APUNTES MÍOS DE JAVA

1. **SOBRE TIPOS DE DATOS**

Tipos de datos primitivos: byte short long int float double boolean char

Tipo especial (cadenas): String (es una clase)

Para meter un float hay que hacer float variable = 23.5f;

Para definir constantes: final;

1. **SOBRE ENTRADA DE DATOS**

Para leer datos por teclado: Scanner entrada = new Scanner(System.in);

Leer números: *int numero = entrada.nextInt();*

Leer flotante: *float numero = entrada.nextFloat();* 🡪 Hay que introducirlo con comas

Leer flotante: *double numero = entrada.nextDouble();* 🡪Hay que meterlo con comas

Leer cadenas: *String cadena = entrada.next ();* 🡪 Sólo guarda palabras (sin espacios)

Leer cadenas: *String cadena = entrada.next****Line*** *();* 🡪 Guarda frases

Leer carácter: *char caract = entrada.next().****charAt****(0);* 🡪 Guarda el primer carácter

1. **Entrada/salida de datos con JoptionPane (Ventanas Emergentes)**

Librería: javax.swing.JOptionPane

*Cadena = JOptionPane.****showInputDialog*** *(“Introduzca una cadena”) -- > Siempre devuelve un String*

Para un entero: *entero = Integer.parseInt (JOptionPane.showInputDialog (“Introduzca una cadena”));*

Para un carácter: *car = JOptionPane.showInputDialog (“Introduzca una cadena”).charAt(0);*

Para un doble: *doble = Double.parseDouble (JOptionPane.showInputDialog (“Introduzca una cadena”));* ***Con JoptionPane el flotante se introduce con punto***

Para un flotante: *float = Float.parseFloat (JOptionPane.showInputDialog (“Introduzca una cadena”));*

Para mostrar los datos: *JOptionPane.****showMessageDialog*** *(null, “la cadena es” + cadena)*

1. **OPERADORES MATEMATICOS (*app U\_Operadores*)**

Ve el +, -, \*, /, %, ++, --, +=, -=, \*=, /=, %=, =++, =--

Clase Math:

Raíz cuadrada: *double raiz = Math.sqrt(9)* 🡪el argumento debe ser double

Potencia: *double potencia = Math.pow(base, exponente)* 🡪 los argumentos deben ser double

Redondear: *long resultado = Math.round(numero)* 🡪 los argumentos deben ser double

*Int resultado = Math.round(flotante)* 🡪 si el argumento es float, el resultado devuelve un int.

Generar un número aleatorio: *double numero = Math.random()*;

**Ejercicios:**

1. Calcular e imprimir la suma de 3 calificaciones. Pedir los números al usuario.
2. Calcular el salario semanal de un empleado a partir de sus horas semanales trabajadas y de su salario por hora.
3. Guillermo tiene N euros. Luis tiene la mitad de lo que tiene Guillermo. Juan tiene la mitad de lo que tienen Luis y Guillermo juntos. Hacer un programa que imprima cuánto dinero tienen cada uno de ellos y entre todos.
4. Una empresa que vende coches usados paga a sus vendedores 1000€ mensuales más una comisión de 150€ por cada coche vendido más el 5% del valor de la venta del coche vendido. Calcular el salario de un vendedor de esta empresa.
5. Un estudiante de informática obtiene su nota final a partir de la siguiente ponderación de notas:
   * Participación: 10% de la nota final
   * Trabajos: 25% de la nota final
   * Primer examen: 25% de la nota final
   * Segundo examen: 40% de la nota final

Crear un programa que imprima su nota final.

1. Calcular el cuadrado de una suma:

(a+b)2 = a2 + b2 + 2ab

1. Construir un programa que, dado un número total de horas,devuelva el número de semanas, días y horas equivalentes.Por ejemplo: 1000 h son 5 semanas, 6 días y 16 horas.
2. Construir un programa que calcule e imprima las raíces de la ecuación de segundo grado de coeficientes reales.

ax2 + bx + c = 0;

1. Hacer el ejercicio 4 pero pidiendo el valor de cada coche vendido (usan bucles).
2. Introducir números enteros hasta que ponga fin (usa bucles).
3. **ALTERNATIVAS O CONDICIONALES (con JoptionPane) (*app U\_Condicionales*)**

* IF/IF … ELSE 🡪 If (condición){ } else { }
* SWITCH: switch (dato) {

Case caso1: hacer lo que sea en caso1;

break;

Case caso2: hacer lo que sea en caso1;

break;

default: qué hago en este caso;

break;

}

Con el break salimos del switch, si no ponemos break sigue preguntando por todos los demás casos.

* Operador Ternario : valor = (Condicion) ? : valor1 : valor2

Si la condición se cumple se guarda valor1 en valor. Si es falsa mete el valor2 en valor. Ejemplo: Cuando metamos un número, que saque si es par o impar

**EJERCICIOS:**

* 1. Hacer un programa que pida un número entero y muestre si es múltiplo de 10
  2. Pedir dos números y decir cuál es el mayor de ellos o si son iguales.
  3. Hacer un programa que pida un carácter y nos diga si es Mayúscula o no. Para ello hay que usar un método de la clase Character llamado isUpperCase(‘ ‘). Por ejemplo: if (Character.isUpperCase(letra))
  4. En MegaPlaza se hace un 20% de descuento a los clientes cuya compra supere los 300€. ¿Cuál será la cantidad que pagará una persona por su compra?
  5. Un obrero necesita calcular su salario semanal, el cual se obtiene de la siguiente manera:
     1. Si trabaja 40 horas o menos se le paga a 16€/hora.
     2. Si trabaja más de 40 horas se le paga 16€/hora por cada una de las primeras 40 horas y 20€ por cada hora extra.
  6. Hacer un programa que tome dos números y me diga si ambos son pares o impares.
  7. Pedir 3 números y ordenarlos de mayor a menor.
  8. Pedir un número entre 0 y 99999 y decir cuántos dígitos tiene.
  9. Pedir el día, mes y año de una fecha e indicar si la fecha es correcta, teniendo en cuenta los meses de 28, 30 y 31 días.
  10. Hacer lo mismo que i pero teniendo en cuenta los años bisiestos (donde febrero tiene 29 días).
  11. Construir un programa que simule el funcionamiento de una calculadora que puede realizar las cuatro operaciones aritméticas básicas (suma, resta, producto y división) con valores numéricos enteros. El usuario debe especificar la operación con el primer carácter del primer parámetro de la línea de comandos: S o s para la suman, R o r para la resta, D o d para la división y M o m para la multiplicación.
  12. Pedir una nota de 0 a 10 y mostrar su equivalente alfabético: Insuficiente, Suficiente, etc…
  13. Hacer un programa que simule un cajero automático con un saldo inicial de 1000€, con el siguiente menú de opciones:
      1. Ingresar dinero en la cuenta
      2. Retirar dinero en la cuenta
      3. Salir
  14. Hacer un programa que pase de kg a otra unidad de medida de masa. Mostrar en pantalla un menú con las opciones posibles (hectogramos, decagramos…miligramos.
  15. (JOptionPane) Hacer un programa que me diga qué hacer según el clima. Para ello vamos a hacer dos preguntas ¿Hace sol? Y ¿Hace viento? Y el usuario tendrá que responder Sí o No. Estas las pasaremos a dos variables booleanas y, dependiendo del valor de las mismas, haremos lo siguiente:
      1. Si hace sol y no hace viento 🡪Salgo a pasear
      2. Si hace sol y viento 🡪 Salgo a navegar
      3. Si no hace sol y no hace viento 🡪 Salgo de cervezas
      4. Si no hace sol y hace viento 🡪 Ir al cine

1. **BUCLES (con JoptionPane) (*app U\_Bucles*)**

* **WHILE:**  while (cond) { … código …}
* **DO WHILE:** do { … código …} while (condición}
* **FOR:** for (inicialización; condición; incremento) { … código …}

**EJERCICIOS DE BUCLES:**

* 1. Leer un número y mostrar su cuadrado. Repetir el proceso hasta que se introduzca un número negativo.
  2. Leer un número e indicar si es positivo o negativo. El proceso se repetirá hasta que se introduzca un 0.
  3. Leer números hasta que se introduzca un 0. Para cada uno indicar si es par o impar.
  4. Pedir números hasta que se teclee uno negativo y mostrar cuántos números se han introducido y cuántos han sido pares y cuántos impares.
  5. Realizar un juego para adivinar un número. Para ello generar un número aleatorio entre 0 y 100 y luego ir pidiendo números indicando “es mayor” o “es menor” según sea mayor o menor respecto al número N aleatorio. El proceso termina cuando el usuario acierta o cuando el número de intentos llega a 10.
  6. Pedir números hasta que se teclee un 0, mostrar la suma de todos los números introducidos.
  7. Pedir números hasta que se introduzca uno negativo y calcular su media aritmética (sin Math y con Math)
  8. Pedir un número N y mostrar todos los números del 1 al N.
  9. Escribir todos los números del 100 al 0 de 7 en 7.
  10. Calcular la suma de 10 números.
  11. Mostrar el producto de los 10 primeros números impares.
  12. Mostrar el factorial de un número. Por ejemplo: 3! = 3 \* 2 \* 1
  13. Pedir 10 números. Mostrar la media de los números positivos, la media de los números negativos y la cantidad de ceros.
  14. Pedir 10 sueldos. Mostrar su suma y cuántos hay mayores de 1500€.
  15. Dadas las edades y alturas de 5 alumnos, mostrar la edad y la estatura media, la cantidad de alumnos mayores de 18 años y la cantidad de alumnos que miden más de 1,75.
  16. Pedir un número entre 0 y 10 y mostrar su tabla de multiplicar.
  17. Una empresa que se dedica a la venta de desinfectantes necesita un programa para gestionar las facturas. En cada factura figura: el código del artículo, la cantidad vendida en litros y el precio por litro. Se pide, de 5 facturas introducidas, la facturación total, cantidad en litros vendidos del artículo 1 y cuántas facturas se emitieron de más de 1000€.
  18. Modificar el ejercicio anterior suponiendo que no se introduce el precio por litro. Solo existen tres productos con precios:
      1. 0,6 €/litro
      2. 3 €/litro
      3. 1,25 €/litro
  19. Dadas 6 notas, escribir la cantidad de alumnos aprobados, suspensos y condicionados (=4).
  20. Pedir un número N, introducir N sueldos y mostrar el sueldo máximo.
  21. Pedir 10 numeros e indicar si hay algún negativo.
  22. Pedir 5 notas y decir si hay algún suspenso.

1. **ARRAYS (*app U\_Arrays)***

**DEFINICIÓN:** Un array es una estructura de datos que nos permite almacenar un conjunto de datos del mismo tipo. El tamaño de los arrays se declara en un primer momento y **YA NO PUEDE** cambiar a lo largo del programa.

**DECLARACIÓN DE ARRAYS:** Tipo\_de\_ariable[ ] nombre\_array = new tipo\_variable[dimensión];

Por ejemplo: int[] edad = new int[4];

Long[] edad = new long[6]; Aquí se reservan 6 celdas de memoria.

**Nombrar elementos de un array:** nombre\_array[índice] (empiezan por 0).

Edad[0] = 5;

Int[] Edad = {5,6,7,1}

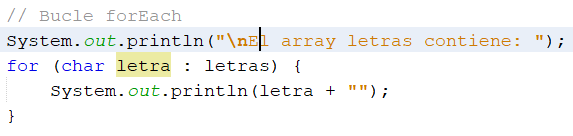
**Recorrer el array (Bucle forEach):** mejorado para arrays (atajo fore):

for (tipo\_array variable\_nueva : nombre\_array) {

……

}

Ejemplo:



**Arrays Bidimensionales:**

Son Arrays dentro de otro array. Por ejemplo:

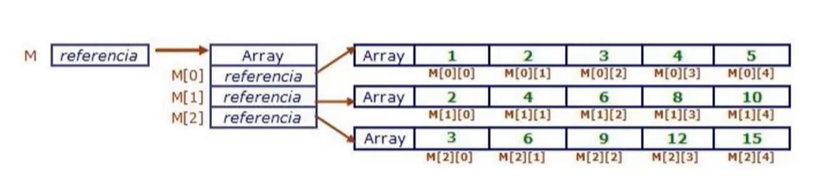
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

Los elementos de este tipo de arrays tienen dos índices: uno para especificar el array y otro para especificar el elemento dentro de ese array. Por ejemplo:

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

int x = myNumbers[1][2];

System.out.println(x); // Outputs 7

Representación en Java:

**FORMA DE EJECUTAR UN PROGRAMA CON ARGUMENTOS DE ENTRADA**

Pinchamos sobre el proyecto con el botón derecho, propiedades, run y en Argumentos le pasamos los argumentos separados por un espacio.

**EJERCICIOS DE ARRAYS:**

* 1. Hacer j, m, n, o, q, u y v de Condicionales, guardando los datos en un array.
  2. Pedir al usuario un número y crear un array con ese número, número-1, número-2..1 (como si fuera un factorial). Luego calcular su multiplicación (para sacar el factorial)
  3. Pedir al usuario un número y crear un array desde -n hasta n. Luego sumar todos los números del array.
  4. Leer 5 números reales(double), guardarlos en un array y mostrarlos en el mismo orden.
  5. Leer 5 números reales (float), guardarlos en un array y mostrarlos en orden inverso.
  6. Leer 5 números enteros, guardarlos en un array y mostrar la media de los números positivos, la media de los negativos y contar el número de ceros.
  7. Leer 10 números enteros y almacenarlos en un array. Mostarlos de la siguiente forma: el primero, el último, el segundo, el penúltimo…
  8. Leer por teclado dos arrays de 3 enteros y mezclarlos en un tercer array de la siguiente forma: el 1º de A, el 1º de B, el 2º de A, el 2º de B,etc….
  9. Leer los datos correspondientes a dos arrays de 12 elementos enteros y mezclarlos en un tercer array de la siguiente forma: 3 del primer array, 3 del segundo, los siguientes 3 del primero, los siguientes 3 del segundo, etc…
  10. Leer por teclado 10 números enteros y meterlos de menor a mayor en un array.
  11. Diseñar una aplicación que declare un array de 10 enteros. Leer mediante teclado sólo 8 números. Luego pedimos introducir un número y una posición e introducimos el número en la posición dada, desplazando el resto de números.
  12. Crear un programa que lea por teclado un array de 10 números y luego crear otro desplazando todos los numeros a la derecha, pasando el último a ser el primero.
  13. Crear un programa que lea por teclado un array de 10 números y desplace X posiciones en el array (X es introducido por el usuario)
  14. Leer 5 enteros. Ordenarlos de menor a mayor e introducirlos en un array de tamaño 10. Luego pediremos un número X y los insertaremos en el lugar adecuado para que la tabla continúe ordenada.
  15. Leer un array de 10 elementos y a continuación pedir una posición. Eliminar el elemento situado en esta posición sin dejar huecos.
  16. Leer 10 enteros, almacenándolos en un array. Luego crear otro array donde se almacenen primero todos los números pares y luego todos los impares.
  17. Leer 2 series de 10 enteros, que estarán ordenados de forma creciente. Copiar (fusionar) las dos tablas en una tercera de forma que sigan ordenados.
  18. Leer 10 enteros ordenados de forma creciente. Leer N y buscarlo en la tabla. Se debe mostrar la posición en que se encuentra. Si no está, indicarlo con un mensaje.
  19. Queremos desarrollar una aplicación que nos ayude a gestionar las notas de un centro educativo. Cada grupo (o clase) está compuesto por 5 alumnos. Se pide leer las notas del primer, segundo y tercer trimestre de un grupo. Debemos mostrar al final: la nota media del grupo en cada trimestre y la media del alumno que se encuentra en la posición Y (leído por teclado).

