Teoría definitiva de JAVA

Comentarios (// - /* - /**)

```
//Comentario de una sola línea

/*
* Comentarios de
* Varias líneas
*/

/**
* Comentarios usados
* para documentación
*/
```

Operadores

Operadores aritméticos

```
X+Y    || X-Y   || X*Y   || X/Y   || X%Y
X+=Y   || X-=Y || X*=Y || X/=Y || X%=Y
X++    || X--
```

Operadores relacionales

```
X==Y   || X!=Y || X<Y  || X>Y  || X<=Y  || X>=Y
```

Operadores lógicos

```
AND    &       &&
OR     |       ||
```

La diferencia entre los caracteres “&” y “|” con “&&” y “||” es que && y || realizan short-circuit evaluations, es decir, que si tenemos if (X && Y) y X no se cumple, no comprobará Y, mientras que if (X & Y) comprobaría ambos a pesar de que la primera condición no se cumpliese.

Convenio para nombrar variables

Identificador	Convención	Ejemplo
Nombre de variable	Comienza por letra minúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas	numAlumnos, suma cantidadTotal numeroAlumnosAprob
Nombre de constante	En letras mayúsculas, separando las palabras con el guión bajo. Por convenio el guión bajo no se utiliza en ningún otro sitio	TAM_MAX, PI
Nombre de una clase	Comienza por letra mayúscula	String, MiTipo
Nombre de función	Comienza por letra minúscula	modificaValor, obtieneValor

Datos primitivos

Tipo	Descripción
boolean	1 byte. Permite representar valores lógicos; true o false.
char	2 bytes. Unicode. Comprende el código ASCII.
byte	1 byte. Entero. Su rango de valores va desde -128 (-2^7) a +127 ($+2^7-1$).
short	2 bytes. Valor entero entre -32768 y 32767
int	4 bytes. Valor entero entre -2.147.483.648 y 2.147.483.647
long	8 bytes. Valor entero entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807
float	4 bytes. Número real. De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38
double	8 bytes. Número real. De -1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308

Conversiones

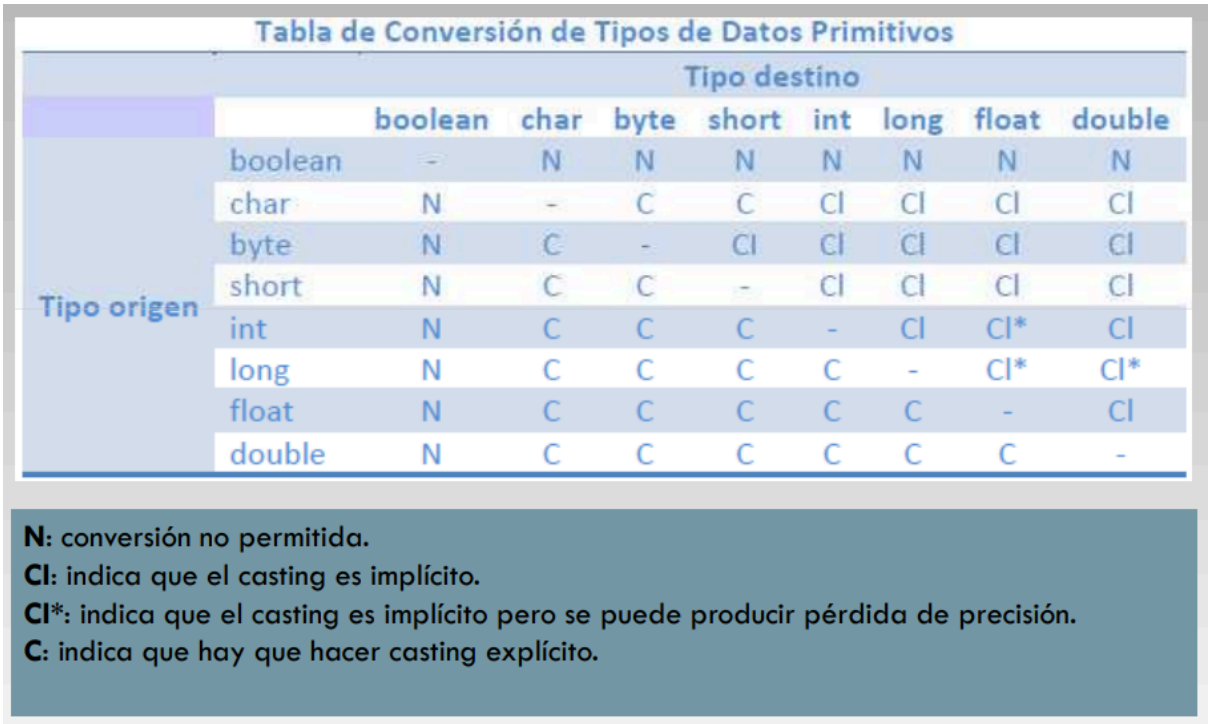


Tabla de Conversión de Tipos de Datos Primitivos

		Tipo destino						
		boolean	char	byte	short	int	long	float	double
Tipo origen	boolean	-	N	N	N	N	N	N	N
	char	N	-	C	C	CI	CI	CI	CI
	byte	N	C	-	CI	CI	CI	CI	CI
	short	N	C	C	-	CI	CI	CI	CI
	int	N	C	C	C	-	CI	CI*	CI
	long	N	C	C	C	C	-	CI*	CI*
	float	N	C	C	C	C	C	-	CI
	double	N	C	C	C	C	C	C	-

N: conversión no permitida.
CI: indica que el casting es implícito.
CI*: indica que el casting es implícito pero se puede producir pérdida de precisión.
C: indica que hay que hacer casting explícito.

Scanner

```
import java.util.Scanner;
Scanner nombreDelScaner = new Scanner(System.in);
nextByte();
nextShort();
nextInt();
nextLong();
nextFloat();
nextDouble();
nextLine();
nextBoolean();
```

Char

No se puede pedir un char con Scanner, pero se puede hacer un bucle que pida un string, si tiene un solo carácter acaba el bucle y se asigna la posición 0 del String al Char.

If

```
if(condición){

} else if(condición){

} else{

}
```

Switch-Case

Número entero

```
switch(variable){
case 0:

        break;
case 1:

        break;
default:

        break;
}
```

Cadena de texto

```
switch (variable) {
case "a":

        break;
case "b":

        break;
default:

        break;
}
```

Si no ponemos break, sigue ejecutando el código hasta que encuentra un break, por lo que podemos hacer que varios resultados hagan lo mismo.

While

```
while (variable) {

}
```

Do while

```
do {

} while (variable);
```

For

```
for (int i = 0; i < 10; i++) {

}
```

Conversión

Implícita

```
byte a=1, b=2;
int c=a+b;
```

Explícita

```
byte a=100, b=30;
double c=(double)a/b;
```

System.out.print (+ ln y f)

```
System.out.println(); // Escribe algo y hace un salto de línea (\n)
System.out.print();   // Escribe algo sin saltar de linea
System.out.printf("%2$d %1$f",c,a);        // Escribir con formato
```

%[posición_dato$][indicador_de_formato][ancho][.precisión]carácter_de_conversión
Los [] indican que es opcional.
posición_dato$: en qué posición está el dato(empieza por 1, no por 0).

indicador_de_formato: carácter que determina el formato de salida.

INDICADORES DE FORMATO			
Indicador	**Significado**	**Indicador**	**Significado**
-	Alineación a la izquierda	+	Mostrar signo + en números positivos
(	Los números negativos se muestran entre paréntesis	0	Rellenar con ceros
,	Muestra el separador decimal		

ancho: cantidad de caracteres mínima que ocupará el dato.
.precisión: cantidad de decimales que se mostraran.
carácter_de_conversión: carácter que indica el tipo de dato que se mostrará.

CARACTERES DE CONVERSIÓN			
Carácter	**Tipo**	**Carácter**	**Tipo**
d	Número entero en base decimal	X, x	Número entero en base hexadecimal
f	Número real con punto fijo	s	String
E, e	Número real notación científica	S	String en mayúsculas
g	Número real. Se representará con notación científica si el número es muy grande o muy pequeño	C, c	Carácter Unicode. C: en mayúsculas

Saltos de línea y tabulación

"\r" Carriage Return.
"\n" Line Feed.
"\r\n" Salto de línea CR + LF.
"\t" Tabulación.
"\s" Espacio en blanco.

