
Politécnico
Internacional

Politécnico Internacional

Java

ESTRUCTURA DE DATOS

Politécnico Internacional

Java

“Celebra cada pequeña victoria.”

AGENDA

- Conceptos Java
- BetPlay
- Conclusiones

Apuntadores

Los **"apuntadores"** son un concepto importante en la programación, especialmente en lenguajes de programación como C y C++. Sin embargo, en el contexto de Java, los apuntadores no están presentes en la misma forma debido a la gestión automática de memoria proporcionada por el recolector de basura y la seguridad que ofrece el lenguaje.

Apuntadores

En lenguajes como C y C++, un apuntador es una variable que almacena la dirección de memoria de otra variable en lugar de su valor real. Los apuntadores permiten trabajar con la memoria directamente, lo que puede ser útil para operaciones avanzadas pero también puede ser propenso a errores y problemas de seguridad si se maneja incorrectamente.

Apuntadores

En lenguajes como C y C++, un apuntador es una variable que almacena la dirección de memoria de otra variable en lugar de su valor real. Los apuntadores permiten trabajar con la memoria directamente, lo que puede ser útil para operaciones avanzadas pero también puede ser propenso a errores y problemas de seguridad si se maneja incorrectamente.

Apuntadores

Los apuntadores se utilizan para:

1. **Acceder a Direcciones de Memoria:** Pueden usarse para acceder directamente a la ubicación en la memoria de una variable o un objeto.
2. **Manipulación de Estructuras de Datos:** Los apuntadores son fundamentales para implementar estructuras de datos como listas enlazadas, árboles y grafos.
3. **Paso de Parámetros Eficiente:** En algunos casos, pasar un apuntador a una función puede ser más eficiente que pasar una copia completa de un objeto.
4. **Gestión de Memoria:** En lenguajes sin recolección de basura, los apuntadores pueden usarse para administrar la memoria de manera más precisa y evitar fugas de memoria.
5. **Implementación de Algoritmos Avanzados:** En algoritmos y operaciones avanzadas, como la manipulación de cadenas, los apuntadores pueden ser útiles para optimizar el rendimiento.

Apuntadores

Los apuntadores se utilizan para:

1.Acceder a Direcciones de Memoria: Pueden usarse para acceder directamente a la ubicación en la memoria de una variable o un objeto.
2.Manipulación de Estructuras de Datos: Los apuntadores son fundamentales para implementar estructuras de datos como listas enlazadas, árboles y grafos.
3.Paso de Parámetros Eficiente: En algunos casos, pasar un apuntador a una función puede ser más eficiente que pasar una copia completa de un objeto.
4.Gestión de Memoria: En lenguajes sin recolección de basura, los apuntadores pueden usarse para administrar la memoria de manera más precisa y evitar fugas de memoria.
5.Implementación de Algoritmos Avanzados: En algoritmos y operaciones avanzadas, como la manipulación de cadenas, los apuntadores pueden ser útiles para optimizar el rendimiento.

Apuntadores

En Java, debido a la abstracción y la administración automática de memoria, los conceptos de apuntadores directos y manipulación de memoria no son tan evidentes como en lenguajes de programación como C o C++. En su lugar, se utilizan referencias para acceder a objetos en la memoria, pero el programador no necesita preocuparse por la administración de memoria o los problemas asociados con los apuntadores tradicionales.

Apuntadores

```java
package com.mycompany.referenciasexample;

/**
 *
 * @author Harol
 */
public class ReferenciasExample {

    public static void main(String[] args) {
        // Crear un objeto de tipo String
        String cadena = "Politecnico Internacional, Implementacion de una Cadena";

        // Asignar una referencia al objeto a otra variable
        String otraCadena = cadena;

        // Modificar el contenido del objeto a través de la referencia
        otraCadena = otraCadena + "Politecnico Internacional, Implementacion de una cadena modificada";

        // Imprimir ambas variables
        System.out.println("Cadena original: " + cadena);
        System.out.println("Otra cadena: " + otraCadena);
    }
}
```

Apuntadores

```
Output - Run (ReferenciasExample)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\ReferenciasExample; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "\"C:\\Program F
Scanning for projects...

------------------< com.mycompany:ReferenciasExample >------------------
Building ReferenciasExample 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ ReferenciasExample ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\ReferenciasExample\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ ReferenciasExample ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ ReferenciasExample ---
Cadena original: Politecnico Internacional, Implementacion de una Cadena
Otra cadena: Politecnico Internacional, Implementacion de una CadenaPolitecnico Internacional, Implementacion de una cadena modificada
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.665 s
Finished at: 2023-08-15T10:32:07-05:00
------------------------------------------------------------------------
```

LIFO

LIFO significa "Last In, First Out", que se traduce como "último en entrar, primero en salir". LIFO es un principio utilizado en estructuras de datos y operaciones de procesamiento donde el último elemento que se agrega es el primero en ser retirado. Esto se asemeja al funcionamiento de una pila, donde los elementos se apilan uno encima del otro y el elemento en la parte superior es el primero en ser retirado.

