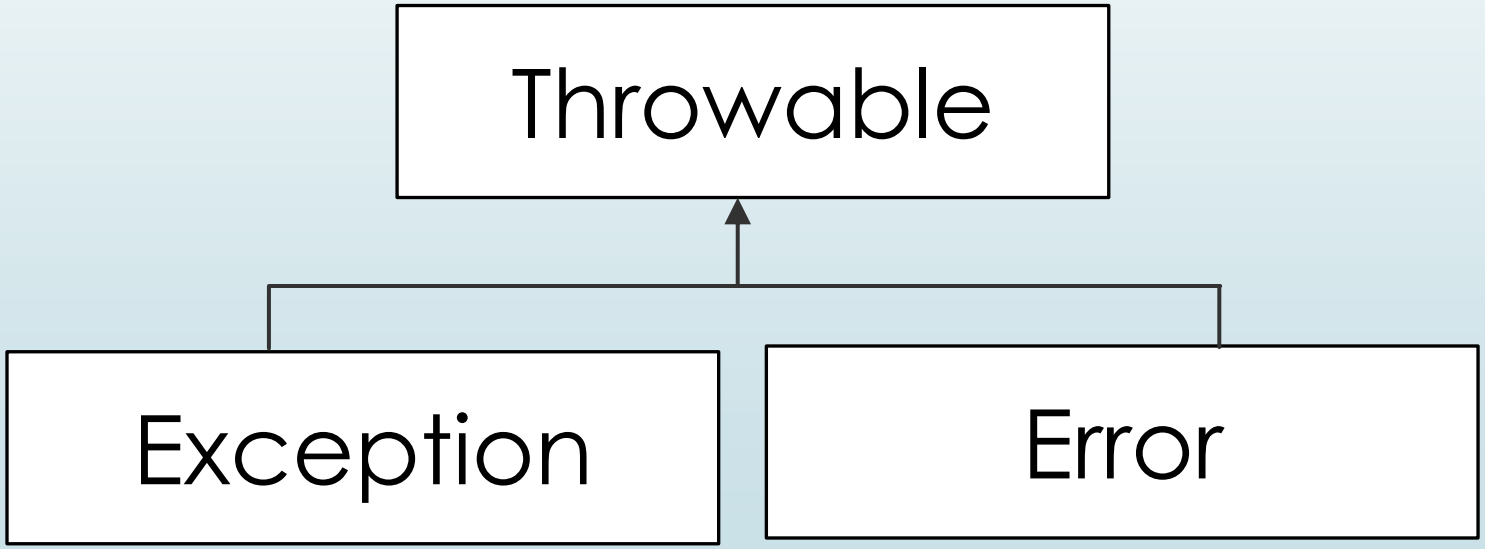
EXCEPCIONES

Throwable

- En Java cuando se produce un error “se lanza” un objeto Throwable



EXCEPCIONES

- Errores en tiempo de ejecución.
- Cuando se producen se muestra un mensaje de error y finaliza el programa.

- Una excepción es un evento que se produce cuando se ejecuta el programa de forma que interrumpe el flujo normal de instrucciones.

EXCEPCIONES

- Excepciones típicas:

- El fichero que se pretende abrir no existe.
- La clase que se pretende utilizar no existe o no está accesible.
- La conexión de red se ha caído.
- Se ha excedido el máximo permitido en el índice de un array.

¿Qué diferencia hay entre un error y una excepción?

- Un **error** generalmente tiene una causa ajena al programa, por ejemplo, OutOfMemoryError, InternalError, etc
- Una **excepción** es algo que debe ser controlado y el programador tiene que predecir que puede ocurrir.

Manejo de excepciones

- El control por excepciones es el sistema para manejar de forma sencilla el control de errores en un programa.

Manejo de excepciones

```
intenta {
        abrir fichero
        leer registro
        mientras hay registros
        procesar registro
        leer registro
        cerrar fichero
}
si (ERROR lectura) {...}
si (ERROR proceso) {...}
si (otro ERROR){...}
```

Manejo de excepciones

- Ventajas:
- El control de errores no interrumpe la lógica del programa.
- Código fuente más fácil de mantener.

