Unidad 13

Manejo de excepciones

Contenidos

Parte 1: Captura, Propagar, Lanzar y Crear excepciones

El manejo de excepciones es una parte fundamental del desarrollo de software en Java. Permite controlar los errores de manera estructurada, mejorando la robustez y estabilidad de las aplicaciones. Este apartado se centrará en los conceptos de **captura, propagación y lanzamiento de excepciones**, así como en la creación de **clases de excepciones personalizadas**.

1. Captura de Excepciones

La captura de excepciones permite manejar errores específicos que ocurren durante la ejecución del programa. Esto se logra utilizando bloques `try` y `catch`.

1.1. Uso del Bloque `try-catch`

- `try`: Contiene el código que puede generar una excepción.
- `catch`: Maneja la excepción lanzada.

Ejemplo Básico

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

Salida:

```
Error: División por cero.
```

1.2. Captura de Excepciones Múltiples

Cuando un bloque de código puede generar diferentes tipos de excepciones, se pueden usar varios bloques `catch`.

Ejemplo

```
  public class Main {
      public static void main(String[] args) {
            try {
                  int[] numeros = {1, 2, 3};
                  System.out.println(numeros[5]); // ArrayIndexOutOfBoundsExcep-
tion
            } 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. Propagar Excepciones

La propagación de excepciones ocurre cuando una excepción no se maneja en el método actual y se "propaga" a su llamador. Esto se indica utilizando la palabra clave `throws` en la declaración del método.

2.1. Ejemplo de Propagación de Excepciones

```
import java.io.FileReader;
public class Main {
    public static void leerArchivo() throws Exception {
        FileReader archivo = new FileReader("archivo.txt");
    }
    public static void main(String[] args) {
        try {
            leerArchivo(); // La excepción se propaga a este método
        } catch (Exception e) {
            System.out.println("Error al leer el archivo: " + e.getMes-
sage());
        }
    }
```

Salida:

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

3. Lanzar Excepciones

La palabra clave `throw` permite lanzar una excepción manualmente desde cualquier parte del código. Es útil cuando queremos validar condiciones y manejar errores personalizados.

3.1. Ejemplo de Lanzar Excepciones

```
  public class Main {
      public static void verificarEdad(int edad) {
            if (edad < 18) {
                  throw new IllegalArgumentException("Edad insuficiente para con-
tinuar.");
            }
      }
      public static void main(String[] args) {
            try {
                  verificarEdad(16); // Lanza IllegalArgumentException
            } catch (IllegalArgumentException e) {
                  System.out.println("Excepción lanzada: " + e.getMessage());
            }
      }
  }
```

Salida:

```
  Excepción lanzada: Edad insuficiente para continuar.
```

4. Crear Clases de Excepciones

Java permite crear clases de excepciones personalizadas que extienden de `Exception` (verificada) o `RuntimeException` (no verificada).

4.1. Crear una Excepción Personalizada

Ejemplo con `Exception` (verificada)

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

Uso de la Excepción Personalizada

```
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 class Main {
    public static void main(String[] args) {
        CuentaBancaria cuenta = new CuentaBancaria(100);
        try {
            cuenta.retirar(150); // Lanza SaldoInsuficienteException
        } catch (SaldoInsuficienteException e) {
            System.out.println("Excepción capturada: " + e.getMessage());
        }
    }
}
```

Salida:

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

4.2. Excepción No Verificada (`RuntimeException`)

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

Uso

```
public class Main {
    public static void verificarNumero(int numero) {
        if (numero < 0) {
            throw new MiExcepcionRuntime("El número no puede ser negati-
vo.");
        }
    }
    public static void main(String[] args) {
        verificarNumero(-5); // Lanza MiExcepcionRuntime
    }
}
```

Salida:

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

5. Ejercicio Guiado

Problema

1. **Crea una clase `Producto` con atributos `nombre` y `precio`.**
2. **Define una excepción personalizada `PrecioInvalidoException` para validar que el precio no sea negativo.**
3. **Implementa un método `setPrecio(double precio)` que lance la excepción personalizada si el precio es negativo.**
4. **Maneja la excepción en la clase principal.**

6. 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. ¿Cuál es la diferencia entre `throw` y `throws`?

a) thr`ow se` utiliza para capturar excepciones, mientras que `throws` se usa para lanzarlas.
b) `throw` lanza una excepción, mientras que `throws` declara excepciones que un método puede lanzar.
c) `throw` es para excepciones no verificadas, mientras que `throws` es para verificadas.
d) No hay diferencia.

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

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