

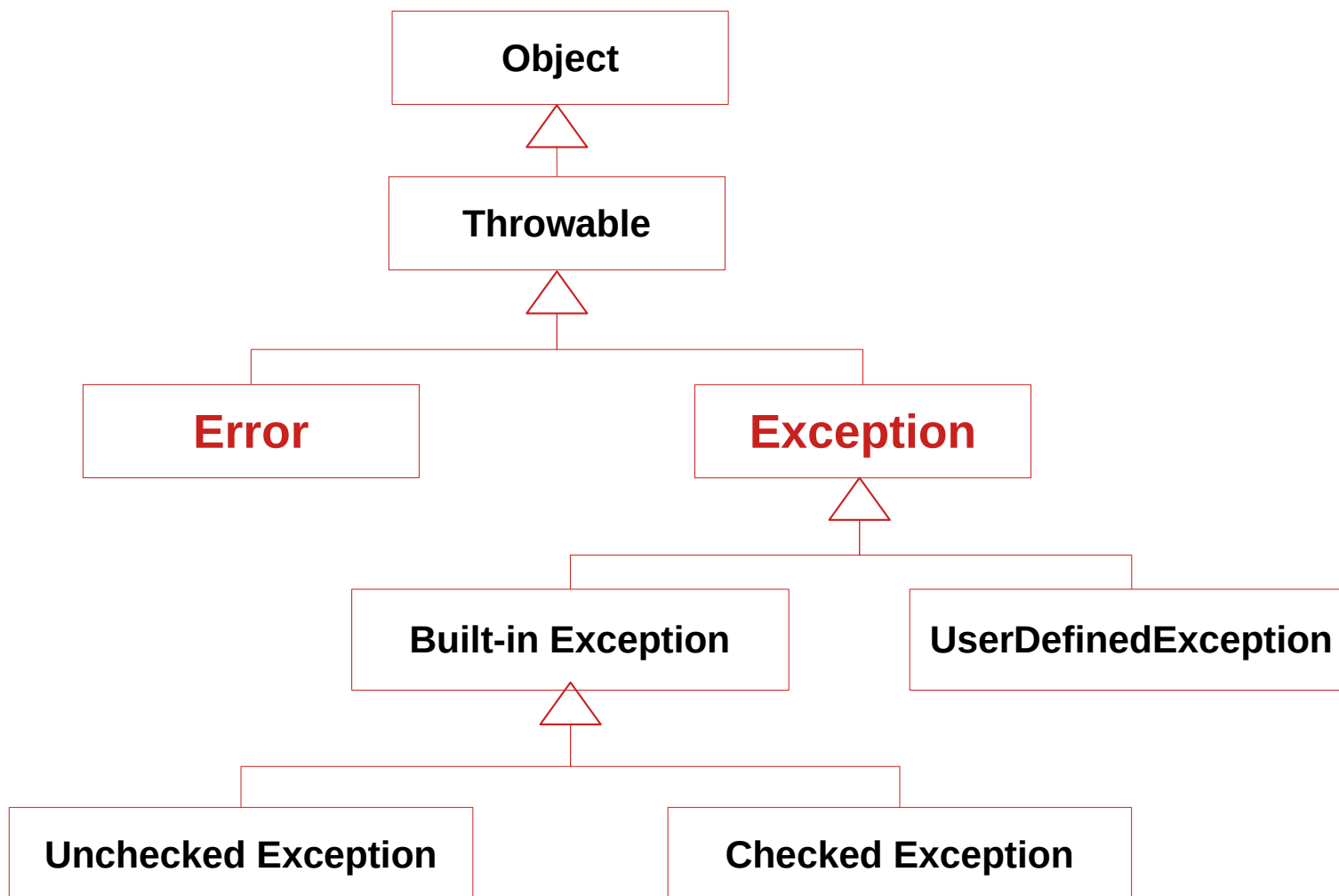
Gestión de Excepciones

Gestión de Excepciones

- Tipos Excepciones
- Capturar Excepciones
- Propagar Excepciones

1. Gestión de Excepciones

- Una **Excepción** es un evento causado por un error inesperado producido en tiempo de ejecución.
- Ejemplo: disco duro lleno, error de red, división por cero, cast inválido, etc.
- Las excepciones o ejecución anómala puede ser de dos tipos: **Error** o **Exception**.
- Ambos heredan de la clase **Throwable** que indica que pueden ser lanzados a clases superiores para tratar de corregir la situación provocada.



1. Gestión de Excepciones

Error:

Errores graves en la máquina virtual de Java (ej: fallos al enlazar con alguna librería).

Normalmente en Java no se tratan estos errores y finalizan la ejecución del programa de forma abrupta.

Exception:

Representan errores que **no son críticos**.

Pueden ser tratados y continuar la ejecución de la aplicación.

