

# Clase 4

Genéricos y Excepciones

# Genéricos (Generics)

## **Definición**

Los genéricos permiten definir clases, interfaces y métodos que trabajan con tipos específicos sin tener que especificar el tipo concreto.

Esto permite crear código más genérico y reutilizable.

## Tipos de genéricos

- Clases genéricas
- Métodos genéricos
- Interfaces genéricas

# Ejemplo de clase y métodos genéricos

```
public class ListaGenerica<T> {  
    private ArrayList<T> lista;  
  
    public ListaGenerica() {  
        lista = new ArrayList<T>();  
    }  
  
    public void agregar(T elemento) {  
        lista.add(elemento);  
    }  
  
    public T obtener(int indice) {  
        return lista.get(indice);  
    }  
  
    public int tamaño() {  
        return lista.size();  
    }  
}
```

# Interfaz Genérica

```
public interface MiInterfaz<T> {  
    public void add(T obj);  
    public T get(int index);  
}
```

# Genéricos

Se pueden utilizar más de un genérico en la definición de clase

```
public class MiClase<T, U> {  
    // ...  
}
```

## Herencia de genéricos

```
public class MiClaseHija<T> extends MiClase<T> {  
    // ...  
}
```

Ejemplo de clase genérica que solo acepta clases que implementan la interfaz comparable

```
public class MiClase<T extends Comparable<T>> {  
    // ...  
}
```

# Genéricos

```
public class Producto implements Comparable<Producto> {  
    private String nombre;  
    private double precio;  
  
    public Producto(String nombre, double precio) {  
        this.nombre = nombre;  
        this.precio = precio;  
    }  
  
    public String getNombre () {  
        return nombre;  
    }  
  
    public void setNombre (String nombre) {  
        this.nombre = nombre;  
    }  
  
    public double getPrecio () {  
        return precio;  
    }  
  
    public void setPrecio (double precio) {  
        this.precio = precio;  
    }  
  
    @Override  
    public int compareTo (Producto otroProducto) {  
        if (this.precio < otroProducto.getPrecio ()) {  
            return -1;  
        }  
        if (this.precio > otroProducto.getPrecio ()) {  
            return 1;  
        }  
        return 0;  
    }  
}
```

```
public class Main {  
  
    public static void main (String[] args) {  
        List<Producto> listaProductos = new ArrayList<>();  
        listaProductos.add(new Producto ("Producto 1", 20.0));  
        listaProductos.add(new Producto ("Producto 2", 15.0));  
        listaProductos.add(new Producto ("Producto 3", 30.0));  
        System.out.println("Lista de productos sin ordenar:");  
        for (Producto producto : listaProductos) {  
            System.out.println (producto.getNombre () + " - " +  
                producto.getPrecio ());  
        }  
        Collections.sort (listaProductos);  
        System.out.println("\nLista de productos ordenada por  
precio:");  
        for (Producto producto : listaProductos) {  
            System.out.println (producto.getNombre () + " - " +  
                producto.getPrecio ());  
        }  
    }  
}
```

# Excepciones

## **Definición**

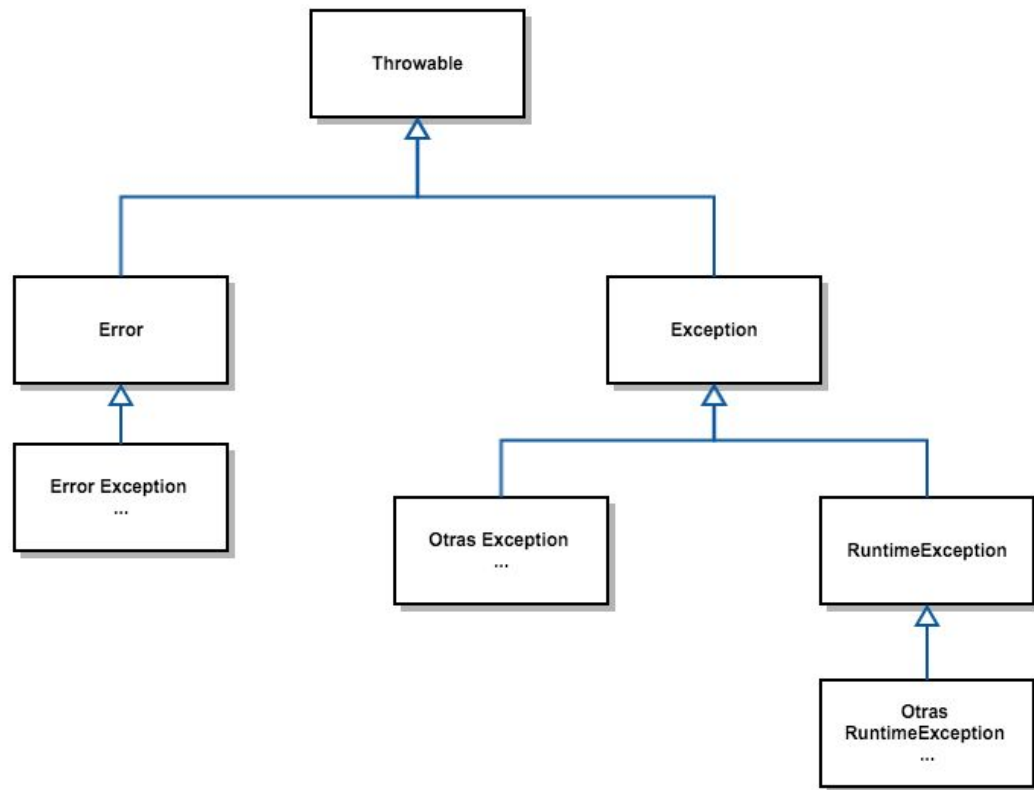
Mecanismo que permite manejar errores de forma eficiente y controlada.

