura (por eso la suma de Juan a 60320).

En definitiva, una variable tipo objeto se comporta de una forma u otra dependiendo del contexto, adquiere una forma u otra dependiendo de cómo se lo aplique.

Como hace la máquina virtual de java (interprete de java) para detectar en este bucle for a qué método tiene que llamar en unas ocasiones y a qué método tiene que llamar en otras, ese comportamiento lo hace java de manera automática y se llama enlazado dinámico.



**Enlazado dinámico:** La máquina virtual de java es capaz en tiempo de ejecución de saber a qué método perteneciente a la clase padre o a la subclase tiene que llamar.

```

1 package polimorfismo_enlazado_43;
2 import java.util.*;
3 public class Uso_Emppleado {
4     public static void main(String[] args) {
5         Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 9, 25);
6         jefe_RRHH.estable_incentivo(2570);
7         Empleado[] misEmpleados=new Empleado[6];
8         misEmpleados[0]=new Empleado("Ariel", 85000, 1990, 12, 17);
9         misEmpleados[1]=new Empleado("Ana", 95000, 1980, 3, 11);
10        misEmpleados[2]=new Empleado("Oswaldo", 55000, 2012, 7, 3);
11        misEmpleados[3]=new Empleado("Rodrigo");
12        /*principio de sustitucion, polimorfismo en accion,
13        almacenamos un objeto de tipo jefatura*/
14        misEmpleados[4]=jefe_RRHH;
15        misEmpleados[5]=new Jefatura("Maria", 95000, 1999, 5, 26);
16        for(Empleado e: misEmpleados){
17            e.subeSueldo(5);
18        }
19        for(Empleado e: misEmpleados){
20            System.out.println("Nombre: "+e.dameNombre()+" Sueldo: "+
21                e.dameSueldo()+" Fecha de alta: "+e.dameFechaContrato());
22        }
23    }
24 }
25 class Empleado{
26     //constructor que recibe parametros o argumentos
27     public Empleado(String nom, double sue, int agno, int mes, int dia){
28         nombre=nom;
29         sueldo=sue;
30         GregorianCalendar calendario=new GregorianCalendar(agno, mes-1, dia);
31         altaContrato=calendario.getTime();
32     }
33     //si solo se conoce nombre
34     public Empleado(String nom){
35         //this esta llamando al otro constructor de la clase y pasarle los parametros correspondientes
36         this(nom, 30000, 2000, 01, 01);
37     }
38     //getter
39     public String dameNombre(){
40         return nombre;
41     }
42     //getter
43     public double dameSueldo(){
44         return sueldo;
45     }
46     //getter
47     public Date dameFechaContrato(){
48         return altaContrato;
49     }
50     //setter, para aumentar el sueldo
51     public void subeSueldo(double porcentaje){
52         double aumento=sueldo*porcentaje/100;
53         sueldo+=aumento;
54     }
55     private String nombre;
56     private double sueldo;
57     private Date altaContrato;
58 }
59
60 class Jefatura extends Empleado{
61     public Jefatura(String nom, double sue, int agno, int mes, int dia){
62         super(nom, sue, agno, mes, dia);
63     }
64     //set
65     public void estable_incentivo(double b){
66         incentivo=b;
67     }
68     //get
69     public double dameSueldo(){
70         double sueldoJefe=super.dameSueldo(); //con super esta llamando a dameSueldo() de la clase Empleado
71         return sueldoJefe+incentivo;
72     }
73     private double incentivo;
74 }

```

Problems @ Javadoc Declaration Console

<terminated> Uso\_Emppleado (3) [Java Application] C:\Program Files\Java\jre1.8.0\_91\bin\javaw.exe (23 ago. 2016 11:28:12)

Nombre: Ariel Sueldo: 89250.0 Fecha de alta: Mon Dec 17 00:00:00 BOT 1990  
Nombre: Ana Sueldo: 99750.0 Fecha de alta: Tue Mar 11 00:00:00 BOT 1980  
Nombre: Oswaldo Sueldo: 57750.0 Fecha de alta: Tue Jul 03 00:00:00 BOT 2012  
Nombre: Rodrigo Sueldo: 31500.0 Fecha de alta: Sat Jan 01 00:00:00 BOT 2000  
Nombre: Juan Sueldo: 60320.0 Fecha de alta: Mon Sep 25 00:00:00 BOT 2006  
Nombre: Maria Sueldo: 99750.0 Fecha de alta: Wed May 26 00:00:00 BOT 1999

## Vídeo 44. Herencia V. Casting (refundición) de objetos. Clases y métodos final

Convertir un objeto de un tipo, en un objeto de un tipo diferente.

En la imagen declaramos a clase jefe con final, con lo cual ya no se puede heredar de ella:



```

Uso_Emppleado.java
1 package casting_clases_metodos_final_44;
2 import java.util.*;
3 public class Uso_Emppleado {
4     public static void main(String[] args) {
5         Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 9, 25);
6         jefe_RRHH.estable_incentivo(2570);
7         Empleado[] misEmpleados=new Empleado[6];
8         misEmpleados[0]=new Empleado("Ariel", 85000, 1990, 12, 17);
9         misEmpleados[1]=new Empleado("Ana", 95000, 1980, 3, 11);
10        misEmpleados[2]=new Empleado("Oswaldo", 55000, 2012, 7, 3);
11        misEmpleados[3]=new Empleado("Rodrigo");
12        /*principio de sustitucion, polimorfismo en accion,
13        almacenamos un objeto de tipo jefatura*/
14        misEmpleados[4]=jefe_RRHH;
15        misEmpleados[5]=new Jefatura("Maria", 95000, 1999, 5, 26);
16        //casting de objetos
17        Jefatura jefa_Finanzas=(Jefatura) misEmpleados[5];
18        jefa_Finanzas.estable_incentivo(5000);
19        for(Empleado e: misEmpleados){
20            e.subeSueldo(5);
21        }
22        for(Empleado e: misEmpleados){
23            System.out.println("Nombre: "+e.dameNombre()+" Sueldo: "+
24                e.dameSueldo()+" Fecha de alta: "+e.dameFechaContrato());
25        }
    }

```

Console

```

<terminated> Uso_Emppleado (4) [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (25 ago. 2016 20:38:07)
Nombre: Ariel Sueldo: 89250.0 Fecha de alta: Mon Dec 17 00:00:00 BOT 1990
Nombre: Ana Sueldo: 99750.0 Fecha de alta: Tue Mar 11 00:00:00 BOT 1980
Nombre: Oswaldo Sueldo: 57750.0 Fecha de alta: Tue Jul 03 00:00:00 BOT 2012
Nombre: Rodrigo Sueldo: 31500.0 Fecha de alta: Sat Jan 01 00:00:00 BOT 2000
Nombre: Juan Sueldo: 60320.0 Fecha de alta: Mon Sep 25 00:00:00 BOT 2006
Nombre: Maria Sueldo: 104750.0 Fecha de alta: Wed May 26 00:00:00 BOT 1999

```

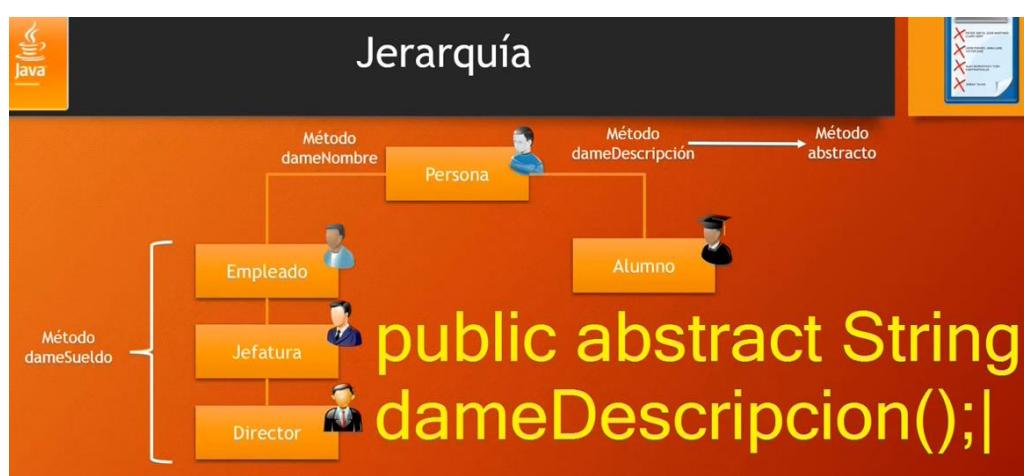
## Vídeo 45. Herencia VI. Clases Abstractas I.

Método dameNombre es común para Empleado, Jefatura, Director y Alumno.

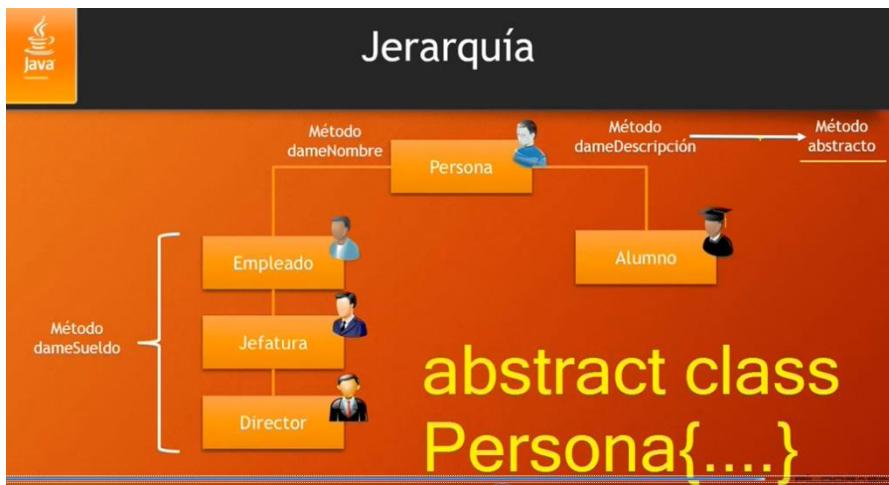
Método dameDescripcion no es común, porque tienen diferentes características.

En Java si tienes un método abstracto, tienes que declarar a la clase también abstracta.

Declaración de un método abstracto:



Declaración de una clase abstracta:



Cuando se declara un método abstracto en una clase, todas las clases que heredan están obligadas a sobrescribir ese método abstracto.

¿Todas las clases que heredan de persona deben tener un nombre?

En nuestro caso sí, entonces se crea al Método dameNombre dentro de la clase Persona.

¿Este método estará construido igual para todas las clases?

Si, con que pongamos un return nombre, será suficiente y funcionará para todas clases.

¿En este diseño de herencia necesito un método dameDescripcion para todos los objetos?

