# MODULO M03: PROGRAMACIÓN B

ILERNA Online

Inicialmente, se va a generar un **bytecode** que, después, va a ser traducido por la máquina en el lugar en el que se ejecute el programa

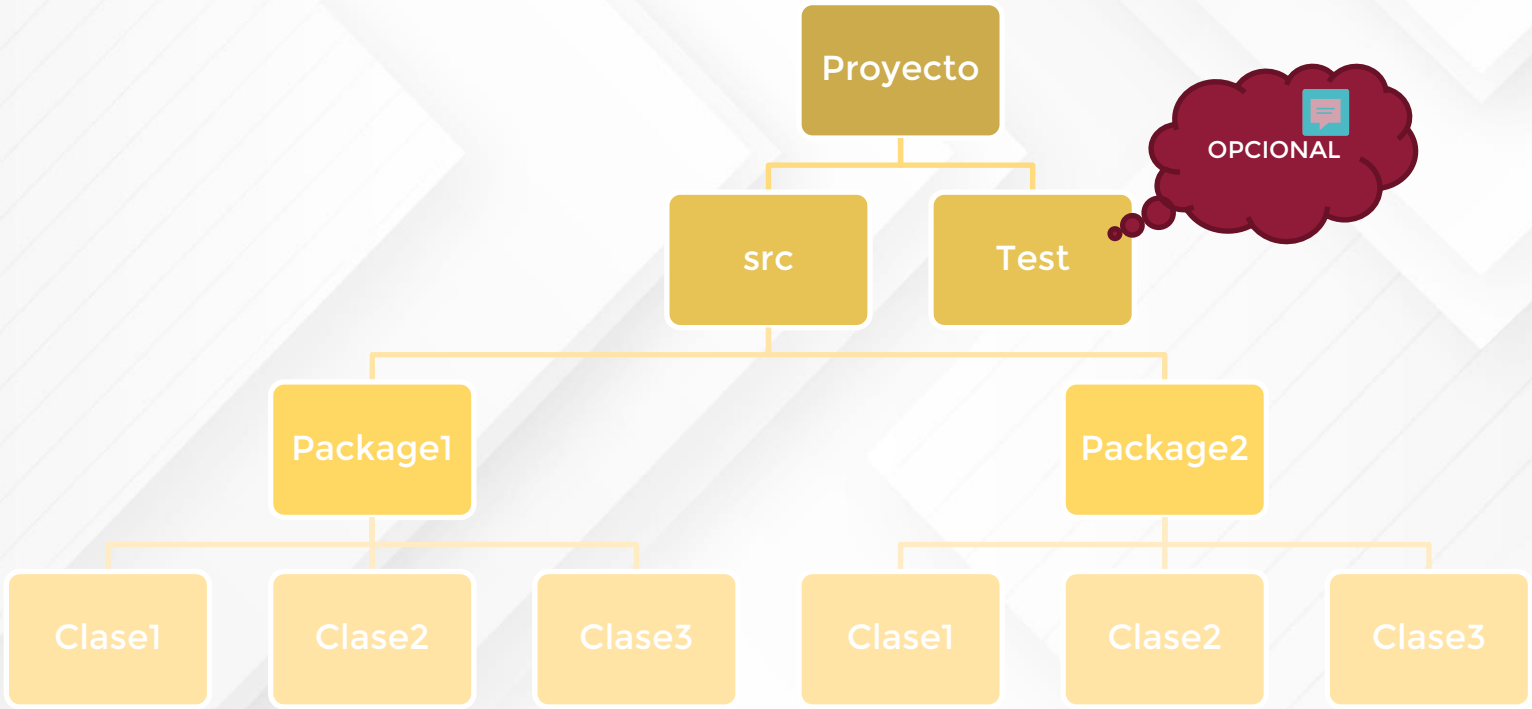
# UF1

---

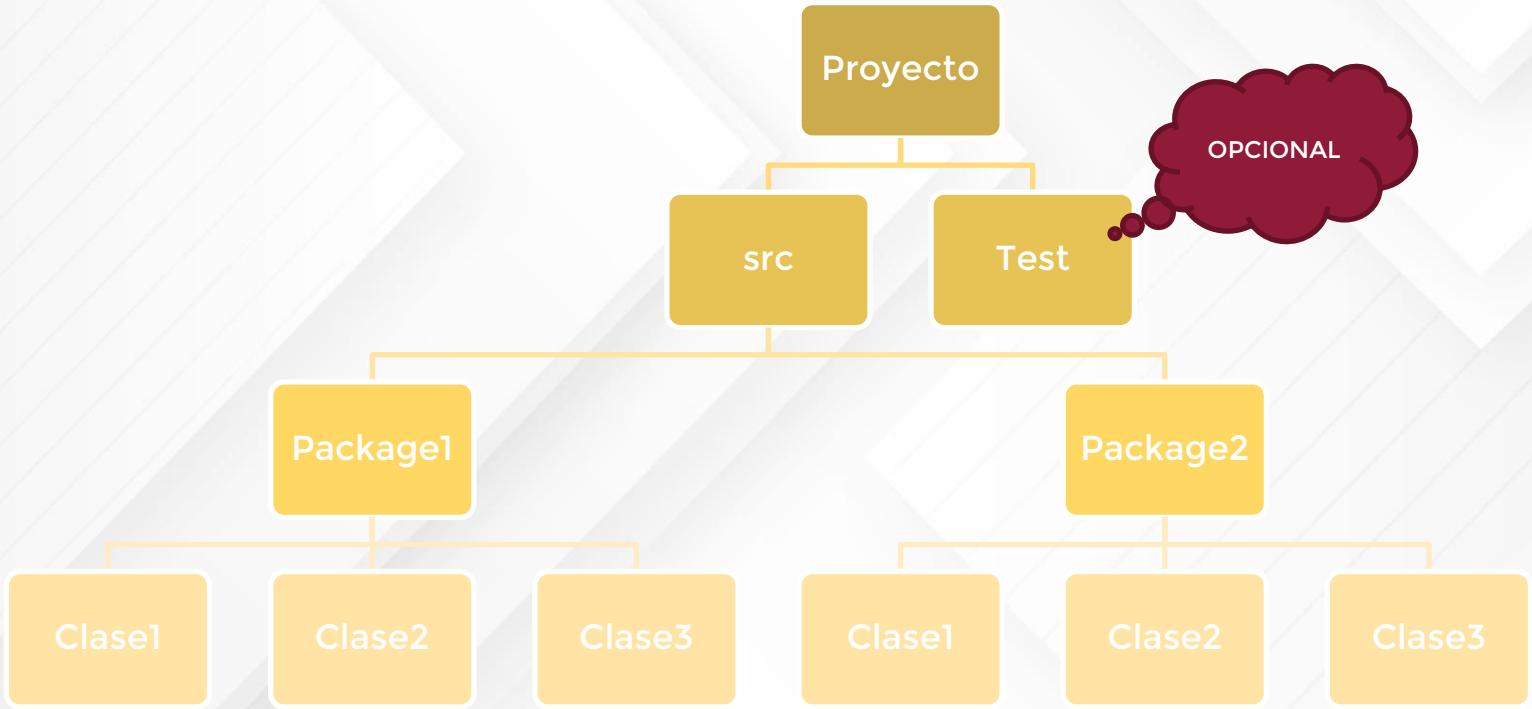
ÁLVARO ORTEGA

>

Las clases en Java siempre se inician con la inicial en mayúscula.