StringBuilder

Constructor

Crear un StringBuilder vacío.
StringBuilder nombre = new StringBuilder();

Crear un StringBuilder partiendo de un CharSequence o un String.
StringBuilder nombre = new StringBuilder(CharSequence);
StringBuilder nombre = new StringBuilder(String);

append

Añade al StringBuilder el argumento que le pasemos, le podemos pasar: -boolean -char -char[] -CharSequence -double -float -int -long -object -String -StringBuffer
sb.append(X).

insert

Introduce lo que le pasemos en la posición indicada. Le podemos pasar String, char, boolean, números enteros, decimales…
sb.insert(4, 8.76); (Introduce en la posición 4 el texto 8.76)

charAt

Devuelve el carácter de la posición indicada. Empezando por 0.
sb.charAt(3)

delete

Borra del StringBuilder el contenido desde start hasta un carácter antes que end.
sb.delete(int start, int end)

deleteChar

Elimina el carácter de la posición indicada.
deleteCharAt(int index)

reverse

Devuelve el StringBuilder pero escrito al revés. (123 → 321)
sb.reverse()

length

Devuelve el tamaño (cantidad de caracteres) del StringBuilder.
sb.length()

indexOf

Da la posición de la primera vez que aparece “perro” en el StringBuilder.
sb.indexOf(“perro”);
Da la posición de la primera vez que aparece “perro” en el StringBuilder empezando por el carácter de la posición 8.
indexOf(“perro”, 8)

lastIndexOf

Da la posición de la última vez que aparece la palabra “gordo” en el StringBuilder.
sb.lastIndexOf(“gordo”);
Da la posición de la última vez que aparece la palabra “gordo” en el StringBuilder empezando desde el carácter en la posición 9.
sb.lastIndexOf(“gordo”, 9);

substring

Devuelve un String desde la posición indicada hasta el final.
substring(int start)
Devuelve un String desde la posición start hasta una posición antes de la indicada en end.
substring(int start, int end)

toString

Devuelve un String con el contenido del StringBuilder.
sb.toString;

Arrays

copyOf

Arrays.copyOf(vector, length a copiar)
Copia el vector hasta la distancia especificada (si ponemos más length de la que tiene el vector se rellenan con el valor por defecto (0 int, null integer p.e.)).
También existe copyOfRange(vector, int from, int to)

copyOfRange

Arrays.copyOfRange(array, 0, array.length);
(T[] original, int from, int to)

deepEquals

Returns true if the two specified arrays are deeply equal to one another.
(Object[] a1, Object[] a2)

fill

Arrays.fill(vector, valor, desde qué posición, hasta que posición -1)
Llena el vector con ese valor desde X hasta Y-1.

setAll

Arrays.setAll(vector, (i) -> (int)(Math.random~~~~~~))
Te rellena el vector con números aleatorios según lo que especifiques.
Arrays.setAll(vector, (i) -> i*i)
Rellena el vector con el cuadrado de cada posición.

equals

Arrays.equals(vector1, vector2)
devuelve true si son idénticos (mismo tipo (int p.e.), misma length, mismos valores).

sort

Arrays.sort(vector) | Arrays.sort(vector, desde qué posición, hasta que posición)
Ordena el contenido de menor a mayor.
Arrays.sort(vector, Collections.reverseOrder());
Ordena de mayor a menor (tiene que ser integer o String).
Si lo hacemos con String lo ordena alfabéticamente.

toString

Arrays.toString(vector)
Muestra un vector

String

length

frase.length();
Da la cantidad de caracteres del string frase.

charAt

frase.charAt(3);
Devuelve el carácter en la posición 3 de frase (empieza en 0).

substring

frase.substring(7, 10);
Devuelve lo que esté escrito desde el carácter 7 hasta el 9.

indexOf

frase.indexOf(“perro”);
Da la posición de la primera vez que aparece “perro”.

lastIndexOf

frase.lastIndexOf(“gordo”);
Da la posición de la última vez que aparece la palabra “gordo”.

endsWith

frase.endsWith(“dos”);
Devuelve true si la frase acaba en “dos”.

startsWith

frase.startsWith(“mec”);
Devuelve true si la frase empieza por “mec”.

replace

frase.replace(‘a’,’e’);
No reemplaza las a por e, pero si lo usamos en un syso muestra la frase cambiando las a por e.

replaceAll

frase.replaceAll(“pepe”,“pepa”);
No reemplaza, solo sustituye un momento. Se pueden usar expresiones regulares. Podemos buscar mediante una expresión regular.

toUpperCase

frase.toUpperCase();
Muestra la frase en mayúsculas pero no la cambia.

toLowerCase

frase.toLowerCase();
Muestra la frase en minúsculas pero no la cambia.

toCharArray

frase.toCharArray();
Devuelve un array de caracteres formado por todos los caracteres de la frase.

split

frase.split(“mortadela”);
Devuelve un array de Strings separando los String de la frase cada vez que aparece mortadela.

contains

frase.contains(“perro”);
Devuelve true si en alguna parte de frase aparece la cadena especificada “perro”.

concat

frase.concat(“aaa”).concat(“fff”) = frase+"aaa”+”fff”

equals

frase.equals(“chistorra”)
devuelve true si el contenido de frase es el especificado “chistorra”.

format

String.format(“”,)
Literalmente printf pero en vez de escribirlo lo convierte en frase.

isEmpty

frase.isEmpty();
Devuelve true si la frase tiene longitud 0.

trim

frase.trim();
muestra la frase sin poner los espacios al inicio y al final de una frase (/t también lo quita).

valueOf

String.valueOf(valor);
Muestra valor como string. (Sería similar a un toString)

matches

String frase=”patata”;
if(frase.matches(“patata”)) TRUE
Podemos poner matches.(regex); Usando expresiones regulares le sacamos más partido.
{8} = MinYMax 8 {8,} Min 8 Max *

```
System.out.println(patata.matches("^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@#$%^&+=]).{8,}$"));
System.out.println(patata.matches("^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#&()–[{}]:;',?/*~$^+=<>]).{8,20}$"));
```

Math

abs

devuelve el valor absoluto del número que pasemos.
Math.abs(X)

max/min

Devuelve el número más grande/pequeño de dos números.
Math.max(X, Y) Math.min(X, Y)

pow

Math.pow(X, Y)

X = num Y = potencia, devuelve X^Y

random

(int) (Math.random() * (MAX - MIN + 1) + MIN)
Número random entre el MAX y MIN indicado.

round

Math.round(X)
Redondea un double o float al número entero que corresponda.

signum

Math.signum(X)
Devuelve 1 si es +, 0 si es 0 y -1 si es -.

sqrt

Math.sqrt(X)
Raíz cuadrada de X

substractExact

Math.subtractExact(X, Y)
Devuelve el valor entre X-Y, solo acepta int o long.

LocalDate

Para crear un objeto ponemos:

LocalDate nombre = [LocalDate.now()];

now

Devuelve la fecha actual.

of

Creamos una fecha específica.
LocalDate.of(yyyy, mm, dd);

getXXX

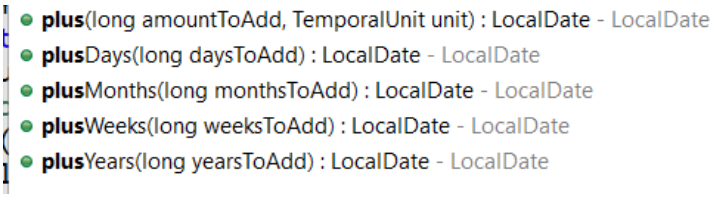
Para obtener alguno de los datos de la fecha ponemos la fecha.getMonth, getDayOfWeek…
LocalDate.getXXX()

isBefore isAfter

Devuelve true si la fecha está entes/después de la otra fecha. fecha.isBefore(otraFecha)
fecha.isAfter(otraFecha)

plus minus

LocalDate.plusXXX()
LocalDate.minusXXX()



until

fecha.until(otraFecha, ChronoUnit.UNIDAD)
System.out.println("He vivido "+fecha1.until(fecha).toString().replace("P", "").replace("D", " Días").replace("M", " Meses ").replace("Y", " Años "));
Period p = fecha1.until(fecha);
System.out.println("He vivido "+p.getDays()+" días "+"estos meses "+p.getMonths()+" y estos años "+p.getYears());

isLeepYear

Devuelve true si es bisiesto. fecha.isLeepYear();

LocalTime

Para crear un objeto ponemos:
LocalTime nombre = [LocalTime.now()];

now

Devuelve la hora actual hasta milisegundos.

truncatedTo

Devuelve una hora truncada a la unidad indicada.
hora.truncated(ChronoUnit.UNIDAD); (Ctrl+space a la hora de poner unidad y ves las opciones)

whitNano

Muestra la hora y le pone los nano segundos indicados, si ponemos 0 no muestra.
hora.whitNano(0)

isBefore isAfter

Devuelve true si la hora está entes/después de la otra hora. hora.isBefore(otraHora) hora.isAfter(otraHora)

LocalDateTime

now

Devuelve la fecha+hora actual.

Static

Un atributo es static cuando es igual para toda la clase.
Un método es static cuando está asociado a una clase pero no a instancias específicas de un objeto. Podemos usarlo llamando a la clase, no necesitamos de un objeto para usarlo.

Extends

Para que una clase herede de otra ponemos
public class patata extends clasePadre {

}

instanceof

objeto instanceof Clase
Devuelve true si el objeto es de la clase indicada.

Implements

```
public class patata implements interfaz{

}
```

clase abstracta

```
public abstract class patata {
	public abstract void metodo();
}
```

Captura de errores

Capturar error

```
try{
  cosas a intentar
  Imaginemos que aquí salta error
  no se llega a esta parte porque ha saltado error y va directo a la captura
} catch (patataException e) {
  que se hace cuando salte la excepción patataException
} finally {
  que se hace siempre, independientemente de si salta o no excepción
}
```

Repetir hasta que acabe el error

```
boolean error = true;
do {
	try {
		lo que intentamos
		error = false;
	} catch (patataException e) {
		System.out.println("ERROR");
	} finally {
		System.out.println("Pasamos por el try :D");
		}
}while(error);
```

Crear excepcion

Creamos una clase llamada loQueSeaException.
Hacemos que herede de Exception.