Si, necesito un método que me dé la descripción de los empleados, jefes, directores y alumnos, entonces declararé ese método en la clase Persona.

¿Esa descripción podrá estar construido de igual forma para todos los objetos?

No, porque la descripción será diferente para un empleado, jefe....

En ese caso tendré que declarar este método como abstracto, porque todas las clases que hereden de persona tienen que tener una descripción, pero será diferente en cada caso.

## Vídeo 46. Herencia VII. Clases Abstractas II.

```
Uso_Persona.java ✘
1 package clases_abstractas_II_46;
2 import java.util.Date;
3 import java.util.GregorianCalendar;
4 public class Uso_Persona {
5     public static void main(String[] args) {
6         Persona[] lasPersonas=new Persona[2];
7         lasPersonas[0]=new Empleado2("Luis Conde", 50000, 2009, 02, 25);
8         lasPersonas[1]=new Alumno("Ana Lopez", "Biologicas");
9         for(Persona p: lasPersonas){
10             System.out.println(p.dameNombre()+" "+p.dameDescripcion());
11         }
12     }
13 }
14 abstract class Persona{
15     private String nombre;
16     public Persona(String nom){
17         nombre=nom;
18     }
19     //get
20     public String dameNombre(){
21         return nombre;
22     }
23     //metodo abstracto
24     public abstract String dameDescripcion();
25 }
26 class Empleado2 extends Persona{
27     private double sueldo;
28     private Date altaContrato;
29     private int id;
30     private static int idSiguiente=1;
31     //constructor que recibe parámetros o argumentos
32     public Empleado2(String nom, double sue, int agno, int mes, int dia){
33         super(nom);
34         sueldo=sue;
35         GregorianCalendar calendario=new GregorianCalendar(agno, mes-1, dia);
36         altaContrato=calendario.getTime();
37         id=idSiguiente;
38         idSiguiente++;
39     }
40     //
41     public String dameDescripcion(){
42         return "Este empleado tiene un ID="+id+" con un sueldo="+sueldo;
43     }
44     //getter
45     public double dameSueldo(){
46         return sueldo;
47     }
48     //getter
49     public Date dameFechaContrato(){
50         return altaContrato;
51     }
52     //setter, para aumentar el sueldo
53     public void subeSueldo(double porcentaje){
54         double aumento=sueldo*porcentaje/100;
55         sueldo+=aumento;
56     }
57 }
58 class Alumno extends Persona{
59     private String carrera;
60     public Alumno(String nom, String car){
61         super(nom);
62         carrera=car;
63     }
64     public String dameDescripcion(){
65         return "Este alumno esta estudiando la carrera de "+carrera;
66     }
67 }
```

Console ✘

<terminated> Uso\_Persona [Java Application] C:\Program Files\Java\jre1.8.0\_91\bin\javaw.exe (26 ago. 2016 17:28:40)  
Luis Conde, Este empleado tiene un ID=1 con un sueldo=50000.0  
Ana Lopez, Este alumno esta estudiando la carrera de Biologicas

## Vídeo 47. Modificadores de acceso. La clase object (superclase cósmica)

Public, Private y otras más.

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

Private se utiliza para encapsular datos o métodos.

Cuando no tenemos constructor, java asume el constructor por defecto:

```
package poo;
public class Clase1 {
    int mivar=5;
    int mivar2=7;
    String mimetodo(){
        return "El valor de mivar2 es: " + mivar2;
    }
}
```

public Clase1(){  
}  
}

La clase object (superclase cósmica): Todas las clases de java heredan de object, tanto las nuestras como las que vienen predefinidas en la API de java:

AbstractSelectableChannel

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3  
java.nio.channels.spi

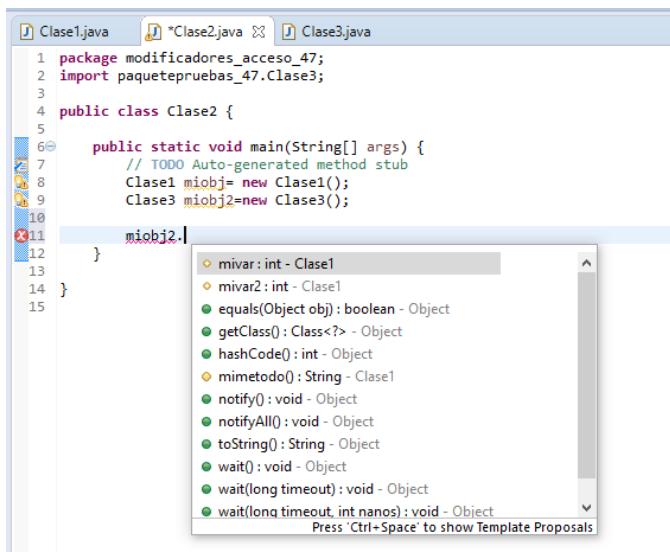
**Class AbstractSelectableChannel**

java.lang.Object  
java.nio.channels.spi.AbstractInterruptibleChannel  
java.nio.channels.SelectableChannel  
java.nio.channels.spi.AbstractSelectableChannel

**All Implemented Interfaces:**  
Closeable, AutoCloseable, Channel, InterruptibleChannel

**Direct Known Subclasses:**  
DatagramChannel, Pipe.SinkChannel, Pipe.SourceChannel, ServerSocketChannel, SocketChannel

Podemos verificar los métodos de la clase object:

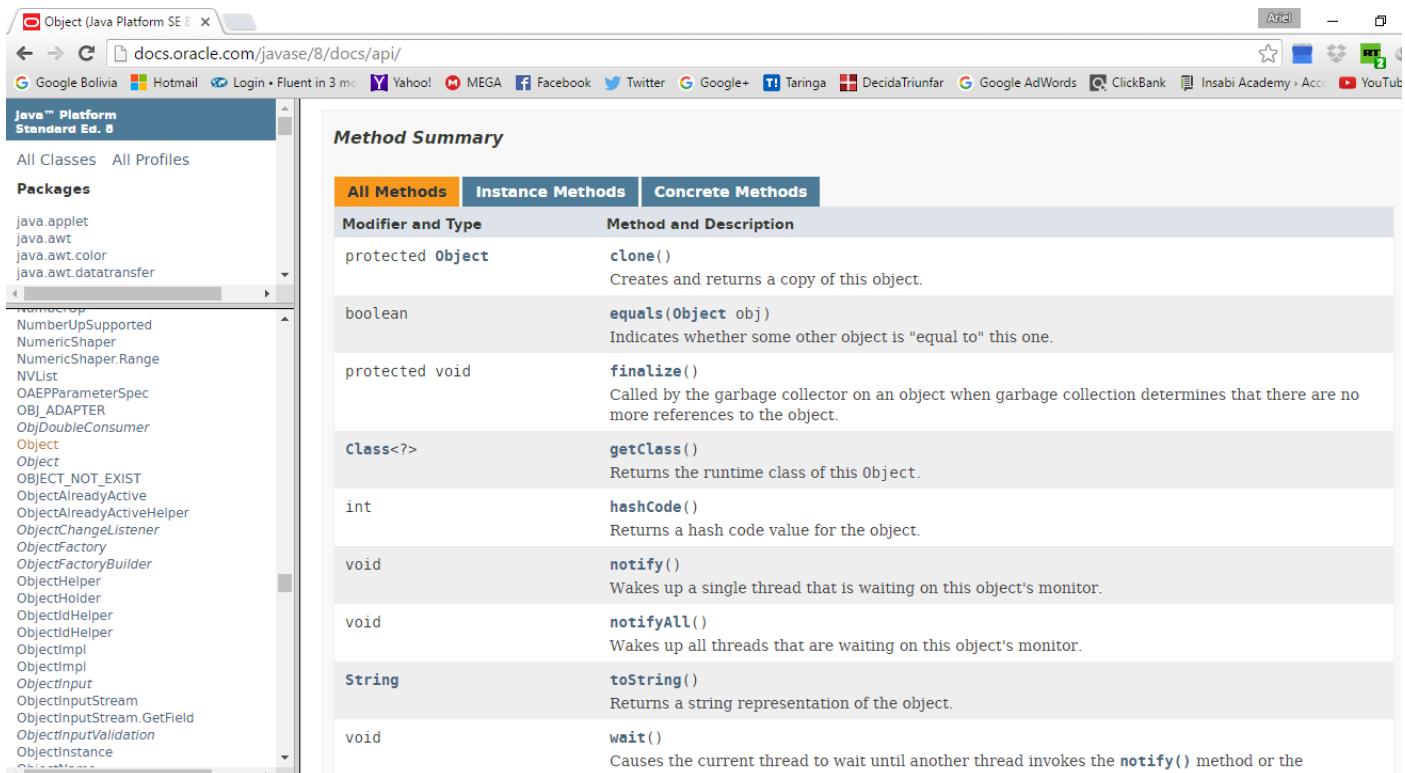


A screenshot of an IDE showing Java code. The code defines a class Clase2 that contains a main method. Inside the main method, there is a line of code: 'Clase1 miobj= new Clase1();'. Below this line, the code 'miobj2.' is typed, and a dropdown menu shows various methods available for the Object class, such as clone(), equals(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), and wait(long timeout).

```
1 package modificadores_acceso_47;
2 import paquetepruebas_47.Clase3;
3
4 public class Clase2 {
5
6     public static void main(String[] args) {
7         // TODO Auto-generated method stub
8         Clase1 miobj= new Clase1();
9         Clase3 miobj2=new Clase3();
10
11     miobj2.|
```

miobj2.  
mivar : int - Clase1  
mivar2 : int - Clase1  
equals(Object obj) : boolean - Object  
getClass() : Class<?> - Object  
hashCode() : int - Object  
mimetodo0 : String - Clase1  
notify() : void - Object  
notifyAll() : void - Object  
toString() : String - Object  
wait() : void - Object  
wait(long timeout) : void - Object  
wait(long timeout, int nanos) : void - Object  
Press 'Ctrl+Space' to show Template Proposals

Los métodos de la clase object se tendrán siempre disponibles:



A screenshot of the Java API documentation for the Object class. The page title is 'Object (Java Platform SE 8)'. On the left, there is a sidebar with a tree view of Java packages, including java.awt, java.awt.color, java.awt.datatransfer, and several subclasses of Object like NumberUpSupported, NumericShaper, and NVList. The main content area is titled 'Method Summary' and contains a table of methods. The table has three tabs at the top: 'All Methods' (which is selected), 'Instance Methods', and 'Concrete Methods'. The table lists the following methods:

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

## Vídeo 48. Tipos enumerados

Tipos enumerados relacionados con la Clase ENUM, que nos permite crear objetos de tipo enumerados.

Es un variable objeto de tipo enum en la cual podemos almacenar un grupo de valores y sólo se podrá almacenar ese grupo de valores.