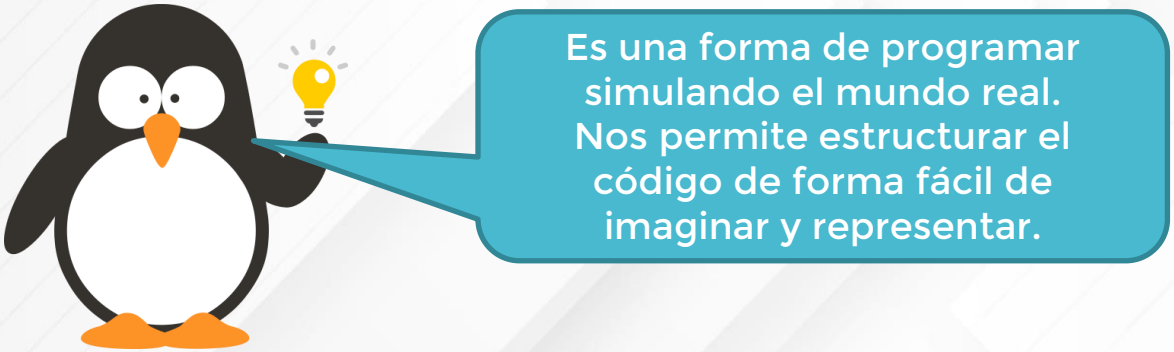


>


Proyecto
OPCIONAL
src
Test
Package1
Package2
Clase1
Clase2
Clase3
Clase1
Clase2
Clase3

| | | |
|---|---|---|
| Numéricos | Enteros | short |
| | | int |
| | | long |
| | Reales | float |
| | | double |
| Char | Representar un único carácter | |
| Boolean | Almacena valores lógicos (*true o false)* | |

# ¿Qué es la programación orientada a objetos?

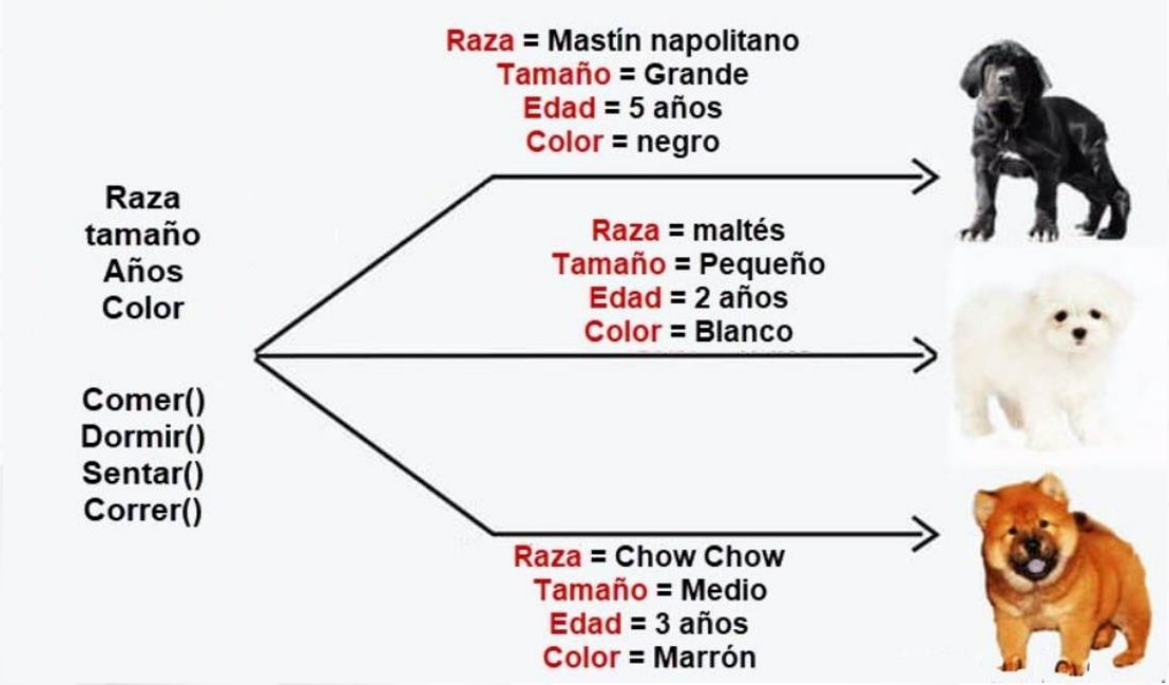



¿Pero qué es una clase y qué es un objeto?


La inicialización consiste en crear un nuevo objeto de esa plantilla.
Le damos valores a sus atributos, mediante constructores

Plano que llevamos a una fabrica para que nos fabrique el objeto

## > Otro ejemplo de clase y objeto



# Variables en Java

## > Buenas practicas

- Declarar las variables al inicio de la clase o función
- Una variable sin inicializar, tiene un valor desconocido
- Siempre que se pueda se le da un valor inicial.

```java
public static void main(String[] args) {
    //Numero entero
    int numero = 0;
    //Numero decimal
    float decimal = 0.1f;
    //Caracter
    char a = 'a';
    //Caracteres
    String aa = "aa";
    //True or False
    boolean condicion = false;

    /* Escritura del código del programa */
}
```

## ¿Cómo se crea un 1r proyecto?

1. Creación de un proyecto
2. Creación de dos clases:
    1. Puerta
    2. Main
3. Creación de atributos
4. Creación de constructores
5. Creación de funciones getter/setter.
6. Método toString()

# VT3 CLASES Y MÉTODOS

# > Clases, Funciones y Atributos

ILERNA Online

## Clase

Es una plantilla

Compuesta de:
- Atributos
- Funciones

Inicialización mediante constructores

Se muestra gracias a la función toString()

## Funciones

Visibilidad
- Public
- Private
- Protected
- Package

Retorno

nombre

Parámetros

## Atributos

Visibilidad

Tipo
- Primitivo
- Clase

Nombre

Valor

## > Getters y Setters

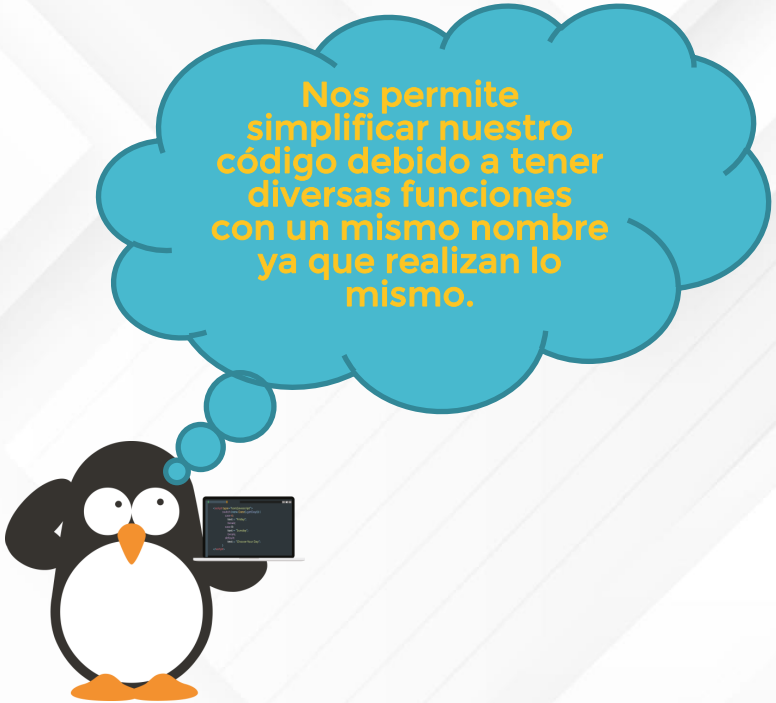
- Son funciones para:
  - Establecer valor a los atributos
  - Obtener valor de los atributos