1. **Funciones**

Aquí he ido introduciendo el concepto de función y explicado el paso de parámetros. Aunque está más adelante, lo he introducido aquí para que se vayan familiarizando con los métodos de las clases.

Además, de esta manera, puedo preguntarlo en el primer examen.

**EJERCICIOS DE FUNCIONES:**

* 1. Hacer una función que me devuelva el MCD de dos números.
  2. Hacer una función que me devuelva el mcm de dos números.
  3. Hacer una función que me devuelva la media aritmética de los números de un array de enteros
  4. Hacer una función que pida X números (pasado como parámetro) y me devuelva un array ya creado con los números introducidos en él.
  5. Hacer una función que me muestre el contenido de un array unidimensional
  6. Hacer una función que me muestre el contenido de un array bidimensional.
  7. Hacer una función que me rellene un array de flotantes con un número determinado (ambos pasados como parámetro). Investigad una función util en las API de Java

1. **ENUMS (*app Enumeraciones)***

Una **enum** es una "clase" especial que representa un grupo de constantes (variables inmutables, como final variables).

Para crear una **enum**, use la palabra clave ***enum*** y separe las constantes con una coma. Tenga en cuenta que deben estar en mayúsculas:

***enum Level {***

***LOW,***

***MEDIUM,***

***HIGH***

***}***

Para acceder a las constantes de los enum se puede hacer mediante la sintaxis de puntos:

***Level miVar = Level.MEDIUM;***

Uso de Enum con Switch:

enum Level {

LOW,

MEDIUM,

HIGH

}

public class Main {

public static void main(String[] args) {

Level myVar = Level.MEDIUM;

switch(myVar) {

case LOW:

System.out.println("Low level");

break;

case MEDIUM:

System.out.println("Medium level");

break;

case HIGH:

System.out.println("High level");

break;

}

}

}

El tipo de enumeración tiene un  método ***values(),*** que devuelve un array de todas las constantes de enumeración. Este método es útil cuando se desea recorrer las constantes de una enumeración:

### Ejemplo

for (Level myVar : Level.values()) {

System.out.println(myVar);}

// Igualmente hay un método para pasar un String EXACTAMENTE IGUAL a la constante de la enumeración, aunque me daría error si me equivoco en alguna letra.

// Level myLevel = Level.valueOf(“MEDIUM”);

// System.out.println(myLevel);

1. **FECHAS EN JAVA (*app Enumeraciones: package Fechas )***

Java no tiene una clase Date incorporada, pero podemos importar el paquete ***java.time***  para trabajar con la API de fecha y hora. El paquete incluye muchas clases de fecha y hora. Por ejemplo:

|  |  |
| --- | --- |
| **Class** | **Description** |
| LocalDate | Represents a date (year, month, day (yyyy-MM-dd)) |
| LocalTime | Represents a time (hour, minute, second and nanoseconds (HH-mm-ss-ns)) |
| LocalDateTime | Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns) |
| DateTimeFormatter | Formatter for displaying and parsing date-time objects |

**MÉTODOS DE USO FRECUENTE**

LocalDate myObj = LocalDate.now() 🡪 Devuelve la fecha actual del sistema

LocalTime myObj = LocalTime.now(); 🡪 Devuelve el tiempo (horas, minutos, sg, nanosg)

LocalDateTime myObj = LocalDateTime.now();🡪 Devuelve la fecha y el tiempo

DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss"); 🡪 Devuelve la fecha y el tiempo con el formato indicado en el método ofPattern

Este último método ofPattern acepta una gran variedad de formatos, como pueden ser:

|  |  |
| --- | --- |
| **Value** | **Example** |
| *yyyy-MM-dd* | "1988-09-29" |
| *dd/MM/yyyy* | "29/09/1988" |
| *dd-MMM-yyyy* | "29-Sep-1988" |
| *E, MMM dd yyyy* | "Thu, Sep 29 1988" |

1. **ENTRADA Y SALIDA POR TECLADO (*app )***

[Lectura del tema 4](Temas/T4.%20Entrada-Salida%20por%20teclado-consola.La%20clase%20scanner.docx)

Lectura sólo de la clase Scanner.

1. **LA CLASE STRING (*app )***

[Lectura del tema 5](Temas/T5.1j.%20La%20clase%20String.doc)

Lectura sólo de la clase Scanner.

**PROGRAMACIÓN ORIENTADA A OBJETOS (POO)**

1. **CONCEPTOS DE POO (*app )***

La Programación Orientada a Objetos es una forma especial de programar, más cercana a cómo expresaríamos las cosas en la vida real.

Una clase es un conjunto de objetos con características similares.

Un objeto debe tener:

* Unas características (que expresaremos mediante variables o atributos).
* Una serie de acciones que puede realizar el objeto (que son los métodos o funciones).

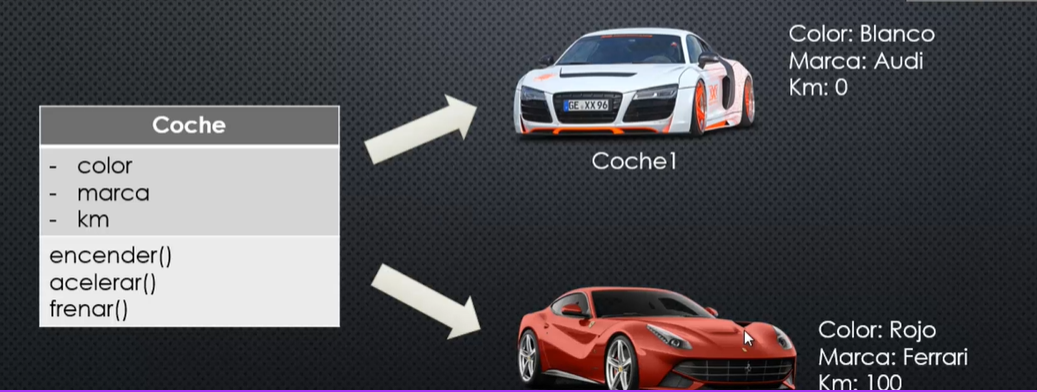
Los objetos pueden también tener sólo variables o sólo métodos, pero no es lo más usual.

Por ejemplo:

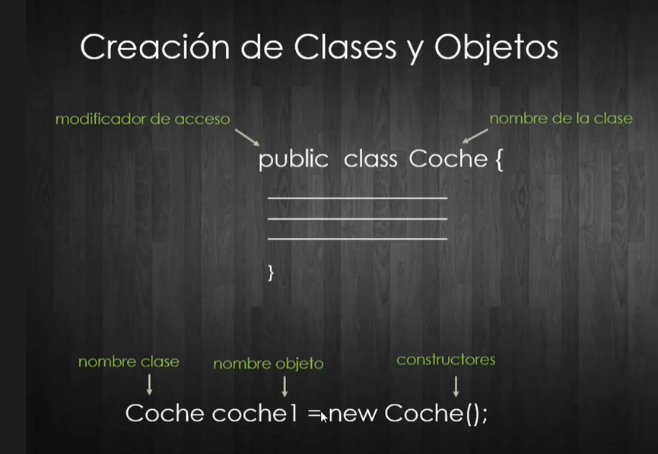


Aparte de los objetos físicos o materiales también podemos pensar en un objeto no físico, como por ejemplo, una fracción numérica.

Las **CLASES** son unas ‘plantillas’ que me dicen qué atributos y qué acciones debe realizar un objeto para pertenecer a esa clase. Un **OBJETO** en concreto será una **INSTANCIA O EJEMPLO** **CONCRETO** de esa clase, con unas características determinadas.

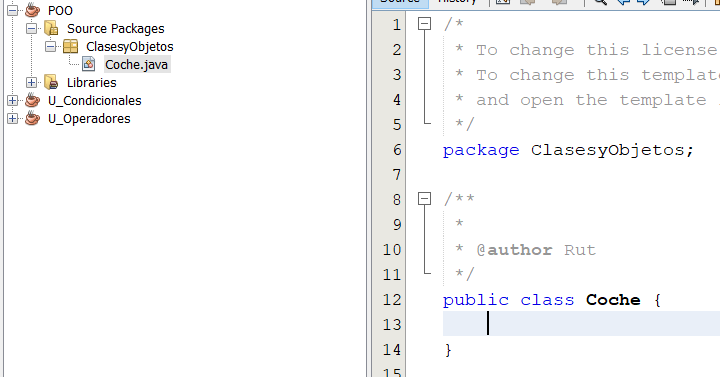


1. **CREACIÓN DE CLASES Y OBJETOS.**

La sintaxis es la siguiente:

Creamos un proyecto nuevo llamado POO.

Vamos a crear nuestra clase de ejemplo en Netbeans, teniendo mucho cuidado de desmarcar la creación de la clase main(). De esta forma, cuando se abre el Netbeans, éste ni siquiera reconoce o pone nombre al paquete o package. ¿Qué es un package? Es un conjunto de clases. Es como si fuera un directorio donde guardamos todas las clases.

Luego, tenemos que crear un nuevo paquete: (clic derecho sobre el paquete/New package y lo llamamos ClasesyObjetos. Ahora nos situamos encima del paquete y con el clic derecho creamos una nueva clase Java llamada Coche.

Ahora le damos a la clase coche sus características (color, marca, kmRecorridos) y sólo creamos, por ahora el método main() (dentro de la clase, como parte de sus métodos).

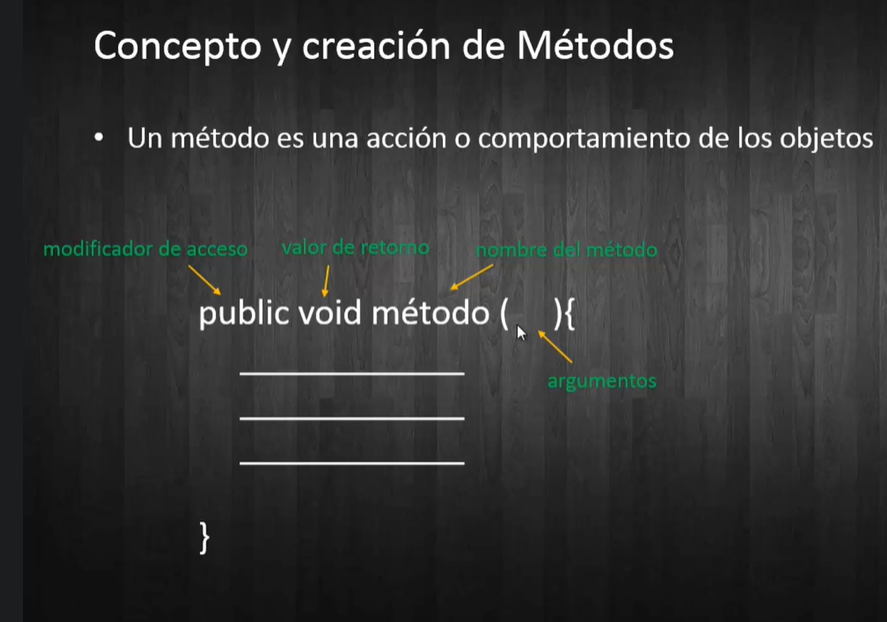
Luego crearemos el objeto lancia, instanciándolo con la palabra new y el objeto mercedes (dos objetos coche de la clase coche)

Ver cómo queda el código en el proyecto POO.

Como a mí no me gusta mucho que el método main quede dentro de la clase, voy a hacer otro proyecto ***POO2*** con el método main marcado y, dentro, voy a hacer mi nueva clase java Coche y voy a crear los objetos en el main del proyecto.

1. **CONCEPTO Y CREACIÓN DE MÉTODOS.**

Un método es una acción que puede realizar ese objeto o describe un comportamiento del mismo.