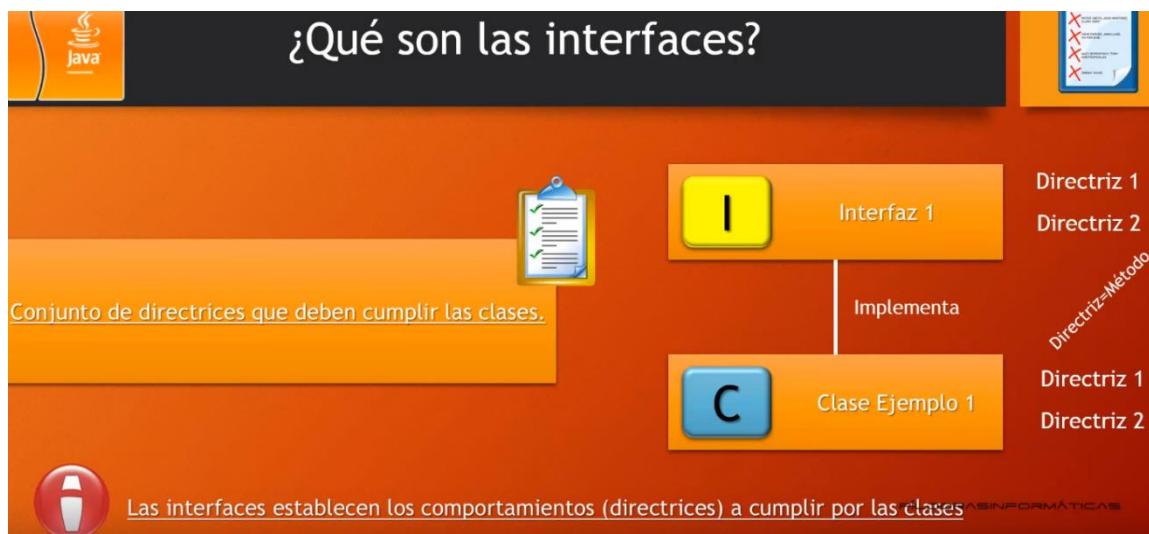
```

1 package tiposenumerados_48;
2 import java.util.*;
3 public class Uso_Tallas {
4     //variable objeto Talla
5     //enum Talla {MINI, MEDIANO, GRANDE, MUY_GRANDE};
6     enum Talla{
7         MINI("S"), MEDIANO("M"), GRANDE("L"), MUY_GRANDE("XL");
8         private String abreviatura;
9         private Talla(String abreviatura){
10             this.abreviatura=abreviatura;
11         }
12         //get
13         public String dameAbreviatura(){
14             return abreviatura;
15         }
16     };
17     public static void main(String[] args) {
18         // TODO Auto-generated method stub
19         /*Talla s=Talla.MINI;
20         Talla m=Talla.MEDIANO;
21         Talla l=Talla.GRANDE;
22         Talla xl=Talla.MUY_GRANDE;*/
23         Scanner entrada=new Scanner(System.in);
24         System.out.println("Escribe una talla: MINI, MEDIANO, GRANDE, MUY_GRANDE");
25         String entrada_datos=entrada.nextLine().toUpperCase();
26         Talla la_talla=Enum.valueOf(Talla.class, entrada_datos);
27         System.out.println("Talla=" + la_talla);
28         System.out.println("Abreviatura=" + la_talla.dameAbreviatura());
29     }

```

Problems @ Javadoc Declaration Console Progress  
 <terminated> Uso\_Tallas [Java Application] C:\Java\jre\bin\javaw.exe (6 sep. 2016 11:53:38)  
 Escribe una talla: MINI, MEDIANO, GRANDE, MUY\_GRANDE  
 MINI  
 Talla=MINI  
 Abreviatura=S

## Vídeo 49. Interfaces y clases internas. Interfaces I





## ¿Y esto no lo hacían ya las clases abstractas? Sí pero.... Problema herencia simple



```
Uso_Emppleado.java
```

```
1 package interfaces_I_49;
2 import java.util.*;
3 public class Uso_Emppleado {
4     public static void main(String[] args) {
5         Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 9, 25);
6         jefe_RRHH.estable_incentivo(2570);
7         Empleado[] misEmpleados=new Empleado[6];
8         misEmpleados[0]=new Empleado("Ariel", 85000, 1990, 12, 17);
9         misEmpleados[1]=new Empleado("Ana", 95000, 1980, 3, 11);
10        misEmpleados[2]=new Empleado("Oswaldo", 55000, 2012, 7, 3);
11        misEmpleados[3]=new Empleado("Rodrigo");
12        /*principio de sustitucion, polimorfismo en accion,
13        almacenamos un objeto de tipo jefatura*/
14        misEmpleados[4]=jefe_RRHH;
15        misEmpleados[5]=new Jefatura("Maria", 95000, 1999, 5, 26);
16        //casting de objetos
17        Jefatura jefa_Finanzas=(Jefatura) misEmpleados[5];
18        jefa_Finanzas.estable_incentivo(5000);
19        for(Empleado e: misEmpleados){
20            e.subeSueldo(5);
21        }
22        Arrays.sort(misEmpleados);
23        for(Empleado e: misEmpleados){
24            System.out.println("Nombre: "+e.dameNombre()+" Sueldo: "+
25                e.dameSueldo()+" Fecha de alta: "+e.dameFechaContrato());
26        }
27    }
28 }
29 class Empleado implements Comparable{
30     private String nombre;
31     private double sueldo;
32     private Date altaContrato;
33     //constructor que recibe parametros o argumentos
34     public Empleado(String nom, double sue, int agno, int mes, int dia){
35         nombre=nom;
36         sueldo=sue;
37         GregorianCalendar calendario=new GregorianCalendar(agno, mes-1, dia);
38         altaContrato=calendario.getTime();
39     }
40     //si solo se conoce nombre
41     public Empleado(String nom){
42         //this está llamando al otro constructor de la clase y pasarse los parametros correspondientes
43         this(nom, 30000, 2000, 01, 01);
44     }
45     //getter
46     public String dameNombre(){
47         return nombre;
48     }
49     //getter
50     public double dameSueldo(){
51         return sueldo;
52     }
53     //getter
54     public Date dameFechaContrato(){
55         return altaContrato;
56     }
57     //setter, para aumentar el sueldo
58     public void subeSueldo(double porcentaje){
59         double aumento=sueldo*porcentaje/100;
60         sueldo+=aumento;
61     }
62     //Creamos el método compareTo
63     public int compareTo(Object miObjeto){
64         //casting
65         Empleado otroEmpleado=(Empleado) miObjeto;
66         if(this.sueldo<otroEmpleado.sueldo){
67             return -1;
68         }
69         if(this.sueldo>otroEmpleado.sueldo){
70             return 1;
71         }
72         return 0;
73     }
74 }
75 class Jefatura extends Empleado{
76     private double incentivo;
77     public Jefatura(String nom, double sue, int agno, int mes, int dia){
78         super(nom, sue, agno, mes, dia);
79     }
80     //set
81     public void estable_incentivo(double b){
82         incentivo=b;
83     }
84     //get
85     public double dameSueldo(){
86         double sueldoJefe=super.dameSueldo(); //con super esta llamando a dameSueldo() de la clase Empleado
87         return sueldoJefe+incentivo;
88     }
89 }
```

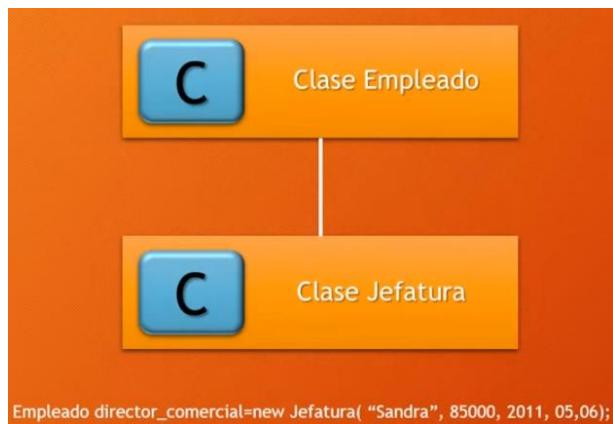
Problems @ Javadoc Declaration Console Progress

<terminated> Uso\_Emppleado (4) [Java Application] C:\Java\jre\bin\javaw.exe (6 sep. 2016 16:51:06)

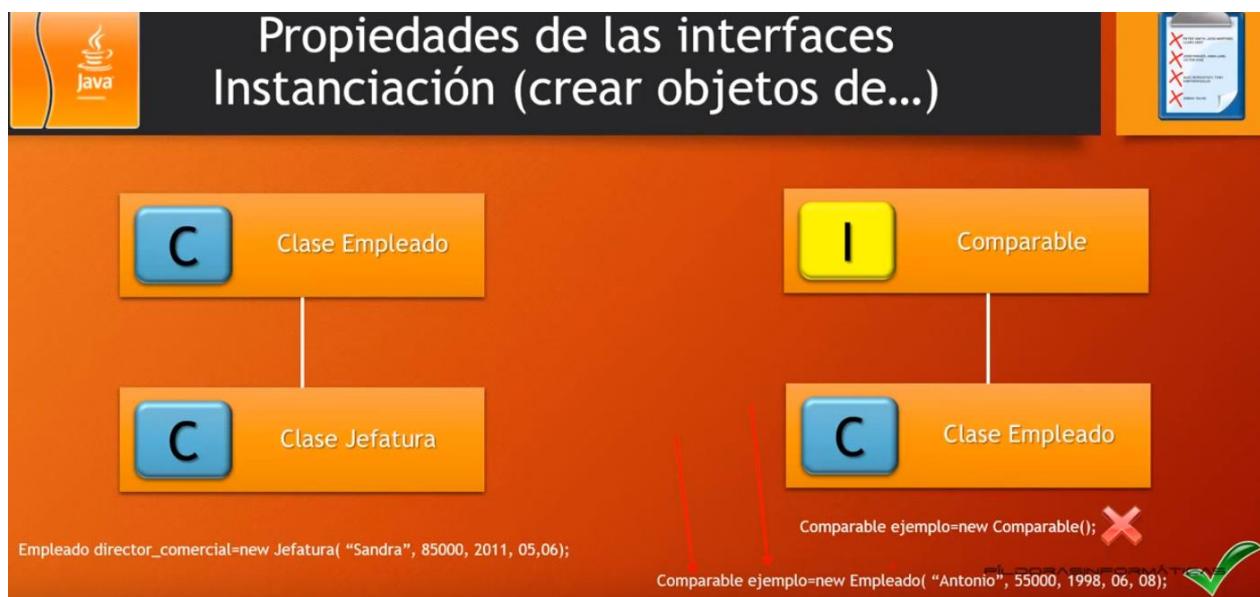
Nombre: Rodrigo Sueldo: 31500.0 Fecha de alta: Sat Jan 01 00:00:00 BOT 2000  
Nombre: Oswaldo Sueldo: 57750.0 Fecha de alta: Tue Jul 03 00:00:00 BOT 2012  
Nombre: Juan Sueldo: 60320.0 Fecha de alta: Mon Sep 25 00:00:00 BOT 2006  
Nombre: Ariel Sueldo: 89250.0 Fecha de alta: Mon Dec 17 00:00:00 BOT 1990  
Nombre: Ana Sueldo: 99750.0 Fecha de alta: Tue Mar 11 00:00:00 BOT 1980  
Nombre: Maria Sueldo: 104750.0 Fecha de alta: Wed May 26 00:00:00 BOT 1999