- Buena praxis, variables privadas y se accede mediante sus funciones.
- Un getter/setter para cada variable que deseemos que tenga acceso el usuario.

```
public float getAltura() { return altura; }

public void setAltura(float altura) { this.altura = altura; }
```

## > Sobrecarga de métodos

- Es una característica de algunos lenguajes como Java, permite:
- Funciones con mismo nombre y diferentes parámetros.
- Ejecutará una función u otra en función de los métodos recibidos (constructores)



# System

- Uso habitual:
  - Entrada por teclado
  - Salida por pantalla
    - Salida estándar
    - Salida de error.
- Contiene las variables:
  - In
  - out
  - err

# > System.in

- La variable in es utilizada para poder entrar datos nuestro programa

```
System.out.println("Introduce un caracter");
int number = System.in.read();
System.out.println(number);
// Flush()
System.in.read();

System.out.println("Introduce 100 caracteres");
byte[] buffer = new byte[100];
System.in.read(buffer);
for (int i = 0; i < 100; i++) {
    System.out.print((char) buffer[i]);
}
```

## > Clase Scanner

- La clase Scanner nos permite simplificar toda la tarea de recuperar valores por teclado.
  - next()
  - nextLine()
  - nextDouble()
  - nextInt()
  - ….

```java
Scanner scanner = new Scanner( source: System.in);
System.out.println("Introduce un String");
String string = scanner.next();
System.out.println(string);

System.out.println("Introduce una linea");
String line = scanner.nextLine();
System.out.println(line);

System.out.println("Introduce un double");
double num = scanner.nextDouble();
System.out.println(num);
```

## > System.out/err

- Salidas por pantalla:
  - out → salida estándar
  - err → salida error.

```
System.out.println("Este es un mensaje estandard");
System.err.println("Este es un mensaje de error");
```

```
Este es un mensaje estandard
Este es un mensaje de error
```

# Repaso general

- **Clases:**
  - Plantilla de la estructura que deseamos crear.
  - Tiene atributos → características
    - Se pueden inicializar en los constructores.
  - Tiene funciones → acciones que realizará.
    - Funciones getter/setter para obtener/establecer valores a los atributos.
- **Para inicializar un nuevo objeto de una clase se debe utilizar la palabra reservada `new`**
- **Visibilidad**

## > Recuperar y sacar datos estándar

- Salida de datos

- Salida estándar;
```
System.out.println();
```

- Salida error:
```
System.err.println();
```

- Entrada estándar:

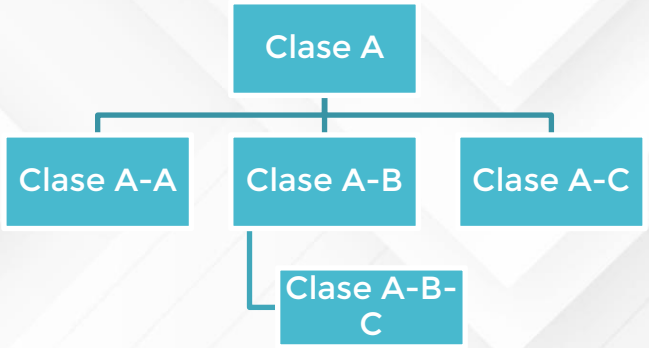
```
System.in.read();
```

```
Scanner teclado = new Scanner(System.in);
```

# Herencia

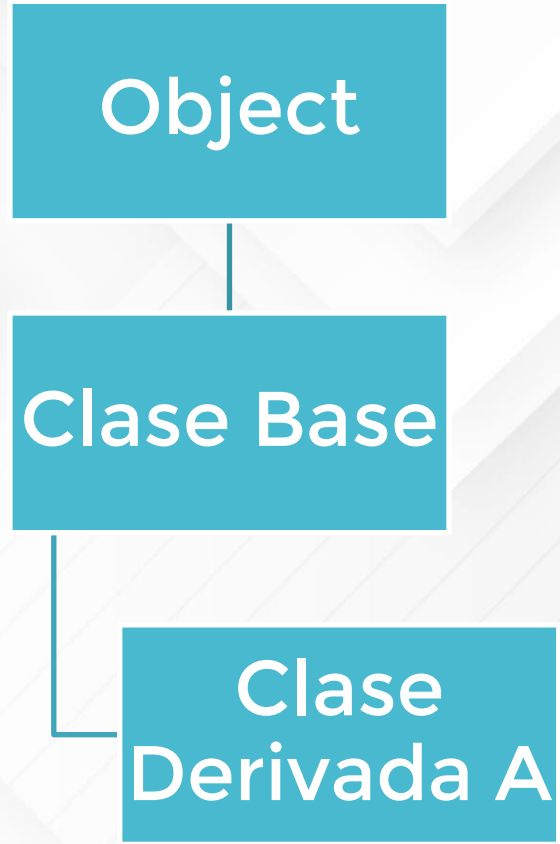
## > ¿Qué es la herencia?

- La herencia es la capacidad de crear clases que reutilicen código de otras ya existentes, adquiriendo sus atributos y funciones.
- Tiene una estructura jerárquica en forma de árbol, esto quiere decir que una clase solo puede heredar de otra.
- En java NO existe la herencia múltiple entre clases.




- ¡Conceptos básicos!