LIFO

En Java, el concepto de LIFO se utiliza comúnmente en el contexto de pilas (stacks). Una pila es una estructura de datos lineal que sigue el principio LIFO

LIFO

1.Gestión de Llamadas a Funciones: En la ejecución de programas, las llamadas a funciones se manejan mediante una pila. Cuando se llama a una función, se agrega a la pila, y cuando la función termina, se retira de la pila.

1.Despliegue de Historial en Navegadores Web: Cuando navegas por diferentes páginas en un navegador web, las URL se agregan a una pila. Al presionar el botón "Atrás", se retira la última URL de la pila, lo que te lleva a la página anterior.

1.Resolución de Expresiones en Evaluación Postfija (Notación Polaca Inversa): La notación polaca inversa utiliza una pila para evaluar expresiones matemáticas. Los operandos se apilan y los operadores se aplican cuando se retiran de la pila.

LIFO

```java
import java.util.Stack;

public class LIFOExample {

    public static void main(String[] args) {
        // Crear una instancia de Stack para implementar LIFO
        Stack<String> lifoStack = new Stack<>();

        // Agregar elementos a la pila
        lifoStack.push(item: "Elemento 1");
        lifoStack.push(item: "Elemento 2");
        lifoStack.push(item: "Elemento 3");

        System.out.println(x: "| Politecnico Internacional | LIFO|");

        // Imprimir la pila actual
        System.out.println("Pila después de agregar elementos: " + lifoStack);

        // Retirar y mostrar el elemento superior de la pila
        String topElement = lifoStack.pop();
        System.out.println("Elemento retirado de la pila: " + topElement);

        // Imprimir la pila después de retirar un elemento
        System.out.println("Pila después de retirar un elemento: " + lifoStack);
    }
}
```

LIFO

```
Output - Run (LIFOExample)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\LIFOExample; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "\'
Scanning for projects...

---------------------< com.mycompany:LIFOExample >----------------------
Building LIFOExample 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ LIFOExample ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\LIFOExample\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ LIFOExample ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ LIFOExample ---
| Politecnico Internacional | LIFO|
Pila después de agregar elementos: [Elemento 1, Elemento 2, Elemento 3]
Elemento retirado de la pila: Elemento 3
Pila después de retirar un elemento: [Elemento 1, Elemento 2]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.647 s
Finished at: 2023-08-15T10:39:29-05:00
------------------------------------------------------------------------
```

Queue - Colas

Una cola, en programación, es una estructura de datos lineal que sigue el principio "First In, First Out" (FIFO), lo que significa que el primer elemento que se agrega a la cola es el primero en ser retirado. Es similar a una fila de personas esperando su turno: la persona que llega primero es la primera en ser atendida.

Queue - Colas

En Java, la interfaz Queue define la funcionalidad de una cola. Las colas son útiles en situaciones donde es importante mantener el orden de llegada y procesar elementos en ese orden, como en procesos de encolamiento y desencolamiento.

Queue - Colas

1.Procesamiento de Tareas en Orden: Las colas son útiles para administrar tareas en orden de llegada. Por ejemplo, en la programación de concurrencia, se pueden utilizar colas para manejar tareas en un orden específico.

2.Implementación de Algoritmos: Algoritmos como BFS (Breadth-First Search) en grafos utilizan colas para procesar los nodos en el orden correcto.

3.Administración de Colas de Trabajo: En sistemas de procesamiento de trabajos, las colas se utilizan para administrar trabajos pendientes y en proceso.

4.Gestión de Hilos: Las colas son una herramienta común en la programación concurrente para coordinar hilos y evitar condiciones de carrera.

5.Implementación de Algoritmos de Planificación: Los sistemas operativos y los planificadores de tareas pueden usar colas para administrar tareas a ejecutar en un sistema.