## Vídeo 50. Interfaces y clases internas. Interfaces II

Utilizando el principio de sustitución, una instancia a la que denomino director\_comercial es de tipo Empleado, a la hora de invocar al constructor es igual a new Jefatura, es decir que es un nuevo jefe, por que un jefe siempre va a ser un empleado:



Una instancia (ejemplo) perteneciente a una interfaz (Comparable), a la hora de invocar al constructor le decimos que es de tipo Empleado:



```

10     misEmpleados[2]=new Empleado("Oswaldo", 55000, 2012, 7, 3);
11     misEmpleados[3]=new Empleado("Rodrigo");
12     /*principio de sustitucion, polimorfismo en accion,
13     almacenamos un objeto de tipo jefatura*/
14     misEmpleados[4]=jefe_RRHH;
15     misEmpleados[5]=new Jefatura("Maria", 95000, 1999, 5, 26);
16     //casting de objetos
17     Jefatura jefa_Finanzas=(Jefatura) misEmpleados[5];
18     jefa_Finanzas.estable_incentivo(5000);
19     //utilizando el principio de sustitución
20     Empleado director_comercial=new Jefatura("Sandra", 85000, 2012, 05, 05);
21     Comparable ejemplo=new Empleado("Elisabeth", 95000, 2011, 06, 07);
22     if(director_comercial instanceof Empleado){
23         System.out.println("Es de tipo Jefatura");
24     }
25     if(ejemplo instanceof Comparable){//si implementa la interfaz comparable
26         System.out.println("Implementa la interfaz comparable");
27     }
28     for(Empleado e: misEmpleados){
29         e.subeSueldo(5);
30     }
31     Arrays.sort(misEmpleados);
32     for(Empleado e: misEmpleados){
33         System.out.println("Nombre: "+e.dameNombre()+" Sueldo: "+
34             e.dameSueldo()+" Fecha de alta: "+e.dameFechaContrato());

```

Problems @ Javadoc Declaration Console Progress

<terminated> Uso\_Empleado (5) [Java Application] C:\Java\jre\bin\javaw.exe (6 sep. 2016 17:23:06)

Es de tipo Jefatura  
 Implementa la interfaz comparable  
 Nombre: Rodrigo Sueldo: 31500.0 Fecha de alta: Sat Jan 01 00:00:00 BOT 2000  
 Nombre: Oswaldo Sueldo: 57750.0 Fecha de alta: Tue Jul 03 00:00:00 BOT 2012  
 Nombre: Juan Sueldo: 60320.0 Fecha de alta: Mon Sep 25 00:00:00 BOT 2006  
 Nombre: Ariel Sueldo: 89250.0 Fecha de alta: Mon Dec 17 00:00:00 BOT 1990  
 Nombre: Ana Sueldo: 99750.0 Fecha de alta: Tue Mar 11 00:00:00 BOT 1980  
 Nombre: Maria Sueldo: 104750.0 Fecha de alta: Wed May 26 00:00:00 BOT 1999



Package Explorer

- Ejercicios
- Java\_Avanzado\_V2B
- Logica\_Programacion
- PildorasInformaticas
  - src
    - casting\_clases\_metodos\_final\_44
    - clases\_abstractas\_II\_46
    - constantes\_36
    - herencia\_I\_40
    - herencia\_II\_41
    - herencia\_III\_42
    - interfaces\_I\_49
    - interfaces\_II\_50
      - Jefes.java
      - Uso\_Empleado.java

Uso\_Empleado.java Jefes.java

```

1 package interfaces_II_50;
2
3 public interface Jefes {
4     //podemos colocar los modificadores public abstract
5     //si no lo colocamos java por defecto los coloca asi
6     String tomar_decisiones(String decision);
7 }
8

```

```

76 class Jefatura extends Empleado implements Jefes{
77     private double incentivo;
78     public Jefatura(String nom, double sue, int agno, int mes, int dia){
79         super(nom, sue, agno, mes, dia);
80     }
81     //set
82     public void estable_incentivo(double b){
83         incentivo=b;
84     }
85     //get
86     public double dameSueldo(){
87         double sueldoJefe=super.dameSueldo(); //con super esta llamando a dameSueldo() de la clase Empleado
88         return sueldoJefe+incentivo;
89     }
90     //se utiliza el método de la interfaz
91     public String tomar_decisiones(String decision){
92         return "Un miembro de la dirección ha tomado la decisión de: " + decision;
93     }

```

Uso\_Emppleado.java Jefes.java

```

1 package interfaces_II_50;
2 import java.util.*;
3 public class Uso_Emppleado {
4     public static void main(String[] args) {
5         Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 9, 25);
6         jefe_RRHH.estable_incentivo(2570);
7         Empleado[] misEmpleados=new Empleado[6];
8         misEmpleados[0]=new Empleado("Ariel", 85000, 1990, 12, 17);
9         misEmpleados[1]=new Empleado("Ana", 95000, 1980, 3, 11);
10        misEmpleados[2]=new Empleado("Oswaldo", 55000, 2012, 7, 3);
11        misEmpleados[3]=new Empleado("Rodrigo");
12        /*principio de sustitución, polimorfismo en acción,
13        almacenamos un objeto de tipo jefatura*/
14        misEmpleados[4]=jefe_RRHH;
15        misEmpleados[5]=new Jefatura("Maria", 95000, 1999, 5, 26);
16        //casting de objetos
17        Jefatura jefa_Finanzas=(Jefatura) misEmpleados[5];
18        jefa_Finanzas.estable_incentivo(5000);
19        System.out.println(jefa_Finanzas.tomar_decisiones("Dar más días de vacaciones a los empleados"));
20        for(Empleado e: misEmpleados){
21            e.subeSueldo(5);
22        }
23        Arrays.sort(misEmpleados);
24        for(Empleado e: misEmpleados){
25            System.out.println("Nombre: "+e.dameNombre()+" Sueldo: "+

```

Problems @ Javadoc Declaration Console Progress

```

<terminated> Uso_Emppleado (5) [Java Application] C:\Java\jre\bin\javaw.exe (6 sep. 2016 17:54:59)
Un miembro de la dirección ha tomado la decisión de: Dar más días de vacaciones a los empleados
Nombre: Rodrigo Sueldo: 31500.0 Fecha de alta: Sat Jan 01 00:00:00 BOT 2000
Nombre: Oswaldo Sueldo: 57750.0 Fecha de alta: Tue Jul 03 00:00:00 BOT 2012
Nombre: Juan Sueldo: 60320.0 Fecha de alta: Mon Sep 25 00:00:00 BOT 2006
Nombre: Ariel Sueldo: 89250.0 Fecha de alta: Mon Dec 17 00:00:00 BOT 1990
Nombre: Ana Sueldo: 99750.0 Fecha de alta: Tue Mar 11 00:00:00 BOT 1980
Nombre: Maria Sueldo: 104750.0 Fecha de alta: Wed May 26 00:00:00 BOT 1999

```

## Vídeo 51. Interfaces y clases internas. Interfaces III

Una clase puede implementar dos o más interfaces:

Uso\_Emppleado.java Jefes.java

```

31 class Empleado implements Comparable, Jefes{
32     private String nombre;
33     private double sueldo;
34     private Date altaContrato;
35     //constructor que recibe parámetros o argumentos.
36     public Empleado(String nom, double sue, int agno, int mes, int dia){
37         nombre=nom;
38         sueldo=sue;
39         GregorianCalendar calendario=new GregorianCalendar(agno, mes-1, dia);
40         altaContrato=calendario.getTime();
41     }
42     //si solo se conoce nombre
43     public Empleado(String nom){
44         //this está llamando al otro constructor de la clase y pasadle los parámetros correspondientes
45         this(nom, 30000, 2000, 01, 01);
46     }
47     //getter
48     public String dameNombre(){
49         return nombre;
50     }
51     //getter
52     public double dameSueldo(){
53         return sueldo;
54     }
55     //getter
56     public Date dameFechaContrato(){
57         return altaContrato;
58     }
59     //setter, para aumentar el sueldo
60     public void subeSueldo(double porcentaje){
61         double aumento=sueldo*porcentaje/100;
62         sueldo+=aumento;
63     }
64     //creamos el método compareTo
65     public int compareTo(Object miObjeto){
66         //casting
67         Empleado otroEmpleado=(Empleado) miObjeto;
68         if(this.sueldo<otroEmpleado.sueldo){
```



*Los métodos de las Interfaces tan sólo se definen, no se desarrollan.*

```

Uso_Emppleado.java Jefes.java Trabajadores.java
1 package interfaces_III_51;
2
3 public interface Trabajadores {
4     //public abstract double establece_bonus(double gratificacion);
5     double establece_bonus(double gratificacion);
6     //Creamos una constante de mínimo
7     //public static final double bonus_base=1500;
8     double bonus_base=1500;
9 }

```

```

Uso_Emppleado.java Jefes.java Trabajadores.java
1 package interfaces_III_51;
2
3 public interface Jefes extends Trabajadores{//establecemos jerarquías de interfaces, también deben implementar de Trabajadores
4     //podemos colocar los modificadores public abstract
5     //si no lo colocamos java por defecto los coloca así
6     String tomar_decisiones(String decision);
7 }

```

```

Uso_Emppleado.java Jefes.java Trabajadores.java
68         Empleado otroEmpleado=(Empleado) miObjeto;
69         if(this.sueldo<otroEmpleado.sueldo){
70             return -1;
71         }
72         if(this.sueldo>otroEmpleado.sueldo){
73             return 1;
74         }
75         return 0;
76     }
77     //desarrollamos el método establece_bonus de la Interfaz Trabajadores
78     public double establece_bonus(double gratificacion){
79         return Trabajadores.bonus_base+gratificacion;
80     }
81 }
82 class Jefatura extends Empleado implements Jefes{
83     private double incentivo;
84     public Jefatura(String nom, double sue, int agno, int mes, int dia){
85         super(nom, sue, agno, mes, dia);
86     }
87     //set
88     public void estable_incentivo(double b){
89         incentivo=b;
90     }
91     //get
92     public double dameSueldo(){
93         double sueldoJefe=super.dameSueldo(); //con super esta llamando a dameSueldo() de la clase Empleado
94         return sueldoJefe+incentivo;
95     }
96     //se utiliza el método de la interfaz
97     public String tomar_decisiones(String decision){
98         return "Un miembro de la dirección ha tomado la decisión de: " + decision;
99     }
100    //get [Interfaz Trabajadores]
101    public double establece_bonus(double gratificacion){
102        double prima=2000;
103        return Trabajadores.bonus_base+gratificacion+prima;
104    }
105 }