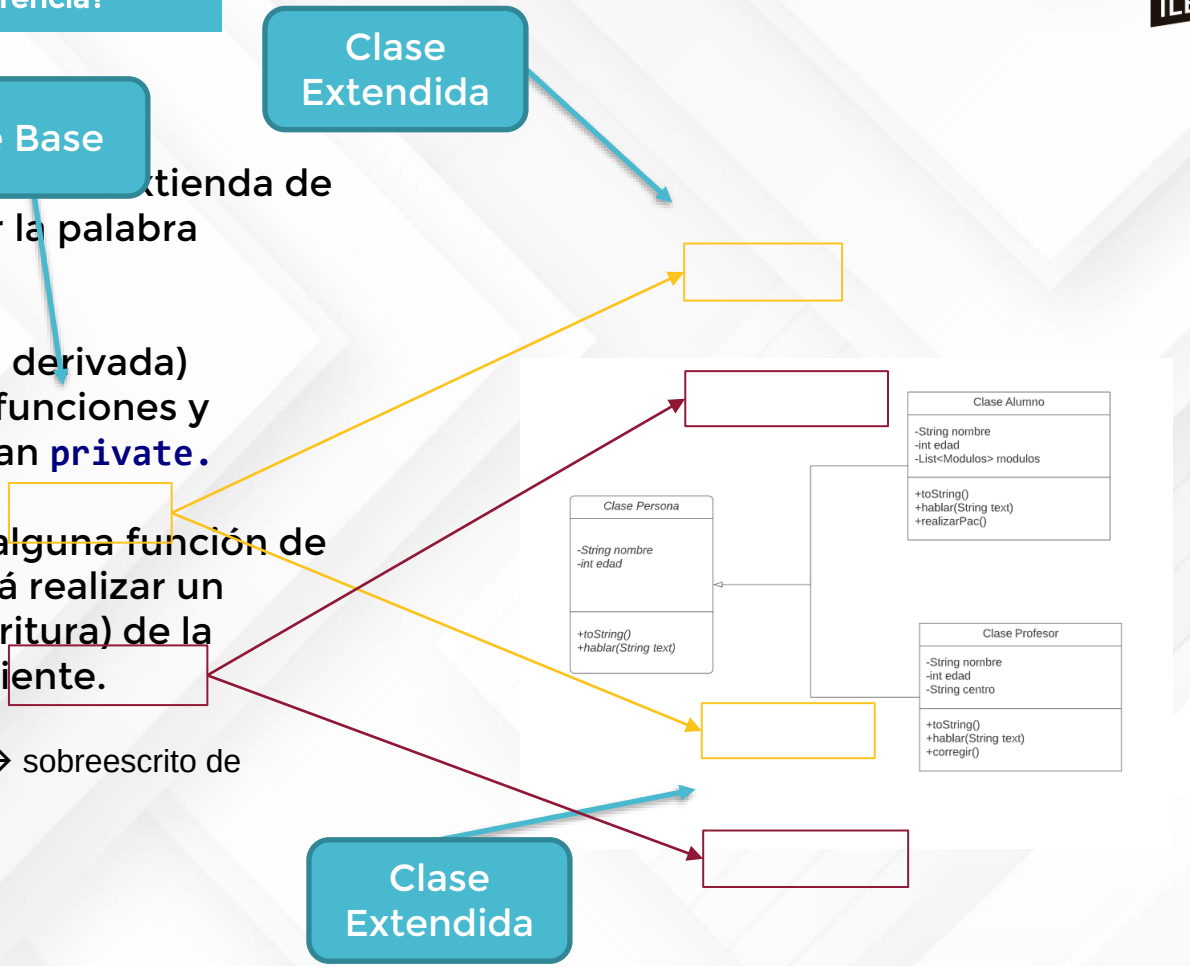
- Clase Base: superclase
  - Clase desde la que se hereda, en la jerarquía es la clase que está en la sección superior
- Clase derivada: subclase
  - Clase que hereda de otras, aprovecha la funcionalidad que le da la clase superior
  - A su vez puede ser la clase base de otras clases.
  - Si deseamos modificar el comportamiento de una función de la clase superior deberemos sobreescribir la clase (override)



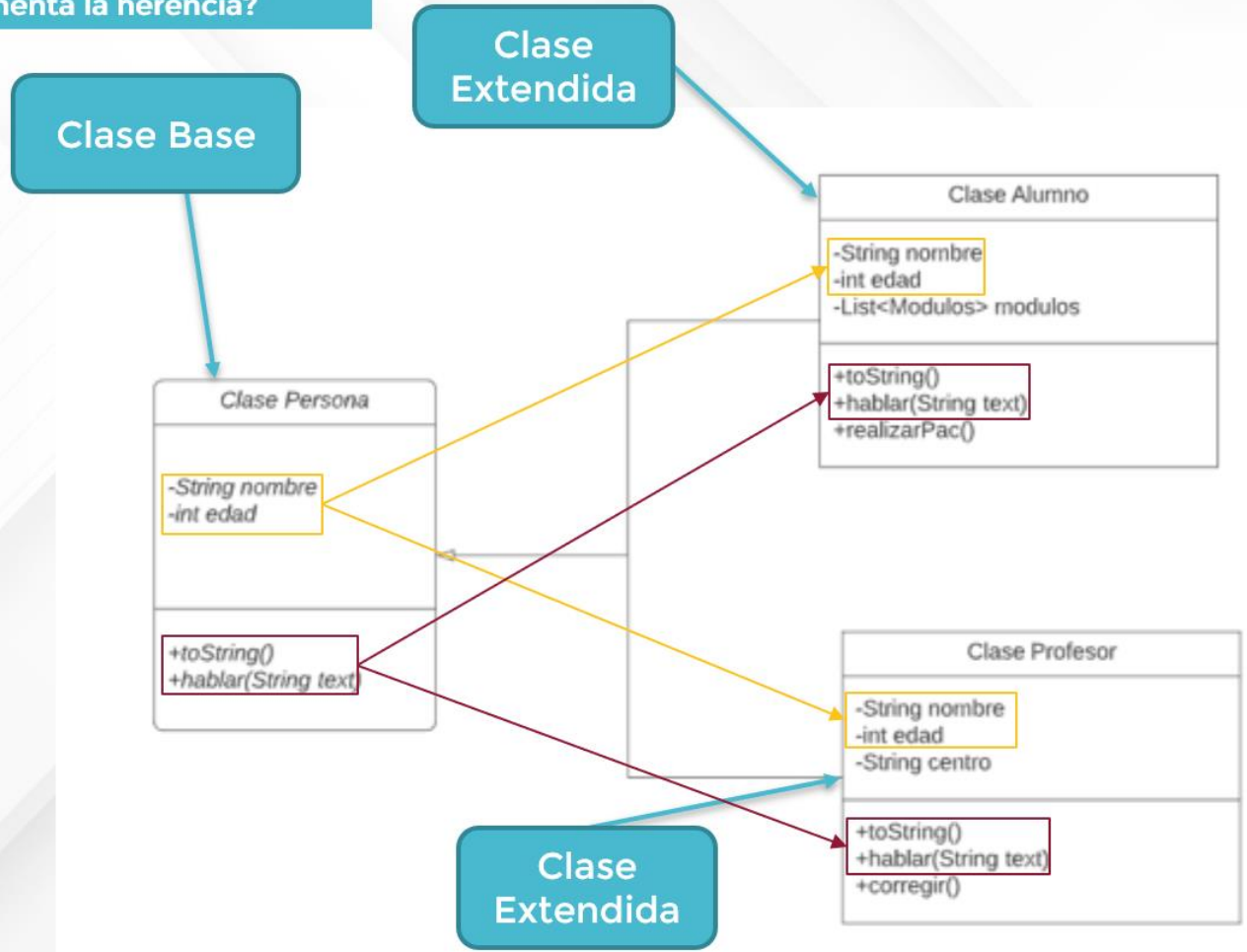
| | Mismo paquete | | Otro paquete | |
|---|---|---|---|---|
| | Subclase | Otra | Subclase | Otra |
| Private | No | No | No | No |
| Package | Si | Si | No | No |
| Protected | Si | Si | Si | NO |
| Public | Si | Si | Si | Si |

## > ¿Cómo se implementa la herencia?

- Para hacer que una clase herede de otra se debe utilizar la palabra reservada `extends`.
- La subclase (o clase derivada) obtendrá todas las funciones y métodos que no sean `private`.
- Si desea modificar alguna función de la superclase deberá realizar un `override` (sobreescritura) de la función correspondiente.
    - Método toString() → sobreescrito de Object.