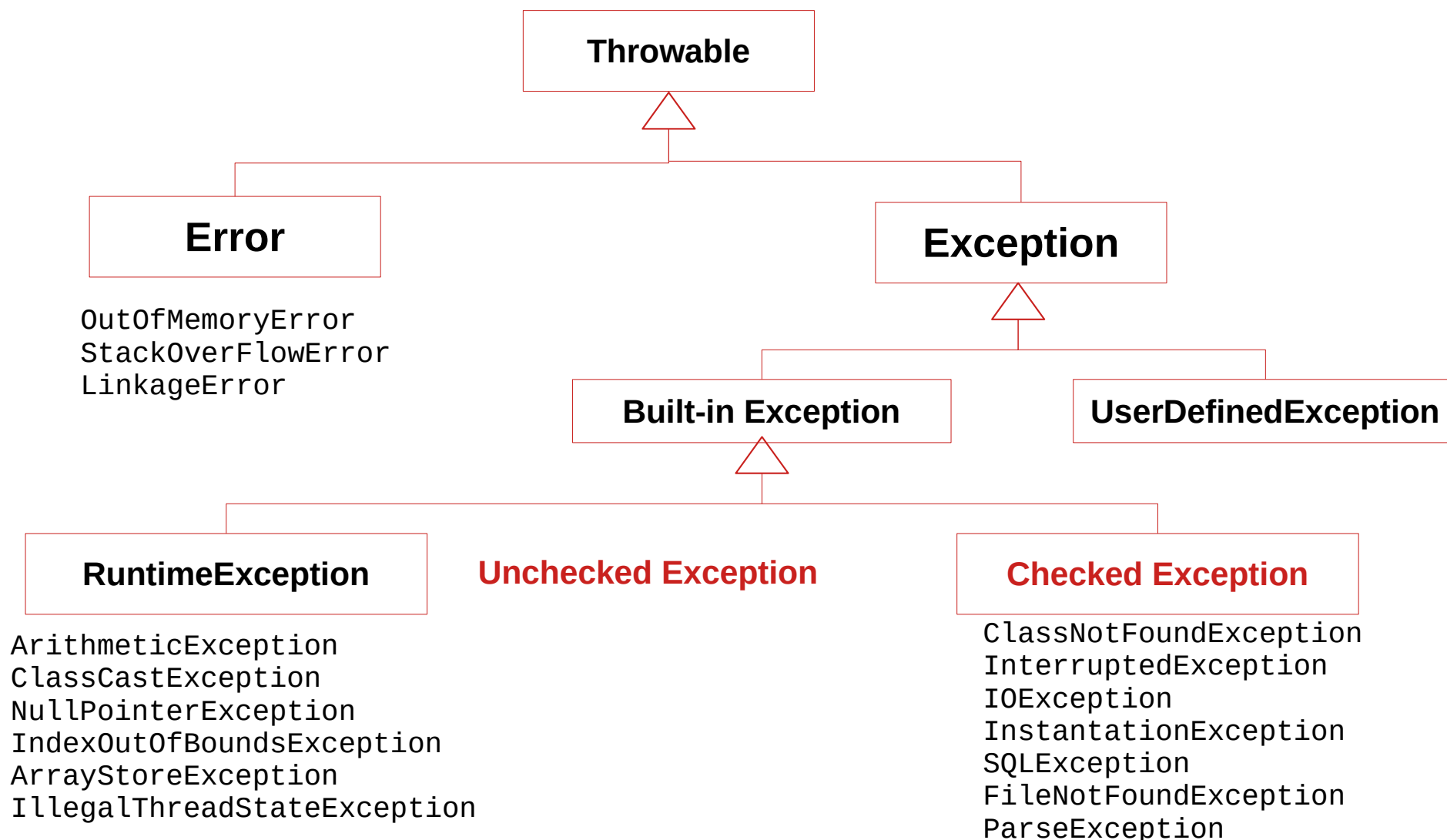
2. Tipos de Excepciones

Dos tipos de excepciones:

Checked: El compilador las detecta y espera que el programador trate la excepción.

Unchecked: Se dan en tiempo de ejecución y no se puede prever.

Las excepciones ***RuntimeException*** son excepciones a errores de código (una referencia a un puntero null, acceder fuera de los límites de un array, no encontrar un fichero, etc). Estas excepciones son unchecked.



2. Tipos de Excepciones

Excepciones:

IndexOutOfBoundsException: esta excepción se produce cuando se intenta acceder fuera de los límites de un array.

NullPointerException: se produce cuando se intenta acceder a un objeto nulo.

NumberFormatException: se produce cuando se intenta convertir una cadena a un número y la cadena no es un número válido.

IllegalArgumentException: pasa cuando se pasa un argumento no válido a un método.

IOException: cuando hay un error de entrada/salida.

2. Tipos de Excepciones

Excepciones:

FileNotFoundException: se produce cuando se intenta acceder a un archivo que no existe.

ClassNotFoundException: se produce cuando se intenta cargar una clase y no se encuentra.

SQLException: cuando hay un error con una base de datos.

NoSuchMethodException: se produce cuando se intenta llamar a un método que no existe en una clase.

OutOfMemoryError: se produce cuando un programa intenta asignar más memoria de la disponible.