Queue - Colas

```java
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {

    public static void main(String[] args) {
        // Crear una instancia de Queue utilizando LinkedList
        Queue<String> cola = new LinkedList<>();

        // Agregar elementos a la cola
        cola.offer(e: "Elemento 1");
        cola.offer(e: "Elemento 2");
        cola.offer(e: "Elemento 3");

        // Mostrar la cola actual
        System.out.println("Cola después de agregar elementos: " + cola);

        // Retirar y mostrar el primer elemento de la cola
        String primerElemento = cola.poll();
        System.out.println("Primer elemento retirado de la cola: " + primerElemento);

        // Mostrar la cola después de retirar un elemento
        System.out.println("Cola después de retirar un elemento: " + cola);
    }
}
```

Queue - Colas

```
Output - Run (QueueExample)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\QueueExample; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "\"C
Scanning for projects...

---------------------< com.mycompany:QueueExample >---------------------
Building QueueExample 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ QueueExample ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\QueueExample\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ QueueExample ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ QueueExample ---
Cola después de agregar elementos: [Elemento 1, Elemento 2, Elemento 3]
Primer elemento retirado de la cola: Elemento 1
Cola después de retirar un elemento: [Elemento 2, Elemento 3]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.716 s
Finished at: 2023-08-15T10:52:07-05:00
------------------------------------------------------------------------
```

Queue - Colas

Una **cola circular** es una variante de la estructura de datos de cola estándar en la que los elementos se almacenan en una secuencia circular. En una cola circular, cuando se agrega un nuevo elemento y la cola está llena, el siguiente elemento se agrega en el principio de la cola, reemplazando el elemento más antiguo. Esto permite que la cola "gire" en lugar de simplemente desbordarse.

Queue - Colas

En Java, puedes implementar una cola circular utilizando un arreglo o una lista enlazada circular. Las colas circulares son útiles en situaciones donde necesitas mantener una cantidad fija de elementos y quieres reciclar los espacios una vez que la cola esté llena, en lugar de simplemente no permitir más inserciones.

Queue - Colas

Algunos ejemplos de situaciones en las que las colas circulares son útiles incluyen:

1.Buffer Circular: En aplicaciones que manejan flujo de datos, como en la reproducción de música o video en tiempo real, una cola circular puede utilizarse como un buffer para almacenar datos temporalmente antes de procesarlos.

1.Cola de Tareas Limitada: En sistemas de administración de tareas o planificación, una cola circular puede garantizar que solo se ejecuten un número limitado de tareas a la vez.

1.Buffer en Comunicaciones: En aplicaciones de redes, una cola circular puede ser útil para manejar paquetes de datos entrantes y salientes.

1.Manejo de Eventos: En aplicaciones que reciben eventos, como aplicaciones gráficas o interacciones de usuario, una cola circular puede utilizarse para manejar eventos en el orden en que se reciben.

Queue - Colas

```java
public class CircularQueueExample {

    public static class CircularQueue {
        private int[] elements;
        private int front, rear, size;

        public CircularQueue(int capacity) {
            elements = new int[capacity];
            front = 0;
            rear = -1;
            size = 0;
        }

        public void enqueue(int element) {
            if (size < elements.length) {
                rear = (rear + 1) % elements.length;
                elements[rear] = element;
                size++;
            } else {
                System.out.println(x: "La cola circular está llena. No se puede encolar.");
            }
        }

        public int dequeue() {
            if (size > 0) {
                int removedElement = elements[front];
                front = (front + 1) % elements.length;
                size--;
                return removedElement;
            } else {
                System.out.println(x: "La cola circular está vacía. No se puede desencolar.");
                return -1;
            }
        }

        public boolean isEmpty() {
            return size == 0;
        }

        public boolean isFull() {
            return size == elements.length;
        }
```

Queue - Colas

```
    public boolean isFull() {
        return size == elements.length;
    }

    public int size() {
        return size;
    }
}

public static void main(String[] args) {
    CircularQueue queue = new CircularQueue(capacity: 5);

    queue.enqueue(element: 1);
    queue.enqueue(element: 2);
    queue.enqueue(element: 3);
    queue.enqueue(element: 4);

    System.out.println("Elementos encolados: " + queue.size());

    int removedElement = queue.dequeue();
    System.out.println("Elemento desencolado: " + removedElement);

    queue.enqueue(element: 5);
    queue.enqueue(element: 6);

    System.out.println("Elementos encolados después de desencolar y encolar: " + queue.size());
}
}
```