## > ¿Cómo se implementa la herencia?



# Clases Abstractas

## > Clases Abstractas

- Son clases que no se pueden instanciar.
  - Tienen atributos y funciones que podemos utilizar posteriormente.
  - Puede contener funciones no implementadas, sin cuerpo. Métodos abstractos
- Se utilizan cuando no se tiene intención de crear objetos de una superclase pero si se desea simplificar código.
- Para evitar que un método se sobreescriba se deberá marcar con la palabra reservada `final`

```java
/*Clase BASE*/
public abstract class Persona {
    protected String nombre;
    protected int edad;

    public Persona() {
        nombre = "Sin nombre";
        edad = 0;
    }

    public Persona(String nombre, int edad) {
        this.edad = edad;
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public String getNombre() {
        return nombre;
    }

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

    /*Override para sobreescribir la clase Padre*/
    @Override
    public String toString() {
        return "Nombre: " + nombre + ", edad: " + edad;
    }

    public abstract void quienSoy();


}
```

## > Clases Abstractas

- Son clases que no se pueden instanciar.
  - Tienen atributos y funciones que podemos utilizar posteriormente.
  - Puede contener funciones no implementadas, sin cuerpo. Método abstractos
- Se utilizan cuando no se tiene intención de crear objetos de una superclase pero si se desea simplificar código.
- Para evitar que un método se sobreescriba se deberá marcar con palabra reservada `final`

```java
/*Clase BASE*/
public abstract class Persona {
    protected String nombre;
    protected int edad;

    public Persona() {
        nombre = "Sin nombre";
        edad = 0;
    }

    public Persona(String nombre, int edad) {
        this.edad = edad;
        this.nombre = nombre;
    }

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

    @Override /*Override para sobreescribir la clase Padre*/
    public String toString() {
        return "Nombre: " + nombre + ", edad: " + edad;
    }

    public abstract void quienSoy();
}
```

Nos permite crear funciones y variables que no se pueden modificar. Constantes.

# Interfaces

## > Interfaces

- Es una colección de métodos 100% abstractos y que se deben implementar.
- Contiene atributos pero estos son constantes y no se pueden modificar en ninguna de las diferentes implementación.
- Nos permite definir la estructura para diversas clases, simulando la herencia múltiple.

```
public interface Figura {

    void area();
    void perimetro();
}
```

# Librerías de clases fundamentales

# Strings

- Hemos trabajado constamente con ellos para representar texto.
- ¿Qué es un String?
    - Nos permite representar una cadena de caracteres

```
String str;
String string = "Esto es un texto";
String texto = new String("Esto es otro texto");
```

**char charAt (int indice)**: devuelve el carácter que se encuentra en la posición de índice.

**Boolean equals (String string)**: devuelve *True* si el objeto que se pasa por parámetro y el *string* son iguales. Si no, devuelve *False*.

**boolean isEmpty ()**: si la cadena es vacía, devuelve *True*, es decir, si su longitud es cero.

**int length ()**: devuelve el número de caracteres de la cadena.

**String [] Split (String expresión)**: devuelve un array de *String* con los elementos de la cadena expresión.

**String toLowerCase ()**: devuelve un array en el que aparecen los caracteres de la cadena que hace la llamada al método en minúsculas.

**String toUpperCase ()**: devuelve un array en el que aparecen los caracteres de la cadena que hace la llamada al método en mayúsculas.

**String trim ()**: devuelve una copia de la cadena, pero sin los espacios en blanco.

**String valueOf (tipo variable)**: devuelve la cadena de caracteres que resulta al convertir la variable del tipo que se pasa por parámetro.

# Ejercicios para practicar

1. Dada una cadena mostrar por pantalla la cantidad de vocales que tiene
2. Dada una cadena invertir la misma y mostrar por pantalla

# Estructuras de datos avanzadas

ILERNA Online

## Conjuntos

- Grupo de elementos no duplicados, de valores únicos, los cuales pueden estar ordenados o no

## Listas

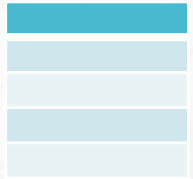
- Secuencia de elementos que ocupan una posición determinada.

## Colas

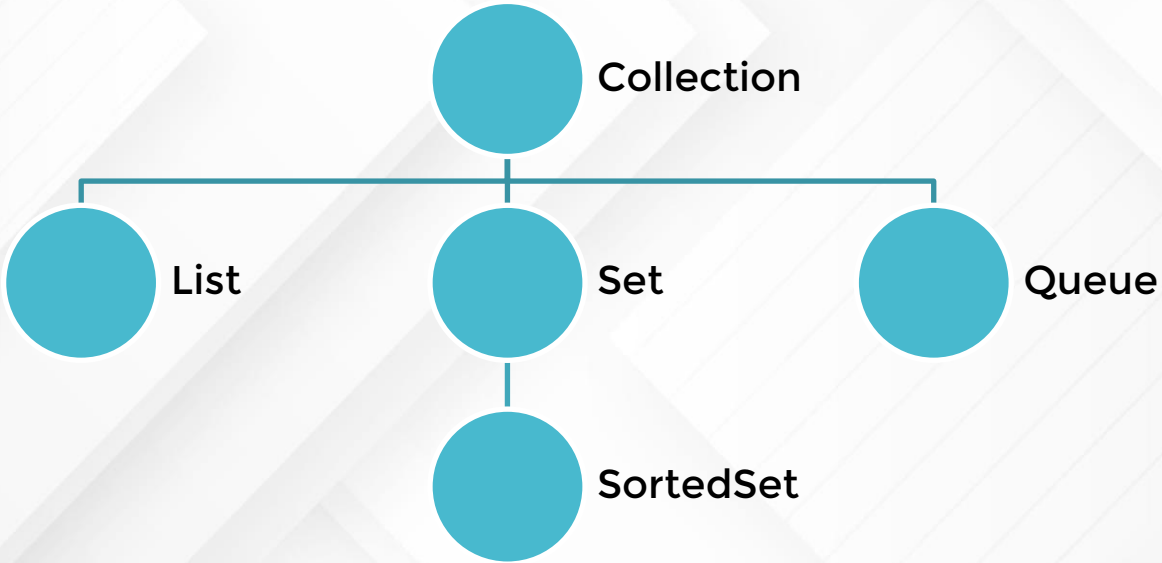
- Listas con la metodología FIFO First In First Out
  - Sus operaciones principales son enqueue & dequeue

## Pilas

- Listas que siguen la metodología LIFO, Last In First Out
  - Tiene dos operaciones push & pop



Una colección representa un grupo de objetos (elementos) que se pueden recorrer.