```

```
Uso_Emppleado.java Jefes.java Trabajadores.java
32 class Empleado implements Comparable, Trabajadores{
33     private String nombre;
34     private double sueldo;
35     private Date altaContrato;
36     //constructor que recibe parámetros o argumentos
37     public Empleado(String nom, double sue, int agno, int mes, int dia){
38         nombre=nom;
39         sueldo=sue;
40         GregorianCalendar calendario=new GregorianCalendar(agno, mes-1, dia);
41         altaContrato=calendario.getTime();
42     }
43     //si solo se conoce nombre
44     public Empleado(String nom){
45         //this está llamando al otro constructor de la clase y pasarle los parámetros correspondientes
46         this(nom, 30000, 2000, 01, 01);
47     }
48     //getter
49     public String dameNombre(){
50         return nombre;
51     }
52     //getter
53     public double dameSueldo(){
54         return sueldo;
55     }
56     //getter
57     public Date dameFechaContrato(){
58         return altaContrato;
59     }
60     //setter, para aumentar el sueldo
61     public void subeSueldo(double porcentaje){
62         double aumento=sueldo*porcentaje/100;
63         sueldo+=aumento;
64     }
65     //desarrollamos el método compareTo
66     public int compareTo(Object miObjeto){
67         //casting
68         Empleado otroEmpleado=(Empleado) miObjeto;
69         if(this.sueldo<otroEmpleado.sueldo){
70             return -1;
71         }
72         if(this.sueldo>otroEmpleado.sueldo){
73             return 1;
74         }
75         return 0;
76     }
77     //desarrollamos el método establece_bonus de la Interfaz Trabajadores
78     public double establece_bonus(double gratificacion){
79         return Trabajadores.bonus_base+gratificacion;
80     }
81 }
```

```
Uso_Emppleado.java Jefes.java Trabajadores.java
1 package interfaces_III_51;
2 import java.util.*;
3 public class Uso_Emppleado {
4     public static void main(String[] args) {
5         Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 9, 25);
6         jefe_RRHH.estable_incentivo(2570);
7         Empleado[] misEmpleados=new Empleado[6];
8         misEmpleados[0]=new Empleado("Ariel", 85000, 1990, 12, 17);
9         misEmpleados[1]=new Empleado("Ana", 95000, 1980, 3, 11);
10        misEmpleados[2]=new Empleado("Oswaldo", 55000, 2012, 7, 3);
11        misEmpleados[3]=new Empleado("Rodrigo");
12        /*principio de sustitucion, polimorfismo en acción,
13        almacenamos un objeto de tipo jefatura*/
14        misEmpleados[4]=jefe_RRHH;
15        misEmpleados[5]=new Jefatura("Maria", 95000, 1999, 5, 26);
16        //casting de objetos
17        Jefatura jefa_Finanzas=(Jefatura) misEmpleados[5];
18        jefa_Finanzas.estable_incentivo(5000);
19        System.out.println(jefa_Finanzas.tomar_decisiones("Dar más días de vacaciones a los empleados"));
20        System.out.println("La/El jefe "+jefa_Finanzas.dameNombre()+" tiene un bono de: "+jefa_Finanzas.establece_bonus(500));
21        System.out.println(misEmpleados[3].dameNombre()+" tiene un bono de: "+misEmpleados[3].establece_bonus(200));
22        for(Empleado e: misEmpleados){
23            e.subeSueldo(5);
24        }
25    }
26
27    Problems @ Javadoc Declaration Console Progress
28 <terminated> Uso_Emppleado (6) [Java Application] C:\Java\jre\bin\javaw.exe (6 sep. 2016 19:05:48)
29 Un miembro de la dirección ha tomado la decisión de: Dar más días de vacaciones a los empleados
30 La/El jefe Maria tiene un bono de: 4000.0
31 Rodrigo tiene un bono de: 1700.0
32 Nombre: Rodrigo Sueldo: 31500.0 Fecha de alta: Sat Jan 01 00:00:00 BOT 2000
33 Nombre: Oswaldo Sueldo: 57750.0 Fecha de alta: Tue Jul 03 00:00:00 BOT 2012
```

## Vídeo 52. Interfaces y clases internas. Interfaces IV

Vamos a elaborar un temporizador, esto es un programa que realiza una acción cada x tiempo.

Constructor Summary  
Constructors  
Constructor and Description  
Timer(int delay, ActionListener listener)  
Creates a Timer and initializes both the initial delay and between-event delay to delay milliseconds.

Tipo Void, es decir no devuelve ningún dato:

```
void start()  
Starts the Timer, causing it to start sending action events to its listeners.
```

See Also:  
ActionEvent, How to Write an Action Listener

Method Summary  
All Methods Instance Methods Abstract Methods  
Modifier and Type Method and Description  
void actionPerformed(ActionEvent e)  
Invoked when an action occurs.

```
1 package interfaces_IV_52;  
2 import javax.swing.*;  
3 import java.awt.event.*;  
4 import java.util.*;  
5 import javax.swing.Timer;  
6  
7 public class PruebaTemporizador {  
8     public static void main(String[] args) {  
9         //Clase DameLaHora no tiene constructor, estamos utilizando el constructor por defecto,  
10        //que es aquel que no recibe parámetros y se da por supuesto.  
11        DameLaHora oyente=new DameLaHora();  
12        DameLaHora oyente=new DameLaHora();  
13        Timer mitemportador=new Timer(5000, oyente);  
14        mitemportador.start();  
15        JOptionPane.showMessageDialog(null, "Pulsa Aceptar para Detener");  
16        System.exit(0);  
17    }  
18 }  
19 class DameLaHora implements ActionListener{  
20     public void actionPerformed(ActionEvent e){  
21         Date ahora=new Date();  
22         System.out.println("Te pongo la hora cada 5 seg.: "+ ahora);  
23     }  
24 }  
25
```

Problems Declaration Console Progress

PruebaTemporizador [Java Application] C:\Java\jre\bin\javaw.exe (7 sep. 2016 10:38:19)  
Te pongo la hora cada 5 seg.: Wed Sep 07 10:38:24 BOT 2016  
Te pongo la hora cada 5 seg.: Wed Sep 07 10:38:29 BOT 2016  
Te pongo la hora cada 5 seg.: Wed Sep 07 10:38:34 BOT 2016

## Vídeo 53. Interfaces y clases internas. Clases Internas I

### ¿Qué son las clases internas? (Inner class)

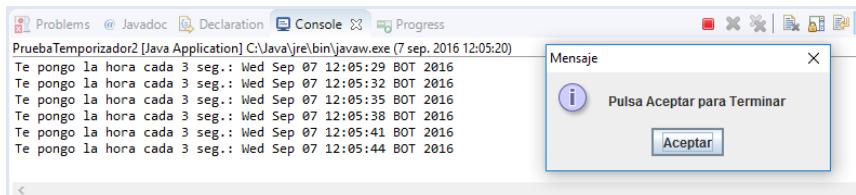
Como su nombre indica, una clase interna es una clase dentro de otra.

```
Public class Clase1 {  
    class Clase2 {  
        Código de la Clase2  
    }  
    Código de la Clase1  
}
```

## ¿Y por qué serían necesarias?

- Para acceder a los campos privados de una clase desde otra clase.
- Para ocultar una clase de otras pertenecientes al mismo paquete.
- Para crear clases internas “anónimas”, muy útiles para gestionar eventos y retrolllamadas
- Cuando solo una clase debe acceder a los campos de ejemplar de otra clase.

```
PruebaTemporizador.java  PruebaTemporizador2.java
1 package clases_internas_I_53;
2 import javax.swing.*;
3 import java.awt.event.*;
4 import java.util.*;
5 import javax.swing.Timer;
6 import java.awt.Toolkit;
7 public class PruebaTemporizador2 {
8     public static void main(String[] args) {
9         Reloj mireloj = new Reloj(3000, true);
10        mireloj.enMarcha();
11        JOptionPane.showMessageDialog(null, "Pulsa Aceptar para Terminar");
12        System.exit(0);
13    }
14 }
15 class Reloj{
16     private int intervalo;
17     private boolean sonido;
18     public Reloj(int intervalo, boolean sonido){
19         this.intervalo=intervalo;
20         this.sonido=sonido;
21     }
22     public void enMarcha(){
23         ActionListener oyente=new DameLaHora2();
24         Timer mitemporizador=new Timer(intervalo, oyente);
25         mitemporizador.start();
26     }
27 //clase interna
28     private class DameLaHora2 implements ActionListener{
29         public void actionPerformed(ActionEvent evento){
30             Date ahora=new Date();
31             System.out.println("Te pongo la hora cada 3 seg.: " + ahora);
32             if(sonido){
33                 Toolkit.getDefaultToolkit().beep();
34             }
35         }
36     }
37 }
```



## Vídeo 54. Interfaces y clases internas. Clases Internas II

### Clases internas locales ¿Qué son?

- Una clase dentro de un método.
- ¿Cuándo se utilizan estos tipos de clases y por qué?
  - Son útiles cuando solo se va a utilizar (instanciar) la clase interna una vez. El objetivo es simplificar aún más el código.
  - Su ámbito queda restringido al método donde son declaradas. ¿Ventajas?
    - Están muy “encapsuladas”. Ni siquiera la clase a la que pertenecen puede acceder a ella. Tan solo puede acceder a ella el método donde están declaradas.
    - El código resulta más simple.