Queue - Colas

```
Output - Run (CircularQueueExample)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\CircularQueueExample; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "\"C
Scanning for projects...

----------------< com.mycompany:CircularQueueExample >----------------
Building CircularQueueExample 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ CircularQueueExample ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\CircularQueueExample\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ CircularQueueExample ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ CircularQueueExample ---
Elementos encolados: 4
Elemento desencolado: 1
Elementos encolados después de desencolar y encolar: 5
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.686 s
Finished at: 2023-08-15T11:02:15-05:00
------------------------------------------------------------------------
```

Arboles B+ & B-

Un **árbol B+ y un árbol B- son estructuras de datos similares a los árboles binarios de búsqueda**, pero optimizadas para operaciones de búsqueda y ordenamiento en bases de datos y sistemas de almacenamiento. Estos árboles son especialmente útiles cuando se manejan grandes cantidades de datos que deben almacenarse y consultarse eficientemente

Arboles B+ & B-

Árbol B+: Un árbol B+ es una estructura de datos de árbol en la que cada nodo interno puede tener múltiples hijos y contiene un rango de claves. Los nodos hoja contienen registros de datos y están vinculados en una lista enlazada para facilitar la búsqueda secuencial. Los árboles B+ son comúnmente utilizados en sistemas de bases de datos y sistemas de almacenamiento en disco, ya que su estructura facilita la búsqueda rápida de registros y permite realizar consultas de rango eficientemente.

Arboles B+ & B-

Árbol B-: Un árbol B- es similar al árbol B+, pero con algunas diferencias en cómo se organizan los nodos internos y cómo se manejan las claves duplicadas. Los árboles B- también están diseñados para mejorar la eficiencia de las operaciones de búsqueda y ordenamiento en bases de datos y sistemas de almacenamiento.
Ambos tipos de árboles se utilizan para mejorar el rendimiento de la búsqueda y manipulación de datos en sistemas que requieren una alta eficiencia, como bases de datos, sistemas de archivos y sistemas de gestión de información.

Arboles B+ & B-

Árbol B-: Un árbol B- es similar al árbol B+, pero con algunas diferencias en cómo se organizan los nodos internos y cómo se manejan las claves duplicadas. Los árboles B- también están diseñados para mejorar la eficiencia de las operaciones de búsqueda y ordenamiento en bases de datos y sistemas de almacenamiento.
Ambos tipos de árboles se utilizan para mejorar el rendimiento de la búsqueda y manipulación de datos en sistemas que requieren una alta eficiencia, como bases de datos, sistemas de archivos y sistemas de gestión de información.

Arboles B+ & B-

En Java, los árboles B+ y B- no son estructuras de datos estándar en la biblioteca Java, pero se pueden implementar utilizando clases personalizadas. Sin embargo, su implementación suele ser más compleja que otros tipos de árboles y podría requerir conocimientos avanzados de estructuras de datos y algoritmos

Arboles B+ & B-

En Java, los árboles B+ y B- no son estructuras de datos estándar en la biblioteca Java, pero se pueden implementar utilizando clases personalizadas. Sin embargo, su implementación suele ser más compleja que otros tipos de árboles y podría requerir conocimientos avanzados de estructuras de datos y algoritmos

Arboles B+ & B-

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

// Clase para representar un nodo en el árbol B+
class NodoBPlus {
    List<Integer> claves;
    List<NodoBPlus> hijos;
    boolean esHoja;

    public NodoBPlus() {
        claves = new ArrayList<>();
        hijos = new ArrayList<>();
        esHoja = false;
    }
}
```

Arboles B+ & B-

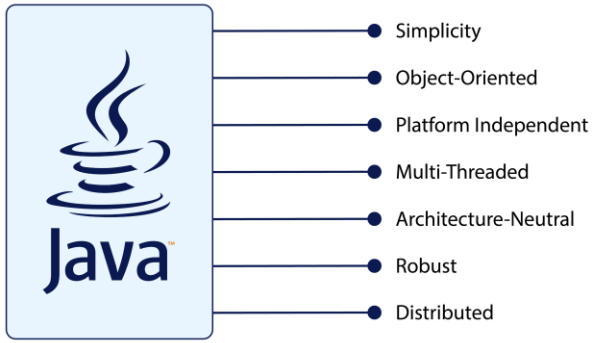
```java
// Clase principal para mostrar la estructura básica de un árbol B+
public class EjemploArbolBPlus {

    public static void main(String[] args) {
        // Crear un nodo raíz
        NodoBPlus raiz = new NodoBPlus();
        raiz.esHoja = true;
        raiz.claves.add(e: 10);
        raiz.claves.add(e: 20);
        raiz.claves.add(e: 30);