**List**:
Pueden estar repetidos, se indexan mediante un índice numérico.

**Set**:
Elementos no repetidos y sin ordenar, es muy eficiente para comprobar si un elemento se encuentra ya en la colección

**Queue**:
No permite el acceso aleatorio y solo se puede acceder al primer y último elemento.

ILERNA Online

Implementa una lista de elementos mediante un array de tamaño variable.

**Métodos más importantes:**

get(int) → Obtener el objeto de la posición indicada.

indexOf(Object) → Obtener la posición del objeto indicado.

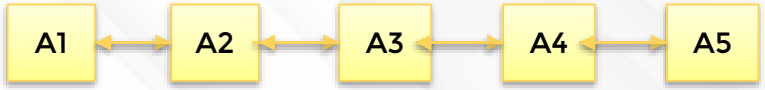
isEmpty() → Booleano para comprobar si existen datos.

add(Object) → Insertar Object en la última posición

add(int,Object) → Insertar Object en la posición indicada.

set(int,Object) → Insertar Object en la posición indicada, reemplaza el anterior valor.

toArray() → Convierte el ArrayList en un array del tipo especificado.




El acceso es secuencial, por lo que localizar un elemento que no se encuentra al principio/final lleva un tiempo proporcional al tamaño de la lista.
El rendimiento en los extremos es constante.

Métodos:

removeFirst() → Elimina el primer elemento de la lista

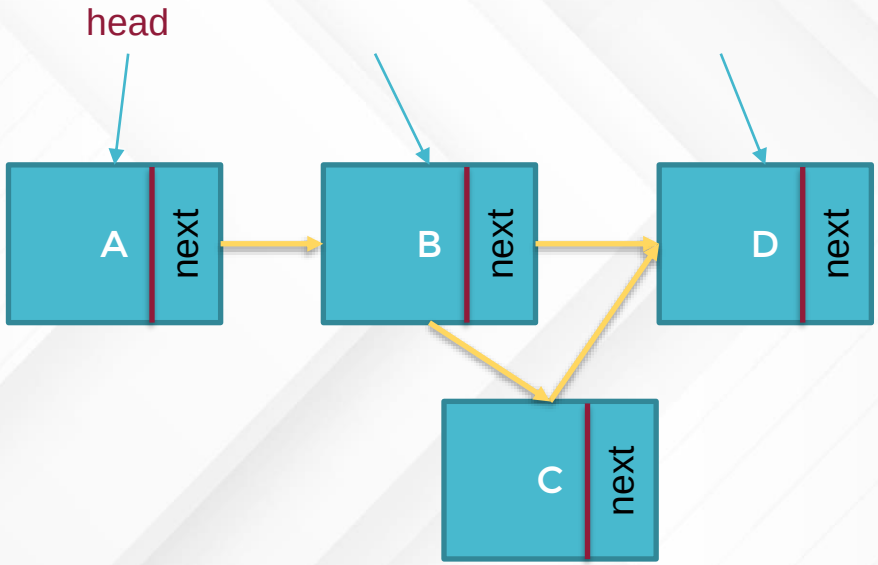
addFirst() → Añade un elemento al principio de la lista

addLast() → Añade un elemento al final de la lista

getFirst() → Devuelve el primer elemento de la lista

getLast() → devuelve el último elemento de la lista.

## > ¿Como funciona Linked List?




addLast → addLast → next → next → add(2,C)

## > Iterator

Para recorrer las colecciones se puede utilizar un Iterator(), este nos permite utilizar recorrer cualquier estructura de datos de exactamente la misma forma, sin importar la implementación interna de la misma.

Este iterador nos proporcionará unos métodos propios de este:
next () → devuelve el siguiente elemento en la iteración.

hasNext () → devuelve verdadero si la iteración tiene más elementos.

remove() → elimina de la colección subyacente el último elemento devuelto por este iterador.

```
ArrayDeque<Integer> arrayDeque = new ArrayDeque<>();
arrayDeque.add(1);
arrayDeque.addFirst(0);
arrayDeque.push(2);

Iterator<Integer> ite = arrayDeque.iterator();
while(ite.hasNext()){
    System.out.println(ite.next());
}
```

## > HashSet

Gracias a una función matemática llamada Hash, obtiene un identificador alfanumérico único para cada elemento.

La iteración a través de sus elementos es más costosa y necesitará recorrer todas las entradas de la tabla.

Funciones más importantes:

isEmpty() → Devuelve un booleano indicando si el conjunto contiene elementos.

Clear() → Borra todos los elementos del conjunto.

Add → Añade un nuevo elemento al set.

Remove → Elimina un elemento

Iterator → Devuelve un iterador.

# GENERICOS y COMODINES

Los genéricos nos permiten crear una clase o función sin un tipo definido.
Por ejemplo:

```
private static <T> void printData(T data) {
    System.out.println(data);

}
```

Beneficios de los genéricos:

1. Comprobación de tipos más fuertes en tiempo de compilación.
2. Eliminación de cast, mejorando la legibilidad del código.
3. Posibilidad de implementar algoritmos genéricos.

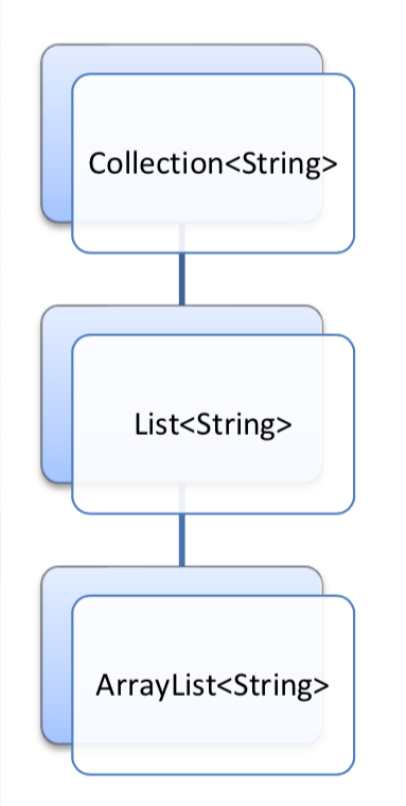
ILERNA Online

Una ventaja principal del código con genéricos es que trabajará automáticamente con el tipo de datos establecido.

Muchos algoritmos son lógicamente idénticos sin importar el tipo de datos que se use. Como por ejemplo un ArrayList

## > Ejemplo

```java
// Definimos la caja de tipo T, ya que no sabemos que tipo va a recibir.
//Puede ser de cualquier tipo.
public class Caja<T> {

    //ArrayList de tipo T, será del tipo que se cree la clase.
    //Aunque a su vez ArrayList es generica, esto nos permite añadirle una restricción.
    ArrayList<T> caja;

    public Caja() {
        //Operador diamond, establece el mismo tipo que declarado en la creación de la variable.
        this.caja = new ArrayList<>();
    }

    //Obtenemos un tipo T que añadimos al ArrayList
    public void add(T element) {
        caja.add(element);
    }

    //Obtenemos un tipo T que eliminaremos de la lista
    public void remove(T element) {
        caja.remove(element);
    }

    //Recorremos la lista mediante un foreach que pintaremos por pantalla.
    public void list() {
        for (T t : caja) {
            System.out.println(t);
        }
    }
}
```




Los comodines respecto a los genéricos nos permite establecer una restricción mayor a nuestras funciones permitiendo a estas recibir datos relacionados con un tipo establecido

# Otras colecciones

## > ArrayDeque

Es una estructura de datos lineal que permite insertar y eliminar elementos por ambos extremos.