## Declaración de clase interna local

```
Class Clase_externa{  
    Public void método(){  
        class clase_interna_local{  
            código de la clase interna;  
        }  
        código del método;  
    }  
    Código de la clase externa;  
}
```

\*PruebaTemporizador2.java

```
1 package clases_internas_II_54;  
2 import javax.swing.*;  
3 public class PruebaTemporizador2 {  
4     public static void main(String[] args) {  
5         Reloj mireloj = new Reloj();  
6         mireloj.enMarcha(3000, true);  
7         JOptionPane.showMessageDialog(null, "Pulsa Aceptar para Terminar");  
8         System.exit(0);  
9     }  
10 }  
11 class Reloj{  
12     public void enMarcha(int intervalo, final boolean sonido){  
13         //clase interna  
14         class DameLaHora2 implements ActionListener{  
15             public void actionPerformed(ActionEvent evento){  
16                 Date ahora=new Date();  
17                 System.out.println("Te pongo la hora cada 3 seg.: " + ahora);  
18                 if(sonido){  
19                     Toolkit.getDefaultToolkit().beep();  
20                 }  
21             }  
22         }  
23         ActionListener oyente=new DameLaHora2();  
24         Timer mitemporizador=new Timer(intervalo, oyente);  
25         mitemporizador.start();  
26     }  
27 }  
28 ActionListener oyente=new DameLaHora2();  
29 Timer mitemporizador=new Timer(intervalo, oyente);  
30 mitemporizador.start();  
31 }
```

Problems Javadoc Declaration Console Mensaje

Mensaje

Pulsa Aceptar para Terminar

Aceptar

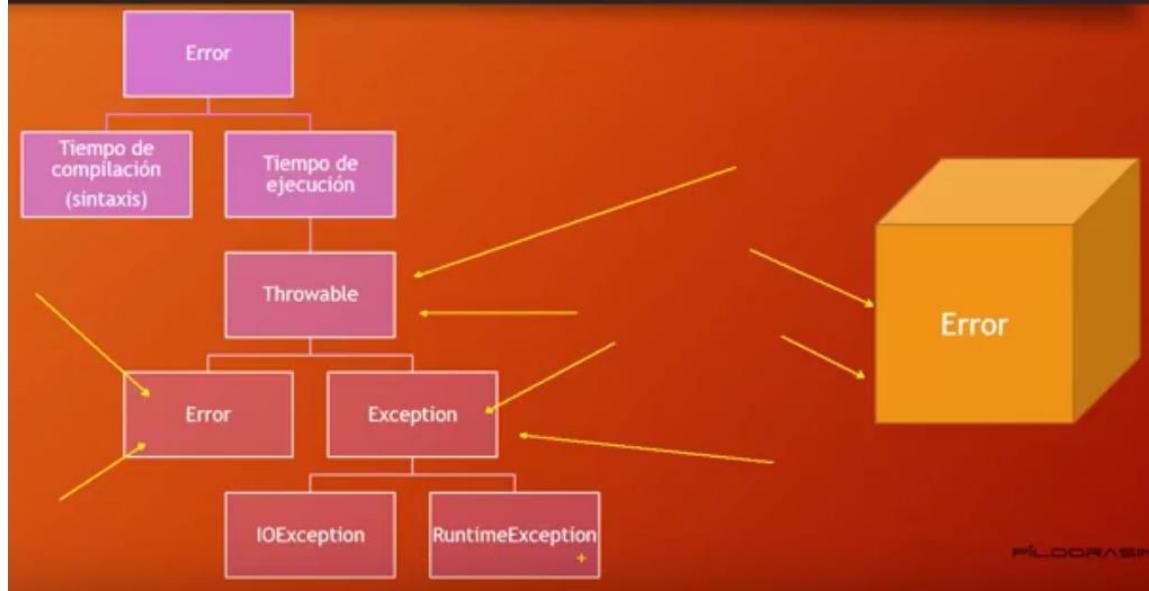
PruebaTemporizador2 (1) [Java Application] C:\Java\jre\bin\javaw.exe (7 sep. 2016 12:29:00)  
Te pongo la hora cada 3 seg.: Wed Sep 07 12:29:15 BOT 2016  
Te pongo la hora cada 3 seg.: Wed Sep 07 12:29:18 BOT 2016  
Te pongo la hora cada 3 seg.: Wed Sep 07 12:29:21 BOT 2016  
Te pongo la hora cada 3 seg.: Wed Sep 07 12:29:24 BOT 2016  
Te pongo la hora cada 3 seg.: Wed Sep 07 12:29:27 BOT 2016

## Vídeo 142. Manejo de errores. Excepciones I

Si la lógica del programa no tiene mucho sentido, eso se denomina error en tiempo de ejecución, una vez que se compila el programa y se crea el archivo class, cuando existe el error se genera un objeto, y a partir de la clase throwable desde donde tenemos la jerarquía de clases que manejan los errores en java.

Todo objeto error que se genera en un programa java hereda de la clase Throwable, debajo del mismo hay dos clases Error y Exception, cuando un error hereda de la clase Error normalmente significa un error de hardware, poco espacio en HDD o corrupción de memoria, etc.

## Jerarquía de errores



IOException: Errores no atribuibles al programador.

RunTimeException: Error atribuibles al programador.



Fallos.java

```
1 package excepciones_I_142;
2 import javax.swing.*;
3
4 public class Fallos {
5
6     public static void main(String[] args) {
7         int[] mi_matriz=new int[5];
8         mi_matriz[0]=5;
9         mi_matriz[1]=38;
10        mi_matriz[2]=-15;
11        mi_matriz[3]=92;
12        mi_matriz[4]=71;
13        mi_matriz[5]=81;
14        mi_matriz[6]=91;
15        for(int i=0; i<5; i++){
16            System.out.println("Posicion " + i + " = " + mi_matriz[i]);
17        }
18        //Petición de datos personales
19        String nombre JOptionPane.showInputDialog("Introduce tu nombre");
20        int edad=Integer.parseInt(JOptionPane.showInputDialog("Introduce tu edad"));
21        System.out.println("Hola " + nombre + " Tienes " + edad + " años. " + "El programa terminó su ejecución");
22    }
23}
```

Console

```
<terminated> Fallos [Java Application] C:\Java\jre\bin\javaw.exe (12 sep. 2016 15:33:20)
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
at excepciones_I_142.Fallos.main(Fallos.java:13)
```

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform Standard Ed.

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAILED: FIELD | CONSTR | METHOD

compact1, compact2, compact3  
java.lang

### Class **ArrayIndexOutOfBoundsException**

```
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.IndexOutOfBoundsException
          java.lang.ArrayIndexOutOfBoundsException
```

#### All Implemented Interfaces:

Serializable

```
public class ArrayIndexOutOfBoundsException
extends IndexOutOfBoundsException
```

Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

#### Since:

JDK1.0

#### See Also:

Serialized Form

## Vídeo 143. Excepciones comprobadas y no comprobadas Lanzamiento de excepciones (throws/try/catch).

### Excepciones II

Throws (lanzar), si ocurre algún fallo en el método, va a lanzar un objeto perteneciente a la clase IOException.

Para poder controlar los errores que pudiera producir este método, Java nos obliga a utilizar la estructura try...catch

Try (intentar), catch (capturar, coger)



## Vídeo 144. Excepciones comprobadas y no comprobadas Lanzamiento de excepciones (throws/try/catch).

### Excepciones III

```
Entrada_datos.java  
1 System.out.println("que deseas hacer");  
2 System.out.println("1. Introducir datos");  
3 System.out.println("2. Salir del programa");  
4 Scanner entrada=new Scanner (System.in);  
5 int decision=entrada.nextInt();  
6 if(decision==1){  
7     pedirDatos();  
8 }else{  
9     System.out.println("Adios");  
10    System.exit(0);  
11 }  
12 entrada.close();  
13 }  
14 static void pedirDatos(){  
15     Scanner entrada=new Scanner(System.in);  
16     System.out.println("Introduce tu nombre, por favor");  
17     String nombre_usuario=entrada.nextLine();  
18     System.out.println("Introduce edad, por favor");  
19     entrada.close();  
20 }  
21 }  
  
Console  
<terminated> Entrada_datos [Java Application] C:\Java\jre\bin\javaw.exe (12 sep. 2016 16:36:49)  
2. Salir del programa  
1  
Introduce tu nombre, por favor  
Ariel  
Introduce edad, por favor  
1kjlj  
Exception in thread "main" java.util.InputMismatchException  
at java.util.Scanner.throwFor(Unknown Source)  
at java.util.Scanner.next(Unknown Source)  
at java.util.Scanner.nextInt(Unknown Source)  
at java.util.Scanner.nextInt(Unknown Source)  
at excepciones_III.Entrada_datos.pedirDatos(Entrada_datos.java:23)  
at excepciones_III.Entrada_datos.main(Entrada_datos.java:11)
```

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX  
PREV CLASS NEXT CLASS FRAMES NO FRAMES  
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CC  
compact1, compact2, compact3  
java.util  
**Class InputMismatchException**  
java.lang.Object  
  java.lang.Throwable  
    java.lang.Exception  
      java.lang.RuntimeException  
      java.util.NoSuchElementException  
        java.util.InputMismatchException

Hereda de RunTimeException, excepción no comprobada, no se está obligado a utilizar un try-catch, pero podemos utilizarlo

```
Entrada_datos.java
14     System.exit(0);
15 }
16 entrada.close();
17 }
18 static void pedirDatos() throws InputMismatchException{
19     try{
20         Scanner entrada=new Scanner(System.in);
21         System.out.println("Introduce tu nombre, por favor");
22         String nombre_usuario=entrada.nextLine();
23         System.out.println("Introduce edad, por favor");
24         int edad=entrada.nextInt();
25         System.out.println("Hola "+nombre_usuario+". El año que viene tendrás "+(edad+1)+" años");
26         entrada.close();
27     }catch(Exception e){
28         System.out.println("Introduce tu edad en numeros");
29     }
30 }
31 System.out.println("Hemos terminado");
32 }
33 }
```

```
Console
<terminated> Entrada_datos [Java Application] C:\Java\jre\bin\javaw.exe (12 sep. 2016 16:50:33)
¿Qué deseas hacer?
1. Introducir datos
2. Salir del programa
1
Introduce tu nombre, por favor
Ariel
Introduce edad, por favor
as
Introduce tu edad en numeros
Hemos terminado
```

## Vídeo 145. throws/try/catch. Excepciones IV

Funciona el código, pero es una mala práctica:

```
Entrada_datos.java
1 package excepciones_IV_145;
2 import java.io.IOException;
3 import java.util.*;
4 public class Entrada_datos {
5     public static void main(String[] args) {
6         System.out.println("¿Qué deseas hacer?");
7         System.out.println("1. Introducir datos");
8         System.out.println("2. Salir del programa");
9         Scanner entrada=new Scanner (System.in);
10        int decision=entrada.nextInt();
11        if(decision==1){
12            try{
13                pedirDatos();
14            }catch(Exception e){
15                System.out.println("Introduce tu edad en numeros");
16            }
17        }else{
18            System.out.println("Adios");
19            System.exit(0);
20        }
21        entrada.close();
22    }
23 static void pedirDatos() throws IOException{
24     //try{
25     Scanner entrada=new Scanner(System.in);
26     System.out.println("Introduce tu nombre, por favor");
27     String nombre_usuario=entrada.nextLine();
28     System.out.println("Introduce edad, por favor");
29     int edad=entrada.nextInt();
30     System.out.println("Hola "+nombre_usuario+". El año que viene tendrás "+(edad+1)+" años");
31     entrada.close();
32     //}catch(Exception e){
33     //    System.out.println("Introduce tu edad en numeros");
34     //}
35
36     System.out.println("Hemos terminado");
37 }
38 }
```

```
Console
<terminated> Entrada_datos (1) [Java Application]
¿Qué deseas hacer?
1. Introducir datos
2. Salir del programa
1
Introduce tu nombre, por favor
Ariel
Introduce edad, por favor
asd
Introduce tu edad en numeros
```