Manejo de excepciones

```
try {
.....
} catch (Exception e){
.....
}
[finally {
.......
}]
```

Lógica del programa

Clase a la que pertenece la excepción

Instancia

Tratamiento de la excepción

Siempre se ejecuta

```
try{
    //Instrucciones que se intentan ejecutar, si se produce una
    //situación inesperada se lanza una excepción
}
catch(tipoExcepcion e){
    //Instrucciones para tratar esta excepción
}
catch(otroTipoExcepcion e){
    //Instrucciones para tratar esta excepción
}
    //Se pueden escribir tantos bloques catch como sean necesarios
finally{
    // instrucciones que se ejecutarán siempre después de un bloque try
    // se haya producido o no una excepción
}
```

Excepciones

- **Importa el orden de los bloques catch**
- Las excepciones más genéricas tienen que capturarse al final.

Try-with-resources

- Siempre que estemos trabajando con un recurso que implemente **Closeable** o **Autocloseable** podemos usar el enfoque try with resources
- Es una característica introducida en Java 7 que permite una gestión más sencilla y eficiente de los recursos, como flujos de archivos, sockets
- Cuando se usa try-with-resources el recurso se cierra automáticamente al final del bloque try, lo que ayuda a evitar posibles fugas de recursos y reduce la necesidad de código de cierre explícito en un bloque finally.

Try-with-resources

try **(recurso autocloseable)** {

……

} catch (Exception e){

……

}

- No se necesita bloque finally

- **Implementan Closeable**
- BufferedReader
- BufferedWriter
- FileInputStream
- FileOutputStream
- InputStreamReader
- OutputStreamWriter
- RandomAccessFile
- Socket
- ServerSocket

- **Implementan AutoCloseable**
- java.sql.Connection
- java.sql.ResultSet
- java.sql.Statement
- java.sql.PreparedStatement

- **Implementan ambas** (Closeable extiende AutoCloseable)
- ZipFile
- Scanner

```
public class TryWithResourceDemo{
public static void main(String[]args){
try(BufferedReader reader=new BufferedReader(new FileReader("fichero.txt"))){
Stringline =reader.readLine();
System.out.println(line);
}catch(IOExceptione){
// Vamos a capturar y manejar la excepción
System.out.println("Se ha producido un error de I/O: "+e.getMessage());
}
}
```