Es una forma de anticiparse a posibles errores.

Al ocurrir un error manejado por excepciones se puede cambiar el curso de acción de la aplicación.

Definidos como clases, herencias

# Excepciones - Jerarquía de herencia





# Excepciones

```
FileReader archivo = new FileReader( fileName: "ruta/al/archivo.txt");
```

```
}
```

Unhandled exception: java.io.FileNotFoundException


```
public FileReader(  
    @NotNull String fileName  
)  
throws java.io.FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read, using the platform's [default charset](#).

Params: `fileName` – the name of the file to read

Throws: [FileNotFoundException](#) – if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

 [java.io.FileReader](#)

 < zulu-17 >

# Excepciones

## Tipos

- Tratadas (manejada).

Debemos manejarlas desde el código como el ejemplo anterior

Bloque try - catch

- No tratadas (no manejada)

No es necesario manejarlas desde el código

# Excepciones tratadas

```
FileReader archivo = new FileReader( fileName: "ruta/al/archivo.txt");
```

```
}
```


Unhandled exception: java.io.FileNotFoundException


```
public FileReader(  
    @NotNull String fileName  
)  
throws java.io.FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read, using the platform's [default charset](#).

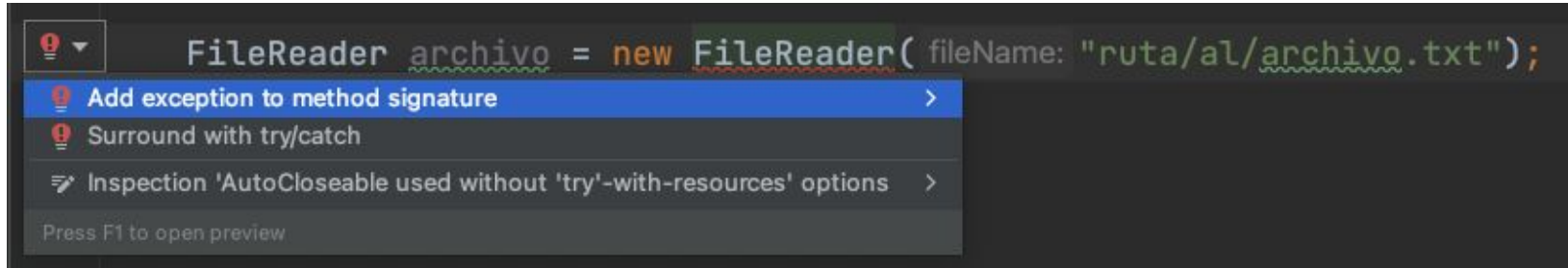
Params: `fileName` – the name of the file to read

Throws: [FileNotFoundException](#) – if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

 [java.io.FileReader](#)

 < zulu-17 >

# Excepciones tratadas



```
66      try {
67          FileReader archivo = new FileReader( fileName: "ruta/al/archivo.txt");
68      } catch (FileNotFoundException e) {
69          throw new RuntimeException(e);
70      }
```

# Excepciones no tratadas

```
67      String dato = "hola";
68      double num1 = Double.parseDouble(dato);
69  }
70  }
71
72
73
```

Main x

```
/Users/Villano/.sdkman/candidates/java/17.0.2-zulu/bin/java -javaagent:/Applications/IntelliJ IDEA
.app/Contents/lib/idea_rt.jar=59620:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
/Users/Villano/Documents/GitHub/UMProgramacion1-2023-ejemplos/programacion2-2023-ejemplos/clase3/clase3/out
/production/clase3 Main
Exception in thread "main" java.lang.NumberFormatException: Create breakpoint : For input string: "hola"
<2 internal lines>
    at java.base/java.lang.Double.parseDouble(Double.java:651)
    at Main.arrancar3(Main.java:68)
    at Main.main(Main.java:12)

Process finished with exit code 1
```

# Excepciones propias

## Definición

Es una clase que extiende de la clase `Exception` o algún hijo en la jerarquía de herencia.

```
// Definición de la clase
public class MiExcepcion extends Exception {

    // Definición de constructores
    public MiExcepcion() {
        super("Texto genérico de excepción")
    }

    public MiExcepcion(String msg) {
        super(msg);
    }
}
```

# Use de excepciones propias

```
public class Ejemplo {

    public void metodo1(Integer valor1) throws MiExcepcion{
        if(valor1>10) {
            throw new MiExcepcion("Esta excepción es porque el valor es mayor a 10");
        }
    }

    public void metodo2(String valor2) throws MiExcepcion{
        if(valor2.equals("texto no permitido")) {
            throw new MiExcepcion("Esta excepción es porque el texto es inválido");
        }
    }

    public void metodo3(String valor2) throws MiExcepcion, Exception{
        if(valor2.equals("texto no permitido")) {
            throw new MiExcepcion("Esta excepción es porque el texto es inválido");
        }
        if(valor2 == null) {
            throw new Exception("Esta excepción es porque el texto es nulo");
        }
    }
}
```

# ¿Qué hacer con las excepciones?

Dos alternativas:

- Tirarlas (throw)

```
public void metodo2() throws MiExcepcion {  
    Ejemplo ejemplo = new Ejemplo();  
    ejemplo.metodo1(11);  
}
```

- Tratarlas (try-catch-finally)

```
public void metodo3() {  
    Ejemplo ejemplo = new Ejemplo();  
    try {  
        ejemplo.metodo3("Texto");  
    } catch (Exception e) {  
        throw new RuntimeException(e);  
    }  
    finally {  
        //Algo mas  
    }  
}
```