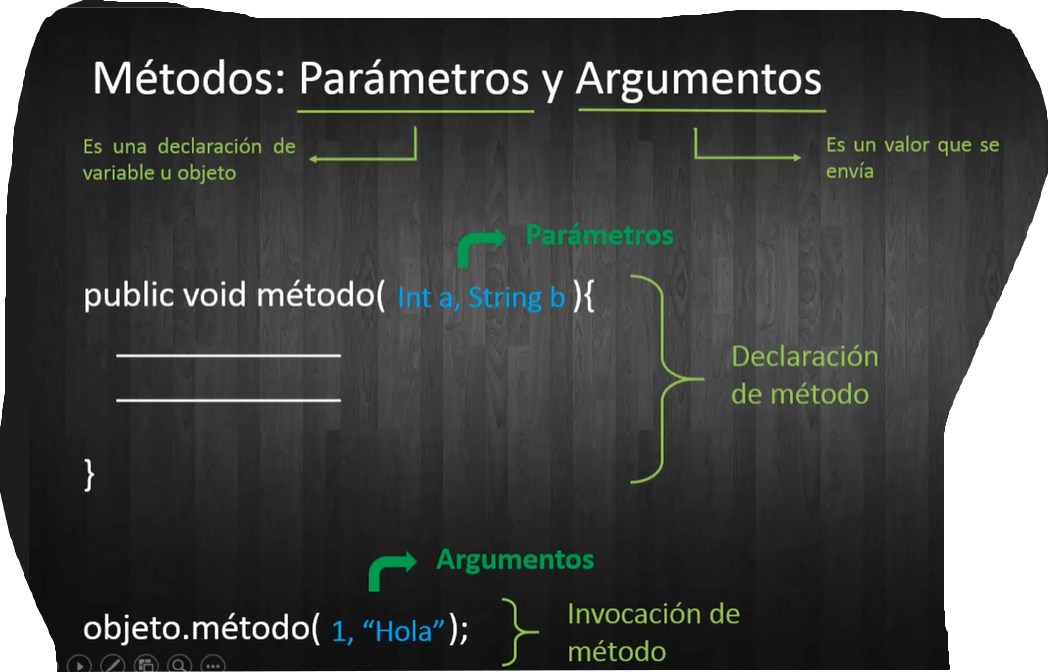
¿Cómo se declaran los métodos? Pues como las funciones:

1. **PARÁMETROS Y ARGUMENTOS DE LOS MÉTODOS.**

Un parámetro es una declaración de una variable en la definición de las funciones o métodos. Estos parámetros recibirán los valores que el programa principal quiera que la función use. Por ejemplo, en el método acelerar, a la hora de llamarlo, le pondría entre paréntesis cuánto quiero acelerar.

Esto es debido a que, como ya habíamos visto, el programa principal no puede ver lo que hay dentro de las funciones ni las funciones pueden ver lo que hay en el programa principal (porque está fuera de la función). Pero es necesario que programa principal y función se comuniquen, por tanto, los parámetros son una forma de pasarse datos entre programa principal y función.

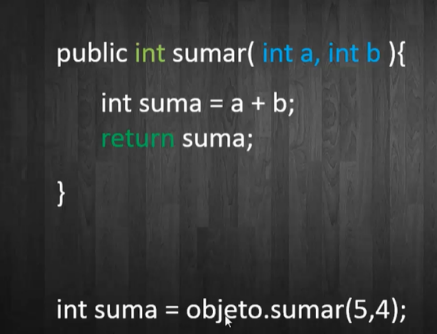
Se les llama parámetros a los que se definen en la función y argumentos a los valores que ponemos entre paréntesis al llamar a los métodos (desde el programa principal).

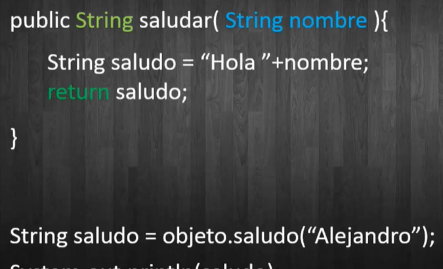


Ver un ejemplo del uso de parámetros y argumentos en el método acelerar de la clase Coche.

1. **MÉTODOS CON RETORNO DE VALORES**

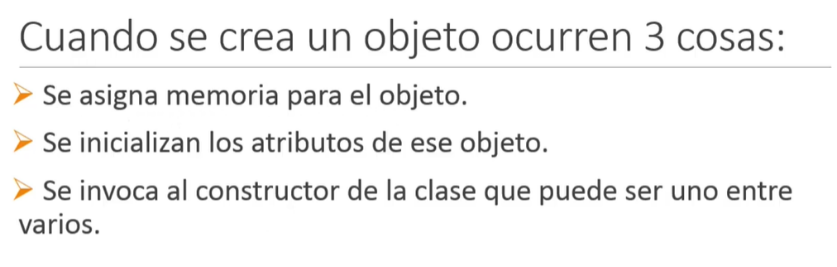
Los métodos o funciones pueden retornar o devolver una salida al programa principal que lo invoca.

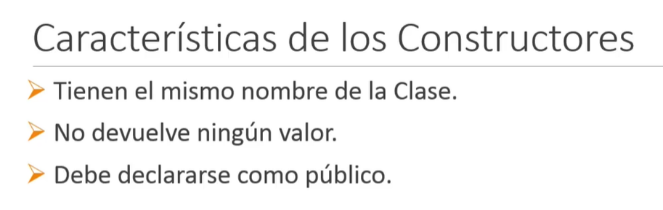
El tipo de dato que la función va a devolver debe ponerse después del modificador de visibilidad. P.eje: public int suma(int a, int b) 🡪 suma me devuelve un entero.



1. **MÉTODO CONSTRUCTOR.**

Un constructor es un método especial de la clase que se invoca cada vez que se crea un objeto.





**Pero atención!!! Aunque no devuelva ningún valor, NO SE LE PONE VOID AL MÉTODO CONSTRUCTOR. Es el único método que no lleva void.**

Como ejemplo vemos la clase Persona en el proyecto ***POO2*** también.

**Ejercicio**: cambiar los nombres de los parámetros de los métodos y ponerles el mismo nombre que los atributos de la clase, a ver qué pasa. (uso del **this**)

TIP!!: Clic derecho en Netbeans e Insert Code me inserta el código del método constructor

1. **SOBRECARGA DE MÉTODOS.**

La sobrecarga de métodos consiste exactamente en que puedo tener varios métodos con el mismo nombre SIEMPRE Y CUANDO estos métodos tengan **DIFERENTES PARÁMETROS** (en número o en tipo).

Observar el método correr en la clase Persona.

1. **MODIFICADORES DE ACCESO.**

Para ver un ejemplo creamos un nuevo proyecto ***MODIFICADORES*** pero sin el main marcado para que no lo cree porque voy a hacer varios paquetes.

A continuación creamos un paquete llamado Paquete1, con dos clases, Clase1 y Clase2.

Luego creamos otro paquete llamado Paquete2, con Clase3.

Ponemos los atributos a probar en la Clase1 y creamos un main “a mano” en la Clase2 y en la Clase3. Así podremos probar los diferentes tipos de atributos de la clase1.

**Modificadores**:

* Si no ponemos nada, ese atributo puede ser accedido desde todas las clases de un mismo paquete.
* Public, el atributo puede ser accedido desde cualquier clase de cualquier paquete.
* Private, el atributo sólo puede ser accedido por métodos de su propia clase.
* Protected, el atributo podrá ser accedido por métodos de su clase y otras subclases.

1. **ENCAPSULAMIENTO y getters y setters.**

Se encarga de **ocultar** los atributos o métodos para que sólo puedan ser accedidos desde la propia clase. ¿Por qué? Pues porque será la propia clase la que controle correctamente el contenido de sus atributos y nadie pueda coger y rellenar los atributos de mi clase con valores erróneos.

El encapsulamiento aisla a los atributos. El aislamiento **protege a los datos asociados** de un objeto **contra su modificación** por quien no tenga derecho a acceder a ellos, **eliminando efectos secundarios** e **interacciones** en cuanto al ocultamiento de los datos miembros de un objeto.

Por tanto, los atributos deben ser accedidos sólo por la clase. ¿Entonces cómo hago yo para manejar estos datos e intercambiarlos? Pues a través de los setters y getters que me devuelven o me establecen el valor de un atributo PERO CONTROLANDO SI HAY ERRORES O NO en su valor.

Además, hay otro método muy útil, llamado toString(), que me muestra por pantalla todos los valores de los atributos del objeto.

Ejemplo de getters y setters en la clase Persona2 de ***POO2***.

**Ejercicio**: Cambiar el método setEdad para que ponga siempre una edad buena.

1. **CONSTANTES.**

Las constantes son variables (o atributos) que NO CAMBIAN. Por ejemplo: el punto de ebullición del agua, el dni de una persona, el número PI.

Se crea con la palabra ‘***final’*** y NO TIENE MÉTODOS SETTERS para este atributo.

1. **MIEMBROS ESTÁTICOS DE UNA CLASE.**

Se consideran **MIEMBROS** de una clase tanto a los atributos como a los métodos de una clase.

A estos miembros estáticos se les llama también ‘**MIEMBROS DE CLASE’**. ¿Por qué se les llama así? Pues porque estos atributos de clase van a tener **el mismo valor para todas las instancias** u objetos que creemos que pertenezcan a la misma clase.

Por ejemplo: supongamos que creamos una clase Persona() y que tiene un atributo llamado numero\_piernas. Lo inicializamos a dos y ya todos los objetos persona que se instancien van a tener este valor puesto a 2.

¿Pero se puede cambiar el valor de estos atributos?

Pues claro que sí (no son final)!! Lo que pasa, que se los cambiarás a todos los objetos de esa clase.

Por ejemplo: si hay alguien que es cojo, dirá ¡¡yo solo tengo una pierna!!. No hay problema 🡪 cambiamos el atributo a 1, pero entonces ya dejaremos a todas las personas cojas.

Los miembros estáticos se declaran poniendo ***Static*** después del modificador de acceso. Por ejemplo:

Public ***static*** String frase = “”;

**Nota importante**: Se puede acceder a los miembros estáticos de una clase **SIN INSTANCIAR** objetos. Hasta ahora, para acceder a los miembros de una clase, teníamos que crear o instanciar un objeto con ‘new’. Sin embargo, si yo sólo quiero acceder a un atributo o método que sea estático, lo puedo hacer sin tener que crear el objeto, es decir, **sin hacer el ‘NEW’**.

¿Entonces cómo accedo a él si no tengo ningún objeto creado? Poniendo el nombre de la clase, un punto y luego el miembro estático. Ej:

Math.sqrt();

**Nota importante 2**: Si un método es estático y llama a otros métodos, éstos también tienen que ser estáticos.

Proyecto: ***MiembrosEstaticos***.

¿Puedes pensar algún atributo más de la clase Persona que pudiera ser estatico?

¿Y de otras clases?

1. **FACTORÍA.**

No es normal en los constructores que, cuando la fecha es mala (es decir, el día 88 del mes 44) me devuelva una fecha por defecto (aquí el 1-1-2000). Lo que es normal es que devuelva una excepción o un null. Lo que hay que hacer es crear la fecha solamente si esta es buena y si es mala devuelve un null. Por tanto, esto se hace poniendo los constructores de fecha privados. Pero entonces ¿cómo llamo al constructor si no tengo ninguno público?. Pues me creo uno público (damefecha del Proyecto ***TestFecha02***) que lo que hace es DEVOLVERME una instancia del objeto. Pero, a la hora de llamarlo desde fuera no puedo usar ese método público damefecha porque aún no he instanciado el objeto. Necesito un objeto para usar el método damefecha. Por tanto, tengo que hacer el **damefecha estático,** para que pueda ser referenciado sin crear el objeto, sólo con el nombre de la clase.

Esta forma de generar objetos se llama **Factoría**, que siempre se devuelve un objeto correcto o null.

Si se me olvida poner el constructor de la fecha en mayúscula puedo cambiarlo, pero esta vez se llama reestructurar no refactorizar.

1. **DIAGRAMA DE CLASES UML.**

Los diagramas UML van a ser representaciones gráficas de las clases que nos van a ayudar a entender mejor cómo están formadas y de quién heredan o qué interfaces implementan ( Proyecto ***TestArea***) .

Para crear estos diagramas UML tenemos que descargar un plugin de Netbeans llamado EasyUML. Como ya no está en los repositorios oficiales de Netbeans es posible que no lo encontremos pero no importa porque podemos descargarnos el paquete desde Moodle (pongo ahí el zip). Pero antes vamos a ver si se puede hacer desde el repositorio oficial yendo al Netbeans/ Tools/ Plugins/ Available Plugins y ahí ponemos EasyUML en la búsqueda y lo instalamos.

Si tenemos que instalarlo desde el zip que yo he colgado en la plataforma, entonces debemos descomprimir la carpeta y luego ir a Netbeans y en Tools/Plugins/downloaded le damos a añadir plugins. Ahí vamos a la carpeta que hemos descomprimido y seleccionamos todos los archivos nbm y le damos a instalar.

Ahora hay que hacer un nuevo Proyecto UML (File/New Project/UML) y ponerle un nombre. Una vez hecho este Proyecto UML nuevo, nos vamos al proyecto TestHerenciaA y sobre el paquete completo (con el símbolo del café) hago clic derecho, y pincho en ‘easyUML Create Class Diagram’ y entonces elijo el proyecto UML creado anteriormente.

**EJERCICIOS DE CLASES:**

1. (**Básico**) Crear una clase EnteroPositivo que sea capaz de leer un entero positivo por teclado (eliminando los posibles errores). Podrá:
   1. Decir si un número es impar o par
   2. Decir si un número es primo o no
   3. Poder decir si un número es primo o no sin tener que instanciar un objeto EnteroPositivo.
2. (**Básico**) Crear una clase Fecha que sea capaz de: detectar si una fecha es incorrecta, sumar x días a una fecha, restar x días a una fecha, sumar x meses a una fecha, restar x meses a una fecha, restar fechas. Esta clase Fecha tendrá los siguientes miembros: dd, mm, aa (todas de tipo entero). El constructor me devolverá la fecla del sistema por defecto.
3. Hacer un programa que calcule el área y el perímetro de un cuadrilátero dada la longitud de sus dos lados. Los valores de la longitud deberán introducirse por linea de comandos. Si es un cuadrado, sólo se proporcionará la longitud de uno de los lados al constructor. (***app TestArea***)
4. Hacer un programa que permita dirigir el movimiento de un objeto dentro de un tablero y actualice su posición dentro del mismo. Los movimientos posibles son ARRIBA, ABAJO, IZQUIERDA Y DERECHA. Tras cada movimiento el programa mostrará la nueva dirección elegida y las coordenada de situación del objeto dentro del tablero. (***app TestTablero***)
5. Hacer un programa que dada una serie de vehículos caracterizados por su marca, modelo y precio, imprima las propiedades del vehículo más barato. Para ello, se deberán leer por teclado las características de cada vehículo y crear una clase que los represente.
6. Crear un programa para una competencia de atletismo. El programa debe gestionar una serie de atletas caracterizados por su número de atleta, nombre y tiempo de carrera. Al final, el programa debe mostrar los datos del atleta ganador de la carrera.
7. Diseñar un programa para trabajar con triángulos isósceles. Para ello, defina los atributos necesarios que se requieren, proporcione métodos de consulta y un método constructor. Implementar métodos para calcular el perímetro y el área de un triángulo y otro que, a partir de un array de triángulos, devuelva el área del triángulo de mayor superficie.
8. Crear un programa para trabajar con 2 números complejos. Tiene que implementar el siguiente menú:
   1. Sumar dos números complejos.
   2. Multiplicar dos números complejos
   3. Comparar dos números complejos (iguales o no)
   4. Multiplicar un complejo por un entero. (***app TestNumeroComplejo***)
9. Hacer un programa sencillo para realizar gestiones en un banco. Tendremos dos clases: Cliente (nombre, apellido y DNI) y Cuenta(número y saldo). El cliente puede consultar el saldo así como ingresar y retirar dinero de sus cuentas. ***(app TestBanco***)
10. Una empresa de envío de paquetes tiene varias sucursales en todo el país. Cada sucursal está definida por su número de sucursal, direccion y ciudad. Para calcular el precio que cuesta enviar cada paquete, las sucursales tienen en cuenta el precio del paquete y la prioridad, sabiendo que se cobra un euro por kilo, 10 euros más si la prioridad es alta y 20 si es exprés. Cada paquete enviado tendrá un número de referencia y el DNI de la persona que lo envía. (***app TestPaquetería***)
11. (**Avanzado, Herencia**) Hacer un programa para calcular el área de Polígonos (Cuadrados, Triángulos y Rectángulos). El programa debe ser capaz de almacenar en un array X triángulos, rectángulos y cuadrados y, al final, mostrar el área y los datos de cada uno. Para ello tendremos:
    * 1. Una superclase llamada Polígono
      2. Una subclase llamada Rectángulo
      3. Una subclase llamada Triángulo
      4. Una subclase llamada Cuadrado
12. (**Avanzado, Herencia**) Crear un programa para simular un equipo de fútbol (futbolista, entrenador y médico). Para ello tendremos:
    * 1. Una superclase Persona, que tendrá los siguientes datos: nombre, apellido, dni y edad.
      2. La subclase Futbolista tendrá los siguientes datos: número y posición
      3. La subclase Entrenador tendrá de dato la estrategia que utiliza.
      4. La subclase Médico, la titulación y los años de experiencia.

