Manejo de Excepciones

Ejercicios

Ejercicio 1: Manejo Básico de Excepciones

Objetivo: Implementar un bloque `try-catch` para manejar una posible excepción de división por cero.

```
public class DivisionPorCero {
    public static void main(String[] args) {
        try {
            int resultado = dividir(10, 0);
            System.out.println("Resultado: " + resultado);
        } catch (ArithmeticException e) {
            System.out.println("Error: No se puede dividir por cero.");
        }
    }

    public static int dividir(int a, int b) {
        return a / b;
    }
}
```

Ejercicio 2: Captura de Múltiples Excepciones

Objetivo: Manejar diferentes tipos de excepciones en un solo bloque `try`.

```
public class ManejoDeExcepciones {
    public static void main(String[] args) {
        try {
            int[] numeros = {1, 2, 3};
            System.out.println(numeros[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Índice fuera de los límites del array.");
        } catch (Exception e) {
            System.out.println("Error general: " + e.getMessage());
        }
    }
}
```

Ejercicio 3: Uso de `finally`

Objetivo: Implementar un bloque `finally` para ejecutar código independientemente de si ocurrió una excepción.

```
public class UsoDeFinally {
    public static void main(String[] args) {
        try {
            int resultado = dividir(10, 2);
            System.out.println("Resultado: " + resultado);
        } catch (ArithmeticException e) {
            System.out.println("Error: No se puede dividir por cero.");
        } finally {
            System.out.println("Ejecución del bloque finally.");
        }
    }

    public static int dividir(int a, int b) {
        return a / b;
    }
}
```

Ejercicio 4: Excepciones Personalizadas

Objetivo: Crear y lanzar una excepción personalizada.

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

public class ValidarEdad {
    public static void main(String[] args) {
        try {
            validarEdad(15);
        } catch (EdadInvalidaException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static void validarEdad(int edad) throws EdadInvalidaException {
        if (edad < 18) {
            throw new EdadInvalidaException("La edad debe ser mayor o igual a
18.");
        }
        System.out.println("Edad válida.");
    }
}
```

Ejercicio 5: Propagación de Excepciones

Objetivo: Propagar una excepción al método llamante usando la palabra clave `throws`.

```
public class PropagacionExcepciones {
    public static void main(String[] args) {
        try {
            metodoA();
        } catch (Exception e) {
            System.out.println("Excepción capturada: " + e.getMessage());
        }
    }

    public static void metodoA() throws Exception {
        metodoB();
    }

    public static void metodoB() throws Exception {
        throw new Exception("Excepción lanzada en metodoB.");
    }
}
```

Ejercicio 6: Validación de Entrada

Objetivo: Implementar un programa que valide la entrada del usuario y maneje posibles excepciones.

```
import java.util.Scanner;

public class ValidacionEntrada {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese un número entero: ");
        try {
            int numero = scanner.nextInt();
            System.out.println("Número ingresado: " + numero);
        } catch (Exception e) {
            System.out.println("Error: Entrada no válida.");
        } finally {
            scanner.close();
        }
    }
}
```

Ejercicio 7: Creación de Excepciones Personalizadas para Validaciones

Objetivo: Crear una clase de excepción personalizada para validar nombres.

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

```
public class ValidarNombre {
    public static void main(String[] args) {
        try {
            validarNombre("");
        } catch (NombreInvalidoException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static void validarNombre(String nombre) throws NombreInvalidoException
{
        if (nombre == null || nombre.isEmpty()) {
            throw new NombreInvalidoException("El nombre no puede estar vacío.");
        }
        System.out.println("Nombre válido.");
    }
}
```

Ejercicio 8: Manejo de Excepciones Checked

Objetivo: Manejar excepciones checked como `IOException` en la lectura de un archivo.

```
import java.io.*;

public class LeerArchivo {
    public static void main(String[] args) {
        try {
            leerArchivo("archivo.txt");
        } catch (IOException e) {
            System.out.println("Error al leer el archivo: " + e.getMessage());
        }
    }

    public static void leerArchivo(String nombreArchivo) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(nombreArchivo));
        String linea;
        while ((linea = br.readLine()) != null) {
            System.out.println(linea);
        }
        br.close();
    }
}
```

Ejercicio 9: Manejo de Excepciones Unchecked

Objetivo: Manejar una excepción `NullPointerException`.

```
public class ManejoNullPointer {
    public static void main(String[] args) {
        String texto = null;
        try {
            System.out.println(texto.length());
        } catch (NullPointerException e) {
            System.out.println("Error: Intento de acceder a un objeto nulo.");
        }
    }
}
```

Ejercicio 10: Excepciones en Operaciones Matemáticas

Objetivo: Manejar excepciones al realizar operaciones matemáticas.

```
public class OperacionesMatematicas {
    public static void main(String[] args) {
        try {
            int resultado = calcularRaizCuadrada(-5);
            System.out.println("Resultado: " + resultado);
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static int calcularRaizCuadrada(int numero) {
        if (numero < 0) {
            throw new IllegalArgumentException("No se puede calcular la raíz
cuadrada de un número negativo.");
        }
        return (int) Math.sqrt(numero);
    }
}
```

Ejercicio 11: Manejo de Excepciones en Entrada de Datos

Objetivo: Implementar un programa que solicite un número y maneje la excepción si el usuario ingresa texto no numérico.

```
import java.util.Scanner;

public class EntradaDatos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese un número: ");
        try {
            int numero = Integer.parseInt(scanner.nextLine());
            System.out.println("Número ingresado: " + numero);
```

```
        } catch (NumberFormatException e) {
            System.out.println("Error: Entrada no es un número válido.");
        } finally {
            scanner.close();
        }
    }
}
```

Ejercicio 12: Control de Excepciones en un Sistema Bancario

Objetivo: Simular un sistema bancario que maneje excepciones al intentar retirar más dinero del que hay disponible.

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

public class CuentaBancaria {
    private double saldo;

    public CuentaBancaria(double saldoInicial) {
        this.saldo = saldoInicial;
    }

    public void retirar(double cantidad) throws FondosInsuficientesException {
        if (cantidad > saldo) {
            throw new FondosInsuficientesException("Fondos insuficientes para
realizar el retiro.");
        }
        saldo -= cantidad;
        System.out.println("Retiro de " + cantidad + " realizado con éxito. Saldo
actual: " + saldo);
    }

    public void ingresar(double cantidad) {
        saldo += cantidad;
        System.out.println("Ingreso de " + cantidad + " realizado con éxito. Saldo
actual: " + saldo);
    }

    public double obtenerSaldo() {
        return saldo;
    }

    public static void main(String[] args) {
        CuentaBancaria cuenta = new CuentaBancaria(1000.0);

        try {
            cuenta.retirar(500.0); // Retiro válido
```

```
            cuenta.retirar(600.0); // Intento de retiro inválido
        } catch (FondosInsuficientesException e) {
            System.out.println(e.getMessage());
        }

        cuenta.ingresar(300.0); // Ingreso de dinero
        System.out.println("Saldo final: " + cuenta.obtenerSaldo());
    }
}
```