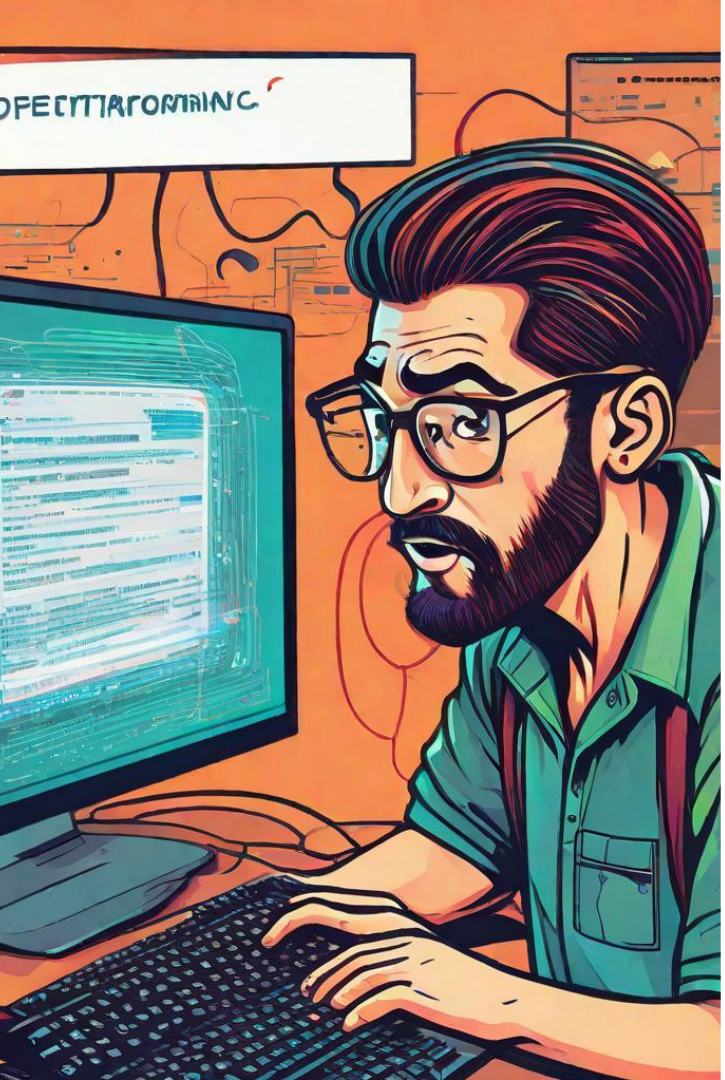
Universidad ICESI

Manejo de excepciones

Esta presentación cubrirá el tema del manejo de excepciones en el lenguaje de programación Java. Discutiremos qué son las excepciones, por qué ocurren y cómo manejarlas de manera efectiva.

Excepciones

- Las excepciones son eventos inesperados que ocurren durante la ejecución de un programa.
- Las excepciones interrumpen el flujo normal de ejecución del programa y pueden llevar a la terminación del programa si no se manejan correctamente.
- Pueden deberse a una variedad de factores, como entradas no válidas, falta de disponibilidad de recursos o errores de programación.



¿Por qué se producen las excepciones?

- Las excepciones se producen cuando se encuentra un error o una condición excepcional durante la ejecución del programa.
- Pueden ser lanzados explícitamente por el programador o automáticamente por el entorno de tiempo de ejecución de Java.

Tipos de excepciones

01 Error: se trata de problemas graves de los que, por lo general, no se pueden recuperar, como OutOfMemoryError o StackOverflowError.