Hacer un menú donde se tengan las siguientes opciones: Viaje de equipo, entrenamiento, partido de fútbol, planificar entrenamiento, entrevista y curar lesión.

1. (**Con ArrayList**) Crear una subclase empleado de Persona que tenga los siguientes atributos: puesto, salario, jefe (si tiene) y departamento al que pertenece. (ArrayList) Crear un programa que sea capaz de:
   * 1. Decir todos los empleados que tiene un determinado departamento
     2. Decir qué empleado tiene el máximo sueldo
     3. Aumentar el sueldo a un empleado
     4. Restar sueldo a un empleado
     5. Sacar un listado de los jefes de cada departamento
     6. Cambiar a un empleado de departamento
     7. Cambiar a un jefe de departamento
     8. Despedir a un empleado
2. **CONCEPTO Y TRATAMIENTO DE EXCEPCIONES**

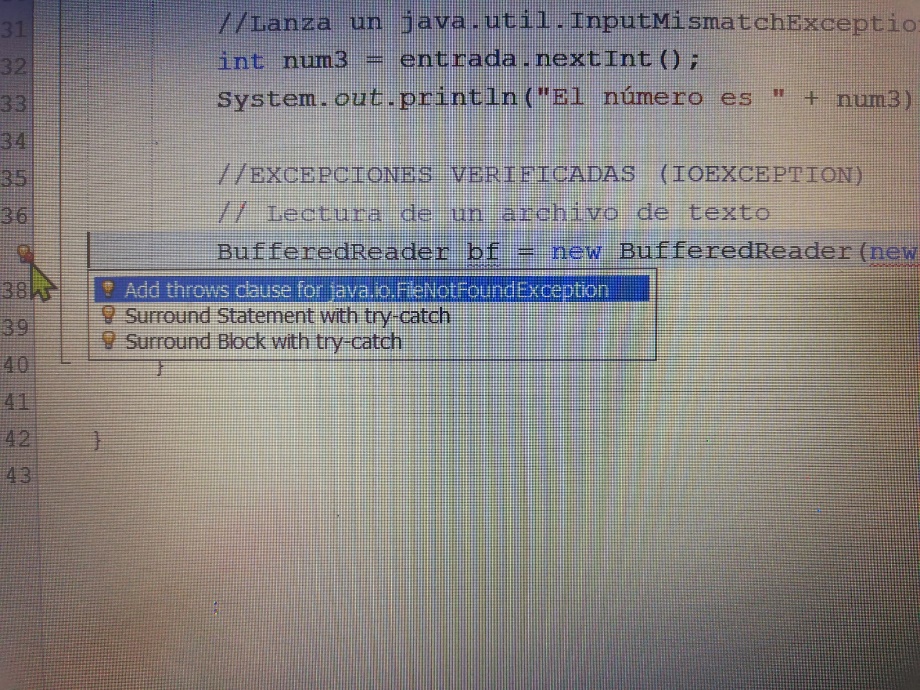
¿Qué son las excepciones?

Cuando un programa viola las restricciones semánticas del lenguaje se produce un error. JVM (Máquina Virtual de Java) comunica este hecho al programa mediante una excepción. Muchas clases de errores pueden provocar un error. Por ejemplo: un intento de dividir por cero, acceder a un elemento fuera de rango en un array, un disco duro estropeado,etc. Cuando esto ocurre, el JVM crea un objeto de la clase Exception y se para la ejecución.

Ejemplo: Proyecto ***ManejoExcepciones***

Jerarquía de Excepciones

La superclase de todas las Excepciones es Throwable. De aquí heredan las subclases Error ( fallos de hardware) y Exception. De la clase Exception heredan las siguientes subclases:

* **IOException**: Se llaman Excepciones Verificadas porque en principio todo está correcto, pero ocurre algo que el programador no puede controlar y entonces se produce la excepción. Se producen en cualquier entrada/salida. Por ejemplo: abrir ficheros. En principio si la ruta y todo está correcto todo funciona bien. Pero si en algún momento alguien borra el archivo, entonces se producirá un error. Java te obliga a tratar estos errores.

Java me obliga a:

* O bien declaramos las excepciones que se pueden encontrar en un método añadiendo, en su cabecera o definición, una cláusula ***throws FileNotFoundException.*** Como FileNotFoundException hereda de la clase IOException (como se ve en las API de Java), también podemos poner ***throws IOException*** e igualmente sería correcto. (Luego, evidentemente, deberíamos encerrar la **llamada a esta función** dentro de un try..catch)
* O bien no pongo la cláusula ***throws*** y la capturo dentro de la función, rodeando todo el trozo de código que causa la excepción con una cláusula ***try..catch***. Dentro del try vamos a poner todo el trozo de código que puede lanzar la excepción. Luego, con el catch, le indicamos que la capture en caso de que exista y le indicamos lo que tiene que hacer con ella. Si algún método lanza excepciones de varias clases (como se ve en el proyecto), lo más recomendable es capturarlas una por una, aunque si no se hace esto y se captura una excepción sólo, de la clase más genérica, también funcionaría.

Opcionalmente, se puede poner al final de todo una cláusula ***finally*** que se ejecutará SIEMPRE, encuentre o no encuentre una excepción.

Ejemplo:

try { leerArchivoTexto();

} catch (FileNotFoundException error){

JOptionPane.showMessageDialog(null,"El fichero no se ha encontrado!!");

} catch (IOException ex) {

JOptionPane.showMessageDialog(null, "Se ha producido un error genérico!");

} finally { System.out.println("Programa Terminado"); }

* **RunTimeException**: Se llaman Excepciones No Verificadas y éstas sí son responsabilidad del programador, pero Java no te obliga a tratarlas. ¿Cómo se hace el manejo de excepciones en este caso? Igualmente, hay que rodear el trozo de código en el que puede ocurrir el error con una cláusula try…catch. Para ello hay que ver qué tipo de Excepción puede lanzar (simplemente ejecutando el programa y viendo qué tipo de Excepción se lanza cuando ocurre el fallo). Y ya está, aquí NO TE OBLIGA A PONER la cláusula throws.

Lo que sí se suele hacer es lo que aparece en el proyecto de ejemplo, y es que se suele poner lo que puede producir errores dentro de un método (divEntreCero) y luego el try..catch se lo ponemos fuera, en otro método que llame a este.

**USO DE THROW** (para excepciones personalizadas)

Lo primero que hay que decir es que **NO HAY QUE CONFUNDIRLO CON THROWS**, que servía para declarar una excepción en un método. Las excepciones personalizadas son aquellas excepciones que nosotros, por algún motivo, queremos tratar de forma especial en nuestros programas. Por ejemplo: en un programa que pida introducir números, que cuando se introduzca un 0 lance una excepción.

Para ello, hay que crear una nueva subclase que trate la excepción. Esta subclase debe heredar de la superclase ***Exception***. Luego, cuando queramos lanzar “a mano” la excepción pondremos ***throw*** y crearemos un nuevo objeto de la subclase creada (que hereda de Exception). Ver el ejemplo en el proyecto ***ManejoExcepciones***.

Ya, al poner throw new Excepcion0() , Java me obliga a declarar en su cabecera (con throws) que este método va a lanzar una excepción.

Lo último sería llamar a este método que lanza la excepción dentro de un try .. catch.

1. **MÉTODO COMPARE-TO y EQUALS**

Para explicar el funcionamiento de estos dos métodos vamos a fijarnos en la aplicación ***TestProducto***.

Aquí, para hacer la comparación basta con usar el método compareTo de los Strings, aunque sería bueno meter esta comparación dentro de otra función que escribiera por pantalla qué objeto es el menor, como hemos hecho en TestEnteroPositivo (método menor).

Sin embargo, a la hora de escribir el método equals vamos a trabajar directamente con las referencias, de manera que primero preguntamos si las dos referencias son iguales (retornamos True). Si el parámetro es null o es de distinta clase a mi objeto devolvemos false. Una vez hecho esto “casteamos” el argumento para convertirlo al objeto de mi clase (en este caso Producto) y una vez hecho esto ya podemos utilizar el método equals del String nombre.

1. **ORDENANDO ARRAYS (interfaces Comparable y Comparator) *OrdenandoArrays***

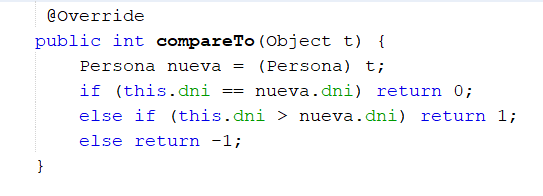
A la hora de comparar String o tipos primitivos de datos nos es suficiente con el == o con equals, pero a la hora de comparar objetos le tenemos que decir por qué atributo vamos a comparar. Es decir, si quisiéramos ordenar un array de personas, tendríamos que decirle cómo se comparan esas Personas, por qué campo, ¿por la edad, por el dni, por el nombre,etc…?

Bien, pues para poder hacer esto, vamos a ver dos formas. Estas dos formas incluyen la implementación de una INTERFAZ, que por ahora, aunque no sepamos qué es, no nos va a hacer falta. Simplemente vamos a hacer los pasos que detallo a continuación y ya entraremos en profundidad en lo que son las interfaces.

**INTERFAZ COMPARABLE**

Usamos esta interfaz para especificar por qué campo vamos a ordenar. La forma que tienen los objetos de ordenar es usando el método compare ó compareTo y es en este apartado donde vamos a intentar ver la diferencia entre estos dos métodos.

Para usar la interfaz Comparable tenemos que poner ***implements Comparable*** en la cabecera de la definición de la clase (como se aprecia en la aplicación OrdenandoArrays).

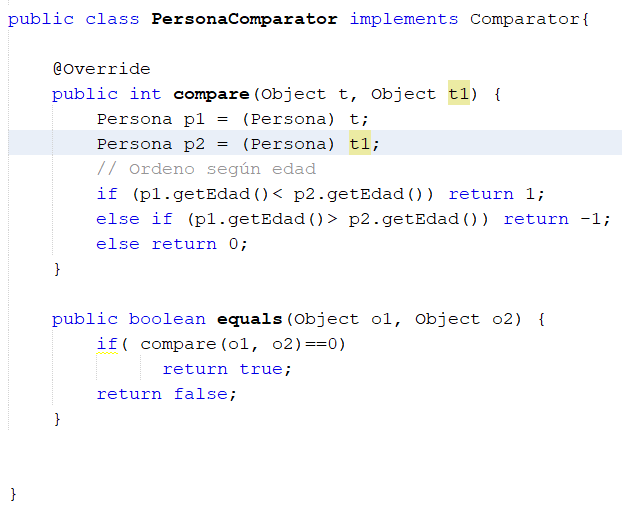
Al poner esto en la cabecera, Netbeans inmediatamente me obliga a codificar o implementar todos los métodos abstractos de dicha interfaz, en este caso, me oblida a implementar el método ***compareTo***. Es en este método donde le diremos por qué atributo va a ordenar (esta vez ordenamos por DNI).

**INTERFAZ COMPARATOR**

Todo esto está muy bien si yo sólo quiero ordenar por un solo campo (a la vez). Pero ¿qué pasa si yo quiero ordenarlo por dni y, además, por edad, y luego por nombre? Para estos casos se usa la interfaz Comparator.

En este caso tendremos **que CREAR UNA NUEVA CLASE de Persona**, (que se llame, por ejemplo, PersonaPorEdad o algo así) que ponga en su cabecera ***implements Comparator*** .

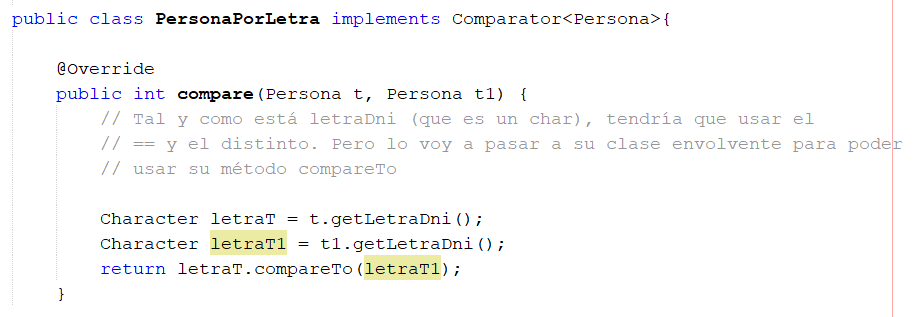
Al poner esto en la cabecera, Netbeans inmediatamente me obliga a codificar o implementar todos los métodos abstractos de dicha interfaz, en este caso, me oblida a implementar el método ***compare***. Es en este método donde le diremos por qué atributo va a ordenar (esta vez ordenamos por edad).



