Programación Orientada a Objetos en Java

PROFESOR JUAN LUIS HERENCIA GUERRA

Arreglos y colecciones

GESTIÓN DE DATOS

Arreglos

- Los arreglos en Java son estructuras de datos que contienen elementos del mismo tipo. Tienen un tamaño fijo que se establece al momento de su creación.

```java
public class ArregloEjemplo {
    public static void main(String[] args) {
        int[] numeros = new int[5];  // Declaración y creación de un arreglo

        // Inicialización de los elementos del arreglo
        numeros[0] = 1;
        numeros[1] = 2;
        numeros[2] = 3;
        numeros[3] = 4;
        numeros[4] = 5;

        // Recorrer el arreglo
        for (int i = 0; i < numeros.length; i++) {
            System.out.println(numeros[i]);
        }
    }
}
```

Colecciones

- Las colecciones en Java son estructuras de datos dinámicas que pueden crecer y decrecer según sea necesario. Proporcionan métodos para manipular los datos de manera eficiente.

```java
import java.util.ArrayList;
import java.util.List;

public class ColeccionesEjemplo {
    public static void main(String[] args) {
        List<String> lista = new ArrayList<>();

        // Agregar elementos a la lista
        lista.add("Elemento 1");
        lista.add("Elemento 2");
        lista.add("Elemento 3");

        // Recorrer la lista
        for (String elemento : lista) {
            System.out.println(elemento);
        }
    }
}
```

Genéricos

- Los genéricos en Java permiten definir clases, interfaces y métodos con tipos de datos parametrizados. Esto proporciona mayor flexibilidad y seguridad de tipos.

```java
public class GenericosEjemplo<T> {
    private T dato;

    public GenericosEjemplo(T dato) {
        this.dato = dato;
    }

    public T getDato() {
        return dato;
    }

    public static void main(String[] args) {
        GenericosEjemplo<String> ejemploString = new GenericosEjemplo<>("Hola");
        System.out.println(ejemploString.getDato());  // Output: Hola

        GenericosEjemplo<Integer> ejemploInteger = new GenericosEjemplo<>(123);
        System.out.println(ejemploInteger.getDato());  // Output: 123
    }
}
```

Java Collections Framework

- Los genéricos en Java permiten definir clases, interfaces y métodos con tipos de datos parametrizados. Esto proporciona mayor flexibilidad y seguridad de tipos.

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
```

```java
public class JCF {
    public static void main(String[] args) {
        // Lista
        List<String> lista = new ArrayList<>();
        lista.add("Elemento 1");
        lista.add("Elemento 2");
        System.out.println("Lista: " + lista);

        // Set
        Set<String> conjunto = new HashSet<>();
        conjunto.add("Elemento A");
        conjunto.add("Elemento B");
        System.out.println("Set: " + conjunto);

        // Map
        Map<String, Integer> mapa = new HashMap<>();
        mapa.put("Clave1", 1);
        mapa.put("Clave2", 2);
        System.out.println("Map: " + mapa);
    }
}
```

Manejo de Listas

- Las listas son colecciones ordenadas que permiten elementos duplicados y acceso posicional a los elementos.

```java
import java.util.ArrayList;
import java.util.List;

public class ListaEjemplo {
    public static void main(String[] args) {
        List<String> lista = new ArrayList<>();

        // Agregar elementos
        lista.add("Elemento 1");
        lista.add("Elemento 2");
        lista.add("Elemento 3");

        // Obtener elemento por índice
        System.out.println("Elemento en índice 1: " + lista.get(1));

        // Recorrer la lista
        for (String elemento : lista) {
            System.out.println(elemento);
        }

        // Eliminar elemento
        lista.remove("Elemento 2");
        System.out.println("Lista después de eliminar: " + lista);
    }
}
```

Manejo de Maps

- Los mapas son colecciones que mapean claves únicas a valores. No permiten claves duplicadas y cada clave puede mapear a un solo valor.

