Unidad 12

Control y manejo de excepciones

Contenidos

Parte 1: Control y Manejo de Excepciones

El manejo de excepciones es una característica fundamental de Java que permite gestionar errores y condiciones anómalas de forma controlada. Esto mejora la robustez y estabilidad de los programas al evitar fallos inesperados durante la ejecución.

1. Excepciones: Concepto

1. ¿Qué es una Excepción?
- Una **excepción** es un evento que interrumpe el flujo normal de ejecución de un programa debido a un error o condición inesperada.
- Se genera cuando el programa encuentra un problema, como dividir por cero, acceder a índices fuera de rango o intentar abrir un archivo inexistente.

2. Características Principales:
- Las excepciones son **objetos** que representan un error o un estado excepcional.
- Java proporciona un mecanismo estructurado para **capturar, manejar y recuperar** de estos errores.

3. Manejo de Excepciones en Java:
- Java utiliza las palabras clave `try`, `catch`, `finally`, y `throw` para manejar excepciones.

1.1. Ejemplo Básico de Manejo de Excepciones

```
public class Main {
    public static void main(String[] args) {
        try {
            int resultado = 10 / 0; // Esto genera una excepción
        } catch (ArithmeticException e) {
            System.out.println("Error: División por cero.");
        } finally {
            System.out.println("Bloque finally ejecutado.");
        }
    }
}
```

Salida:

```
Error: División por cero.
Bloque finally ejecutado.
```

1.2. Flujo de Control en Excepciones

Bloque	Descripción
`try`	Contiene el código que puede generar una excepción.
`catch`	Captura y maneja la excepción generada en el bloque `try`.
`finally`	Contiene código que se ejecuta siempre, independientemente de si hubo excepción.

`throw`	Lanza manualmente una excepción en el programa.

2. Jerarquías de Excepciones

Java organiza las excepciones en una jerarquía basada en clases. Todas las excepciones derivan de la clase base `Throwable`.

2.1. Diagrama Simplificado de la Jerarquía de Excepciones

```
Throwable
|
|---Error
|  |__ Ejemplos: OutOfMemoryError, StackOverflowError
|
|- Exception
    |-- RuntimeException
    |   |-- Ejemplos: NullPointerException, ArithmeticException
    |-- Checked Exceptions
          |__ Ejemplos: IOException, SQLException
```

2.2. Tipos de Excepciones

1. Errores (`Error`):
- Representan problemas graves relacionados con el entorno del programa.
- No deben manejarse directamente en la mayoría de los casos.
- Ejemplo: `OutOfMemoryError`.

```
throw new OutOfMemoryError("No hay suficiente memoria.");
```

2. Excepciones (`Exception`):
- Representan problemas que pueden ser manejados y recuperados.
- Se dividen en:
  - **Excepciones Verificadas (`Checked Exceptions`):**
    - Son verificadas en tiempo de compilación.
    - Ejemplo: `IOException`.

```
try {
    FileReader file = new FileReader("archivo.txt");
} catch (IOException e) {
    System.out.println("Archivo no encontrado.");
}
```

  - **Excepciones No Verificadas (`Unchecked Exceptions`):**
    - Son errores en tiempo de ejecución.
    - Ejemplo: `NullPointerException`, `ArithmeticException`.

```
                String texto = null;
                System.out.println(texto.length()); // Lanza NullPointerExcep-
tion
```

2.3. Captura de Excepciones Múltiples

Es posible capturar diferentes tipos de excepciones utilizando varios bloques `catch`.

```
  public class Main {
      public static void main(String[] args) {
            try {
                int[] numeros = {1, 2, 3};
                System.out.println(numeros[5]); // Genera ArrayIndexOutOf-
BoundsException
            } catch (ArithmeticException e) {
                System.out.println("Error aritmético.");
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Índice fuera de rango.");
            } catch (Exception e) {
                System.out.println("Ocurrió un error.");
            }
      }
  }
```

Salida:

```
  Índice fuera de rango.
```

2.4. Relación Entre Excepciones

Cuando se manejan excepciones, es importante capturarlas desde las más específicas a las más generales. Esto asegura que el bloque correcto maneje la excepción.

