Material descargable

egg

Excepciones

Excepciones

Las excepciones son eventos que ocurren durante la ejecución de un programa y pueden interrumpir su flujo normal. En Java, existen dos tipos principales de excepciones: marcadas y no marcadas.

Excepciones Marcadas:

Las excepciones marcadas deben ser manejadas explícitamente en el código. **Estas excepciones son previsibles y se espera que ocurran durante la ejecución del programa**. Algunos ejemplos comunes son IOException, ClassNotFoundException y SQLException. El compilador de Java exige que estas excepciones sean declaradas en el código, ya que son conocidas y potencialmente pueden interrumpir el flujo normal del programa. Esto garantiza que el programador las maneje de manera adecuada y tome las medidas necesarias para mitigar su impacto. Vemos algunos ejemplos de implementaciones:

1. **IOException:**
    - Descripción: Se lanza durante operaciones de entrada y salida fallidas.
    - Ejemplo:

```
 try {
        BufferedReader lector = new BufferedReader(new
FileReader("ruta/al/archivo"));
    } catch (IOException e) {
        System.out.println("Ocurrió un error al leer el archivo");
    }
```

2. **ClassNotFoundException:**
    - Descripción: Se lanza cuando intentamos cargar una clase que no se encuentra.
    - Ejemplo:

```
   try {
        Class.forName("ClaseNoExistente");
    } catch (ClassNotFoundException e) {
        System.out.println("La clase no existe");
    }
```

3. **FileNotFoundException:**

- Descripción: Se lanza cuando se intenta acceder a un archivo que no existe.
- Ejemplo:

```
try {
     new FileInputStream("archivo_no_existente.txt");
 } catch (FileNotFoundException e) {
     System.out.println("Archivo no encontrado");
 }
```

4. SQLException:
   - Descripción: Se lanza cuando ocurre un error en una operación de Base de Datos.
   - Ejemplo:

```
try {
     // Supongamos que tienes un objeto de conexión "conexion"
     conexion.executeQuery("SELECT * from tabla_no_existente");
 } catch (SQLException e) {
     System.out.println("Error en la consulta SQL");
 }
```

Excepciones No Marcadas:

Son conocidas como Runtime Exceptions, estas excepciones son problemas que ocurren debido a errores en el código y son consideradas bugs. Java no obliga a capturarlas ya que asume que se escribirá un código que evite estas situaciones. Algunos ejemplos son **NullPointerException, ArrayIndexOutOfBoundsException y ArithmeticException.**

Las relacionadas con la lógica de negocio creada por el programador, no pueden ser conocidas por Java durante la compilación. Por lo tanto, es responsabilidad del programador anticipar y manejar estas excepciones de manera adecuada en el código. Algunas de ellas son:

1. **NullPointerException**: Se lanza cuando se intenta acceder a un miembro de un objeto nulo.

```
String str = null;
try {
    System.out.println(str.length());
} catch (NullPointerException e) {
    System.out.println("El objeto es nulo");
}
```

2. **ArrayIndexOutOfBoundsException**: Se lanza cuando se intenta acceder a un índice de un arreglo que no existe.

```
int[] arr = new int[5];
try {
    int numero = arr[10];
} catch (ArrayIndexOutOfBoundsException e) {
     System.out.println("El índice está fuera de los límites del
arreglo");
}
```

3. **ArithmeticException**: Se lanza cuando se intenta realizar una operación aritmética ilegal, como dividir por cero.

```
try {
    int resultado = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("División por cero");
}
```

4. **IllegalArgumentException**: Se lanza cuando se pasa un argumento ilegal o inapropiado a un método.

```
try {
    Thread.sleep(-1000);
} catch (IllegalArgumentException e) {
    System.out.println("Argumento ilegal");
}
```

5. **NumberFormatException**: Se lanza cuando se intenta convertir un String a un tipo numérico, pero el formato del String no es apropiado para la conversión.

```
try {
    int num = Integer.parseInt("no_es_un_numero");
} catch (NumberFormatException e) {
    System.out.println("Formato de número inválido");
}
```

6. **ClassCastException**: Se lanza cuando se intenta hacer un casting a un tipo de dato que no es el correcto.

```
Object x = new Integer(0);
try {
    System.out.println((String)x);
} catch (ClassCastException e) {
    System.out.println("No se puede convertir a String");
}
```

7. **NegativeArraySizeException**: Se lanza cuando se intenta crear un array con un tamaño negativo.

```
try {
    int[] arr = new int[-1];
} catch (NegativeArraySizeException e) {
       System.out.println("El tamaño del arreglo no puede ser
negativo");
}
```