Junta en una única estructura las colas y las pilas con sus métodología

- Estructura **LIFO** (colas)
- Estructuras **FIFO** (pilas)

Alguno de sus métodos más importantes son:

push() → Añade un elemento al principio

pop() → Elimina el elemento último elemento

Peek() → Selecciona el último elemento

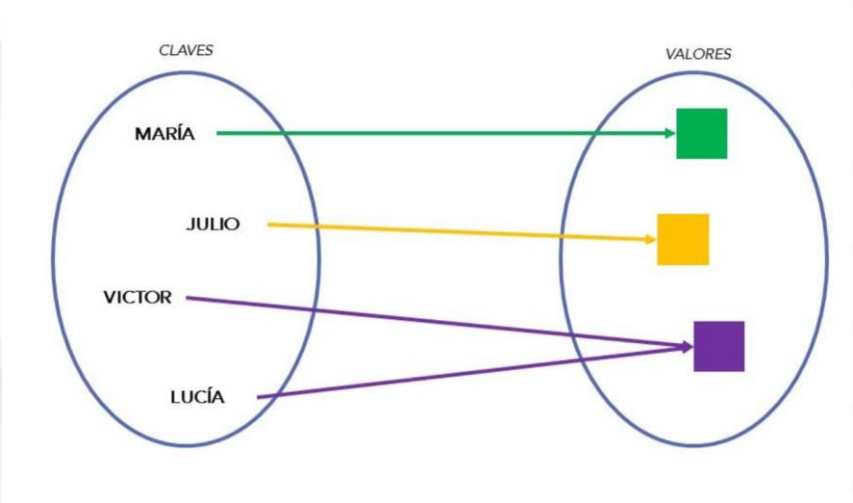
pollFirst() → Elimina el 1r elemento de la cola

pollLast() → Elimina el último elemento de la cola

## > ArrayDeque

```
ArrayDeque<Integer> arrayDeque = new ArrayDeque<>();
arrayDeque.add(1);
arrayDeque.addFirst(0);
System.out.println(arrayDeque.peek());
arrayDeque.push(2);
System.out.println(arrayDeque.toString());
System.out.println(arrayDeque.pollFirst());
System.out.println(arrayDeque.pollLast());
```

Un Map es un conjunto de valores, donde cada uno de los valores tiene asociado una clave mediante el cual se puede realizar la búsqueda.
Suelen ser llamados Map<K,V>



ILERNA Online

Algunos de sus métodos más importantes son:
get(Key) → Obtiene el objeto correspondiente a la Key

put(Key,Object) → Añade un nuevo par K,V, si ya existe un valor para K sobrescribe V.

keySet() → Devuelve todas las claves.

values() → Devuelve todos los valores.

La implementación más utilizada es HashMap();

# Ejercicios para practicar

>

ILERNA Online

1. Crea una nueva ArrayList, de tipo String, añade 5 elementos, posteriormente recupéralos.

2. Crea una nueva ArrayDeque, de tipo Integer, añade 5 elementos (de diferentes formas), posteriormente recupéralos.

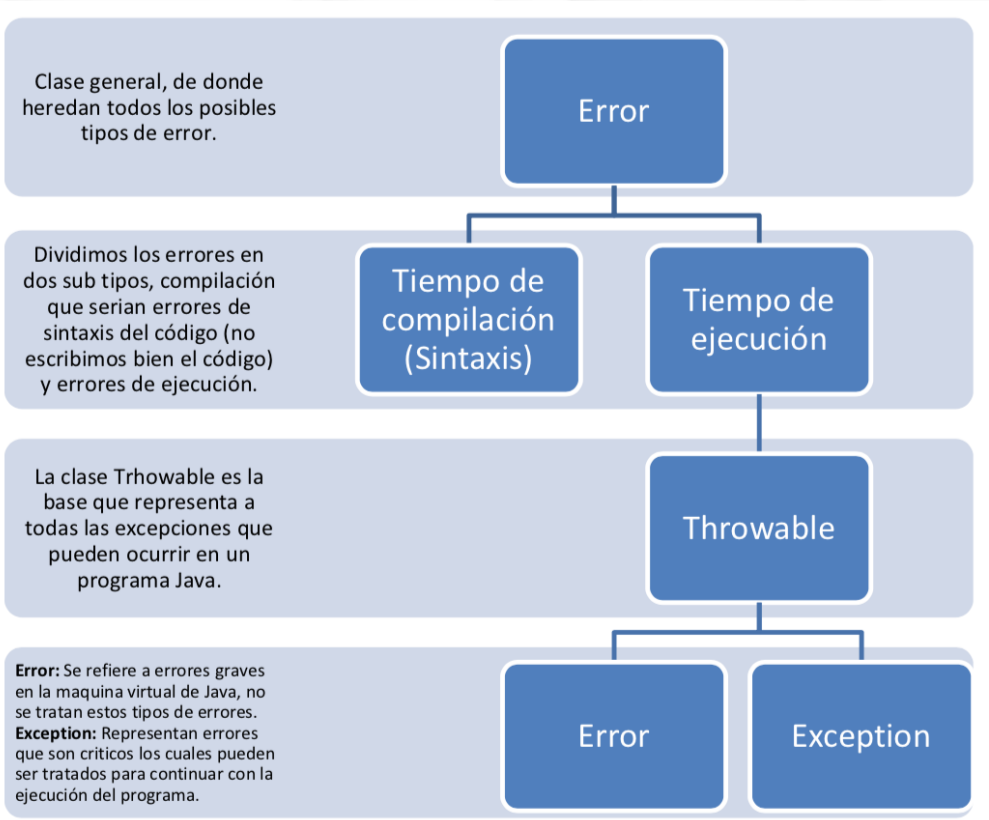
3. Crea un nuevo HashMap e inserta varios elementos, posteriormente recupéralos.

# Exceptions

Estos errores se denominan errores de compilación, mientras que los errores que se muestran durante el tiempo de ejecución se denominan errores de excepción

Cuando un error no se controla el programa termina sin continuar su ejecución

# > Excepciones



```
try{
//Bloque de código que en caso de error lanzara una excepción
//que capturara el bloque catch
}catch (ErrorLoginException error) {
//Código que se ejecutará si el código del bloque try lanza
//una excepción
}finally {
//Código que se va a ejecutar siempre (aunque se entre en //catch)
}
```

Capturaremos la Exception cuando:

- Podemos recuperarnos del error que se ha producido y seguir con la ejecución.
- Queremos registrar un error o mostrarlo (por ejemplo, en un log por consola).
- Queremos lanzar el error, pero con un tipo de excepción distinta (por ejemplo, *MiErrorException*).