Hacemos un constructor en el que no pidamos nada y que envíe a la clase padre un String (el mensaje de error).
Después ya gestionamos cuando o donde se puede generar.
Si creamos un método que pueda generar un error, tenemos que escribir nombreMetodo() throws patataException.
Luego ya hacemos un if o lo que sea para lanzar el error, cuando queramos que se lance ponemos: throw new patataExcepcion().

Colecciones :D

Hash, equals y comparable

Set

No permite valores duplicados, cada elemento es único. El orden depende del tipo de Set.

HashSet

No puede haber duplicados, no importa el orden.

Constructor

HashSet()
HashSet(Collection<? extends E> c)

add

Devuelve boolean, patata.add(X)

clear

Se carga todo lo que haya en el set. patata.clear()

contains

devuelve true si contiene X. patata.contains(X)

isEmpty

Devuelve true si está vacío. patata.isEmpty()

remove

Devuelve true si se elimina dicho elemento del set. patata.remove(X)

size

Devuelve un int con el tamaño del set. patata.size()

LinkedHashSet

No puede haber duplicados, da igual el orden (por inserción).

Constructor

LinkedHashSet()

Métodos

Hereda los de HashSet.

TreeSet

No puede haber duplicados, se ordena de forma natural.

Constructor

TreeSet()
TreeSet(Collection<? extends E> c)
TreeSet(Comparator<? super E> comparator)

add

Devuelve boolean, patata.add(X)

clear

Se carga todo lo que haya en el set. patata.clear()

contains

devuelve true si contiene X. patata.contains(X)

first

Returns the first (lowest) element currently in this set. No funciona si el padre es Set. patata.first()

lower

Devuelve el primer valor inferior a X. Devuelve null si no hay. patata.floor(X)

higher

Devuelve el primer valor superior a X. Devuelve null si no hay. patata.higher(X)

isEmpty

Devuelve true si está vacío. patata.isEmpty()

last

Devuelve el último valor del Set. Excepción si vacio. patata.last()

pollFirst pollLast

Elimina el primer/último valor del Set. Si lo metemos en un syso dice que ha eliminado (null si no quita nada). patata.pollFirst()/patata.pollLast()

remove

Devuelve true si se elimina dicho elemento del set. patata.remove(X)

size

Devuelve un int con el tamaño del set. patata.size()

List

Permite valores duplicados, se puede acceder a los elementos por un índice.

ArrayList

Constructor

ArrayList()
ArrayList(Collection<? extends E> c)

add

Devuelve boolean, patata.add(X)

clear

Se carga todo lo que haya en el set. patata.clear()

contains

devuelve true si contiene X. patata.contains(X)

forEach

patata.forEach(x -> System.out.println(x));

get

patata.get(indice)

indexOf

patata.indexOf(objeto)

isEmpty

Devuelve true si está vacío. patata.isEmpty()

lastIndexOf

Devuelve la última ocurrencia de X, si no hay devuelve -1. patata.lastIndexOf(X)
Elimina el objeto guardado en la posición indicada. patata.remove(índice)

remove

Elimina el valor almacenado en la posición indicada. patata.remove(índice)

removeAll

Elimina todos los elementos de patata que aparezcan en patata2. patata.removeAll(patata2)

removeIf

Elimina los elementos de patata que coincidan con X>3. patata.removeIf(X -> X>3);

replaceAll

Reemplaza cada elemento por la condición indicada. patata.replaceAll(x -> x*x);

set

Reemplaza el valor de la posición indicada por X. patata.set(indice, X)

size

Devuelve un int con el tamaño del set. patata.size()

sort

patata.sort((n1, n2)->-n1.compareTo(n2));

toArray

numeros2 = perro.toArray(numeros2);

Map

Almacena pares de datos Clave-Valor donde cada clave es única.

HashMap

Constructor

HashMap()
HashMap(Map<? extends K,? extends V> m)

clear

Vacía madalenas. madalenas.clear();

containsKey

Devuelve true si el mapa contiene la clave indicada. madalenas.containsKey(key)

containsValue

Devuelve true si el mapa contiene el valor indicado. madalena.containsValue(value)

entrySet

Devuelve un Map.Entry del mapa indicado. madalenas.entrySet()

get

Devuelve el valor asociado a dicha clave. madalenas.get(key)

getOrDefault

Devuelve el valor asociado a la clave indicada, si no existe esa clave, se devuelve X. mandarinas.getOrDefault(key, X);

isEmpty

Devuelve true mientras el mapa esté vacío. madalenas.isEmpy()

keySet

Devuelve un Set con todas las claves del mapa indicado. madalenas.keySet()

put

Añade/reemplaza una clave y su valor. madalenas.put(key, value)

putAll

Añade al primer mapa los valores del segundo mapa. madalenas.putAll(mandarinas)

putIfAbsent

Añade al mapa la clave y el valor si no existen en el mapa (o si está vacío, que viene a ser lo mismo). madalenas.putIfAbsent(key, value);

remove

Elimina el elemento del mapa si coinciden los valores pasados.
madalenas.remove(key);
madalenas.remove(key, value);

replace

Reemplaza (si existe) en el mapa los valores indicados.
madalenas.replace(key, newValue);
madalenas.replace(key, oldValue, newValue);

size

Devuelve cuantos elementos hay en el mapa. madalenas.size()

values

Devuelve un set con todos los valores del mapa. madalenas.values()

LinkedHashMap

Constructor

LinkedHashMap()
LinkedHashMap(Map<? extends K,? extends V> m)

clear

Vacía madalenas. madalenas.clear();

containsKey

Devuelve true si el mapa contiene la clave indicada. madalenas.containsKey(key)

containsValue

Devuelve true si el mapa contiene el valor indicado. madalena.containsValue(value)

entrySet

Devuelve un Map.Entry del mapa indicado. madalenas.entrySet()

get

Devuelve el valor asociado a dicha clave. madalenas.get(key)

getOrDefault

Devuelve el valor asociado a la clave indicada, si no existe esa clave, se devuelve X. mandarinas.getOrDefault(key, X);

isEmpty

Devuelve true mientras el mapa esté vacío. madalenas.isEmpy()

keySet

Devuelve un Set con todas las claves del mapa indicado. madalenas.keySet()

put

Añade/reemplaza una clave y su valor. madalenas.put(key, value)

putAll

Añade al primer mapa los valores del segundo mapa. madalenas.putAll(mandarinas)

putIfAbsent

Añade al mapa la clave y el valor si no existen en el mapa (o si está vacío, que viene a ser lo mismo). madalenas.putIfAbsent(key, value);

remove

Elimina el elemento del mapa si coinciden los valores pasados.
madalenas.remove(key);
madalenas.remove(key, value);

replace

Reemplaza (si existe) en el mapa los valores indicados.
madalenas.replace(key, newValue);
madalenas.replace(key, oldValue, newValue);

size

Devuelve cuantos elementos hay en el mapa. madalenas.size()

values

Devuelve un set con todos los valores del mapa. madalenas.values()

TreeMap

Constructor

TreeMap()
TreeMap(Comparator<? super K> comparator)
TreeMap(Map<? extends K,? extends V> m)

clear

Vacía madalenas. madalenas.clear();

comparator

Devuelve null o el comparador que le hayamos pasado para ordenarlo. madalenas.comparator()

containsKey

Devuelve true si el mapa contiene la clave indicada. madalenas.containsKey(key)

containsValue

Devuelve true si el mapa contiene el valor indicado. madalena.containsValue(value)

descendingMap

Devuelve el mapa ordenado del revés. madalenas.descendingMap()

entrySet

Devuelve un Map.Entry del mapa indicado. madalenas.entrySet()

firstKey

Devuelve la primera clave del mapa. madalenas.firstKey()

firstEntry

Devuelve la primera pareja clave-valor del mapa. madalenas.firstEntry()

floorKey

Devuelve el valor más alto que sea igual o esté por debajo de la clave indicada. madalenas.floorKey(key)

floorEntry

Devuelve la pareja clave-valor que sea igual o esté por debajo de la clave indicada. madalenas.floorEntry(key)

get

Devuelve el valor asociado a dicha clave. madalenas.get(key)

higherKey / lowerKey

Devuelve la clave que sea mayor/menor a la clave indicada. madalena.higherKey(key)

higherEntry / lowerEntry

Devuelve la pareja clave-valor cuya clave sea mayor/menor a la clave indicada. madalena.higherEntry(key)

keySet

Devuelve un Set con todas las claves del mapa indicado. madalenas.keySet()

lastkey

Devuelve la última clave del mapa. madalenas.lastKey()

lastEntry

Devuelve la última pareja clave-valor del mapa. madalenas.lastEntry()

pollFirstEntry pollLastEntry

Elimina el primer/último valor del Set. Si lo metemos en un syso dice que ha eliminado (null si no quita nada).madalenas.pollFirstEntry()/madalenas.pollLastEntry()

put

Añade/reemplaza una clave y su valor. madalenas.put(key, value)

putAll

Añade al primer mapa los valores del segundo mapa. madalenas.putAll(mandarinas)

remove

Elimina el elemento del mapa si coinciden los valores pasados.
madalenas.remove(key);
madalenas.remove(key, value);

replace

Reemplaza (si existe) en el mapa los valores indicados.
madalenas.replace(key, newValue);
madalenas.replace(key, oldValue, newValue);

size

Devuelve cuantos elementos hay en el mapa. madalenas.size()

values

Devuelve un set con todos los valores del mapa. madalenas.values()