8. **StringIndexOutOfBoundsException**: Se lanza cuando se intenta acceder a un índice de un String que no existe.

```
String str = "Hola";
try {
    char ch = str.charAt(10);
} catch (StringIndexOutOfBoundsException e) {
        System.out.println("Índice fuera de los límites del
String");
}
```

Excepciones personalizadas

Puedes crear tus propias excepciones personalizadas extendiendo una de las clases en la jerarquía de excepciones. Esto permite un manejo más específico y significativo de errores.

```
public class MiExcepcionPersonalizada extends Exception {
    public MiExcepcionPersonalizada() {
        super();
    }

    public MiExcepcionPersonalizada(String mensaje) {
        super(mensaje);
    }

    public MiExcepcionPersonalizada(String mensaje, Throwable causa) {
        super(mensaje, causa);
    }

}
```

Bloque *finally* y *try with resources*

El bloque *finally* se utiliza para ejecutar código importante como la limpieza de recursos, no importa si una excepción fue lanzada o no. Siempre se ejecuta.

```
public class Main {
   public static void main(String[] args) {
        try {
            // Código que puede lanzar una excepción
        } catch (Exception e) {
```

```
            // Manejar la excepción
        } finally {
            // Limpiar recursos
        }
    }
}
```

Palabras clave throw y throws

La utilización de las palabras clave throw y throws es fundamental para el manejo de excepciones en Java. Veamos cada una de ellas por separado:

throw:

- **throw** se utiliza para lanzar una excepción explícitamente en el código.
- Se emplea dentro del cuerpo de un método junto con el operador **new**, permitiendo lanzar tanto instancias de excepciones existentes como nuevas.
- Por ejemplo:

```
    public void metodoA() throws MiExcepcion {
        if (condicion) {
            throw new MiExcepcion("Mensaje de error");
        }
    }
```

throws:

- **throws** se utiliza en la firma de un método para declarar que este método puede lanzar una excepción específica.
- Indica que el método puede propagar la excepción al método que lo llama, por lo que el método invocador debe manejarla o declararla a su vez.
- Por ejemplo:

```
  public void metodoB() throws MiExcepcion {
        // Código que puede lanzar MiExcepcion
    }
```

En resumen, throw se utiliza para lanzar excepciones dentro de un método, mientras que throws se utiliza en la firma del método para indicar qué excepciones pueden ser lanzadas por dicho método y deben ser manejadas adecuadamente por el código que lo llama.

Las excepciones y la herencia

En Java, todas las excepciones son subclases de la clase java.lang.Throwable, que tiene dos subclases principales: Error y Exception.

Clase Throwable:

- **Atributos**:
  - **detailMessage**: Es el mensaje que describe por qué se produjo el lanzamiento.
- **Métodos principales:**
  - **getMessage():** Se utiliza para obtener el detailMessage.
  - **printStackTrace():** Este método imprime en la consola la pila de llamadas que llevó a la excepción. Es útil para depurar.
  - **getCause():** Retorna la causa de la excepción, el Throwable que causó esta excepción. Puede ser null si la causa no se conoce o no se inicializó.
  - g**etStackTrace():** Retorna un array que contiene cada elemento en la pila de llamadas.

Clase Error

La clase Error es utilizada por la JVM para indicar errores serios que no deberían intentar ser manejados por la aplicación. A menudo son errores que suceden a nivel de la máquina virtual de Java y están más allá del control del código que estamos escribiendo.

- **OutOfMemoryError**: Se lanza cuando la JVM no puede asignar más memoria para un nuevo objeto debido a la falta de espacio de memoria. Ejemplo de cómo puede ocurrir:

```
int[] matriz = new int[Integer.MAX_VALUE];
```

- **StackOverflowError**: Se lanza cuando una aplicación recursiva consume todo el espacio de la pila de llamadas. Ejemplo de cómo puede ocurrir:

```
void funcionRecursiva() {
    funcionRecursiva();
}
```

Clase Exception

Esta es la clase que normalmente se utiliza para manejar errores en nuestro código. Las subclases de Exception pueden ser Excepciones Marcadas (Checked Exceptions) o no Marcadas (Unchecked Exceptions), y hablaré más sobre ellas en un momento.

En el siguiente **enlace** encontrarás la documentación oficial de la clase Exception, donde podrás acceder a información detallada sobre su uso y funcionalidades.