```java
import java.util.HashMap;
import java.util.Map;

public class MapEjemplo {
    public static void main(String[] args) {
        Map<String, Integer> mapa = new HashMap<>();

        // Agregar pares clave-valor
        mapa.put("Uno", 1);
        mapa.put("Dos", 2);
        mapa.put("Tres", 3);

        // Obtener valor por clave
        System.out.println("Valor para la clave 'Dos': " + mapa.get("Dos"));

        // Recorrer el mapa
        for (Map.Entry<String, Integer> entrada : mapa.entrySet()) {
            System.out.println(entrada.getKey() + ": " + entrada.getValue());
        }

        // Eliminar por clave
        mapa.remove("Tres");
        System.out.println("Mapa después de eliminar: " + mapa);
    }
}
```

Manejo de Sets

- Los conjuntos (sets) son colecciones que no permiten elementos duplicados. Proporcionan operaciones eficientes para agregar, eliminar y verificar la existencia de elementos.

```java
import java.util.HashSet;
import java.util.Set;

public class SetEjemplo {
    public static void main(String[] args) {
        Set<String> conjunto = new HashSet<>();

        // Agregar elementos
        conjunto.add("Elemento A");
        conjunto.add("Elemento B");
        conjunto.add("Elemento C");

        // Verificar si un elemento existe
        System.out.println("Conjunto contiene 'Elemento B': " + conjunto.contains("Elemento B"));

        // Recorrer el conjunto
        for (String elemento : conjunto) {
            System.out.println(elemento);
        }

        // Eliminar elemento
        conjunto.remove("Elemento A");
        System.out.println("Conjunto después de eliminar: " + conjunto);
    }
}
```

Manejo de Sets

- Los conjuntos (sets) son colecciones que no permiten elementos duplicados. Proporcionan operaciones eficientes para agregar, eliminar y verificar la existencia de elementos.

```java
import java.util.HashSet;
import java.util.Set;

public class SetEjemplo {
    public static void main(String[] args) {
        Set<String> conjunto = new HashSet<>();

        // Agregar elementos
        conjunto.add("Elemento A");
        conjunto.add("Elemento B");
        conjunto.add("Elemento C");

        // Verificar si un elemento existe
        System.out.println("Conjunto contiene 'Elemento B': " + conjunto.contains("Elemento B"));

        // Recorrer el conjunto
        for (String elemento : conjunto) {
            System.out.println(elemento);
        }

