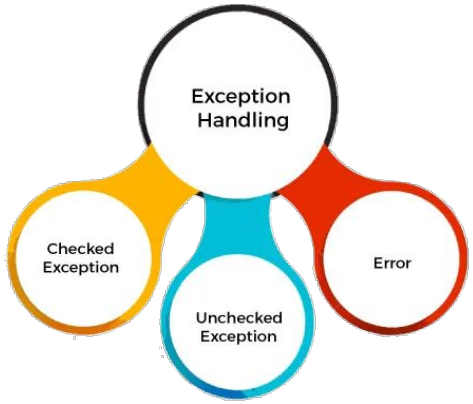


Gestión de excepciones

UNJu - Programación Orientada a Objetos

Ing. José Zapana

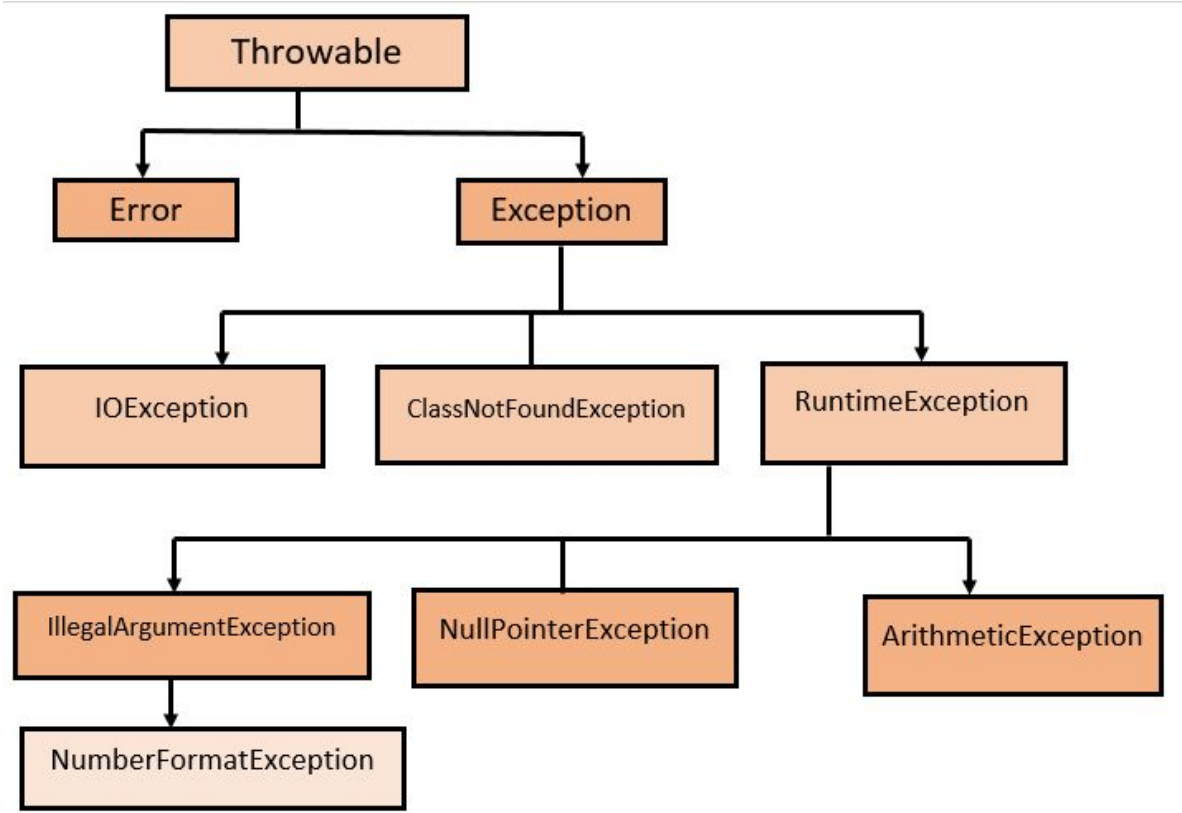
Concepto

- Es una situación anómala que se puede producir durante la ejecución de un programa.
- Se produce debido a fallos de programación o situaciones que escapan del control total del programador (error en la introducción de un dato, corrupción de un fichero, etc).
- Un programa puede recuperarse de una excepción a través de una gestión de excepciones.

Ventajas sobre las excepciones

- Manejo estructurado de errores: legibilidad
- Prevención de comportamientos inesperados: error 500
- Capacidad de recuperación
- Comunicación de errores
- Seguridad y Fiabilidad: El uso adecuado de excepciones puede aumentar la seguridad y la fiabilidad de una aplicación al garantizar que los errores se manejen de manera controlada en lugar de propagarse sin control.