Es una buena práctica de programación también codificar siempre el ***equals*** (para saber cuándo dos objetos son iguales), aunque no es obligatorio.

**ATENCIÓN:**

Es verdad que podemos codificar nuestro objeto poniendo un método que se llame “compareTo” a pelo, es decir, como si fuera un método más de la clase Persona (o sea, sin implementar Comparable), pero entonces yo no podría usar todas las funcionalidades que me da implementar Comparable. Por ejemplo, no podría usar el método sort de los Arrays, Colecciones, etc…

En la aplicación se ve cómo también podemos ordenar de forma distinta con el Comparator, poniendo ‘***implements Comparator<Persona>***’. Lo único que cambia es que ya no tengo que castear los objetos al tipo Persona.

**Ver** el código de OrdenandoArrays ya que aquí se ve cómo ordenar de forma ascendente (Arrays.sort()), de forma descendente (Arrays.sort(nums2,Collections.reverseOrder()), de forma parcial y se ordenan, además, objetos Persona.

**COLECCIONES (ARRAYLIST, HASHMAP y HASHSET)**

1. **ARRAYLIST (*app Colecciones* )**

La clase ArrayList es una [matriz de](https://www.w3schools.com/java/java_arrays.asp) tamaño [variable](https://www.w3schools.com/java/java_arrays.asp) , que se puede encontrar en el paquete java.util.

La diferencia entre un array y un ArrayList en Java es que el tamaño de un array no se puede modificar (si queremos agregar o eliminar elementos a / de un array, debemos crear uno nuevo). Mientras que los elementos se pueden agregar y eliminar de un ArrayList cuando se desee.

**SINTAXIS**

ArrayList<*tipo\_base*> nombre\_variable = new ArrayList<*tipo\_base*>**()**;

ArrayList<String> cars = new ArrayList<String>();

**MÉTODOS**

* Add(elemento) : añadir un elemento a la lista. 🡪 cars.add("Volvo");
* Get(índice): obtiene el elemento situado en la posición índice. 🡪 cars.get(0);
* Contains (elemento): devuelve true si existe este elemento en el ArrayList. 🡪 cars.contains(“Volvo”);
* Set(índice, nuevo elemento): actualiza el elemento situado en la posición “índice” con el nuevo elemento 🡪 cars.set(0,"Mazda");
* Remove(índice): borrar el elemento situado en la posición índice. 🡪 cars.remove(0);
* Clear(): elimina TODOS los elementos del arraylist. 🡪 cars.clear( );
* Size(): me devuelve el tamaño del ArrayList 🡪 cars.size( );

Un array puede ser recorrido bien mediante un for bien mediante un foreach. Más adelante veremos la clase Iterator() que también se usa para recorrer ArrayList.

Los elementos de una ArrayList son en realidad objetos. Por tanto, podemos usar los String, pero para usar otros tipos, tales como int, se debe especificar su clase [contenedora](https://www.w3schools.com/java/java_wrapper_classes.asp) equivalente: Integer. Para otros tipos primitivos: Boolean para booleano, Character para char, Double para doble, etc.

Además, podemos hacer uso del método estático ***sort()*** de la clase Collections para ordenar un ArrayList. He aquí un ejemplo:

1. **HASHMAP (*app Colecciones* )**

Como vimos anteriormente, en los [***ArrayList***](https://www.w3schools.com/java/java_arraylist.asp) , los elementos se almacenaban en una colección ordenada y debíamos acceder a ellos con un número de índice ( de tipo *int*). Sin embargo, HashMap almacena los elementos en pares " **clave** / **valor** " y podemos acceder a ellos mediante un índice de otro tipo (por ejemplo, un índice de tipo *String*).

Un objeto se utiliza como clave (*índice*) para otro objeto (*valor*). Podemos almacenar diferentes tipos:  claves *String* y valores *Integer*, o el mismo tipo, como:  claves *String* y valores *String*.

**SINTAXIS**

HashMap<*tipo\_indice, tipo\_valor*> nombre\_variable = new HashMap<*tipo\_indice, tipo\_valor*> **()**;

HashMap<String, String> capitalCities = new HashMap<String, String>();

**MÉTODOS**

* Put (índice, elemento) : añadir un elemento al HashMap. Si el HashMap ya tenía un elemento con el mismo índice, el valor de este elemento es reemplazado por el nuevo valor 🡪 capitalCities.put("England", "London");
* Get(índice): obtiene el elemento situado en la posición índice. Si no hay ningún elemento con este índice, devuelve null 🡪 capitalCities.get("England");
* Remove(índice): borrar el elemento situado en la posición índice. 🡪 capitalCities.remove("England");
* Clear(): elimina TODOS los elementos del arraylist. 🡪 capitalCities.clear( );
* Size(): me devuelve el tamaño del ArrayList 🡪 capitalCities.size( );

**RECORRER UN HASHMAP**

Los elementos de un HashMap se recorren con un bucle **foreach** **para cada uno** . Para ello habrá que usar el método keySet() si solo deseamos recorrer las claves y el método values() si deseamos recorrer los valores:

// Print keys

for (String i : capitalCities.keySet()) {

System.out.println(i);

}

// Print values

for (String i : capitalCities.values()) {

System.out.println(i);

}

// Print keys and values

for (String i : capitalCities.keySet()) {

System.out.println("key: " + i + " value: " + capitalCities.get(i));

}

Los elementos de un HashMap son en realidad objetos. Por tanto, podemos usar los String, pero para usar otros tipos, tales como int, se debe especificar su clase [contenedora](https://www.w3schools.com/java/java_wrapper_classes.asp) equivalente: Integer. Para otros tipos primitivos: Boolean para booleano, Character para char, Double para doble, etc.

1. **HASHSET (*app Colecciones* )**

Tanto en los ***HashMap*** como en los [***ArrayList***](https://www.w3schools.com/java/java_arraylist.asp) , los elementos eran una colección ordenada de objetos que podían repetirse, sin embargo aquí los objetos NO pueden repetirse y cada elemento es único. Además, estos objetos no tienen ningún ORDEN.

**SINTAXIS**

HashSet<*tipo\_base*> nombre\_variable = new HashSet<*tipo\_base*> **()**;

HashSet<String> cars = new HashSet<String>();

**MÉTODOS**

* Add(elemento) : añadir un elemento a la lista. 🡪 cars.add("BMW"); (Si yo agregara dos veces este mismo elemento, me deja, pero luego, en la colección, sólo aparece uno)
* Contains (elemento): devuelve true si existe este elemento en el ArrayList. 🡪 cars.contains(“Volvo”);
* ~~Get(índice): obtiene el elemento situado en la posición índice. 🡪 cars.get(0);~~
* Remove(índice): borrar el elemento situado en la posición índice. 🡪 cars.remove(0);
* Clear(): elimina TODOS los elementos del arraylist. 🡪 cars.clear( );
* Size(): me devuelve el tamaño del ArrayList 🡪 cars.size( );

**RECORRER UN HASHSET**

Los elementos de un HashSet se recorren con un bucle **foreach**.

for (String i : cars) {

System.out.println(i);

}

Los elementos de un HashSet son en realidad objetos. Por tanto, podemos usar los String, pero para usar otros tipos, tales como int, se debe especificar su clase [contenedora](https://www.w3schools.com/java/java_wrapper_classes.asp) equivalente: Integer. Para otros tipos primitivos: Boolean para booleano, Character para char, Double para doble, etc.

**DIFERENCIAS ENTRE HASHSET Y OTRAS CLASES EXISTENTES EN JAVA**

* **HashSet**: El conjunto de datos no se almacena en un orden específico, si bien se garantiza que no hay duplicados.
* **TreeSet**: Los elementos del conjunto se almacenan de menor a mayor.
* **LinkedHashSet**: Los elementos del conjunto se ordenan según el orden en que fueron introducidos. Es similar a una lista, pero no deja introducir repetidos.

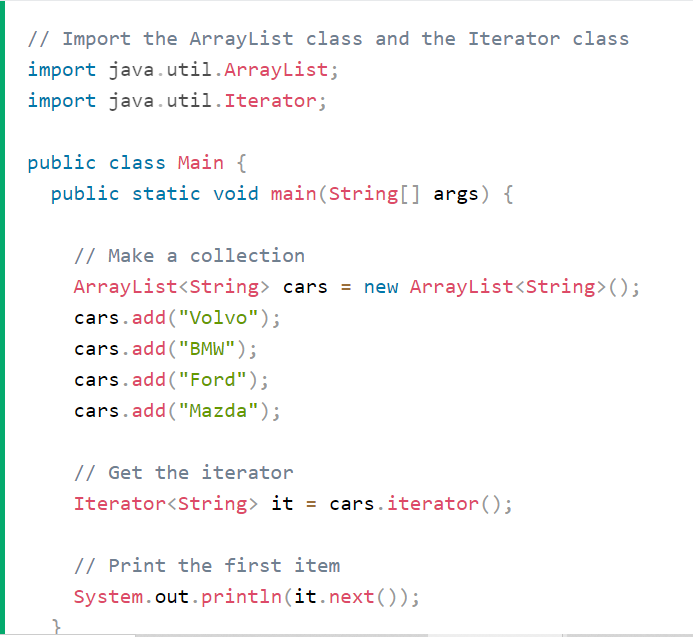
1. **ITERATOR (*app Colecciones* )**

Un ***Iterator*** es un objeto que se puede usar para recorrer colecciones, como [***ArrayList***](https://www.w3schools.com/java/java_arraylist.asp) y [***HashSet***](https://www.w3schools.com/java/java_hashset.asp) . Se le llama "iterador" porque "iterar" es el término técnico para hacer un bucle.

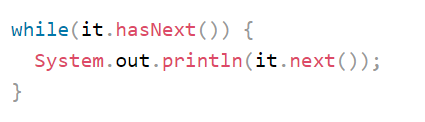
Para utilizar un iterador, debe importarlo desde el paquete ***java.util***.

**OBTENER UN ITERADOR**

El método iterator() se puede usar para obtener un objeto de tipo Iterator para cualquier colección.



**RECORRER UNA COLECCIÓN**

Para recorrer una colección se usan los métodos hasNext() y next() del iterador.

**ELIMINAR ELEMENTOS DE UNA COLECCIÓN**

Los iteradores están diseñados ESPECIALMENTE para que las colecciones que se recorren puedan cambiar de tamaño mientras el iterador las recorre. El  método remove() puede eliminar elementos de una colección mientras se repite.

**Nota:** Intentar eliminar elementos usando un **bucle for** o un **bucle for-each** no funcionaría correctamente porque la colección está cambiando de tamaño al mismo tiempo que el código lo está recorriendo.

1. **HERENCIA.**

La herencia es una forma de reutilización de software en la que se crea una nueva clase (subclase o clase hijo) que contiene los mismos miembros de una ya existente (superclase o clase padre).

Ejemplo: Clase Persona y subclase Estudiante en el proyecto Herencia.



Un estudiante “es una” persona, por tanto, tendrá todos los atributos de la clase padre más sus atributos propios. Igualmente, la subclase estudiante contendrá o podrá usar todos los métodos de la clase padre más los suyos propios.

**Nota importante**: Las subclases SOLO pueden heredar de una única superclase.

¿Cómo se indica que una clase hereda de otra?

Con la palabra clave ‘***extends nombre\_superclase***’.

Constructores, getters y setters de las subclases.

Tanto en el constructor como en los getters y setters de la subclase NO HAY QUE TOCAR LOS MIEMBROS de la superclase porque esos ya los inicializa la superclase.

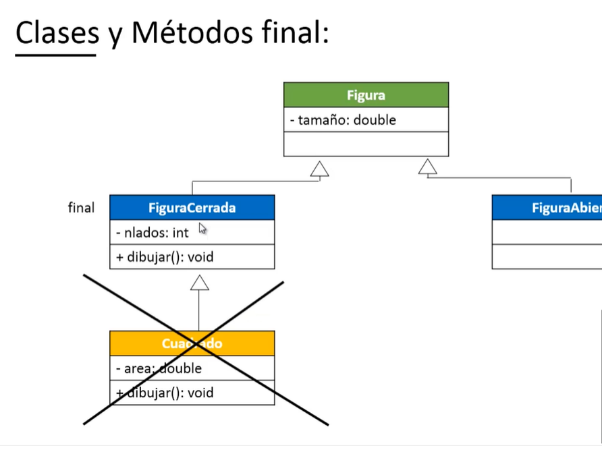
Por tanto, en el constructor, por ejemplo, para decirle que esos atributos los inicializa la superclase le pondremos ; ***super(attr1\_Superclase, attr2\_Superclase, attr3\_Superclase…).***

Los getters y setters de estos atributos no hace falta ni definirlos dentro de la subclase porque ya los tenemos definidos en la superclase.

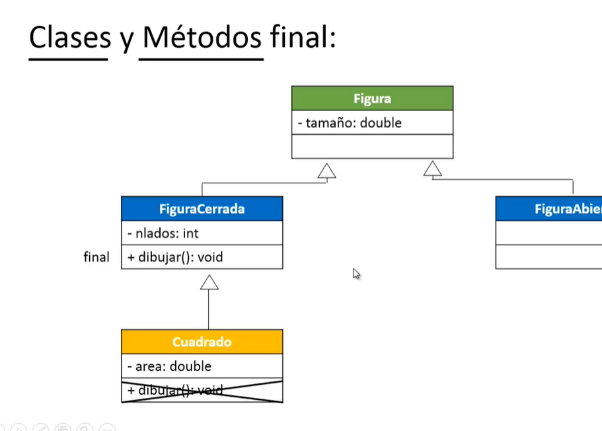
Ejemplo : ***Proyecto Herencia***.

1. **CLASES Y MÉTODOS FINAL.**

La palabra ‘final’ ya habíamos visto anteriormente que me iba a indicar que una variable no podía cambiar su contenido, es decir, que era una constante.

CLASES FINAL: Pues, en las clases final, lo que me va a indicar es que esta clase ya no va a poder tener más hijos. Esto se hace porque, a la hora de entregar un código a otro programador, y para que no estropee el código existente, podamos vetar que esta clase pueda tener más hijos.

