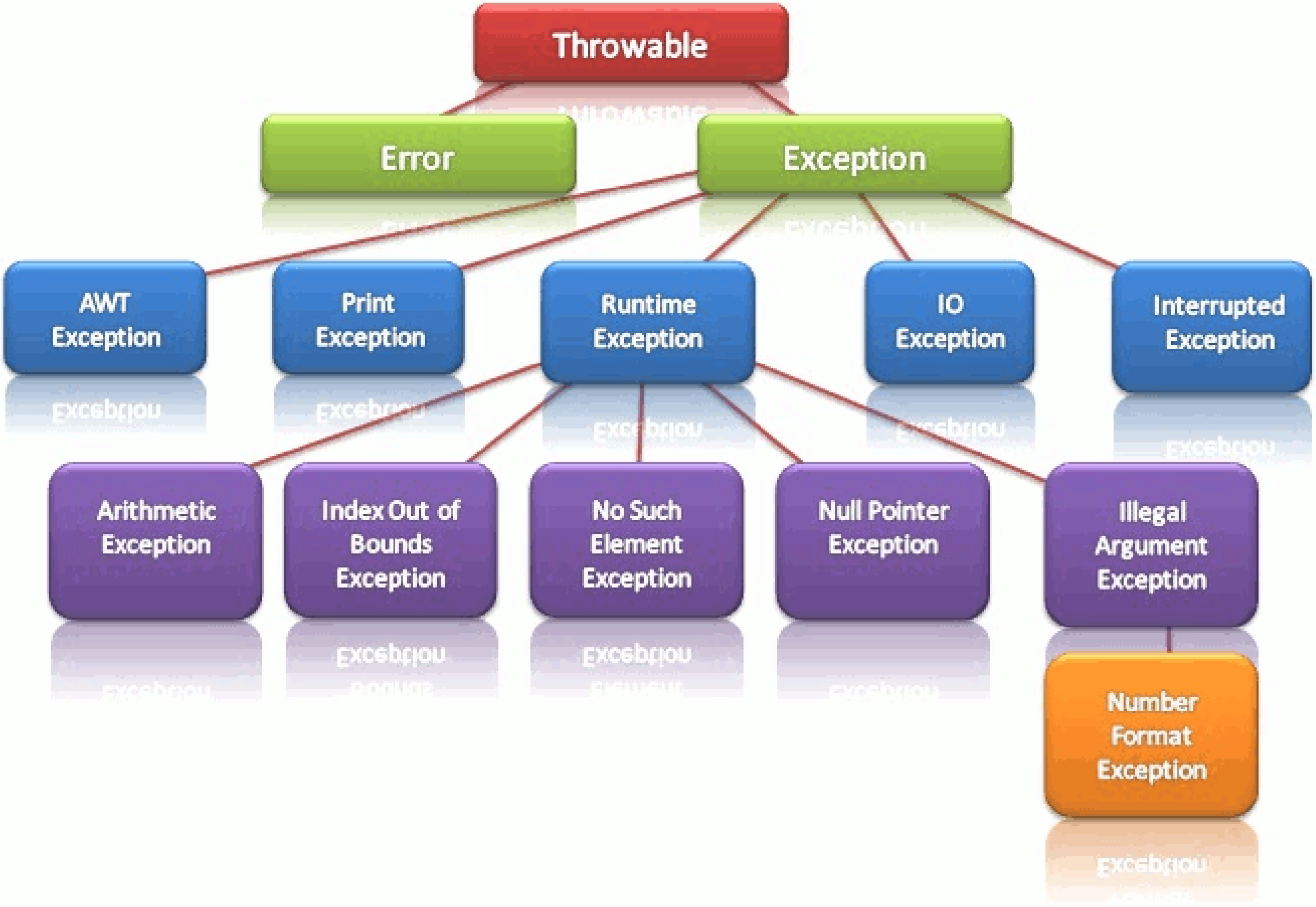
Centro Asociado Palma de Mallorca

# Introducción Práctica de Programación Java


Antonio Rivero Cuesta

# Sesión V

# Excepciones


Throwable
Error
Exception
AWT Exception
Print Exception
Runtime Exception
IO Exception
Interrupted Exception
Arithmetic Exception
Index Out of Bounds Exception
No Such Element Exception
Null Pointer Exception
Illegal Argument Exception
Number Format Exception

Las excepciones son la manera que ofrece Java de manejar los errores en tiempo de ejecución.