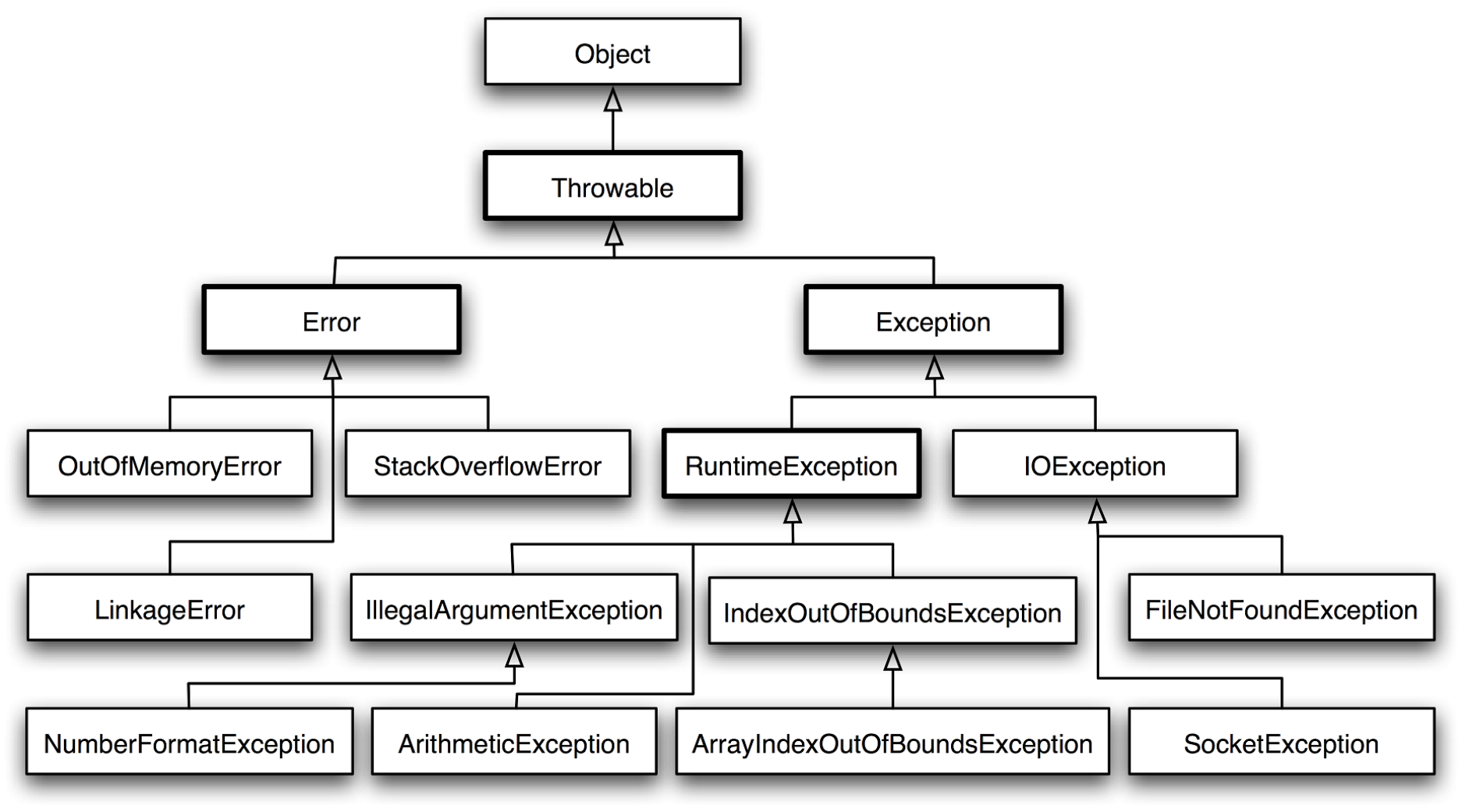
Excepciones típicas

- **ArithmeticException**: Errores aritméticos, como por ejemplo dividir por cero.
- **NullPointerException**: Intento de acceso de una propiedad o de un método de un objeto inexistente.
- **ArrayIndexOutOfBoundException**: Intento de acceso de un array fuera de sus límites.
- **StringIndexOutOfBoundsException**: Intento de acceso a un String a una posición fuera de su contenido.

Jerarquía de excepciones

- Las excepciones son **objetos**.
- Las clases que definen las diferentes excepciones pertenecen a una jerarquía de clases.

Jerarquía de excepciones



Jerarquía de excepciones

- La raíz de todas las excepciones es **Throwable** (desciende directamente de Object)
- Todas las excepciones posteriores heredan de:
- **Error**
  - Todas terminadas en "Error", normalmente irrecuperables, no deben ser tratadas por el programa.
- **Exception**
  - Todas terminadas en "Exception".

Jerarquía de excepciones

- **Excepciones checked:** Objetos de la clase Exception o cualquier otra clase que hereda de ella, excepto si heredan de RuntimeException. **Estamos obligados a tratarlas.**

- **Excepciones Unchecked**: Objetos de la clase RuntimeException o de cualquiera otra clase que herede de ella. **No estamos obligados a tratar.**

Jerarquía de excepciones

- **Excepciones checked:** son condiciones excepcionales del flujo del programa pero que no son debido a un error del propio programa.
- Estamos obligados a tratarlas.
- El compilador obligará a tratarlas bien con bloques **try-catch**(capturarlas) o con **throws** (lanzarlas).
- No son fallos del programador.
- Por ejemplo, IOException.

Jerarquía de excepciones

- **Excepciones uncheked:** extienden de la clase RunTimeException o de la clase Error.
- Son fallos del programador (generalmente)
- Ejemplos de RunTimeException:
- NumberFormatException
- IllegalArgumentException
- NullPointerException

Throws

- Un método puede propagar una excepción, puede provocar una excepción que no maneja él.
- **throws objetoException**

throw

- Java permite al programador lanzar excepciones mediante la palabra reservada **throw**.

- **throw objetoExcepcion**;

- La excepción que se lanza es un objeto, por lo que hay que crearlo como cualquier otro objeto mediante **new**.

Excepciones personalizadas

```
public class MiExcepcion extends Exception{
    private int codigoError;
    public MiExcepcion(int codigoError){
    super();
    this.codigoError=codigoError;
    }
@Override
public String getMessage(){
    String mensaje="";
    switch(codigoError){
    case 111:
    mensaje="Error, el numero esta entre 0 y 10";
    break;
    case 222:
    mensaje="Error, el numero esta entre 11 y 20";
    break;
    }
    return mensaje;
    }
}
```

Excepciones personalizadas

```
public class EjemploMiExcepcion {

public static void main(String[] args) {
    int num;
    try {
        num = 5;
        if (num >= 0 && num <= 10) {
            throw new MiExcepcion(111);
        } else if (num > 10 && num <= 20) {
            throw new MiExcepcion(222);
        }
    } catch (MiExcepcion ex) {
    System.out.println(ex.getMessage());
    }
}
}
```