MÉTODOS FINAL:

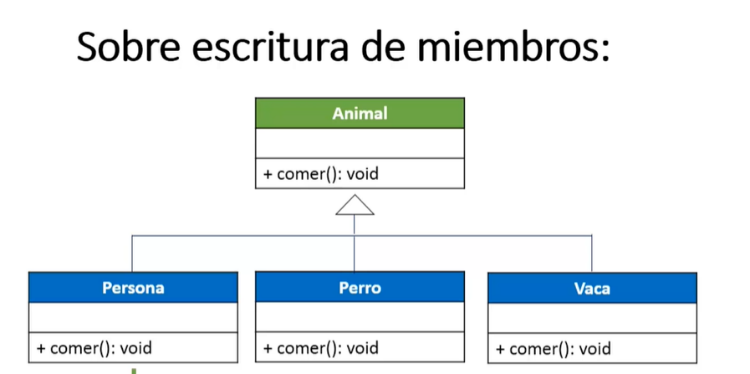
Existen métodos final para decirle a Java que **ya no puede haber otro método que se llame igual en ninguna de las subclases**.

Ejemplo : Proyecto ***FigsGeometricas***.

1. **SOBREESCRITURA DE MIEMBROS.**

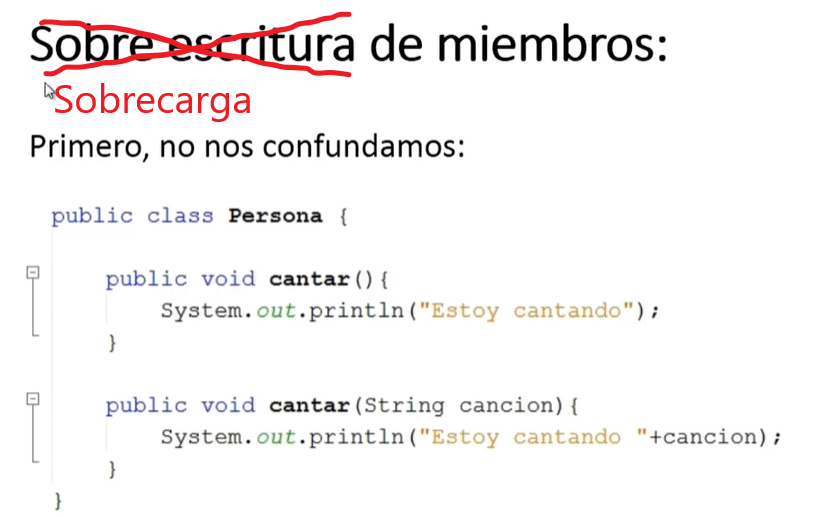
La sobreescritura de miembros significa que yo puedo tener en una subclase un método que se llame exactamente igual que otro de la superclase PERO que haga cosas diferentes. La sobreescritura de métodos está íntimamente ligada con la herencia.

Por ejemplo: el método hablar() no sería el mismo en un ser vivo persona, que en un perro o en un gato.



**NO CONFUNDIR CON LA SOBRECARGA DE MÉTODOS**.

La sobrecarga de métodos ocurre DENTRO DE UNA MISMA CLASE. Son métodos que tienen el mismo nombre, pero el número de parámetros no es el mismo.



Para indicar que un método está sobreescribiendo a otro que hay en la clase padre, se antepone la anotación @Override.

Si un método sobreescribe a otro de la clase padre quiere decir que hay otro método que se llama igual. Por tanto, cuando llamo a este método ¿cuál se ejecuta? Siempre el de la subclase.

Ejemplo : Proyecto ***Sobreescritura***.

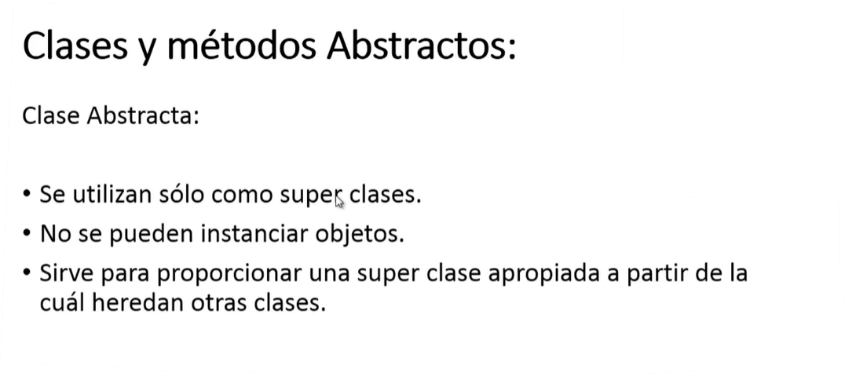
1. **CLASES Y MÉTODOS ABSTRACTOS.INTERFACES.**

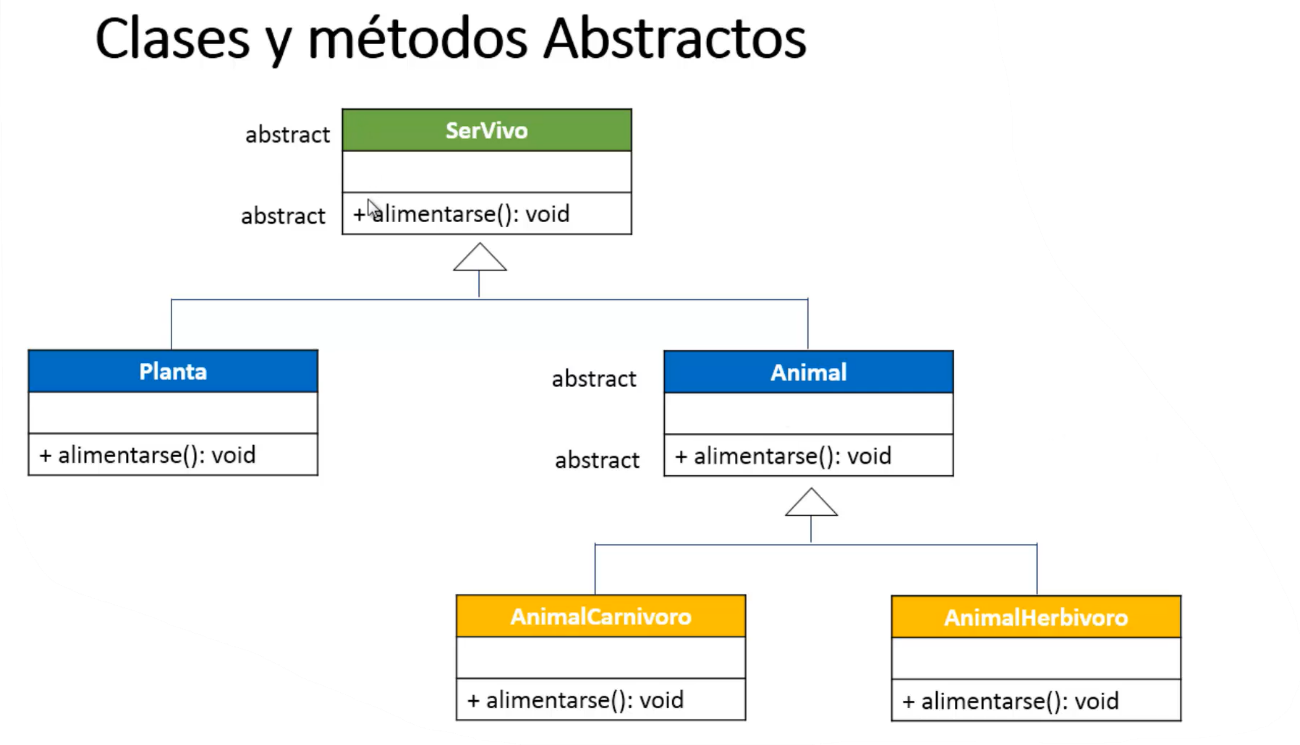
Las clases abstractas son clases que SOLAMENTE me indican qué cosas debería tener una determinada clase, es decir, funcionan como ***plantillas*** que dicen qué cosas tiene que tener una determinada subclase que herede de una clase abstracta.

Por tanto, igual que con las plantillas de word, estas plantillas nunca se rellenan, o lo que es lo mismo, **NUNCA SE INSTANCIAN**.

Además, los métodos abstractos nunca se implementan, **SÓLO SE DEFINEN**. Pero sí que es obligatorio que sean implementados por las subclases.

Las características de las clases abstractas, por tanto, se resumen a continuación:

Veamos el siguiente ejemplo:



El serVivo es una clase abstracta que me define que todo ser vivo tiene que tener A LA FUERZA un método que sea alimentarse. Pero como la clase ser vivo es muy general, no podemos implementarla, porque no sabemos cómo describir cómo se alimentan los distintos seres vivos. Sin embargo, en su hijo Planta sí sabemos cómo se alimentan (mediante fotosíntesis), pero en su hijo Animal (clase abstracta) aún no sabemos cómo se alimentan porque depende de que sean Carnívoros o Herbívoros.

En cuanto haya un método abstracto, la clase entera ya tiene que ser abstracta. Por tanto, serVivo y Animal son clases abstractas. Planta, sin embargo, no podría serlo, porque no tiene hijos y las clases abstractas siempre deben funcionar como superclases.

¿Cuáles, de todos estos métodos alimentarse(), deben ser implementados realmente?

Pues los de las clases últimas, Planta, Carnívoro y Herbívoro.

Las clases abstractas se señalan poniendo la palabra reservada ***Abstract*** después del modificador de visibilidad.

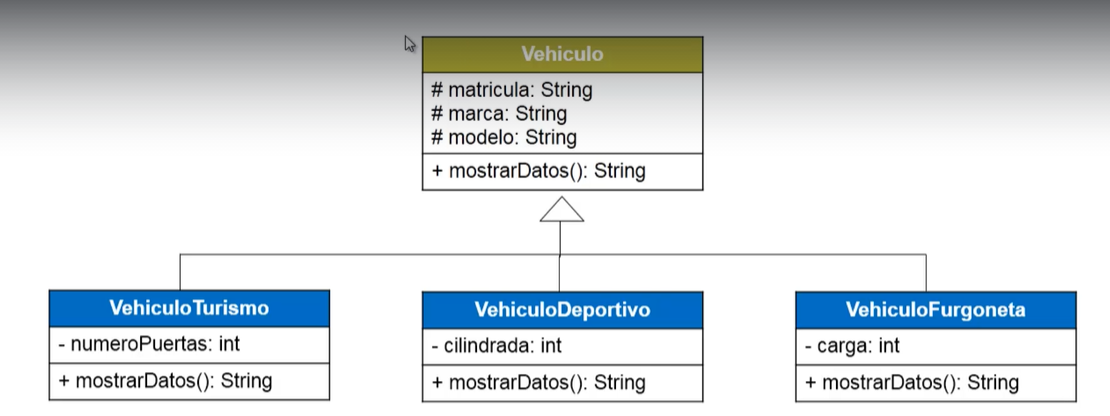
Ejemplo en el Proyecto ***ClasesAbstractas***.

1. **POLIMORFISMO.**

El polimorfismo siempre tiene que darse dentro de la herencia.

El polimorfismo signigica que, en una relación de tipo herencia, un objeto de la superclase puede almacenar un objeto de cualquiera de sus subclases.

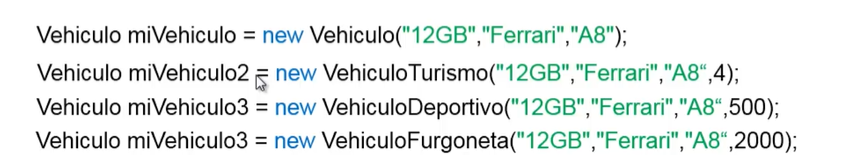
Por tanto, la superclase o clase padre debe ser compatible con los tipos que se derivan de ella, pero no al revés. Es decir, la subclase no tiene que ser compatible con la superclase.

Veamos un ejemplo:

Tal y como tenemos estas clases, Vehículo es una superclase y Turismo, Deportivo

y Furgoneta son sus subclases. Además, todas las subclases están sobreescribiendo el método mostrarDatos().

El polimorfismo me indica que un objeto de tipo Vehículo podría contener a un objeto de cualquiera de sus clases. Es decir, puedo crear objetos instanciando de la superclase directamente o de cualquiera de sus subclases.

He aquí un ejemplo:

Si nos damos cuenta, todos los objetos son de clase Vehículo, pero están instanciados de diferentes formas, o bien directamente instanciando de la superclase, o bien instanciando algunas de sus subclases. PERO NO SE PUEDE HACER AL REVÉS. Es decir, un VehiculoTurismo no puede hacer un new de tipo Vehículo.

Ejemplo en el proyecto ***Polimorfismo***.

1. **UPCASTING y DOWNCASTING.**

La conversión ascendente de tipos se denomina Upcasting y se entiende como parte del polimorfismo.

La conversión descendente de tipos se denomina Downcasting y también se entiende como parte del polimorfismo. Para que pueda haber un Downcasting tiene que haber habido previamente un upcasting. Además, esta parte ya no es tan directa, sino que tengo que usar un casting explícito para decirle a cuál de sus subclases se está refiriendo.

Ejemplo en el proyecto ***Polimorfismo***.