Cuando ocurre una situación que es preciso gestionar como excepcional, el código genera una excepción.

Una excepción es cualquier situación que interrumpe el flujo natural de la ejecución de un programa.

# Excepciones de uso Frecuente

**ArithmeticException;** Excepción en una operación aritmética.

**ArrayIndexOutOfBoundsException;** se produce cuando intentas acceder a un elemento del array que está fuera del rango declarado.

**NullPointerException;** Excepción producida por una referencia a un objeto nulo.

**SecurityException;** Excepción en el sistema de seguridad.

**IOException;** Excepción en un proceso de entrada y salida.

**FileNotFoundException;** Excepción por no encontrar un fichero.

**NumberFormatException;** Excepción en la conversión de una cadena de caracteres a un valor numérico.

Java divide las clases de excepciones en dos categorías:

- Excepciones comprobadas.
- Excepciones no comprobadas.

Todas las subclases de la clase estándar de Java ***RunTimeException*** son:

- Excepciones no comprobadas.

Todas las restantes subclases de ***Exception*** son:

- Excepciones comprobadas.

# Excepciones no Comprobadas

Las excepciones no comprobadas están pensadas para aquellos casos que no deben fallar en una operación normal.

Generalmente indican un error en el programa.

# Excepciones Comprobadas

Las excepciones comprobadas están pensadas para aquellos casos en los que el cliente debe esperar que una operación falle

Por ejemplo: cuando grabamos en un disco, sabemos que el disco puede estar lleno.

# La clase Runtime Exception

La clase **`RuntimeException`** (excepción de ejecución) es una subclase de la clase `Exception`.

Las **`RuntimeException`** son errores evitables que pueden prevenirse con solo seguir buenas técnicas de programación.

Por ejemplo, **`IndexOutOfBoundsException`** (excepción de índice fuera de rango), es una subclase de **`RuntimeException`** y es una excepción que Java lanza cuando el programa intenta acceder el índice de un array que está fuera de rango.

Una `RuntimeException` es una bandera roja que indica que debe arreglar el programa.

Típicamente el costo de chequear las excepciones de tiempo de ejecución excede el beneficio de especificarlas o capturarlas por lo tanto es aconsejable no lanzar y capturar este tipo de excepciones.

Las otras subclases de `Exception`, como `IOException` (excepción de entrada/salida), indican condiciones de error que deben atraparse en forma dinámica, ya que no pueden preverse.

Por ejemplo si el usuario intenta abrir un archivo que no existe, Java lanza una:

`FileNotFoundException` (excepción de archivo no encontrado), que el programa debe manejar de forma dinámica.

# Manejo de Excepciones

# Manejo de Excepciones

- Como lanzar una excepción.
- Como atrapar una excepción.
- Estructura general del manejo de excepciones.
- El bloque finally.

# Como lanzar una Excepción

En Java puede considerarse una excepción como un segundo tipo de valor que puede devolver un método.

El primer requerimiento del compilador es que un método que lanza una excepción comprobada debe declarar que lo hace mediante una ***cláusula throws*** que se agrega en su encabezado.

Si bien se permite el uso de la cláusula ***throws*** para las excepciones no comprobadas, el compilador no lo requiere.

Recomendamos que se use una cláusula ***throws*** solamente para enumerar las excepciones comprobadas que lanza un método.

Para devolver una excepción, un método debe crear un objeto de la clase **`Exception`** y utilizar la instrucción **`throw`** (la instrucción **`throw`** es parecida a la instrucción **`return`**).

```
throw new objetoException();
```

El siguiente código muestra cómo crear y lanzar un objeto **`InterruptedException`** (excepción de interrupción):

```
// especifica que el método puede lanzar dos excepciones
void pruebaExcepcion()throws InterruptedException,IOException{
// Instrucciones
// algunError es true si durante la ejecución de las
   instrucciones ocurre algún error
if (algunError)
//crea un objeto de Exception y lanza la excepción
throw (new InterruptedException());
// otroError es true si durante la ejecución de las
   instrucciones ocurre algún error de E/S
else if (otroError)
//crea un objeto de Exception y lanza la excepción
throw (new IOException());
}
```