Clases de excepciones



Clasificación

En función de su naturaleza una excepción puede ser:

- **Unchecked:** Conocidas también como excepciones de sistema, son todas las excepciones de subclases de RuntimeException
- **Checked:** Son lanzadas por métodos de la API de Java, específicas de cada clase. Es obligatorio capturarlas

Excepciones Unchecked

- **ArrayIndexOutOfBournException:** intento de acceder fuera de los límites de un array
- **NullPointerException:** Acceso a métodos de un objeto con referencia null
- **SecurityException:** producida por una violación de seguridad
- **ClassCastException:** error de conversión de tipos de datos
  - `Object ob = new String("34");`
  - `Integer nro = (Integer) ob;`
- **ArithmeticException:** operación aritmética incorrecta.
  - `int n = 5 / 0;`
  - `double r = 3 / 0.0;`
- **IllegalArgumentException:** un método recibe como parámetro un valor no válido: `thread.sleep(-100)`

Errores

- A diferencia de una excepción, un error es una situación que se produce en un programa de la que este no se puede recuperar, como fallo en la JVM, falta de espacio en memoria, etc.
- Sin embargo los errores también están por clases que heredan de Error: OutOfMemoryError, StackOverflowError, InternalError, etc

Captura de excepciones

Se capturan a través de los bloques try catch:

```
try{
    //instrucciones
}catch(TipoExcepcion1 ex){
    //tratamiento de la excepción
}catch(TipoExcepcion2 | TipoExcepcion2 ex){
    //tratamiento de las dos excepciones
}
```

- Si los catch tienen excepciones en relación de herencia, las subclases deben ir antes que las superclases
- Si no están en relación de herencia se pueden agrupar en un multicatch mediante l

Métodos de exception

Todas las clases de excepción heredan los siguientes métodos de Exception:

- **String getMessage():** devuelve una cadena de caracteres con un mensaje de error asociado a la excepción
- **void printStackTrace():** genera un volcado de error que es enviado a la consola

Bloque finally

- Se ejecuta siempre se produzca o no la excepción

```
try{
    int n = 4 / 0;
}catch(ArithmeticException ex){
     System.out.println(“Error división por cero”);
     return;
}finally{System.out.println(“Final”)}
```

- Si se produce una excepción y no hay ningún catch para capturarla, se propagara la excepción al punto de llamada, pero antes ejecutará el bloque finally

Propagación de una excepción

- Si un método que debe capturar una excepción y no desea hacerlo, puede propagarla al lugar de la llamada del método
- Se debe declarar la excepción en la cabecera del método con la instrucción throws:

```
void metodo(){
    BufferedReader bf = …;
    try{
        String s = bf.readLine()
    }catch(IOException ex){
    }
}
```

Propagación

```
void metodo()throws IOException{
    BufferedReader bf = …;
    String s = bf.readline();
}
```

Lanzamiento de una excepción

- Desde el método de una clase se puede lanzar una excepción para que sea capturada desde el punto de llamada del método.
- Para lanzar una excepción se utiliza la instrucción **`throw objeto_exception`**

```
void metodo()throws IOException{
    …

    //creación y lanzamiento de la excepción
    throw new IOException();
}
```

Si es RuntimeException no es necesario declararla

Ejemplo

Devolver el valor de un elemento de un vector dado su índice

```
/**
 *
 * @param vector: con elementos enteros
 * @param indice: posición desde donde se quiere obtener un valor
 * @return: elemento del vector en el indice indicado
 * @throws java.lang.IndexOutOfBoundsException
 */
public int getElementoVector(int[]vector, int indice)
                                        throws IndexOutOfBoundsException{

        if(indice < 0)
            throw new IndexOutOfBoundsException("Indice incorrecto");

        return vector[indice];
}
```

Excepciones personalizadas

Se puede crear una excepción personalizada definiendo una clase que herede de Exception

```
class TestException extends Exception{

}
```

```
class C1{
 public void metodo() throws TestException{
        …
        throw TestException();
 }
}
```

```
C1 c = new C1();
try{
    c.metodo();
}catch(TestException ex){
    …
}
```

Ejemplo: Realizar extracciones de una cuenta bancaria

```
public class SaldoInsuficienteException extends Exception{
    public SaldoInsuficienteException(String mensaje){
        super(mensaje);
    }
}
```

```
public class CuentaBancaria(){
     public void retirar(double cantidad)throws SaldoInsuficienteException,
     IllegalArgumentException{
        if(cantidad < 0)
          throw new IllegalArgumentException("La cantidad debe ser mayor a cero");
        if (cantidad > saldo)
          throw new SaldoInsuficienteException("Saldo insuficiente para el retiro");
        saldo -= cantidad;
     }
}
```

Ejemplo (continuación)

```
public void metodo()throws {
     CuentaBancaria cuenta = new CuentaBancaria(1000);
     try{
        cuenta.retirar(500);
        cuenta.retirar(800);
     }catch(SaldoInsuficienteException ex){
        System.err.println(ex.getMessage());
     }catch(IllegalArgumentException ex){
        System.err.println(ex.getMessage());
     }
}
```

Referencias

- Oracle Class Exception
- Java - Exceptions