Ejemplo de Captura en Orden Correcto

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

Salida:

```
Error específico: División por cero.
```

Ejemplo de Captura en Orden Incorrecto

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

Salida:

```
Error general.
```

El segundo bloque `catch` nunca se ejecutará porque `Exception` captura todos los errores, incluyendo `ArithmeticException`.

3. Ejercicio Guiado

Problema

1. Diseña un programa que:

- Pida al usuario que introduzca dos números.
- Divida el primer número por el segundo.
- Maneje las siguientes excepciones:
  - `ArithmeticException` si el segundo número es cero.
  - `InputMismatchException` si el usuario introduce un valor no numérico.
  - `Exception` para cualquier otro error.
- Siempre imprima un mensaje en el bloque `finally`.

4. Preguntas de Evaluación

1. ¿Qué bloque se utiliza para manejar excepciones específicas?
- a) `try`.
- b) `catch`.
- c) `finally`.
- d) `throw`.

2. ¿Qué ocurre si una excepción no se maneja en un programa?
- a) Se genera un error de compilación.
- b) El programa continúa normalmente.
- c) El programa termina abruptamente.
- d) La excepción se ignora.

3. ¿Cuál es la jerarquía principal de excepciones en Java?
- a) `Exception` -> `Throwable` -> `Error`.
- b) `Throwable` -> `Error` -> `Exception`.
- c) `Error` -> `Exception` -> `Throwable`.
- d) `Exception` -> `Error` -> `Throwable`.

4. ¿Qué tipo de excepción es `NullPointerException`?
- a) Verificada (`Checked`).
- b) No verificada (`Unchecked`).
- c) Error crítico (`Critical Error`).
- d) Ninguna de las anteriores.

Parte 2: Manejo de Excepciones Personalizadas y Uso de `throw` y `throws`

Además de utilizar las excepciones estándar de Java, podemos crear excepciones personalizadas para manejar situaciones específicas en nuestras aplicaciones. Además, las palabras clave `throw` y `throws` permiten controlar explícitamente la generación y propagación de excepciones.

1. Excepciones Personalizadas

1.1. ¿Qué son las Excepciones Personalizadas?

Las **excepciones personalizadas** son clases definidas por el usuario que extienden la clase base `Exception` o `RuntimeException`. Estas excepciones permiten modelar errores específicos de una aplicación, proporcionando más claridad y control sobre el manejo de errores.

1.2. Creación de Excepciones Personalizadas

1. Extender `Exception`:
- Usar cuando se requiere que la excepción sea verificada (checked).
- El compilador obliga a manejarla o declararla con `throws`.

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

2. Extender `RuntimeException`:
- Usar cuando se requiere una excepción no verificada (unchecked).
- El compilador no obliga a manejarla.

```
public class MiExcepcionRuntime extends RuntimeException {
    public MiExcepcionRuntime(String mensaje) {
        super(mensaje);
    }
}
```

1.3. Uso de Excepciones Personalizadas

Ejemplo con Excepción Verificada

```
public class Main {
    public static void verificarEdad(int edad) throws MiExcepcion {
        if (edad < 18) {
            throw new MiExcepcion("La edad mínima es 18 años.");
        }
    }
    public static void main(String[] args) {
        try {
            verificarEdad(16);
        } catch (MiExcepcion e) {
            System.out.println("Excepción capturada: " + e.getMessage());
        }
    }
}
```

Salida:

```
Excepción capturada: La edad mínima es 18 años.
```

Ejemplo con Excepción No Verificada

```
public class Main {
    public static void verificarSaldo(double saldo) {
        if (saldo < 0) {
            throw new MiExcepcionRuntime("El saldo no puede ser negati-
vo.");
        }
    }
    public static void main(String[] args) {
        verificarSaldo(-50); // Lanza excepción sin necesidad de try-catch
    }
}
```

Salida:

```
Exception in thread "main" MiExcepcionRuntime: El saldo no puede ser negativo.
```

2. Uso de `throw` y `throws`

2.1. Palabra Clave `throw`

La palabra clave `throw` se utiliza para lanzar explícitamente una excepción en un programa.