ILERNA Online

```
try {
//código que se va ejecutar
throw new Clase_de_error(mensaje);
//lanzamiento de error
} catch (clase_de_error) {
//código que se va a ejecutar si hay error
}
```

## > Excepciones - Ejemplo

```
public static void main(String[] args) {
    int[] ints = new int[5];
    Scanner scanner = new Scanner(System.in);
    int i;
    do {
        System.out.println("Introduce otro número. Para salir -1");
        i = scanner.nextInt();
        try {
            if (i != -1) {
                ints[i] = i;
            }
        } catch (IndexOutOfBoundsException ex) {
            throw new FueraDeIndiceException();
        } finally {
            System.out.println("Número introducido: " + i);
        }
    } while (i != -1);
    System.out.println("Números introducidos:");
    for (i = 0; i < 5; i++) {
        System.out.println(ints[i]);
    }
}
```

Para crear una nueva Exception personalizada debemos seguir los pasos:

1. Crear una nueva clase
2. Extender Exception, RuntimeException u otra descendiente de estas.
3. Creación de los constructores necesarios.

## > Excepciones - Ejemplo

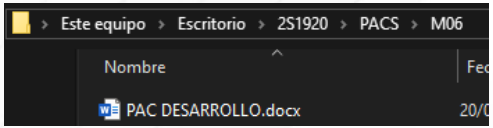
```
public class FueraDeIndiceException extends RuntimeException {
    public FueraDeIndiceException() {
        super("Número introducido incorrecto.\n");
    }
}
```

# Ficheros

# ¿QUÉ SON LOS FICHEROS?

ILERNA Online

- Están formados por bits
- Se guardan como registros
- Identificados por ruta y nombre
- No son volátiles
- Tienen una extensión

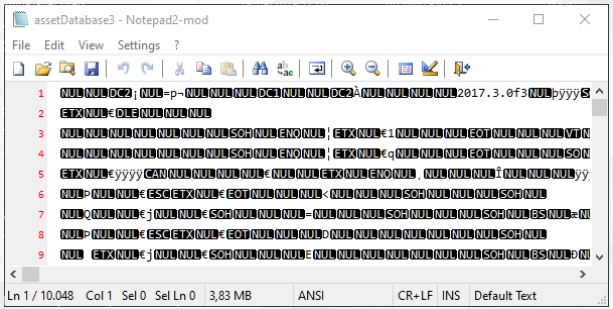


ILERNA Online

# Ficheros

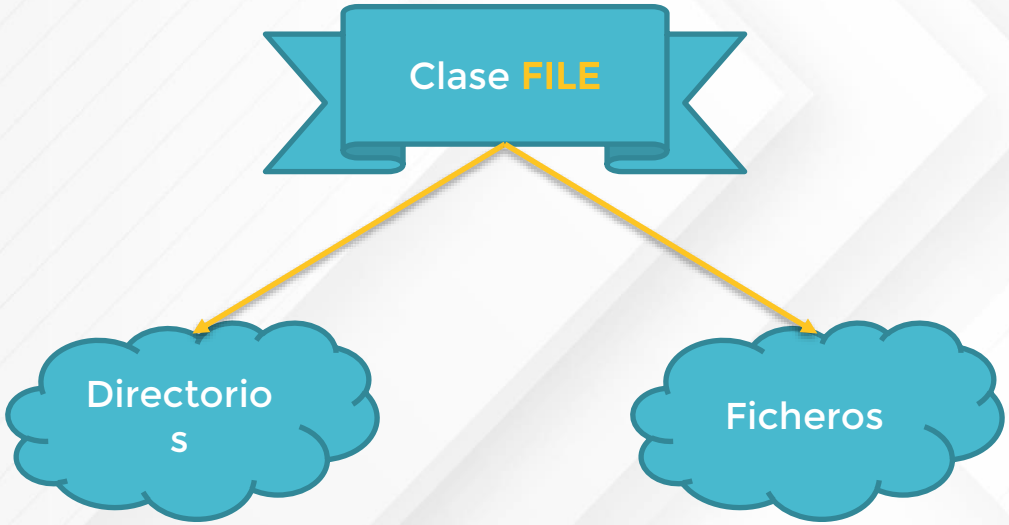
- Caracte (o text
- Binarios (o bytes)

Existen **2 tipos** de ficheros.




Formados por bytes, contienen información generalmente de tipo multimedia

# ¿Cómo trabajamos en Java con ficheros?




La clase File se utiliza para trabajar con ficheros y directorios. Un objeto de esta clase puede representar a cualquiera de los dos elementos

# ¿Cómo se puede crear un objeto File?

ILERNA Online

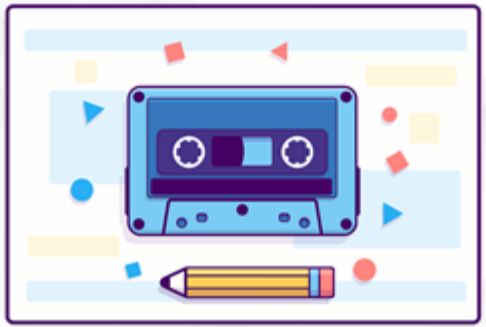
# Constructores

- File (String nombre)
- File (File ruta, String nombre)
- File (String ruta, String nombre)

# Acceder a ficheros

ILERNA Online

El acceso es secuencial → Casete antiguo o VHS.



Data1 → Data2 → Data3

Los datos o registros se leen de forma ordenada.

| | | | |
|---|---|---|---|
| Binario | FileInputStream | Caracteres | FileReader |
| | FileOutputStream | | FileWriter |

# > Ejemplo FileWrite/FileReader

```java
private static void funcion2() throws IOException {
    File fichero1 = new File( pathname: "ficheros/filewriter.txt");
    FileWriter writer = new FileWriter(fichero1);
    writer.write( str: "Esto es un mensaje desde la función2");
    writer.flush();
    writer.close();
}
```

```java
private static void funcion3() throws IOException {
    File fichero1 = new File( pathname: "ficheros/filewriter.txt");
    FileReader reader = new FileReader(fichero1);
    int valor = reader.read();
    while (valor != -1) {
        System.out.println("Char: "+(char)valor+"\t\tascii: "+valor);
        valor = reader.read();
    }
    reader.close();
}
```

| | |
|---|---|
| Char: E | ascii: 69 |
| Char: s | ascii: 115 |
| Char: t | ascii: 116 |
| Char: o | ascii: 111 |
| Char: | ascii: 32 |
| Char: e | ascii: 101 |
| Char: s | ascii: 115 |
| Char: | ascii: 32 |

...

El acceso puede ser a cualquier posición deseada → CD.

Se puede acceder a un dato sin recorrer los datos anteriores.

Aleatorio **RandomAccesFile**

```
private static void randomAccesFile() {
    File fichero1 = new File( pathname: "ficheros/filewriter.txt");
    try {
        RandomAccessFile randomAccess = new RandomAccessFile(fichero1, mode: "rw");

        long pos = fichero1.length();
        randomAccess.seek(pos);
        randomAccess.writeBytes( s: "Linea desde randomAccesFile");
        randomAccess.seek(pos);
        String next;
        while ((next = randomAccess.readLine()) != null) {
            System.out.println(next + ": Tamaño = " + next.length());
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

# Operaciones sobre ficheros

## > ¿Qué operaciones se pueden realizar?

Altas → incluir un nuevo registro al archivo.

Modificaciones → Cambiar una parta del registro.

Bajas→ Dar de baja el registro.

Consultas → Buscar un registro.