02 Excepciones verificadas: son excepciones que deben declararse en la firma de un método o controlarse mediante un bloque try-catch. Algunos ejemplos son IOException y SQLException. (Compilación)

03 Excepciones no verificadas: no es necesario declarar o detectar explícitamente estas excepciones. A menudo son causados por errores de programación, como acceder a una referencia nula o dividir por cero. (Ejecución)

Jerarquía de excepciones en Java

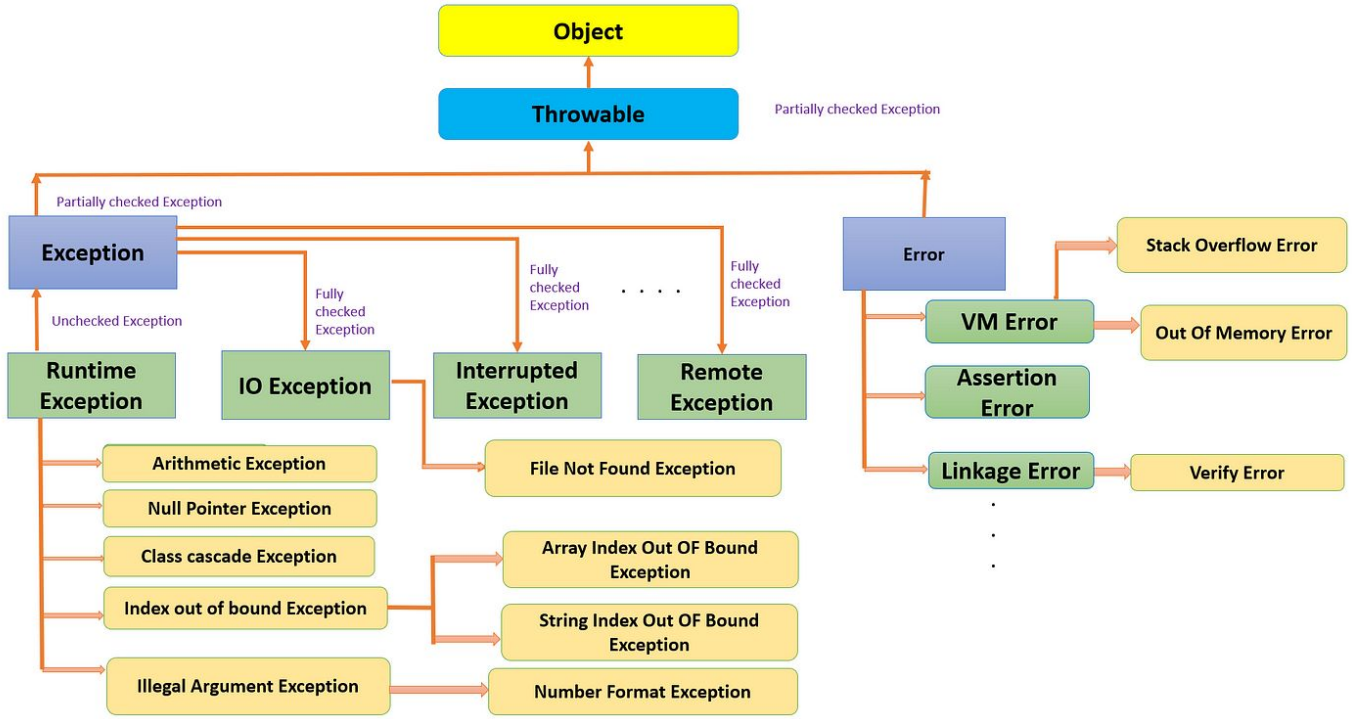


Fig. Exception Hierarchy in Java ~ by Deepti Swain

Control de excepciones

Java proporciona un mecanismo para manejar excepciones mediante bloques try-catch.

El bloque try contiene el código que podría producir una excepción.

El bloque catch detecta y controla la excepción, lo que permite que el programa se recupere o finalice correctamente si es necesario.

Se pueden usar varios bloques catch para controlar diferentes tipos de excepciones.

El bloque finally es opcional y se ejecuta independientemente de si se ha producido una excepción o no.