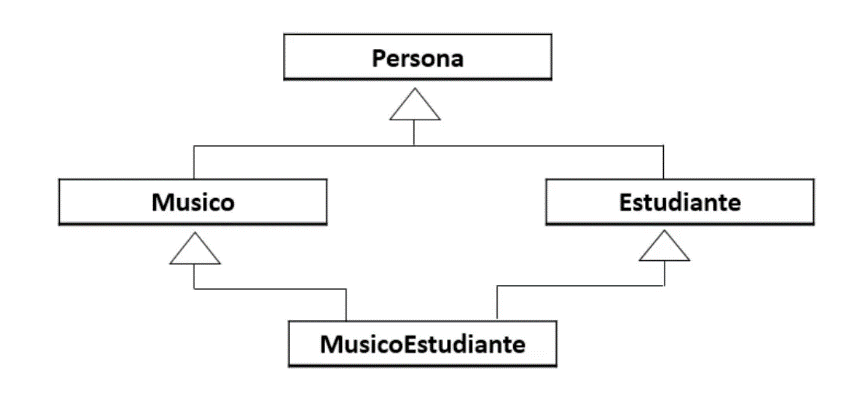
**EJERCICIOS POO**

* 1. Crear una clase EnteroPositivo que sea capaz de leer un entero positivo por teclado (eliminando los posibles errores). Podrá:
     1. Decir si un número es impar o par
     2. Decir si un número es primo o no
     3. Poder decir si un número es primo o no sin tener que instanciar un objeto EnteroPositivo.
  2. Crear una clase Fecha que sea capaz de: detectar si una fecha es incorrecta, sumar x días a una fecha, restar x días a una fecha, sumar x meses a una fecha, restar x meses a una fecha, restar fechas. Esta clase Fecha tendrá los siguientes miembros: dd, mm, aa (todas de tipo entero). El constructor me devolverá la fecla del sistema por defecto.
  3. Hacer un programa para calcular el área de Polígonos (Cuadrados, Triángulos y Rectángulos). El programa debe ser capaz de almacenar en un array X triángulos, rectángulos y cuadrados y, al final, mostrar el área y los datos de cada uno. Para ello tendremos:
     1. Una superclase llamada Polígono
     2. Una subclase llamada Rectángulo
     3. Una subclase llamada Triángulo
     4. Una subclase llamada Cuadrado
  4. Crear un programa para simular un equipo de fútbol (futbolista, entrenador y médico). Para ello tendremos:
     1. Una superclase Persona, que tendrá los siguientes datos: nombre, apellido, dni y edad.
     2. La subclase Futbolista tendrá los siguientes datos: número y posición
     3. La subclase Entrenador tendrá de dato la estrategia que utiliza.
     4. La subclase Médico, la titulación y los años de experiencia.

Hacer un menú donde se tengan las siguientes opciones: Viaje de equipo, entrenamiento, partido de fútbol, planificar entrenamiento, entrevista y curar lesión.

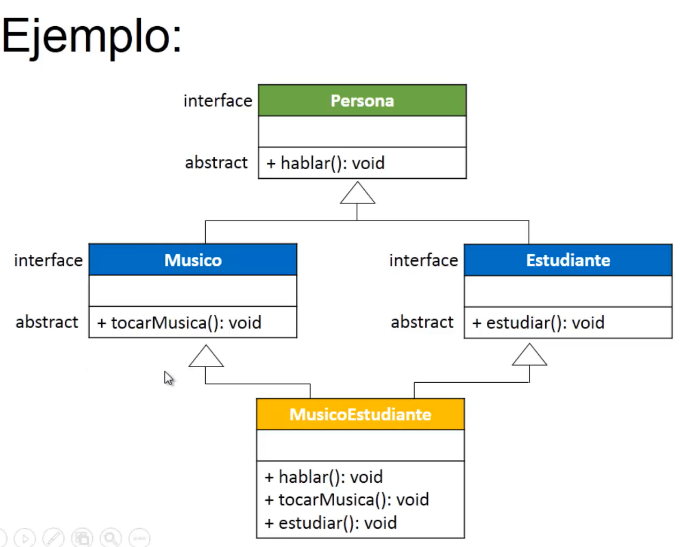
* 1. Crear una subclase empleado de Persona que tenga los siguientes atributos: puesto, salario, jefe (si tiene) y departamento al que pertenece. (ArrayList) Crear un programa que sea capaz de:
     1. Decir todos los empleados que tiene un determinado departamento
     2. Decir qué empleado tiene el máximo sueldo
     3. Aumentar el sueldo a un empleado
     4. Restar sueldo a un empleado
     5. Sacar un listado de los jefes de cada departamento
     6. Cambiar a un empleado de departamento
     7. Cambiar a un jefe de departamento
     8. Despedir a un empleado

1. **INTERFACES**

**Herencia múltiple**: hace referencia a la característica de los lenguajes de programación orientados a objetos en la que una clase puede heredar atributos y métodos de más de una superclase. En Java esto no se puede hacer directamente sino que se puede simular con las Interfaces.

**Interfaces en Java:**

* Permite simular la herencia múltiple
* La interfaz solo es public o default
* Todos sus métodos deben ser abstractos
* Todos sus atributos deben ser final.

 Ejemplo:

En este ejemplo, el único que realmente tiene que implementar los métodos es la clase MusicoEstudiante, porque los demás son Interfaces.

Para crear Interfaces hace falta clicar con el botón derecho en New/Java Interface.

Para indicar que una clase hereda de una Interfaz, hay que poner ***implements*** en vez de extends.

Ejemplo: Proyecto ***Interfaz***

**Creación de un proyecto para gestionar un Aeropuerto.**

Diseñar y codificar una aplicación informática para una compañía de gestión aeroportuaria. Requisitos:

1. Para cada aeropuerto es necesario saber:
   1. Todas las compañías de vuelos que operan en él
   2. Nombre del aeropuerto, la ciudad donde se sitúa y el país al que pertenece.
2. Cada compañía se caracteriza por el nombre y la lista de vuelos que ofrece.
3. Los vuelos están definidos por su identificador, la ciudad de origen, la de destino, el precio del vuelo, la lista de pasajeros, el número máximo de pasajeros permitidos en el vuelo y el número real de pasajeros que ha reservado asiento en el avión.
4. Los aeropuertos pueden ser privados o públicos:
   1. Los aeropuertos privados tienen una serie de empresas que los patrocinan y es necesario saber el nombre de cada una de estas empresas.
   2. Para los aeropuertos públicos se requiere saber la cantidad de dinero correspondiente a la subvención gubernamental.
5. Se necesita gestionar también la información de los pasajeros. Para cada uno se necesita saber:
   1. Nombre, número de pasaporte y nacionalidad.
6. La aplicación tendrá un menú con las siguientes opciones:
   1. Consultar los aeropuertos gestionados, indicando separadamente los aeropuertos públicos y los privados. Para cada uno de ellos deberá mostrar su nombre, la ciudad de ubicación y el país al que pertenece.
   2. Visualizar las empresas que patrocinan un determinado aeropuerto en caso de que sea privado, o la cuantía de la subvención en caso de que sea público.
   3. Para un determinado aeropuerto, se debe poder mostrar la lista de compañías que operan en él.
   4. Para una determinada compañía que opera en un aeropuerto concreto, listar todos los posibles vuelos que dicha compañía ofrece, mostrando su identificador, la ciudad origen y destino y el precio del vuelo.
   5. Mostrar todos los posibles vuelos (identificador) que parten de una ciudad origen a otra ciudad destino (indicadas por el usuario) y mostrar su precio.
7. **API DE JAVA**

La API DE JAVA es una interfaz de programación de aplicaciones provista por los creadores del lenguaje de programación y que da a los programadores los medios para desarrollar aplicaciones Java.

Como Java es POO, la API de Java nos da un conjunto de clases que se pueden usar para efectuar toda clase de tareas necesarias dentro de un programa.

Además, está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente.

La lista de todas estas clases se encuentran en la página oficial de Java: <https://docs.oracle.com/javase/8/docs/api/>.

Nosotros estamos trabajando con la versión 8, donde hay multitud de paquetes y otro apartado con todas las clases posibles.

El paquete básico con el que estamos trabajando es java.lang.

Cada vez que accedamos a una clase me da la explicación de sus campos o atributos y la lista de métodos que se pueden usar.

1. **ARCHIVOS DE TEXTO Y BINARIOS**

Hasta ahora hemos ido guardando nuestra información en la memoria, por tanto, cada vez que salíamos del programa, la información generada por el programa se perdía. Ahora vamos a traspasar esta información a ficheros para que ésta pueda permanecer para siempre en nuestro sistema (a esto se le llama PERSISTENCIA).

**¿QUÉ SON LOS ARCHIVOS?**

* Un archivo o fichero informático es un conjunto de bits almacenados en un dispositivo.
* Un archivo se identifica a través de su nombre y su path dentro del sistema operativo.
* Los archivos informáticos facilitan una manera de organizar los recursos usados para almacenar permanentemente los datos en un sistema informático virtual.

**TIPOS DE ARCHIVOS**

1. DE TEXTO: son reconocidos por su contenido, ya que éstos puede abrirse y leerse con cualquier editor de textos.
2. BINARIOS: pueden contener información de cualquier tipo codificada en binario para el propósito de almacenamiento y procesamiento en ordenadores. Estos archivos podrán ser leídos desde el mismo programa que los creó. Es decir, si se crearon con Java han de leerse con Java, si desde PHP pues con PHP, etc..

**LA CLASE FILE**

La clase Java que se usa para representar cualquier fichero es la clase File.

**Constructores:** no hay constructor vacío y hay muchos tipos de constructores con diferentes parámetros. El constructor que más se suele usar es el que tienen una cadena de texto como parámetro con el nombre del fichero. (File archivo = new File(“nombre del fichero.ext”))

**Métodos más comunes:**

archivo.exists(): me indica si el archivo existe o no

archivo.getAbsolutePath: me devuelve la ruta absoluta donde se encuentra mi archivo

archivo.isFile() || archivo.isDirectory(): me indica si es un fichero o un directorio

archivo.canRead(): me indica si tenemos permisos para leer el fichero

**ARCHIVOS DE TEXTO**

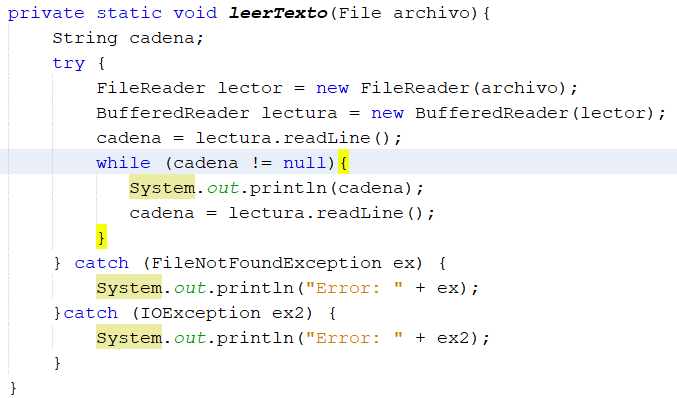
Ver la app ***ArchivosTexto***.

**Clases implicadas: File** archivo, **FileReader**, **BufferedReader**;

**Métodos:**

* **archivo.createNewFile().** Si el archivo no existe pues lo crea y, si existe, pues no lo vuelve a crear PERO NO DA NINGÚN ERROR, es decir, no salta ninguna excepción.
* **archivo.mkdir();**
* Para escribir texto sobre un archivo ya creado: hay que crear un **objeto del tipo FileWriter**. Escribir con el método **write** y CERRARLO SIEMPRE (con archivo.**close()**)!!!
* Para escribir un retorno de carro y luego un salto de línea (para que escriba en líneas nuevas) habría que poner ‘\r\n’ antes de escribir la nueva línea.
* Para que abra el fichero para añadir más líneas y que no se cargue lo que había anteriormente, hay que poner un ‘true’ como segundo parámetro al crear el fichero.

*FileWriter escribeFich = new FileWriter(fich, true);*

* Para LEER de un fichero de texto. Necesitaremos dos objetos más; el FileReader y el BufferedReader.
* El FileReader crea un objeto que maneja los archivos para leerlos (aquí lo llamo lector) y luego hay que pasarle ese lector al objeto BufferedReader. Ver el ejemplo en la app.

EJERCICIO 1 FICHEROS

**(pensar si hacer estos ejercicios antes o después de la interfaz gráfica) (Yo los tengo con GUI)**

Realizar una aplicación para grabar una agenda de contactos en un fichero de texto. Grabaremos en cada línea el nombre y apellidos, el teléfono y el correo electrónico(separados por un # entre ellos). También debe ser capaz de mostrar todos los contactos y añadir todos los que quiera.

**ARCHIVOS BINARIOS**

Son archivos que pueden guardar cualquier tipo de datos, no solamente texto, como por ejemplo, los objetos. ¿Y los objetos pueden guardarse en archivos? Claro que sí, ya que al final todo se graba en forma de un ‘chorro’ de bits. Aunque, para hacer esto, tenemos que transformar nuestros objetos a este chorro de bits, es decir, vamos a **SERIALIZARLOS**. Si queremos que los objetos de una clase puedan ser guardados en archivos, tenemos que hacer que dicha clase ***implemente*** la interfaz ***Serializable***.

Las extensiones de estos ficheros pueden ser cualquiera pero normalmente suelen ser ‘.bin’.

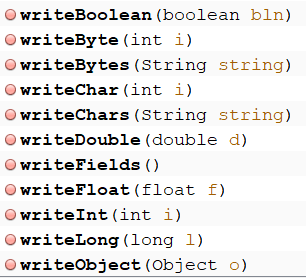
Ver la app ***ArchivosBinarios***.

**Clases implicadas: FileOutputStream** archivo, **ObjectOutputStream**, ;

**Métodos:**

* **Declaración. ObjectOutputStream fileEscritura = new ObjectOutputStream (new FileOutputStream(“personas.bin”);**
* **Escritura: writeObject** para **Objetos**. **fileEscritura.writeObject(new Persona());**

Si queremos escribir otro tipos de datos usaremos los siguientes métodos write:



* **Cerrar** archivo: (con fileEscritura.**close()**)!!!
* Para **LEER** de un fichero binario. Vamos a **DESERIALIZAR**, es decir, convertiremos el chorro de bits en un objeto a través de un simple casting. Usaremos los tipos File**Input**Stream y Object**Input**Stream. A la hora de leer vamos a usar el método **readObject()**. Este método lanza una excepción **EOFException** cuando acaba de leer, por lo que lo meteremos en un bucle simple infinito y nos saltará automáticamente fuera de él, al llegar la excepción (que tenemos que capturar, claro).
* En principio, sería recomendable tener un método que escribiera primero y luego de lectura, como en la aplicación, ya que los objetos ObjectOutputStream y el ObjectInputStream tienen que corresponderse, aunque funcionaría de todas formas con un método de solo lectura sobre el fichero ya creado.
* **Añadir contenido** a un fichero binario:
  + Al hacer el new del FileOutputStream añadimos el segundo parámetro poniendo true, como en los archivos de texto**.**
  + Luego lo intentamos llamar desde el main, pero dará un error, una **excepción StreamCorruptedException: invalid type code: AC.** ¿Por qué? Se debe a que cada vez que hacemos un new ObjectOutputStream, éste escribe una cabecera inicial en el fichero, al volver a abrirlo para añadir el objeto vuelve a escribir esta cabecera en mitad del fichero y ya es una cabecera distinta y eso descoloca al método, que lanza la excepción.