        // Eliminar elemento
        conjunto.remove("Elemento A");
        System.out.println("Conjunto después de eliminar: " + conjunto);
    }
}
```

Excepciones

GESTIÓN DE ERRORES

Tipos de Errores

En Java, los errores se pueden clasificar en tres categorías principales:

1. Errores de Sintaxis: Son errores en el código que violan las reglas del lenguaje y son detectados por el compilador. Ejemplo: falta de punto y coma, paréntesis no emparejados.
2. Errores de Ejecución: Ocurren durante la ejecución del programa y son detectados por la máquina virtual de Java (JVM). Ejemplo: dividir por cero, acceder a un índice de matriz fuera de rango.
3. Errores Lógicos: Son errores en la lógica del programa que producen un comportamiento incorrecto. No son detectados por el compilador ni la JVM. Ejemplo: uso incorrecto de operadores.

JUAN LUIS HERENCIA GUERRA

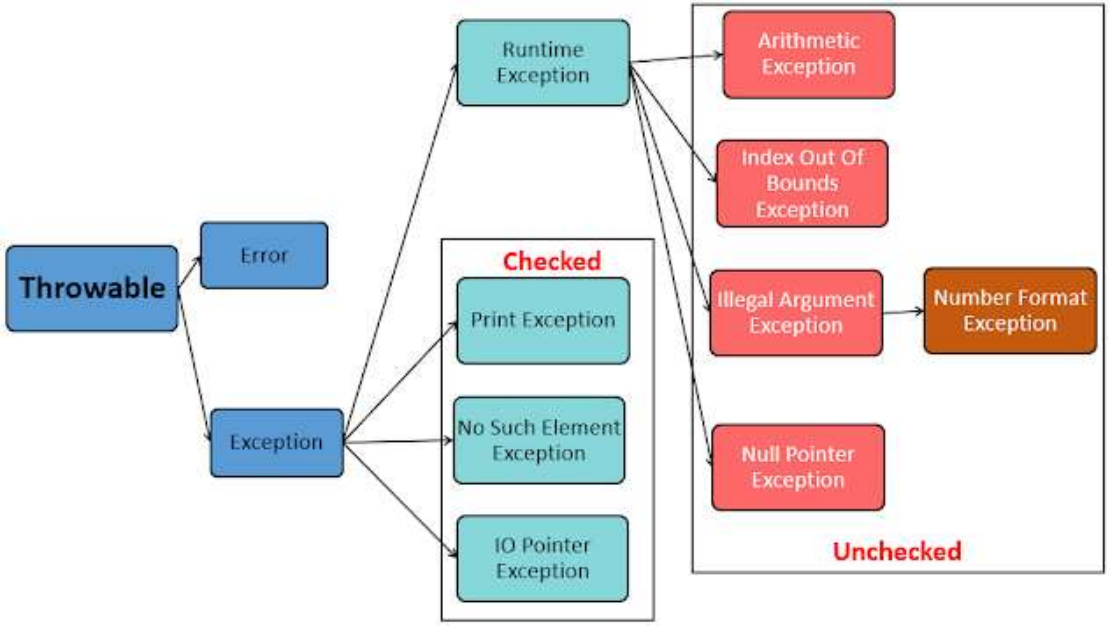
Excepción

- Una excepción en Java es un evento que interrumpe el flujo normal de ejecución del programa. Las excepciones son objetos que describen un error o una condición inusual que ha ocurrido.

```
try {
 //bloque codigo
}
catch (Exception e) {
  //Captura error
}
finally {
 //limpieza y liberación menoria
}
```

Tipos de Excepciones

- Excepciones Verificadas (Checked Exceptions): Son excepciones que se deben manejar explícitamente en el código utilizando try-catch o declarando en la firma del método con throws. Ejemplo: IOException, SQLException.
- Excepciones No Verificadas (Unchecked Exceptions): Son subclases de RuntimeException y no es obligatorio manejarlas explícitamente. Ejemplo: NullPointerException, ArrayIndexOutOfBoundsException.
- Errores (Errors): Son subclases de Error y representan problemas serios que una aplicación normalmente no debería intentar manejar. Ejemplo: OutOfMemoryError, StackOverflowError.



Gestión de Excepciones

- La gestión de excepciones en Java se realiza utilizando bloques try-catch y el bloque finally.

```java
public class GestionExcepciones {
    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.");
        } finally {
            System.out.println("Bloque finally ejecutado.");
        }
    }

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

Excepciones Personalizadas

- Podemos definir nuestras propias excepciones creando clases que extiendan Exception o RuntimeException.

```java
// Definición de la excepción personalizada
class MiExcepcionPersonalizada extends Exception {
    public MiExcepcionPersonalizada(String mensaje) {
        super(mensaje);
    }
}

public class ExcepcionesPersonalizadas {
    public static void main(String[] args) {
        try {
            validarEdad(15);
        } catch (MiExcepcionPersonalizada e) {
            System.out.println("Excepción capturada: " + e.getMessage());
        }
    }

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

Programación Orientada a Objetos en Java

Acerca del profesor Juan Luis Herencia Guerra

Con maestría de especialización en Computer Science, graduado en la carrera de Ingeniería Eléctrica en la Universidad Nacional de Ingeniería con muchos años de experiencia en dirección y desarrollo de software en diferentes lenguajes y arquitecturas para las empresas privadas y públicas.