## Vídeo 146. Clausula throw. Excepciones V

The screenshot shows an IDE interface with two panes. The top pane is titled 'Comprueba\_mail.java' and contains the following Java code:

```
1 package excepciones_V_146;
2 import java.io.EOFException;
3 import javax.swing.JOptionPane;
4 public class Comprueba_mail {
5     public static void main(String[] args) {
6         String el_mail=JOptionPane.showInputDialog("Introduce mail");
7         try{
8             examina_mail(el_mail);
9         }catch(EOFException e){
10             System.out.println("La dirección de email no es correcta");
11         }
12     }
13     static void examina_mail(String mail) throws EOFException{
14         int arroba=0;
15         boolean punto=false;
16         if(mail.length()<=3){
17             //ArrayIndexOutOfBoundsException mi_expcion=new ArrayIndexOutOfBoundsException();
18             throw new EOFException();
19         }else{
20             for(int i=0; i<mail.length(); i++){
21                 if(mail.charAt(i)=='@'){
22                     arroba++;
23                 }
24                 if(mail.charAt(i)=='.'){
25                     punto=true;
26                 }
27             }
28             if(arroba==1 && punto==true){
29                 System.out.println("Es correcto");
30             }else{
31                 System.out.println("No es correcto");
32             }
33         }
34     }
35 }
```

The bottom pane is titled 'Console' and shows the execution output:

```
<terminated> Comprueba_mail [Java Application] C:\Java\jre\bin\javaw.exe (12 sep. 2016 18:26:53)
La dirección de email no es correcta
```

## Vídeo 147. Creación de excepciones propias. Excepciones VI

Creamos clases que hereden de:

Exception  
IOException  
RuntimeException

Si heredamos de las dos primeras (Exception e IOException) nos obligaría a capturar la excepción con un try-catch, mientras que si heredamos de RuntimeException es un error no controlado, no obliga a utilizar try-catch.

```

1 package excepciones_VI_147;
2 import javax.swing.JOptionPane;
3 public class Comprueba_mail {
4     public static void main(String[] args) {
5         String el_mail=JOptionPane.showInputDialog("Introduce mail");
6         //examina_mail(el_mail);
7         try{
8             examina_mail(el_mail);
9         }catch(Exception e){
10             System.out.println("La dirección de email no es correcta");
11             e.printStackTrace();
12         }
13     }
14     static void examina_mail(String mail) throws Longitud_mail_erronea{
15         int arroba=0;
16         boolean punto=false;
17         if(mail.length()<=3){
18             //ArrayIndexOutOfBoundsException mi_excepcion=new ArrayIndexOutOfBoundsException();
19             //throw new EOFException();
20             throw new Longitud_mail_erronea("El mail no puede tener menos de tres caracteres");
21         }else{
22             for(int i=0; i<mail.length(); i++){
23                 if(mail.charAt(i)=='@'){
24                     arroba++;
25                 }
26                 if(mail.charAt(i)=='.'){
27                     punto=true;
28                 }
29             }
30             if(arroba==1 && punto==true){
31                 System.out.println("Es correcto");
32             }else{
33                 System.out.println("No es correcto");
34             }
35         }
36     }
37 }
38 class Longitud_mail_erronea extends Exception{
39     public Longitud_mail_erronea(){}
40     public Longitud_mail_erronea(String msj_error){
41         super(msj_error);
42     }
43 }

```

**Console** ✎  
<terminated> Comprueba\_mail (1) [Java Application] C:\Java\jre\bin\javaw.exe (13 sep. 2016 05:51:35)  
La dirección de email no es correcta  
**excepciones\_VI\_147.Longitud\_mail\_erronea: El mail no puede tener menos de tres caracteres**  
at excepciones\_VI\_147.Comprueba\_mail.examina\_mail([Comprueba\\_mail.java:20](#))  
at excepciones\_VI\_147.Comprueba\_mail.main([Comprueba\\_mail.java:8](#))

### Vídeo 148. Captura de varias excepciones. Excepciones VII

Overview Package Class Use Tree Deprecated Index Help  
Prev Class Next Class Frames No Frames  
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method  
java.lang  
**Class ArithmeticException**

java.lang.Object  
java.lang.Throwable  
java.lang.Exception  
java.lang.RuntimeException  
java.lang.ArithmeticException

All Implemented Interfaces:  
Serializable

public class ArithmeticException  
extends RuntimeException

no controlada  
unchecked!

No es una buena práctica de programación capturar con el try-catch, excepciones no controladas, se supone que son errores de programación y que debes solventarlos de otra forma.

**getMessage()**  
**getClass()** +  
**getName()**

```

[Varias_Excepciones.java]
1 package excepciones_VII_148;
2
3 import javax.swing.JOptionPane;
4
5 public class Varias_Excepciones {
6     public static void main(String[] args) {
7         try{
8             division();
9         }catch(ArithmaticException e){
10            System.out.println("No se permite la división por cero");
11        }catch(NumberFormatException e){
12            System.out.println("Introduce un número");
13            //System.out.println(e.getMessage());
14            System.out.println("Se ha generado un error de este tipo: "+e.getClass().getName());
15        }
16    }
17
18    static void division(){
19        int num1=Integer.parseInt(JOptionPane.showInputDialog("Introduce el dividendo"));
20        int num2=Integer.parseInt(JOptionPane.showInputDialog("Introduce el divisor"));
21        System.out.println("El resultado es: "+num1/num2);
22    }
23 }

```

**Console** ✎  
<terminated> Varias\_Excepciones [Java Application] C:\Java\jre\bin\javaw.exe (13 sep. 2016 09:01:30)  
Introduce un número  
Se ha generado un error de este tipo: [java.lang.NumberFormatException](#)

## Vídeo 149. Cláusula finally. Excepciones VIII

```

[Areas_Peso.java]
1 package excepciones_VIII_149;
2 import java.util.Scanner;
3 import javax.swing.JOptionPane;
4 public class Areas_Peso {
5     public static void main(String[] args) {
6         Scanner entrada=new Scanner(System.in);
7         System.out.println("Elige una opción: \n1: Cuadrado \n2: Rectángulo \n3: Triángulo \n4: Círculo");
8         try{
9             figura=entrada.nextInt();
10            //entrada.close();
11        }catch(Exception e){
12            System.out.println("Ha ocurrido un error");
13        }finally{
14            entrada.close();
15        }
16        switch (figura){
17            case 1:
18                int lado=Integer.parseInt(JOptionPane.showInputDialog("Introduce el lado"));
19                System.out.println(Math.pow(lado, 2));
20                break;
21            case 2:
22                int base=Integer.parseInt(JOptionPane.showInputDialog("Introduce la base"));
23                int altura=Integer.parseInt(JOptionPane.showInputDialog("Introduce la altura"));
24                System.out.println("El área del rectángulo es "+base*altura);
25                break;
26            case 3:
27                base=Integer.parseInt(JOptionPane.showInputDialog("Introduce la base"));
28                altura=Integer.parseInt(JOptionPane.showInputDialog("Introduce la altura"));
29                System.out.println("El área del triángulo es "+(base*altura)/2);
30                break;
31            case 4:
32                int radio=Integer.parseInt(JOptionPane.showInputDialog("Introduce el radio"));
33                System.out.println("El área del círculo es ");
34                System.out.println(Math.PI*(Math.pow(radio, 2)));
35                break;
36            default:
37                System.out.println("La opción no es correcta");
38            }
39            int altura=Integer.parseInt(JOptionPane.showInputDialog("Introduce tu altura en cm"));
40            System.out.println("Si eres hombre tu peso ideal es: "+(altura-100)+" Kg");
41            System.out.println("Si eres mujer tu peso ideal es: "+(altura-110)+" Kg");
42        }
43        static int figura;
44    }

```

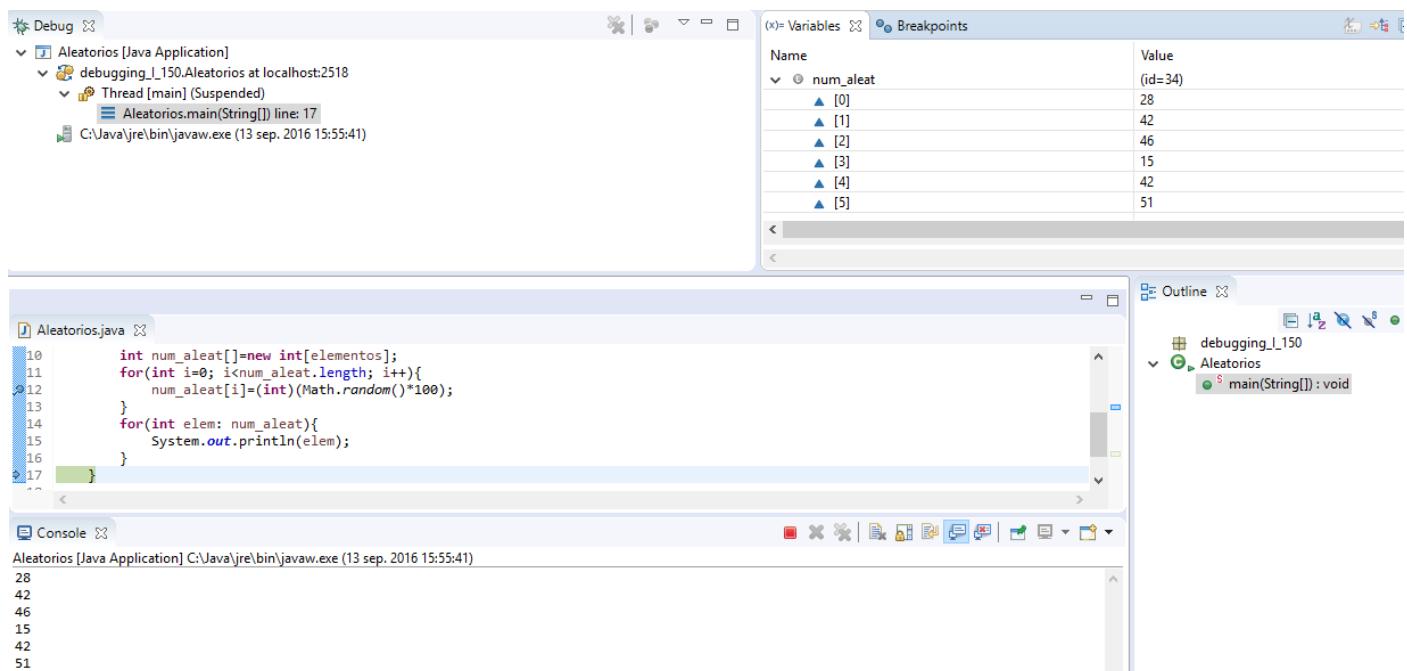
**Console** ✎  
<terminated> Areas\_Peso [Java Application] C:\Java  
Elige una opción:  
1: Cuadrado  
2: Rectángulo  
3: Triángulo  
4: Círculo  
sdaasd  
Ha ocurrido un error  
La opción no es correcta  
Si eres hombre tu peso ideal es: 70 Kg  
Si eres mujer tu peso ideal es: 60 Kg