File

Constructor

File f = new File(“ruta”);
Creamos un File, si ponemos una ruta relativa usará la ruta del proyecto de eclipse. En estos caso crearíamos patata.txt y Patata.
File f1 = new File("patata.txt");
File f2 = new File("Patata");

createNewFile mkdir mkdirs

Estos File puede que no existan en nuestro equipo, por lo que hay que crearlos. Si intentamos crear un archivo o un directorio que ya existan, simplemente no hará nada y devolverá false.
Para crear ficheros hay que usar try/catch.
f1.createNewFile();
Para crear directorios.
f2.mkdir();
Para crear un directorio y sus directorios padre (si no existen) podemos usar mkdirs.
File f2 = new File("puerro/Chimichanga/Patata");
f2.mkdirs();
Esto es lo mismo que mkdir pero también crea los padres, es decir, que si ponemos al final de la ruta patata.txt, lo que creará es un directorio llamado patata.txt. Si intentamos hacer después createNewFile nos dará error.

canExecute canRead canWrite

Para comprobar si se puede ejecutar, leer o escribir un File usamos:
f1.canExecute();
f1.canRead();
f1.canWrite();
Devuelve un booleano, true si se puede hacer, false si no.

delete

Eliminamos el archivo o directorio indicado. Devuelve true si lo borra y false si no lo borra.
f1.delete();

deleteOnExit

Elimina el archivo o directorio cuando se acabe la ejecución del programa.
f2.deleteOnExit();

exists

Devuelve true si el file existe y false si no existe. Comprueba que la ruta con la que creamos el File existe.
f1.exists();

getAbsolutePath getAbsoluteFile

Ambos métodos nos sirven para saber la ruta de un File, la diferencia es que getAbsolutePath nos devuelve un String y getAbsoluteFile nos devuelve un File.
f1.getAbsolutePath();
f1.getAbsoluteFile(); → File f3 = f1.getAbsoluteFile();

getfreeSpace getUsableSpace

Nos devuelve la cantidad de bytes disponibles en la partición en la que se encuentra el File indicado.
Nos devuelve los bytes libres de la partición en la que se encuentra f1.
f1.getFreeSpace();
Nos devuelve los GigaBytes libres de la partición en la que se encuentra f1 (bytes/1024 = KB, KB / 1024 = MB y MB / 1024 = GB).
f1.getFreeSpace()/1024/1024/1024;
getUsableSpace es básicamente lo mismo pero nos devuelve el espacio disponible que puede usar la máquina virtual de java.

getTotalSpace

Nos devuelve la cantidad de bytes totales de la partición en la que se encuentra el File indicado.
Nos devuelve los bytes totales de la partición en la que se encuentra f1.
f1.getTotalSpace();
Nos devuelve los GigaBytes totales de la partición en la que se encuentra f1 (bytes/1024 = KB, KB / 1024 = MB y MB / 1024 = GB).
f1.getTotalSpace()/1024/1024/1024;

getName

Devuelve un String con el nombre del archivo o directorio.
f1.getName();

getParent

Devuelve un String con el nombre del directorio padre.
f1.getParent();

getParentFile

Devuelve un File del directorio padre.
File f3 = f1.getParentFile();

getPath

Nos devuelve la ruta relativa (tomando como directorio actual el directorio del proyecto).
f1.getPath();

isDirectory isFile isHidden

Devuelve true si es un directorio.
f2.isDirectory();
Devuelve true si es un archivo.
f2.isFile();
Devuelve true si el File está oculto.
f1.isHidden();

isAbsolute

Devuelve true si la ruta usada en el File es absoluta o relativa.
f1.isAbsolute();
Ruta relativa:
File f1 = new File("Patata/patata.txt");
Ruta absoluta:
File f1 = new
File("C:\\Users\\Marko\\Documents\\Programación\\TeoriaT8\\Patata\\patata.txt");

lastModified

Nos devuelve un long con la última vez que hemos modificado el File. Si nos dan 1713523705711 no sabemos que es, por lo que podemos usar la clase Date para interpretarlo.
Date f = new Date(f1.lastModified());
System.out.println(f);

length

Nos devuelve el tamaño en bytes del File. Lo cual nos sirve para saber si un archivo está vacío o no, ya que en el momento que tiene contenido un archivo, su tamaño deja de ser 0.
f1.length();

list

Devuelve un array de String con el nombre de los archivos/directorios que estén dentro del File indicado.

```
for(String s : f4.list()) {
	System.out.println(s);
}
```

listFile

Devuelve un array de File con todos los archivos/directorios que estén dentro del File indicado.

```
for(File f : f4.listFiles()) {
	System.out.println(f);
}
```

listRoots

Método estático que muestra los directorios raíz del SO.

```
for(File f : File.listRoots()) {
	System.out.println(f);
}
```

renameTo

Cambia el nombre del fichero por el fichero que le pasemos.

```
f1.renameTo(new File("Patata/patata.txt4"));
```

FileInputStream

Todos los métodos de FileInputStream lanzan excepciones, así que hay que usar throws o try/catch.

Constructor

Para crearlo, podemos pasarles un String o un File.

```
FileInputStream fIn = new FileInputStream("patata.txt");
FileInputStream fIn = new FileInputStream(new File("patata.txt"))
```

available

Indica una estimación de cuántos bytes quedan disponibles por leer.

```
fIn.available();
```

read

Lee un byte, por lo que en el caso de que haya datos, nos devuelve un byte, si no hay más datos, devuelve -1.
b = fIn.read();

skip

Se salta el número de bytes indicados. Hay que tener en cuenta que los saltos de línea en windows son CRLF (\r\n), es decir, un salto de línea son 2 bytes.
fIn.skip(10);

close

Cierra el FileInputStream, es decir, libera los recursos que estuviesen asociados a él y si intentamos volver a usarlo tras cerrarlo saltará una excepción.

FileOutputStream

Todos los métodos de FileOutputStream lanzan excepciones, así que hay que usar throws o try/catch.
Al instanciar el fichero, en caso de no existir, se crea.

Constructor

Podemos crearlos usando un String o un File. De normal, se sobreescribe el contenido del fichero que vamos a escribir. Para indicar si se machaca o no el contenido, podemos añadir un booleano al constructor. true hace que se conserve el contenido y false (por defecto) hace que se machaque.
FileOutputStream fOut = new FileOutputStream("copia.txt");
FileOutputStream fOut = new FileOutputStream(new File("copia.txt"))
FileOutputStream fOut = new FileOutputStream(String name, boolean append)
FileOutputStream fOut = new FileOutputStream(File file, boolean append)

write

Escribe un byte en el fichero.
fOut.write(b);

close

Cierra el FileOutputStream, es decir, libera los recursos que estuviesen asociados a él y si intentamos volver a usarlo tras cerrarlo saltará una excepción.