Ejemplo: Creando un archivo

```
import java.nio.file.Files;
import java.io.IOException;
import java.nio.file.Path;

private Path dataFolder;
private Path dataJson;
private Path dataCsv;

public void inicialize(dataFolder) throws IOException {
        if(!Files.exists(dataFolder)){
            Files.createDirectories(dataFolder);
            if(!Files.exists(dataJson)){
                Files.createFile(dataJson);
            }
            if(!Files.exists(dataCsv)){
                Files.createFile(dataCsv);
            }
        }
    }
```

1. API java.nio (New I/O) (Input/Output)
   - Mejora del sistema de I/O original de Java
   - Mayor flexibilidad en la lectura y escritura de datos.
2. Clases e Interfaces Principales:
   - Path: Representa la ubicación de un archivo o directorio.
   - Paths: Métodos de utilidad para crear instancias de Path.
   - Files: Métodos de utilidad para manipular archivos y directorios.

Ejemplo: Creando un archivo

```
import java.nio.file.Files;
import java.io.IOException;
import java.nio.file.Path;

private Path dataFolder;
private Path dataJson;
private Path dataCsv;

public void inicialize(dataFolder) throws IOException {
        if(!Files.exists(dataFolder)){
            Files.createDirectories(dataFolder);
            if(!Files.exists(dataJson)){
                Files.createFile(dataJson);
            }
            if(!Files.exists(dataCsv)){
                Files.createFile(dataCsv);
            }
        }
    }
```

1. Objetivo: Inicializar una carpeta y archivos (dataJson y dataCsv) si no existen.
2. Funcionamiento:
   - Verifica si dataFolder existe.

     Si no existe:

   - Crea el directorio dataFolder.
   - Verifica y crea los archivos dataJson y dataCsv si no existen.
3. Manejo de Excepciones: Lanza IOException en caso de errores durante la creación.

Ejemplo: Excepción verificada en arreglo

```
// Java Program to handle a java.lang.IndexOutOfBoundsException
import java.util.Arrays;
public class IndexOutOfBoundsException {
    public static void main(String args[])
    {
        int[] array = {1, 2, 3};
        try
        {
            // Attempt to access an element
            // outside the bounds of the array
            int value = array[3];  //  throw IndexOutOfBoundsException
            System.out.println("Value: " + value);
        }
        catch (IndexOutOfBoundsException e)
        {
            // handle the exception and prints the message
            System.out.println("IndexOutOfBoundsException: " +
            e.getMessage());
        }
    }
}
```

1. Objetivo: Demostrar cómo manejar una excepción de tipo IndexOutOfBoundsException.
2. Funcionamiento:
   - Se define un arreglo de 3 elementos: {1, 2, 3}.
   - Se intenta acceder al índice 3 (fuera de los límites del arreglo).
   - Esto lanza una excepción IndexOutOfBoundsException.
   - El bloque catch captura la excepción y muestra un mensaje descriptivo.
3. Manejo de Excepciones:
   - Usa un bloque try-catch para capturar y manejar la excepción.

Recomendación: NO LAS ATRAPES TODAS

Pokémon Exception Handling

For when you just Gotta Catch 'Em All.

```
try {
}
catch (Exception ex) {
// Gotcha!
}
```

Procedimientos recomendados para el control de excepciones

Utilice mensajes de error significativos para proporcionar información útil al usuario y ayudar en la solución de problemas.

Registre las excepciones para ayudar a diagnosticar y depurar problemas.

Detecte excepciones específicas en lugar de generales para controlar adecuadamente los diferentes tipos de errores.

Evite capturar con la clase genérica Exception, ya que puede enmascarar y ocultar detalles importantes sobre el error.

Hoja de trabajo

https://docs.google.com/document/d/1GrG7jfQ7swJkHEheqNEgfr-POwsEyL3CNz-Jre9UIZ0/edit?usp=sharing

Gracias por su tiempo y atención 🙂