3. Gestión de Excepciones

Características:

- Las excepciones provocan un **salto en el flujo** normal del código.
- Permite separar el código para el **tratamiento de errores** del código normal del programa.
- Evitan la terminación abrupta de un programa.

3. Gestión de Excepciones

Tratamiento de las excepciones:

- **Capturar una Excepción:** una Excepción se captura y se trata el error que produce. Se evita que termine la ejecución del programa repentinamente.
- **Propagar la Excepción:** una Excepción se propaga y se delega su gestión a un objeto superior en la jerarquía de herencia.

4. Capturar excepciones

Estructura **try-catch-finally**

- Cuando un fragmento de código pueda lanzar una excepción.
- Tratar un error producido.
- Gestionar una excepción **checked**.

try: Contiene el código que puede producir una excepción/error.

catch: Contiene el código de tratamiento del error, si se produjese. Se puede crear un bloque catch por cada tipo de Excepción. (orden secuencial y se ejecuta la primera)

finally: [no obligatorio] Se ejecuta tras el tratamiento de la excepción. (cerrar ficheros, tablas, desconectar BD, etc.)

4. Capturar excepciones. Ejemplo

```
public class excepcion1 {  
    public static void main(String[] args) {  
        int[] numbers = {1, 2, 3, 4};  
        try {  
            System.out.println(numbers[10]);  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("Ha ocurrido una ArrayIndexOutOfBoundsException");  
        } finally {  
            System.out.println("Este código se ejecutará siempre");  
        }  
    }  
}
```

4. Capturar excepciones. Ejemplo

```
public class excepcion2 {  
    public static void main(String[] args) {  
        String number = "ten";  
        try {  
            int num = Integer.parseInt(number);  
            System.out.println(num);  
        } catch (NumberFormatException e) {  
            System.out.println("Ha ocurrido una NumberFormatException");  
        } finally {  
            System.out.println("Este código se ejecutará siempre");  
        }  
    }  
}
```

4. Capturar excepciones. Ejemplo

```
public class excepcion3 {  
    public static void main(String[] args) {  
        String str = null;  
        try {  
            System.out.println(str.length());  
        } catch (NullPointerException e) {  
            System.out.println("Ha ocurrido una NullPointerException");  
        } finally {  
            System.out.println("Este código se ejecutará siempre");  
        }  
    }  
}
```

4. Capturar excepciones. Ejemplo

```
public class excepcion4 {  
    public static void main(String[] args) {  
        int numerador = 5;  
        int denominador = 0;  
        try {  
            int resultado = numerador / denominador;  
        } catch (ArithmeticException e) {  
            System.out.println("Ha ocurrido una ArithmeticException");  
        }  
    }  
}
```


5. Propagar excepciones

En la cabecera de la función indicamos que puede lanzar la Excepción con **throws**. Y en la implementación de la función lanzamos la excepción al método que lo ha invocado con **throw**, como se muestra en el ejemplo:

```
public void lee_fichero() throws IOException, FileNotFoundException{  
    ...  
    throw new IOException(mensaje_error);  
    ...  
}
```

5. Propagar excepciones. Ejemplo

```
public class excepcion_propagada1 {  
    public static void main(String[] args) {  
        try {  
            funcion1();  
        } catch (Exception e) {  
            System.out.println("Excepción capturada main: " + e.getMessage());  
        }  
    }  
  
    public static void funcion1() throws Exception {  
        funcion2();  
    }  
  
    public static void funcion2() throws Exception {  
        throw new Exception("Excepción generada en la funcion2");  
    }  
}
```

5. Propagar excepciones. Ejemplo

```
public class excepcion_propagada2 {  
    public static void main(String[] args) {  
        try {  
            int result = divide(10, 0);  
            System.out.println("Resultado: " + result);  
        } catch (ArithmeticException e) {  
            System.out.println("Excepción capturada en main: " + e.getMessage());  
        }  
    }  
  
    public static int divide(int nume, int denom) throws ArithmeticException{  
        if (denom == 0) {  
            throw new ArithmeticException("No se puede dividir por cero");  
        }  
        return nume / denom;  
    }  
}
```

6. Ejercicios excepciones

Crea una excepción que genere un mensaje de error si el valor introducido no está incluido en un rango.

Después crea una función rango donde se compruebe si el número introducido está entre 0 y 100. Si no está en el rango, lanza la excepción creada.

En el programa principal, llama a la función rango(50) y rango(150), donde se captura la excepción

5. Ejemplo

```
# Necesitamos crear una excepcion que no existe
public class ExcepcionRango extends Exception {
    public ExcepcionRango(String mensaje) {
        super(mensaje);
    }
}

public class PruebaExcepcionRango {
    public static void rango(int num) throws ExcepcionRango {
        if ((num > 100) || (num < 0)) {
            throw new ExcepcionRango("Número fuera del rango [0,100]");
        }
    }
    public static void main(String[] args) {
        try {
            System.out.println("Probamos con un número 50");
            rango(50);
            System.out.println("Probamos con un número 150");
            rango(150);
        } catch (ExcepcionRango e) {
            System.out.println("Excepcion: " + e.getMessage());
        }
    }
}
```