Copiar ficheros

int b = 0;

```
try {
	while((b = fIn.read()) != -1) {
		fOut.write(b);
	}
} catch (IOException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

DataInputStream

Constructor

Lo creamos con un FileInputStream.
dIn = new DataInputStream(new FileInputStream(""));
dIn = new DataInputStream(new FileInputStream(new File("")));

read

Lee un byte.
dIn.read();

readXXX

Para leer tipos de datos primitivos tenemos varios métodos, todos son read+tipo_dato, por ejemplo: readBoolean(), readDouble()…

readAll

Guardamos todos los bytes del fichero indicado en un array.
byte[] array = dIn.readAllBytes();

readUTF

Lee un String, primero lee dos bytes que indican cuántos bytes ocupa la frase y posteriormente lee dichos bytes.

skipBytes

Salta la cantidad de bytes indicados. Nos puede servir para no leer ciertos datos.
dIn.skip(4);

DataOutputStream

Constructor

Lo creamos con un FileOutputStream.
dOut = new DataOutputStream(new FileOutputStream("",true/false));
dOut = new DataOutputStream(new FileOutputStream(new File("")),true/false));

flush

Vacía el contenido del buffer en el fichero.
dOut.flush();

size

Devuelve la cantidad de bytes escritos, si el fichero ya tiene datos pero no hemos escrito nada devolverá 0.

write

Escribe un byte o varios bytes si le pasamos un array de bytes.
dOut.write(byte);
dOut.write(byte[]);
Si pasamos un array de bytes podemos indicar desde qué byte hasta que byte queremos que copie.
dOut.write(array,0,4);

writeXXX

Para escribir tipos de datos primitivos tenemos varios métodos, todos son write+tipo_dato, por ejemplo: writeBoolean(), writeDouble()…

writeBytes

Escribe un byte por cada uno de los caracteres escritos en el String que le pasamos. Lo que escribe es el valor ascii de cada carácter, por lo que si alguno de estos caracteres supera el valor ascii 255, escribirá otro valor. Cada 256 números se reinicia el número, por lo que si escribimos el carácter con valor 256, nos escribiría 0.
dOut.writeBytes("Hola caracola");
Si interpretamos lo anterior como byte nos devuelve:
72 111 108 97 32 99 97
Esto nos puede servir para escribir en un bloc de notas, ya que interpreta el ascii. Pero hay que tener cuidado, ya que si queremos escribir algún caracter como ඞ, no está entre los 256 primeros, en este caso nos escribiría ž.

writeChars

Escribe cada carácter de un String usando 2 bytes para cada carácter, es como usar writeChar(), pero en vez de tener que hacerlo por cada uno de los caracteres directamente lo hace con todos. Es decir, el número de bytes que vamos a escribir es el doble de la longitud del String.
dOut.writeChars("There is a impostor among us ඞ");

writeUTF

Escribe el String que pasamos, escribiendo primero dos bytes que van a indicar la cantidad de bytes que se van a escribir con la frase.
dOut.writeUTF("Feliz navidad");

Explicación

Usamos Los DataInput/OutputStream para escribir tipos de datos exactos. Cada tipo de dato usa una cantidad de bytes, por lo que tenemos que saber que datos hay almacenados. Si solo hay int (4 bytes) no habrá problema, siempre leeremos int. Pero si queremos guardar/leer un fichero que contiene distintos tipos de datos tendremos que saber cómo están escritos.
Si hay primero un int, luego un boolean y por último un String(utf), tendremos que leerlos siempre en ese orden. Si los leemos en otro orden estaríamos interpretando mal los datos, es como si intentamos leer un texto que está en inglés interpretándolo como si estuviese en ruso.

BufferedReader

Constructor

BufferedReader bR = new BufferedReader(new FileReader("datos.txt"));

close

Cierra el BufferedReader, es decir, libera los recursos que estuviesen asociados a él y si intentamos volver a usarlo tras cerrarlo saltará una excepción.
bR.close();

lines

Nos devuelve un flujo (Stream) de datos con las líneas del fichero.
bR.lines().forEach(System.out::println);

read

Devuelve un int con el carácter actual. Cómo nos devuelve un carácter, tenemos que castearlo a char si queremos que nos lo muestre como tal.
System.out.print((char)bR.read());

readLine

Devuelve un String con la línea actual del fichero.
bR.readLine()

ready

Devuelve true si se puede leer el fichero, cuando
bR.ready();

skip

Salta el número de caracteres indicados y no los lee. Los saltos de línea son \r\n, es decir, un salto de línea son 2 caracteres.
bR.skip(4);

BufferedWriter

Constructor

BufferedWriter bW = new BufferedWriter(new FileWriter("datos.txt"));
BufferedWriter bW = new BufferedWriter(new FileWriter("datos.txt",true));

close

Cierra el BufferedWriter, es decir, libera los recursos que estuviesen asociados a él y si intentamos volver a usarlo tras cerrarlo saltará una excepción.
bW.close();

flush

Vacía el contenido del buffer en el fichero.
bW.flush();

newLine

Inserta una nueva línea en el fichero.
bW.newLine();

write

Nos permite añadir texto. Podemos pasar directamente el String a escribir o pasar el String, desde que caracter queremos que escriba y cuantos caracteres queremos que escriba.
bW.write("HOLA:)");
bW.write("HOLA:)",1,3); (Esto escribiría “OLA”)

ObjectInputStream y ObjectOutputStream

Son clases que nos permiten almacenar objetos en ficheros mediante la serialización.

Constructor

ObjectInputStream oIn = new ObjectInputStream(new FileInputStream(f));
oOut = new ObjectOutputStream(new FileOutputStream(f));
oOut = new MyObjectOutputStream(new FileOutputStream(f, true));

Objeto

El objeto de dicha clase tiene que tener dos cosas:
-Implementar la interfaz Serializable.
public class Patata implements Serializable {
-Tener una constante para indicar cómo realiza la serialización:
private static final long serialVersionUID = 1L;

MyObjectOutputStream

ObjectOutputStream primero escribe un encabezado en el documento para indicar cómo se van a guardar los datos. Si volvemos a ejecutar el programa para guardar más datos escribiremos un segundo encabezado, lo cual nos dará problemas.
Para evitar que esto pase, creamos nuestra propia clase ObjectOutputStream en la cual vamos a hacer que el encabezado esté en blanco.

```
public class MyObjectOutputStream extends ObjectOutputStream {

	public MyObjectOutputStream(OutputStream out) throws IOException  {
		super(out);
	}

	@Override
	protected void writeStreamHeader() throws IOException{
		reset();
	}

}
```

Gestionar encabezados

Para saber si tenemos un encabezado o no vamos a crear un File.
File f = new File("X");
Declaramos un ObjectOutputStream.
ObjectOutputStream oOut = null;
Comprobamos si el File tiene contenido o no.
if (f.length() == 0)
En caso de que la longitud sea 0, es decir, que tenga 0 bytes, significa que no contiene datos, por lo que tendremos que añadir un encabezado, en caso contrario, ya habrá datos y tendremos que usar nuestra clase MyObjectOutputStream.

```
if (f.length() == 0) {
	try {
		oOut = new ObjectOutputStream(new FileOutputStream(f));
	} catch (FileNotFoundException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
} else {
	try {
		oOut = new MyObjectOutputStream(new FileOutputStream(f, true));
	} catch (FileNotFoundException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}
```

Guardar Objetos

Cogemos el objeto, suponemos que es un Coche. Lo sacamos de un Arraylist, lo creamos en ese momento, lo que sea.
Y para guardarlo simplemente ponemos:

```
try {
	oOut.writeObject(c);
} catch (IOException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

Leer Objetos

Creamos un objeto a leer, supongamos que es Coche.
Coche c;
Para leer un coche usamos:
c = (Coche) oIn.readObject();
Una vez leído ya hacemos lo que queramos, como guardarlo en un ArrayList.

Si queremos leer todos los objetos de un fichero tenemos que tener un booleano para comprobar que haya datos en el fichero, una vez hayamos leído todo el fichero saltará la excepción EOFException.

```
boolean hayDatos = false;
try (ObjectInputStream oIn = new ObjectInputStream(new FileInputStream(f));) {
        if (f.length() != 0) {
                System.out.println("DATOS");
                hayDatos = true;
        }
        while (hayDatos) {
                c = (Coche) oIn.readObject();
                System.out.println("COCHE " + index + "\r\n" + c);
                index++;
        }
} catch (EOFException e) {
//      hayDatos = false;
```

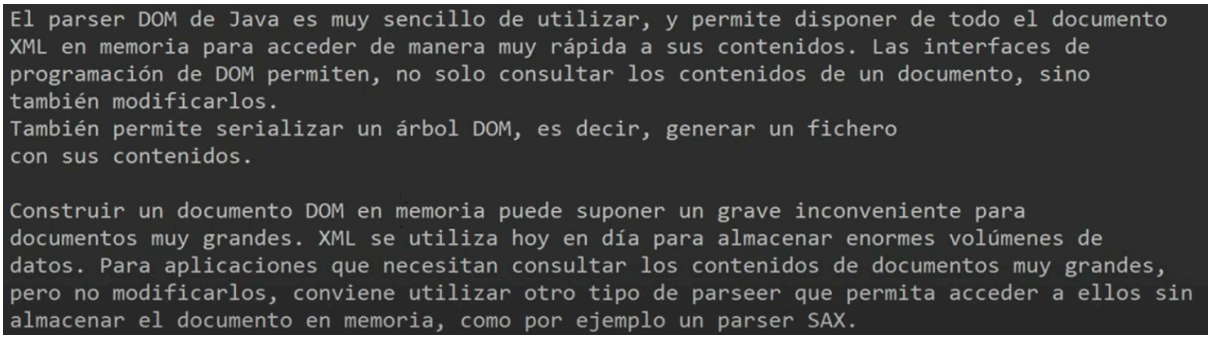
```
public class EOFException extends IOException {
```

Tenemos que tener en cuenta que la excepción EOFException hereda de IOException, por lo que si atrapamos primero IOException no atrapara EOFException, ya que esta excepción es una Excepción del tipo IOException.

Clases para leer/escribir secuencial

Clase	Info
FileInputStream FileOutputStream	Leer/escribir byte a byte, al leer devuelve -1 si ha acabado
DataInputStream DataOutputStream	Leer/escribir datos primitivos, al leer byte a byte devuelve -1 si se ha acabado, si leemos un tipo concreto salta EOFException
BufferedReader BufferedWriter	Leer/escribir texto, al leer carácter a carácter devuelve -1 y al leer línea a línea devuelve null si se ha acabado
ObjectInputStream ObjectOutputStream MyObjectOutputStream	Leer/escribir objetos, al leer objetos lanza la excepción EOFException si se ha acabado

DOM (xml)



Tratamiento de XML en JAVA | Leer un XML con DOM

Crear documento DOM

Para crear el documento DOM, usado para interpretar el XML, podemos hacerlo todo en una línea:
doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse("documento.xml");
O por partes:
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
doc = db.parse("documento.xml");

Normalizar el documento

Para normalizar el documento usamos el siguiente método:
doc.getDocumentElement().normalize();
Normalizar es eliminar los espacios en blanco, como tabulaciones o espacios.

getElementsByTagName

Este método nos devuelve un NodeList (como si fuese un ArrayList que contiene nodos) con los nodos que tengan como nombre el indicado.
doc.getElementsByTagName("nombre")

item

Se puede aplicar a un NodeList o a un Element.
Lo que nos permite es obtener el nodo indicado.
dias.item(i); Nos devuelve el elemento que pasemos como parámetro, si ponemos 0, nos pasará el primer nodo que haya dentro del Element o NodeList sobre el que apliquemos el método.
doc.getElementsByTagName("nombre").item(0);

getTextContent

Si lo aplicamos sobre un objeto Node cuyo contenido sea texto, nos devolverá un String con el contenido de dicho nodo.
doc.getElementsByTagName("nombre").item(0).getTextContent();

getNodeType

Aplicado sobre un Node nos devuelve cuál es el tipo de nodo.