# Como atrapar una Excepción

Para atrapar (detectar) una excepción, se debe llamar al método que lanza la excepción dentro de un bloque **`try`** y procesar la excepción dentro de un bloque **`catch.`**

# Estructura General del Manejo de Excepciones

```
try {
 //Bloque de sentencias que podrían generar una excepción.
} catch (clase_de_excepcion_1 e){
  //sentencias que se ejecutan si se ha producido una
    excepción de la clase clase_de_excepcion_1.
} catch (clase_de_excepcion_2 e){
  //sentencias que se ejecutan si se ha producido una
    excepción de la clase clase_de_excepcion_2.
} catch (Exception e){
  //sentencias que se ejecutan si se ha producido una
    excepción no capturada anteriormente.
} finally {
  //Bloque de sentencias que se ejecutan siempre.
}
```

# Bloque try/catch/finally

En Java se pueden tratar las excepciones previstas por el programador utilizando unos mecanismos, los ***manejadores de excepciones***.

Se estructuran en tres bloques:

- El bloque **`try`**.
- El bloque **`catch`**.
- El bloque **`finally`**.

# El Bloque try

Define un bloque de código donde se puede generar una excepción.

Lo primero que hay que hacer para que un método sea capaz de tratar una excepción generada por la máquina virtual Java o por el propio programa mediante una instrucción `throw` es encerrar las instrucciones susceptibles de generarla en un bloque `try`.

```
try {
// Código que puede hacer que se eleve la excepción
}
```

Cualquier excepción que se produzca dentro del bloque **`try`** será analizado por el bloque o bloques **`catch`**.

En el momento en que se produzca la excepción, se abandona el bloque **`try`** y las instrucciones que sigan al punto donde se produjo la excepción no serán ejecutadas.

Cada bloque **`try`** debe tener asociado por lo menos un bloque **`catch`**.

# El Bloque catch

```
try {
// Código que puede hacer que se
   eleve la excepción
} catch(TipoExcepción nombreVariable){
// Gestor de la excepción
} catch(TipoExcepción nombreVariable){
// Gestor de la excepción
}
```

Define el bloque de sentencias que se ejecutarán cuando se haya producido una excepción en un bloque **`try`**.

Para declarar el tipo de excepción que es capaz de tratar un bloque **`catch`**, se declara un objeto cuya clase es la clase de la excepción que se desea tratar o una de sus superclases.

# El Bloque finally

El bloque **`finally`** se utiliza para ejecutar un bloque de instrucciones sea cual sea la excepción que se produzca.

Este bloque se ejecutará en cualquier caso, incluso si no se produce ninguna excepción.

Sirve para no tener que repetir código en el bloque **`try`** y en los bloques **`catch`**.

```
try {
 //Bloque de sentencias que podrían generar una excepción.
} catch (clase_de_excepcion_1 e){
  //sentencias que se ejecutan si se ha producido una
    excepción de la clase clase_de_excepcion_1.
} catch (clase_de_excepcion_2 e){
  //sentencias que se ejecutan si se ha producido una
    excepción de la clase clase_de_excepcion_2.
} catch (Exception e){
  //sentencias que se ejecutan si se ha producido una
    excepción no capturada anteriormente.
} finally {
  //Bloque de sentencias que se ejecutan siempre.
}
```

Si la excepción producida no es de la clase del primer catch se investiga el siguiente, y así hasta que algún bloque **`catch`** lo capture.

El último bloque captura cualquier clase, ya que todas las clases de excepciones derivan de la clase **`Exception`**.

```java
import java.io.*;
public class PedirNumero {

    public static void main (String[] args) {
        int numero = -1;
        int intentos = 0;
        String linea;

        BufferedReader teclado = new BufferedReader (new InputStreamReader(System.in));

        do {
            try{
                System.out.print("Introduzca número entre 0 y 100: ");
                linea = teclado.readLine();
                numero = Integer.parseInt(linea);
            }catch(IOException e){
            System.out.print("Introduzca número entre 0 y 100: ");
            }finally{
                intentos++;
            }
        }while(numero < 0 || numero >100);
        System.out.print("El número introducido es: " + numero);
        System.out.print(", Número de intentos: "+ intentos);
    }
}
```