Ejemplo Básico

```
public class Main {
    public static void lanzarExcepcion() {
        throw new IllegalArgumentException("Argumento no válido.");
    }
    public static void main(String[] args) {
        lanzarExcepcion();
    }
}
```

Salida:

```
Exception in thread "main" java.lang.IllegalArgumentException: Argumento no válido.
```

2.2. Palabra Clave `throws`

La palabra clave `throws` se utiliza en la declaración de un método para indicar que puede lanzar una o más excepciones verificadas.

Ejemplo Básico

```
import java.io.File;
import java.io.FileReader;
public class Main {
    public static void leerArchivo(String ruta) throws Exception {
          File archivo = new File(ruta);
          FileReader lector = new FileReader(archivo);
    }
    public static void main(String[] args) {
          try {
                leerArchivo("archivo.txt");
          } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
          }
    }
}
```

Salida:

```
Error: archivo.txt (No such file or directory)
```

2.3. Diferencias entre `throw` y `throws`

Aspecto	`throw`	`throws`
Uso	Se utiliza para lanzar una excepción.	Se utiliza para declarar excepciones que puede lanzar un método.
Momento	Se encuentra dentro del cuerpo del método.	Se encuentra en la declaración del método.
Tipos	Lanza una única instancia de excepción.	Declara una o más excepciones.

3. Ejercicio Guiado

Problema

1. Define una clase `CuentaBancaria` con un atributo `saldo`.
2. Crea una excepción personalizada `SaldoInsuficienteException`.
3. Implementa un método `retirar(double cantidad)` que:
- Lanza la excepción si el saldo es insuficiente.

4. En la clase principal:
- Crea una cuenta con saldo inicial.
- Intenta retirar más dinero del disponible, manejando la excepción.

Código Propuesto

```
  public class SaldoInsuficienteException extends Exception {
      public SaldoInsuficienteException(String mensaje) {
            super(mensaje);
      }
  }
  public class CuentaBancaria {
      private double saldo;
      public CuentaBancaria(double saldoInicial) {
            this.saldo = saldoInicial;
      }
      public void retirar(double cantidad) throws SaldoInsuficienteException {
            if (cantidad > saldo) {
                  throw new SaldoInsuficienteException("Saldo insuficiente. Saldo
disponible: " + saldo);
            }
            saldo -= cantidad;
      }
      public double getSaldo() {
            return saldo;
      }
  }
  public class Main {
      public static void main(String[] args) {
            CuentaBancaria cuenta = new CuentaBancaria(100);
            try {
                  cuenta.retirar(150);
            } catch (SaldoInsuficienteException e) {
                  System.out.println("Excepción capturada: " + e.getMessage());
            } finally {
                  System.out.println("Saldo actual: " + cuenta.getSaldo());
            }
      }
  }
```

Salida:

```
Excepción capturada: Saldo insuficiente. Saldo disponible: 100.0
Saldo actual: 100.0
```

4. Preguntas de Evaluación

1. ¿Qué palabra clave se utiliza para lanzar una excepción en Java?

a) `throws`.
b) `throw`.
c) `catch`.
d) `finally`.

2. ¿Qué ocurre si un método lanza una excepción verificada y no se maneja?

a) El programa lanza un error en tiempo de compilación.
b) La excepción se ignora.
c) El programa continúa normalmente.
d) El compilador genera un warning.

3. ¿Qué tipo de excepción se recomienda extender para excepciones personalizadas no verificadas?

a) `RuntimeException`.
b) `Exception`.
c) `Throwable`.
d) `Error`.