Element

Crear un Element es necesario para poder obtener de un nodo sus atributos o un NodeList con sus nodos.
Para crearlo, tenemos que castear un Node a Element.
Element dia = (Element) nodo;

getAttribute

Nos permite obtener el atributo indicado de un Element.
dia.getAttribute("fecha");

Recorrer un NodeList

Para poder recorrer un NodeList no podemos usar for each ya que no implementa la interfaz Iterable.
Para recorrerlo usaremos un for normal.

```
for (int i = 0; i < dias.getLength(); i++) {
}
```

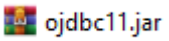
Ejemplo de recorrer un NodeList comentado

```
NodeList dias = doc.getElementsByTagName("dia");
for (int i = 0; i < dias.getLength(); i++) {
	Node nodo = dias.item(i);
	if (nodo.getNodeType() == Node.ELEMENT_NODE) {
		Element dia = (Element) nodo;
		String fecha = dia.getAttribute("fecha"),
				tempMax =
dia.getElementsByTagName("maxima").item(0).getTextContent(),
				tempMin =
dia.getElementsByTagName("minima").item(0).getTextContent();
		System.out.println("Día " + fecha + ", máxima " + tempMax + ", mínima
" + tempMin);
	}
}
```

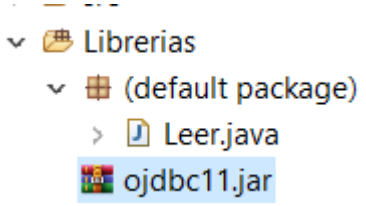
java + oracle

IMPORTANTE, las instrucciones que hagamos desde oracle, son como si tuviésemos puesto el autocommit, es decir, se guardarán directamente los cambios realizados. Si modificamos tablas desde oracle y no tenemos el autocommit activado, los cambios que hagamos desde oracle no se verán reflejados en nuestro programa hasta que hagamos un commit.

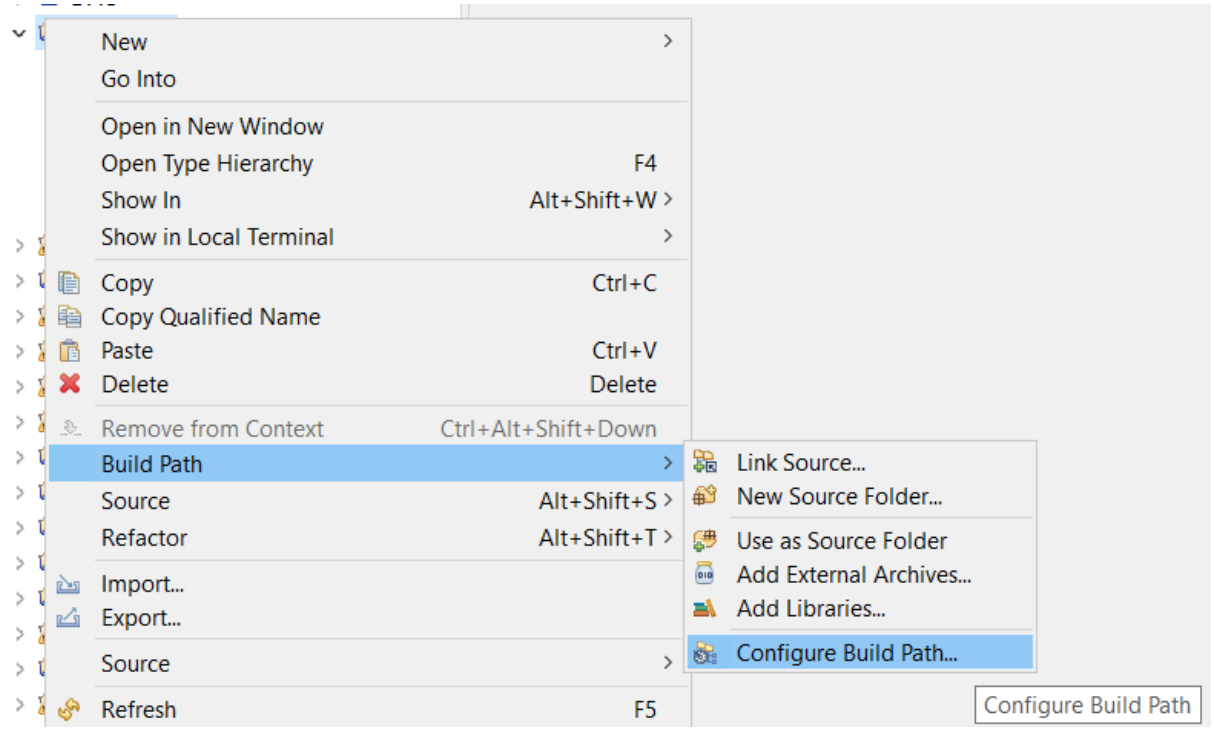
Para poder hacer esto, tenemos que descargar jdbc de oracle para poder conectar oracle con java. Para descargarlo, vamos a la página de descarga, si no tenemos la página de descarga, con buscar “descargar jdbc oracle” el primer resultado seguramente ya sea la página de descarga.

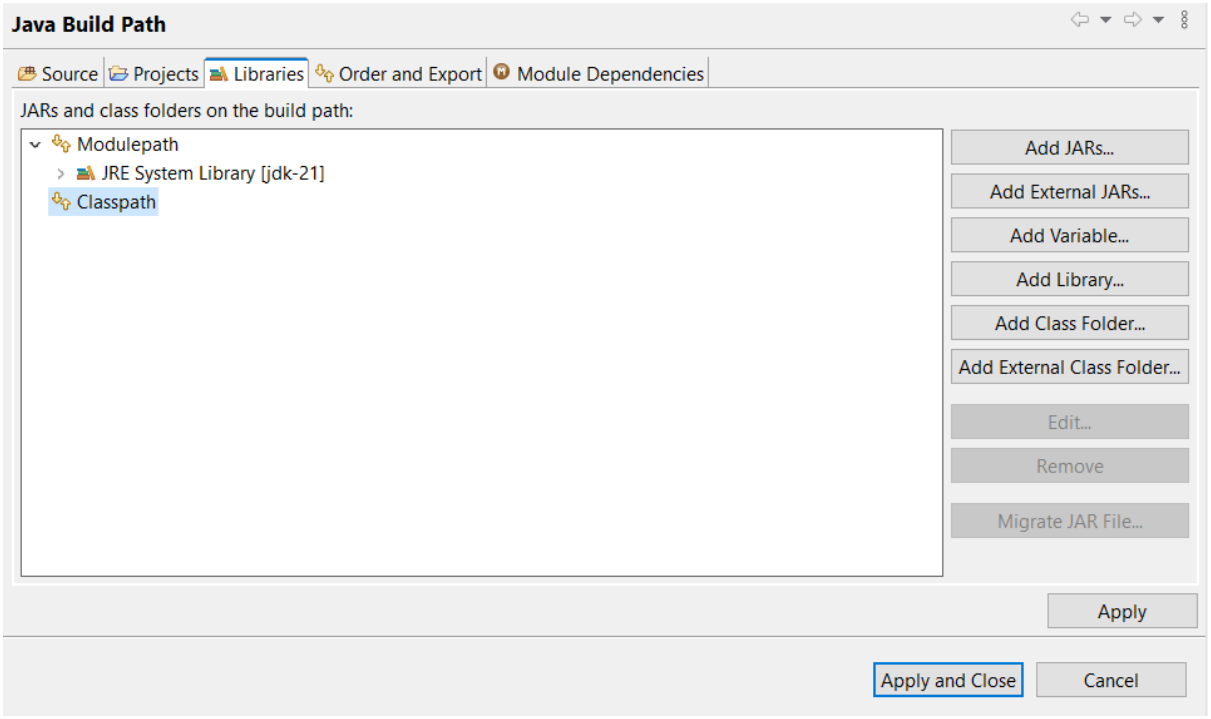


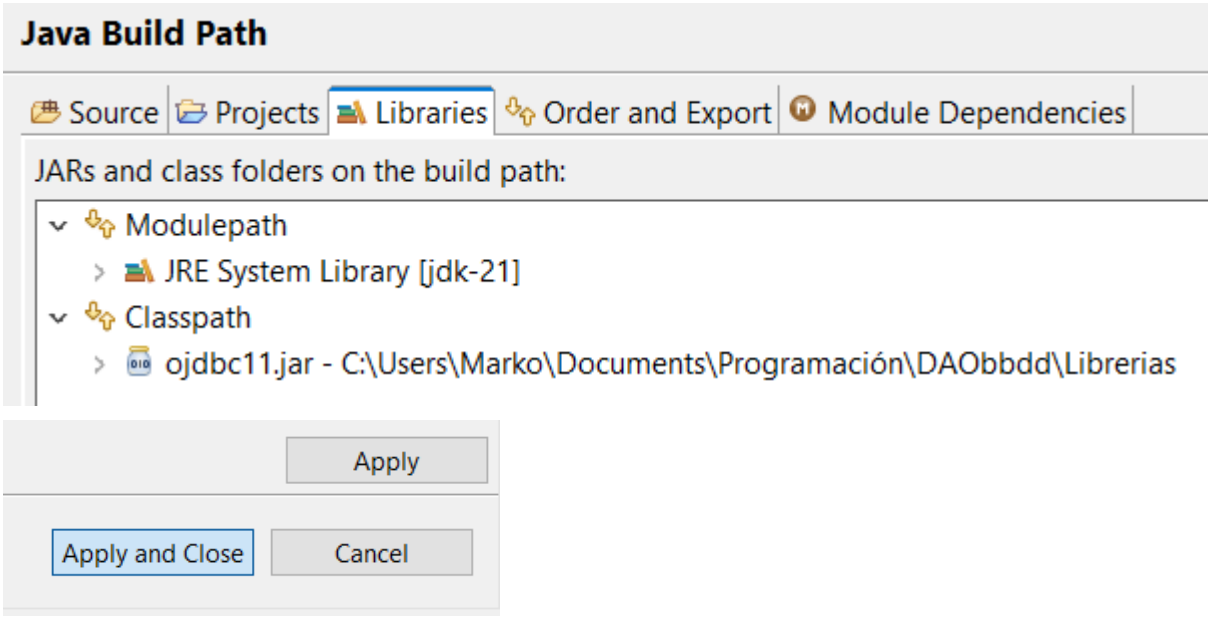
Una vez descargado, lo ponemos en el proyecto, en una carpeta librerías por ejemplo, en la que podemos tener también la clase leer.