## Cláusula finally. Ejemplo utilidad. Conexión BBDD

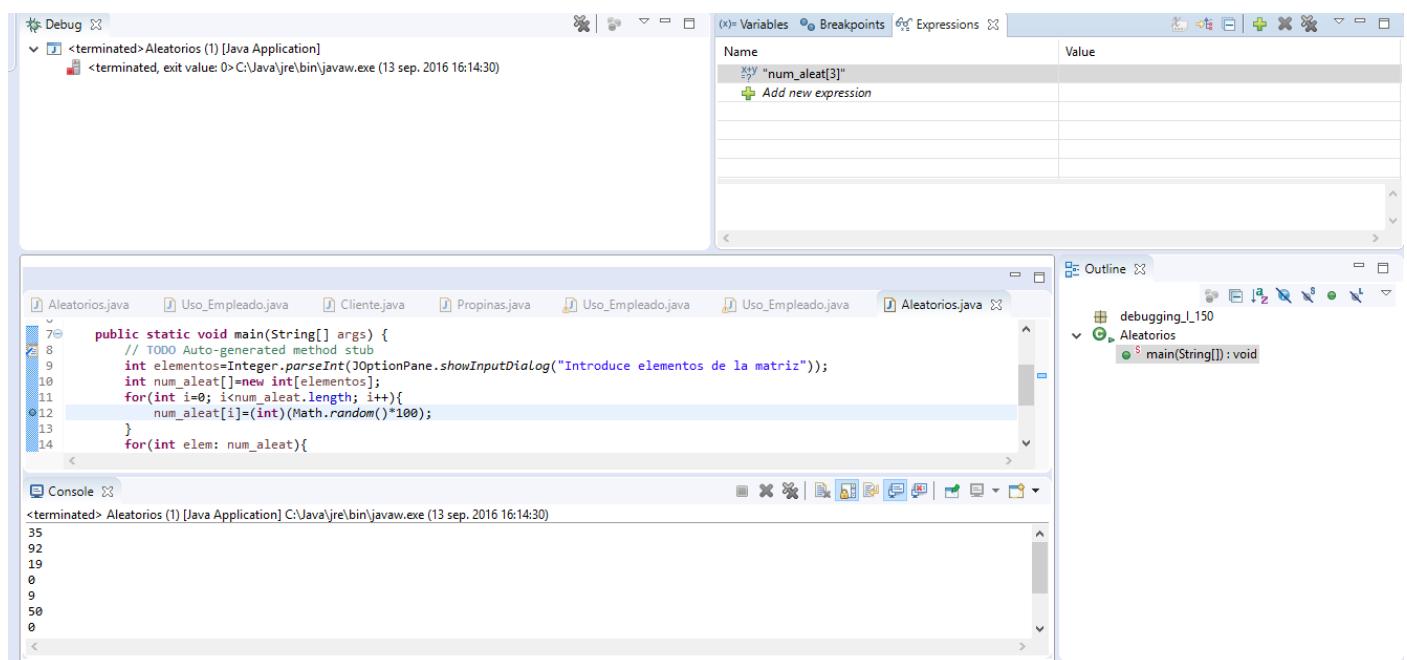
**Try{**  
Abrimos conexión BBDD;  
Ejecutamos sentencia SQL;  
Almacenamos datos en resulset;  
Accedemos y leemos resulset;  
Cerramos el resulset;  
  
**}catch(SQLException){**  
Instrucciones de control de Excepción;  
**}finally{**  
Cerramos conexión BBDD;  
**}**



## Vídeo 150. Depurando con Eclipse. Debugging I



## Vídeo 151. Depurando con Eclipse. Debugging II



## Vídeo 23. Arrays I

### Matrices (Arrays, Arreglos)

- **¿Qué es?** Estructura de datos que contiene una colección de valores del mismo tipo.
- **¿Para qué?** Para almacenar valores que normalmente tienen alguna relación entre sí.
- **Sintaxis:**
  - Declaración: `int[] mi_matriz=new int[10];`

Array es una serie de espacios en la memoria del ordenador donde iremos almacenando valores





```

Uso_Arrays.java ☐
1 package arrays_1_23;
2 public class Uso_Arrays {
3     public static void main(String[] args) {
4         /*int [] mi_matriz = new int[5];
5         mi_matriz[0]=5;
6         mi_matriz[1]=38;
7         mi_matriz[2]=-15;
8         mi_matriz[3]=92;
9         mi_matriz[4]=71;*/
10
11        //System.out.println(mi_matriz[0]);
12        System.out.println(mi_matriz[1]);
13        System.out.println(mi_matriz[2]);
14        System.out.println(mi_matriz[3]);
15        System.out.println(mi_matriz[4]);*/
16
17        int [] mi_matriz={5, 38, -15, 92, 71};
18
19        for(int i=0; i<mi_matriz.length; i++){
20            System.out.println("Valor del indice " + i + " = " + mi_matriz[i]);
21        }
22    }
23 }

Console ☐
<terminated> Uso_Arrays [Java Application] C:\Java\jre\bin\javaw.exe (16 sep. 2016 11:47:04)
Valor del indice 0 = 5
Valor del indice 1 = 38
Valor del indice 2 = -15
Valor del indice 3 = 92
Valor del indice 4 = 71

```

## Vídeo 24. Arrays II Bucle For Each

```

Uso_Arrays_II.java ☐ Uso_Arrays_II.java ☐
1 package arrays_II_24;
2 public class Uso_Arrays_II {
3     public static void main(String[] args) {
4         /*String [] paises = new String [8];
5         paises[0]="España";
6         paises[1]="Bolivia";
7         paises[2]="USA";
8         paises[3]="China";
9         paises[4]="Rusia";
10        paises[5]="Alemania";
11        paises[6]="Colombia";
12        paises[7]="México";*/
13        /*for (int i=0; i<paises.length; i++){
14            System.out.println("País " + (i+1) + " " +paises[i]);
15        }*/
16        String [] paises={"España", "Bolivia", "USA", "China", "Rusia", "Alemania", "Colombia", "México"};
17        for (String i:paises){
18            System.out.println("País: " + i);
19        }
20    }
21 }

Console ☐
<terminated> Uso_Arrays_II [Java Application] C:\Java\jre\bin\javaw.exe (16 sep. 2016 12:13:24)
País: España
País: Bolivia
País: USA
País: China
País: Rusia
País: Alemania
País: Colombia
País: México

```

```

Uso_Arrays_III.java ☐ Uso_Arrays_III.java ☐
1 package arrays_II_24;
2
3 import javax.swing.JOptionPane;
4
5 public class Uso_Arrays_III {
6     public static void main(String[] args) {
7         String [] paises = new String [3];
8         for (int i=0; i<3; i++){
9             paises [i] = JOptionPane.showInputDialog("Introduce país");
10        }
11        for (String i:paises){
12            System.out.println("País: " + i);
13        }
14    }
15 }

Console ☐
<terminated> Uso_Arrays_III [Java Application] C:\Java\jre\bin\javaw.exe (16 sep. 2016 12:14:55)
País: Bolivia
País: Chile
País: Peru

```

## Vídeo 25. Arrays III Matrices Bidimensionales

```
Arrays_Bidimensionales.java ✘
1 package arrays_III_Arrays_Bidimensionales;
2 public class Arrays_Bidimensionales {
3     public static void main(String[] args) {
4         int [][] matrix = new int [4][5];
5         matrix[0][0]=15;
6         matrix[0][1]=21;
7         matrix[0][2]=8;
8         matrix[0][3]=96;
9         matrix[0][4]=55;
10        matrix[1][0]=10;
11        matrix[1][1]=31;
12        matrix[1][2]=85;
13        matrix[1][3]=67;
14        matrix[1][4]=59;
15        matrix[2][0]=19;
16        matrix[2][1]=20;
17        matrix[2][2]=81;
18        matrix[2][3]=94;
19        matrix[2][4]=5;
20        matrix[3][0]=5;
21        matrix[3][1]=1;
22        matrix[3][2]=9;
23        matrix[3][3]=69;
24        matrix[3][4]=51;
25        //System.out.println("valor almacenado en la posición 2, 3: " + matrix[2][3]);
26        for (int i=0; i<4; i++){
27            for (int j=0; j<5; j++){
28                System.out.println("valor almacenado en la posición " + i + " " + j + ": " + matrix[i][j]);
29            }
30        }
31    }
32 }
33 }
```

```
Console ✘
<terminated> Arrays_Bidimensionales [Java Application] <
valor almacenado en la posición 0 0: 15
valor almacenado en la posición 0 1: 21
valor almacenado en la posición 0 2: 8
valor almacenado en la posición 0 3: 96
valor almacenado en la posición 0 4: 55
valor almacenado en la posición 1 0: 10
valor almacenado en la posición 1 1: 31
valor almacenado en la posición 1 2: 85
valor almacenado en la posición 1 3: 67
valor almacenado en la posición 1 4: 59
valor almacenado en la posición 2 0: 19
valor almacenado en la posición 2 1: 20
valor almacenado en la posición 2 2: 81
valor almacenado en la posición 2 3: 94
valor almacenado en la posición 2 4: 5
valor almacenado en la posición 3 0: 5
valor almacenado en la posición 3 1: 1
valor almacenado en la posición 3 2: 9
valor almacenado en la posición 3 3: 69
valor almacenado en la posición 3 4: 51
```

## Vídeo 26. Arrays IV Matrices Bidimensionales II

<https://www.youtube.com/watch?v=xEHkuRApCno&index=26&list=PLU8oAIHdN5BktAXdEVCLUYzvDyaRQJ2Ik>

**Lunes, 4 de Septiembre de 2017**

**Subir proyecto java a repositorio de github con eclipse**

[https://www.youtube.com/watch?v=pGTjdeX\\_Y48](https://www.youtube.com/watch?v=pGTjdeX_Y48)