¿Cómo lo arreglamos?

Vamos a crearnos otra clase diferente a ObjectOutputStream, por ejemplo, la clase **AñadeContenido**, que va a heredar de ObjectOutputStream, que lo va a tener todo igual menos que, al escribir, vamos a escribir sin cabeceras. Para ello sobreescribimos el método writeStreamHeader (que es el que escribe las cabeceras) y le ponemos reset(); para que no escriba la cabecera.

Luego, en AñadirBinario hay que escribir con la clase AñadeContenido, en vez de con ObjectOutputStream.

**RECORDAR**:

Solamente puedo usar ObjectOutputStream, si es la primera vez que voy a escribir en el fichero binario!!!

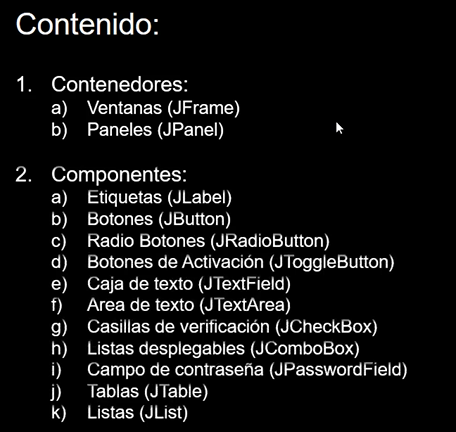
EJERCICIO 2 FICHEROS

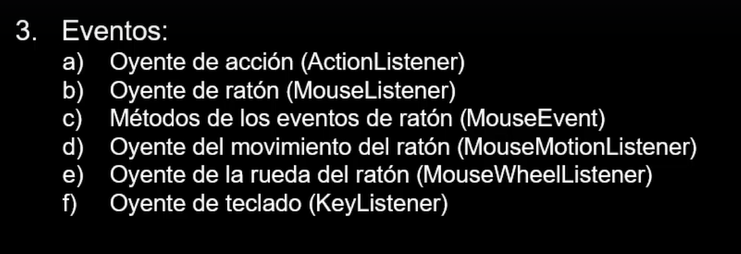
Realizar una aplicación para gestionar las inscripciones de una competencia de atletismo. Vamos a tener un apartado para dar de alta a un atleta (pidiendo nombre, edad, peso y estatura) y otro para mostrar todos los atletas registrados. Estos atletas se guardarán en un archivo binario.

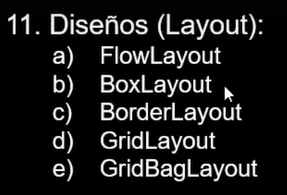
**(pensar si hacer estos ejercicios antes o después de la interfaz gráfica) (Yo los tengo con GUI)**

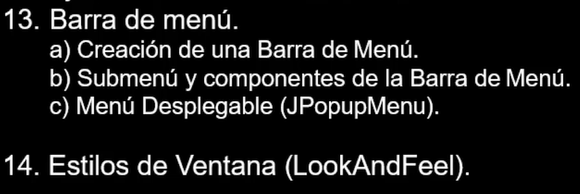
1. **INTERFAZ GRÁFICA DE USUARIO**

La INTERFAZ GRÁFICA DE USUARIO es conocida como la GUI por sus siglas en inglés. Mediante esta interfaz seremos capaces de dotar de elementos gráficos tales como ventanas, botones, etc… a nuestros programas, de forma que tengan una interfaz mucho más amigable. También aprenderemos a crear .jar y .exe para ejecutar el proyecto en cualquier máquina.

****Antes de pasar a mayores, aclararemos el contenido que vamos a ver en este punto.







**VENTANAS (Jframe)**

Todo el entorno gráfico va a usar el paquete javax.swing.

**CREACIÓN DE VENTANAS**

Las Ventanas deben heredar de Jframe. Dado que el JFrame hereda de un montón de clases (Frame, Window, Container, Component, etc..) vamos a tener a nuestra disposición un montón de métodos heredados de estas super clases (echar un vistazo a las API de JAVA 8).

Métodos que intervienen en la creación de ventanas (dentro de la clase ventana):

* Darle un ancho y un largo: *setSize(ancho, largo)*
* Establecer un título: *setTitle(“Título”)*

Métodos que intervienen en la creación de ventanas (en el MAIN):

* Darle VISIBILIDAD*: v1.setVisible(true)* (por defecto no son visibles)

**CIERRE DE VENTANAS**

Terminar la ejecución de la ventana: *setDefaultCloseOperation(EXIT\_ON\_CLOSE).*

**POSICIÓN INICIAL, REDIMENSIONAR Y COLOR.**

Al ejecutar, todas las ventanas aparecen en la esquina superior izquierda (son las coordenadas (0,0)). Vamos a especificar las nuevas coordenadas y así aparecerá en otro sitio.

* Establecer la posición inicial: *setLocation(110, 200)* (en pixeles)
* Hay otro método que engloba a setSize y setLocation, que es *setBounds(coord x, coord y, ancho, alto)*
* Centrar la ventana en todo tipo de pantallas: *setLocationRelativeTo(null)*
* Deshabilitar el redimensionado: *setResizable(false)*
* Establecer el tamaño mínimo (a la hora de redimensionar el usuario): *setMinimumSize (new Dimension(200,200));*
* Cambiar el color de la ventana: El color de la ventana se considera como PARTE DEL CONTENIDO de la ventana, por tanto, primero tengo que obtener el contenido de la ventana y luego cambiarle el color. *this.getContentPane().setBackground(Color.yellow);*

**PANELES (JPanel)**

A la hora de añadir más componentes a nuestra ventana tales como etiquetas, botones, etc… se han de añadir a un JPanel directamente, no a la ventana, por tanto, siempre habrá que añadir un JPanel a las ventanas para poder añadirle cosas. Es decir, la ventana es como si fuera nuestra mesa y el Jpanel como si yo pongo encima de la mesa un folio para pintar.

**CREACIÓN DE PANELES**

* Creación: *new Jpanel();*
* Añadimos el panel a la ventana*: this.getContentPane().add(panel);*
* Cambiamos el color del panel (para que se pueda ver): *panel.setBackground(Color.blue)*

Tanto los paneles como las ventanas son contenedores. Cada contenedor tiene un Layout, es decir, un tipo de diseño (linear, en tabla, etc…). Por defecto tiene un Layout que lo centra todo. Por tanto, si queremos usar las coordenadas, es decir, el setBounds de algún componente lo primero que tenemos que hacer es desactivar ese Layout con *panel.setLayout(null)*.

**ETIQUETAS (JLabel)**

**CREACIÓN DE ETIQUETAS**

Son Jlabel que me ayudan a poner título o escribir encima del Jpanel.

* Creación*: new JLabel(“Hola”);*
* Cambiar el texto a la etiqueta: *etiqueta.setText("Hola Mundo");*
* Agregamos la etiqueta al panel: *panel.add(etiqueta)*
* Cambiamos las coordenadas: Primero desactivamos el layout y luego *etiqueta.setBounds(10, 10, 50, 30);*
* Cambiar el color de la letra: *etiqueta.setForeground (Color.MAGENTA);*
* Cambiar el color de fondo de la etiqueta: Podemos usar el setBackground, pero antes hay que desactivar el diseño por defecto de las etiquetas porque en este método, el fondo de la etiqueta es transparente (porque así se ve todo mejor).

*etiqueta.setOpaque(true); // Cambiar color de fondo*

*etiqueta.setBackground(Color.yellow);*

**UBICACIÓN Y FUENTE DEL TEXTO.**

* Centrar el texto dentro de la etiqueta. Esto se tiene que hacer usando otro tipo de constructor que me especifique la alineación horizontal:

*JLabel etiqueta = new JLabel("Hola", SwingConstants.CENTER);*

* Poner el texto a la derecha dentro de la etiqueta: Lo mismo que antes pero con la derecha. Para diferenciarlo del anterior vamos a hacerlo con un método específico:

*etiqueta.setHorizontalAlignment(SwingConstants.CENTER);*

* Modificar la fuente del texto (consultar las API de Java):
  + Hay que crear un nuevo tipo de Fuente: new Font(nombre, estilo, tamaño)
  + Por ejemplo: *etiqueta.setFont(new Font("arial", Font.PLAIN, 30));*
  + Para ver todos los estilos de letra disponibles los podemos copiar simplemente de cualquier editor de texto, el word, el writer, etc…

**IMAGEN en etiqueta y TAMAÑO de la imagen.**

Aquí vamos a usar etiquetas de tipo imagen. Para ello el fichero de la imagen debe estar en la misma carpeta que el proyecto.

* Creación de la imagen: *JLabel imagen = new JLabel(new ImageIcon("alpes.jfif"));*
* Cambiar de tamaño a la imagen:

*etiqueta2.setIcon(new ImageIcon(imagen.getImage().getScaledInstance(200, 200, Image.SCALE\_SMOOTH)));*

* Ajustar el tamaño de la imagen a la etiqueta:

*etiqueta2.setIcon(new ImageIcon(imagen.getImage().getScaledInstance(etiqueta2.getWidth(), etiqueta2.getHeight(), Image.SCALE\_SMOOTH)));*

**BOTONES (JButton)**

**CREACIÓN DE BOTONES**

Son JButton que me ayudan a interactuar con el usuario.

* Creación*: Jbutton boton1 = new JButton(“Clica aquí”);*
* Cambiar el texto al botón: *boton1.setText("Hola Mundo");*
* Agregamos la etiqueta al panel: *panel.add(boton1)*
* Habilitar/Deshabilitar el botón: *boton1.setEnabled(true)*;
* Establecer la combinación Alt+letra para que funcione como un clic de ratón, pero desde teclado: *boton1.setMnemonic('a');*
* AJUSTES DE BOTONES CON TEXTO:
  + Cambiar color texto: *boton1.setForeground(Color.BLUE);*
  + Cambiar fuente texto: *boton1.setFont (new Font("arial", Font.BOLD, 30));*
  + Cambiar color de fondo: *boton1.setBackground(Color.BLUE);*  En algunas versiones de Java, para que este color se vea necesitaremos hacer un boton1.setOpaque(true)
* AJUSTES DE BOTONES CON IMAGEN:
  + Poner imagen: Hay una forma de poner la imagen que sería usando el constructor, pero de esta forma me pone la imagen con el tamaño real de la misma. Sin embargo, yo quiero que meta la imagen con el tamaño que tenga el botón. Para ello hacemos lo mismo que en las etiquetas:
    - *boton2.setIcon (new ImageIcon(nuclear.getImage().getScaledInstance(boton2.getWidth(), boton2.getHeight(), Image.SCALE\_SMOOTH)));*
* BORDES

Para cambiar el borde, necesitamos pasarle al método un objeto tipo Border, pero si nos vamos a las API de Java, nos dice que Border no es una clase sino una interfaz, por lo tanto, vamos a tener que fabricar un borde con BorderFactory. Usamos el método createLineBorder (color, ancho, ¿esquinas redondeadas?).

*boton3.setBorder(BorderFactory.createLineBorder(Color.RED, 3, false));*

**RADIO BUTTON (JRadioButton)**

**CREACIÓN DE BOTONES de opción**

* Creación (con título y si va a estar chekeado o no) *: JRadioButton radioBoton1 = new JRadioButton("Opcion 1", true);*
* Agregamos los botones al panel: *panel.add(radioBoton1);*
* Para que sólo pueda haber una opción marcada, tengo que meter todos los radioButton dentro de un mismo grupo: *ButtonGroup grupoRB1 = new ButtonGroup ();*
* Agregamos los botones al grupo: *panel.add(boton1)*
* Deshabilitar un radioButton: *radioBoton1.setEnabled(false);*
* Cambiar el texto con *setText* y la fuente como hemos hecho con los anteriores componentes

Asdf

asdf

**BOTONES DE ACTIVACIÓN (JToggleButton)**

**CREACIÓN DE BOTONES de activación**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

Asdf

**CAJAS DE TEXTO (JTextField)**

**CREACIÓN DE Cajas de Texto**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

**ÁREAS DE TEXTO (JTextArea)**

**CREACIÓN DE Áreas de Texto**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

**AÑADIR BARRAS DE DESPLAZAMIENTO (JscrollPane)**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

**CASILLAS DE VERIFICACIÓN (JCheckBox)**

**CREACIÓN DE CASILLAS DE VERIFICACIÓN**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

Asdf

**LISTAS DESPLEGABLES (JComboBox)**

**CREACIÓN DE BOTONES de activación**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

Asdf

**TABLAS (JTable)**

**CREACIÓN DE TABLAS**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

Asdf

**LISTAS (JList)**

**CREACIÓN DE BOTONES de activación**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

Asdf

**GESTIÓN DE EVENTOS O LISTENER**

**ActionListener (Eventos de acción)**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

Asdf

**MouseListener (Eventos de ratón)**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

asdf

**MouseMotionListener (Eventos de movimiento del ratón)**

DFGSDFG sldfj asljfa sdf

Sdf

**MouseWheelListener (Eventos de movimiento del ratón)**

DFGSDFG sldfj asljfa sdf

Sdf

**MouseEvent (Métodos de los eventos del ratón)**

DFGSDFG sldfj asljfa sdf

Sdf

**KeyListener (Eventos de teclado)**

DFGSDFG sldfj asljfa sdf

Sdf

**LAYOUTS**

**FLOWLAYOUT**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

Asdf

**BOXLAYOUT**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

asdf

**BORDERLAYOUT**

DFGSDFG sldfj asljfa sdf

Sdf

**GRIDLAYOUT**

DFGSDFG sldfj asljfa sdf

Sdf

**MENÚS**

**Creación de una barra de menú, submenús y componentes de la barra de menú.**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

Asdf

**Menús desplegables (JpopUp)**

DFGSDFG sldfj asljfa sdf

Sdf

Asdf

asdf

**CREACIÓN DE EJECUTABLES**

**CREACIÓN DE UN .JAR Y UN .EXE**

FGHDFGH

EJERCICIOS

1. Contabilizar clics de ratón
2. Escoger deporte favorito
3. Jugar con los colores RGB
4. Interfaz para un grupo de personas (CON EL DISEÑADOR GUI DE NETBEANS)
5. Creación de un cronómetro
6. Creación de un login
7. Crear una cartelera de cine
8. Hacer una calculadora y un conversor de divisas
9. **BASES DE DATOS**
10. **sfd**