Ahora tenemos que meter la librería en el proyecto, pese a estar dentro del proyecto, no hemos configurado el proyecto para que lo use.
Hacemos clic derecho en el proyecto → “Build Path” → “Configure Build Path” → vamos al apartado “Libraries” → clic en “Classpath” → clic en “Add External JARs” → seleccionamos el jdbc de nuestro proyecto (si lo usamos en varios proyectos, puede que seleccionemos el jdbc de otro proyecto en vez de usar el de nuestro proyecto) y pulsamos abajo a la izquierda “Apply and Close”.







Conexión

Para poder interactuar con una base de datos, primero tendremos que conectarnos a dicha bbdd. Para realizar la conexión, tenemos que crear un objeto de la clase Connection.
Connection con = DriverManager.getConnection("url_de_la_bbdd", "usuario", "contraseña");
Al finalizar la conexión, la cerramos.

```
try {
	con.close();
} catch (SQLException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

Conexión con una bbdd en otro equipo

```
Connection con = null;
try {
	con = DriverManager.getConnection("jdbc:oracle:thin:@//10.6.1.32:1521/xe",
"MARKO", "alumno");
} catch (SQLException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

Conexión con una bbdd local

```
Connection con = null ;
try {
	 con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","programacion","1234")
;
} catch (SQLException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

Statement y PreparedStatement

Statement y PreparedStatement sirven para ejecutar instrucciones sql, PreparedStatement hereda de Statement. Ambas clases nos permiten realizar instrucciones sql, hay distintos métodos para ejecutar las instrucciones. Todos ejecutan las instrucciones, pero varía lo que devuelven, los dos principales métodos que usamos son:
executeUpdate(); Usado para instrucciones DML, devuelve el total de filas afectadas por la instrucción (INSERT, UPDATE, DELETE).
executeQuery(); Usado para realizar consultas, devuelve un objeto de la clase ResultSet, el cual contiene el resultado de la consulta.

Statement

Para realizar sentencias sql que siempre van a ser iguales, usamos la clase Statement.
Para crear un Statement usamos un objeto de la clase Connection.
Creamos el Statement.
Statement s = con.createStatement();
Ejecutamos el Statement indicando con un String cual es la instrucción.
s.executeQuery("SELECT * FROM alumnos");

PreparedStatement

Para realizar sentencias en las cuales pueden variar los datos, usamos la clase PreparedStatement. para crear un PreparedStatement usamos un objeto de la clase Connection.
En las instrucciones ponemos un ?, el cual indicaremos su valor utilizando el método setXXX(), donde XXX varía según el tipo de dato.
Creamos el PreparedStatement, en el cual indicamos la instrucción.
PreparedStatement ps = con.prepareStatement("DELETE FROM alumnos WHERE id_alumno=?");
Cambiamos el valor del ?, indicando primero con un número a qué interrogante queremos asignarle un valor y luego el valor a asignar. Los interrogantes empiezan a contarse desde el número 1, no como los vectores por ejemplo, que empiezan en 0.
ps.setInt(1, id);
Ejecutamos la instrucción.
ps.executeUpdate()

Select

Mediante un Statement o un PreparedStatement ejecutamos un select y lo guardamos en un ResultSet, para eso tenemos que usar ExecuteQuery().
Statement s = con.createStatement();
ResultSet rs = s.executeQuery("SELECT * FROM alumnos");

Ahora tenemos que recorrer el ResultSet.
El ResultSet podríamos imaginarnos que es como la siguiente tabla, en la cual tenemos que recorrer los resultados uno a uno. Para pasar a la siguiente línea, tenemos que usar rs.next(). Empezamos en la primera fila, la cual no tiene contenido, es el nombre de las columnas, por lo que primero tenemos que hacer un next() para leer la primera fila.

id	nombre	apellido
1	Paco	Sandemetrio
2	Manolo	Torcuato
3	Antonia	Gutierrez

Para sacar los datos usamos rs.getXXX(nº_de_columna) donde XXX es el tipo de dato. En este ejemplo sería rs.getInt(1), rs.getString(2) y rs.getString(3).
No sabemos cuántas filas nos ha devuelto el select, pero rs.next(), a parte de pasar a la siguiente fila, devuelve un booleano indicando si la fila a la que nos ha cambiado tiene contenido, por lo que podemos recorrer todo el contenido del ResultSet con un while.
Ejemplo en el que recorremos los resultados y los añadimos a un List:

```
while (rs.next()) {
        lista.add(new Alumno(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getInt(4), rs.getInt(5)));
}
```

Ventanitas

JFrame

Constructor

Para crear una ventana usamos JFrame.
Creamos una ventana llamada Alumnos.
JFrame frame = new JFrame("Alumnos");

Añadir contenido a la ventana

Para añadir contenido a la ventana, usamos add. Normalmente, a un frame solo le añadimos paneles y a dicho paneles, les pondremos el contenido de la ventana.
JPanel panel = new JPanel(); // Creamos un panel
frame.add(panel); // Añadimos el panel en el frame

Indicar tamaño

Con setSize indicamos el tamaño de la ventana, pasando primero el eje x (ancho) y luego el y (altura).
Indicamos que la ventana tiene un ancho de 300 pixeles y un alto de 150 pixeles.
frame.setSize(300,150);

Para hacer que el tamaño de la ventana se asigne según el tamaño de la ventana usamos Toolkit.
Toolkit tk = Toolkit.getDefaultToolkit(); // Nos sirve para sacar el ancho y alto
int xSize = ((int) tk.getScreenSize().getWidth()); // Sacamos el tamaño del ancho
int ySize = ((int) tk.getScreenSize().getHeight()); // Sacamos el tamaño del alto
frame.setSize((int) Math.round(xSize * 0.50), (int) Math.round(ySize * 0.50)); // Indicamos el tamaño

Permitir modificar el tamaño de la ventana

Para indicar si queremos que se pueda cambiar el tamaño de una ventana usamos setResizable.
frame.setResizable(true); // Indicamos si podemos o no cambiar su size

Posicionar la ventana

Usamos setLocationRelativeTo para indicar cual es la posición de la ventana.
frame.setLocationRelativeTo(null); // Lo ponemos en el centro

Cerrar ventana

Para indicar que hace el programa al cerrar la ventana, usamos setDefaultCloseOperation(). Al indicar “JFrame.EXIT_ON_CLOSE” indicamos que al cerrar la ventana se cerrará el programa.
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

Mostrar ventana

Para que se vea la ventana, tenemos que hacerla visible. Hay que hacerla visible después de añadir el contenido a la ventana/paneles de la ventana, así se mostrará la ventana con todo el contenido añadido.
frame.setVisible(true); // Muestra el frame, la hacemos visible

Métodos comunes

getXXX

Podemos obtener información de los componentes.
getName() // Devuelve un objeto String
Gets the name of the component.
getParent() // Devuelve un objeto Container
Gets the parent of this component.
getFont() // Devuelve un objeto Font
Gets the font of this component.
getBackground() // Devuelve un objeto Color
Gets the background color of this component.
…

isXXX

Podemos obtener información de los componentes que se puede resumir en true o false, por lo que todos estos métodos devuelven true o false.
isBackgroundSet()
Returns whether the background color has been explicitly set for this Component.
isEnabled()
Determines whether this component is enabled.
isOpaque()
Returns true if this component is completely opaque, returns false by default.
isShowing()
Determines whether this component is showing on screen.
isVisible()
Determines whether this component should be visible when its parent is visible.
…

setXXX

Podemos establecer varios parámetros de un objeto.
setBounds(int x, int y, int width, int height)
Indicamos el tamaño que ocupa un componente, depende del layout.
setEnabled(boolean b)
Enables or disables this component, depending on the value of the parameter b.
setFocusable(boolean focusable)
Sets the focusable state of this Component to the specified value.
setFont(Font f)
nombre.setFont(new Font("Arial", Font.PLAIN, 20));
Sets the font of this component.
setForeground(Color c)
nombre.setForeground(Color.RED);
Sets the foreground color of this component.
setBackground(Color c) // Quizá tengamos que usar setOpaque()
panel.setBackground(Color.red);
Sets the background color of this component.
…

Otras opciones

Según el componente podremos hacer unas cosas u otras, ya que al final cada componente tiene sus métodos.
Hay ciertas acciones que nos pueden resultar interesantes para distintos componentes:
setText() o getText() para modificar u obtener el texto de etiquetas o campos de texto.
setEnabled(), para activar o desactivar componentes como campos de texto o botones para decidir cuándo se pueden usar.

JPanel

Constructor

En el panel vamos a añadir el contenido que queremos que aparezca en la ventana.
Tenemos que añadir el panel a un frame.
JPanel panel = new JPanel(); // Creamos un panel
frame.add(panel); // Añadimos el panel en el frame

Añadir contenido al panel

Para añadir cualquier elemento al panel, usamos add.
panel.add(userLabel); // Añadimos un componente al panel, en este caso, una etiqueta

Indicar la disposición del panel

Para indicar cual va a ser la disposición del panel usamos setLayout

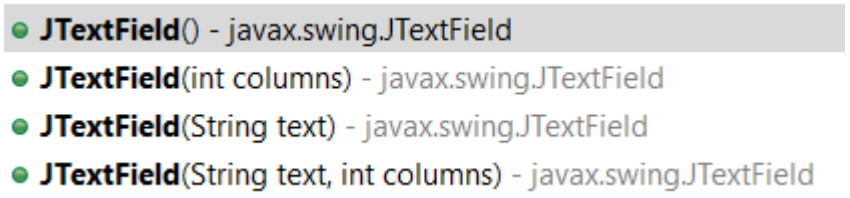
Con el siguiente layout indicamos que no usará ningún diseño, por lo que tendremos que indicar en qué posición poner nosotros los elementos.
panel.setLayout(null); // Indicamos la disposición

JLabel

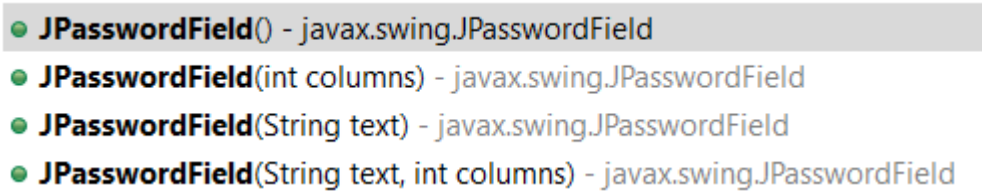
Creamos una etiqueta con el nombre lblEjemplo cuyo texto es "Hola, soy una etiqueta :)".
JLabel lblEjemplo = new JLabel("Hola, soy una etiqueta :)");

JTextField y JPasswordField

Ambas clases sirven para escribir texto, JPasswordField hereda de JTextField, la diferencia es que el JPasswordField no te permite ver que se ha escrito.
Para declarar un campo de texto, podemos no declarar nada, declarar el número de columnas o el texto que aparece por defecto o ambos.



Para crear un JPasswordField lo mismo.

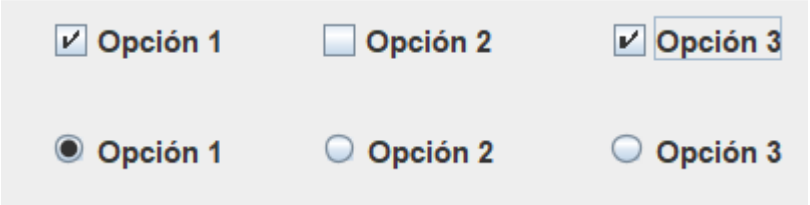


JComboBox

Creamos un comboBox que nos va a mostrar String.
JComboBox<String> comboBox = new JComboBox<>();
comboBox.addItem("Opción1");
comboBox.addItem("Opción2");
comboBox.addItem("Opción3");
Cuando añadimos un action listener a un comboBox, se ejecutará siempre que seleccionemos una opción (incluso si es la misma opción).
comboBox.addActionListener();
Para obtener la opción elegida de un comboBox usamos getSelectedItem.
comboBox.getSelectedItem()

JCheckBox y JRadioButton

Ambas clases sirven para añadir casillas que podemos seleccionar, la diferencia es que en JCheckBox podemos seleccionar ninguno o tantos como queramos, mientras que en JRadioButton vamos a agruparlos y únicamente podremos seleccionar una de las casillas que pertenezcan a un grupo.
Para comprobar si están seleccionadas podemos usar isSelected().



JCheckBox

Creamos un JCheckBox con el texto "Opción 1".
JCheckBox chckbxOp1 = new JCheckBox("Opción 1");
Creamos un JCheckBox con el texto "Opción 1" que por defecto estará activado.
JCheckBox chckbxOp1 = new JCheckBox("Opción 1", true);

JRadioButton

Creamos un JRadioButton con el texto "Opción 1".
JRadioButton rdbtnOp1 = new JRadioButton("Opción 1");
Creamos un JRadioButton con el texto "Opción 1" que por defecto estará activado.
JRadioButton rdbtnOp1 = new JRadioButton("Opción 1", true);
Para que solo podamos seleccionar un único JRadioButton, tenemos que crear un ButtonGroup y añadirlos al ButtonGroup.

```
ButtonGroup bg = new ButtonGroup();
                bg.add(rdbtnOp1);
                bg.add(rdbtnOp2);
                bg.add(rdbtnOp3);
```

JTextArea

Creamos un campo de texto.
JTextArea textAreaEjemplo = new JTextArea();
Creamos un campo de texto con el texto "Soy un campo de texto :D".
JTextArea textAreaEjemplo = new JTextArea("Soy un campo de texto :D");

JTable

-Creamos un DefaultTableModel indicando el número de filas y el número de columnas.
DefaultTableModel modelo = new DefaultTableModel(0, 5);
-Creamos una tabla a la que le pasamos el DefaultTableModel creado.
JTable table = new JTable(modelo);
-Usamos sobre el objeto DefaultTableModel el método setColumnIdentifiers(), al que le pasamos un vector de String para indicar los nombres de las columnas.
Opción 1:
String[] titulos = { "id_alumno", "nombre", "apellido", "nota1", "nota2" };
modelo.setColumnIdentifiers(titulos);
Opción 2:
modelo.setColumnIdentifiers(new String[] { "id_alumno", "nombre", "apellido", "nota1", "nota2" });
-Creamos un objeto JScrollPane y al que le pasamos la tabla creada.
JScrollPane js = new JScrollPane(table);
-Añadimos el JScrollPane al frame.
frame.add(js);

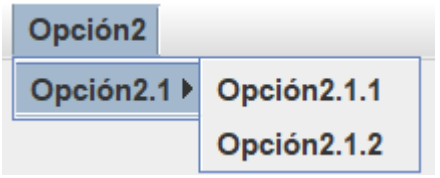
Para añadir filas, usamos addRow() sobre el objeto DefaultTableModel creado, al cual le tenemos que pasar un vector de Object.
modelo.addRow(new Object[] {22,"a","b",4,8});
Para eliminar filas, usamos removeRow sobre el objeto DefaultTableModel creado, al cual le pasamos qué número de fila queremos quitar (empieza en 0).
model.removeRow(0);
Si queremos quitar todas las líneas, usamos el siguiente bucle.

```
while (model.getRowCount() > 0) {
	model.removeRow(0);
}
```

Menú

-Creamos un objeto JMenuBar e indicamos al frame que añadimos una barra de menú con setJMenuBar().
JMenuBar menuBar = new JMenuBar();
frame.setJMenuBar(menuBar);
-Creamos un objeto indicando su nombre y lo añadimos al JMenuBar creado con add.
JMenu mnNewMenu = new JMenu("New menu");
menuBar.add(mnNewMenu);
-Creamos un objeto JMenuItem y lo añadimos al menú creado con add().
JMenuItem mntmNewMenuItem = new JMenuItem("New menu item");
mnNewMenu.add(mntmNewMenuItem);

También podemos añadir un menú a un menú, para tener submenús.




```
JMenu mnMenu2 = new JMenu("Opción2");
menuBar.add(mnMenu2);
JMenu mnMenu2_1 = new JMenu("Opción2.1");
mnMenu2.add(mnMenu2_1);
JMenuItem mntmItem2_1_1 = new JMenuItem("Opción2.1.1");
mnMenu2_1.add(mntmItem2_1_1);
JMenuItem mntmItem2_1_2 = new JMenuItem("Opción2.1.2");
mnMenu2_1.add(mntmItem2_1_2);
```

Cuando añadimos un action listener a un JMenuItem, se ejecutará siempre que seleccionemos ese JMenuItem.

```
mntmItem2_1_2.addActionListener(e -> System.out.println("Hola"));
```

ActionListener()

Para establecer acciones en los componentes de una ventana, usamos la interfaz ActionListener.

Tenemos tres opciones:

-Crear una clase anónima.

```
btn1.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {
		System.out.println("Mortadela");
	}
});
```

-Usar una clase que implemente la interfaz ActionListener.

Añadir el ActionListener.

```
btn2.addActionListener(new EjemploListener());
```

Crear clase que implementa ActionListener.

```
public class EjemploListener implements ActionListener{

	@Override
	public void actionPerformed(ActionEvent e) {
		System.out.println("Patata");
	}

}
```

-Usando expresiones Lambda.

```
btn3.addActionListener(e -> ejemploLambda());
```

Si usamos lambda solo podemos realizar una acción, por lo que lo mejor sería que pongamos un método.