        // Crear un nodo hoja
        NodoBPlus nodoHoja = new NodoBPlus();
        nodoHoja.esHoja = true;
        nodoHoja.claves.add(e: 40);
        nodoHoja.claves.add(e: 50);
        nodoHoja.claves.add(e: 60);

        // Agregar el nodo hoja como hijo del nodo raíz
        raiz.hijos.add(e: nodoHoja);

        // Imprimir las claves del nodo raíz y el nodo hoja
        System.out.println(x: "| Politecnico Internacional | Arbol B+ & B-|");
        System.out.println("Claves del Nodo Raíz: " + raiz.claves);
        System.out.println("Claves del Nodo Hoja: " + nodoHoja.claves);
    }
}
```

Arboles B+ & B-

```
Output - Run (EjemploArbolBPlus)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploArbolBPlus; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "
Scanning for projects...

------------------< com.mycompany:EjemploArbolBPlus >-------------------
Building EjemploArbolBPlus 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ EjemploArbolBPlus ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploArbolBPlus\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ EjemploArbolBPlus ---
Changes detected - recompiling the module!
Compiling 1 source file to C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploArbolBPlus\target\classes

--- exec:3.1.0:exec (default-cli) @ EjemploArbolBPlus ---
| Politecnico Internacional | Arbol B+ & B-|
Claves del Nodo Raiz: [10, 20, 30]
Claves del Nodo Hoja: [40, 50, 60]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  1.083 s
Finished at: 2023-08-15T14:20:03-05:00
------------------------------------------------------------------------
```

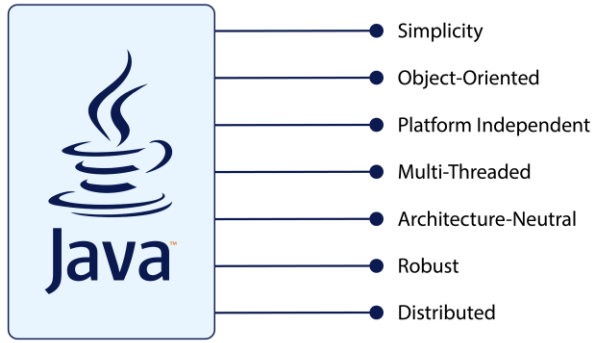
TADs

Los TADs (Tipos Abstractos de Datos) son un concepto fundamental en programación y estructuras de datos. Representan un conjunto de valores y las operaciones que se pueden realizar sobre esos valores. Los TADs son una abstracción que permite separar la implementación de los detalles internos de una estructura de datos de la forma en que se utilizan en un programa.



TADs

En Java, los TADs se implementan a través de clases y estructuras de datos que encapsulan los datos y las operaciones relacionadas con esos datos. Los TADs proporcionan una forma organizada y coherente de trabajar con diferentes estructuras de datos y tipos, lo que mejora la modularidad y la legibilidad del código.



TADs

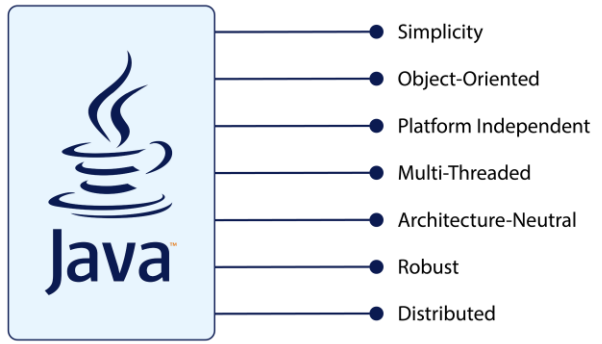
1.Lista: Representa una colección de elementos ordenados donde puedes realizar operaciones como agregar, eliminar y acceder a elementos.

1.Pila: Representa una estructura de datos LIFO (Last In, First Out) donde los elementos se apilan y desapilan en el mismo orden.

1.Cola: Representa una estructura de datos FIFO (First In, First Out) donde los elementos se encolan y desencolan en el mismo orden.

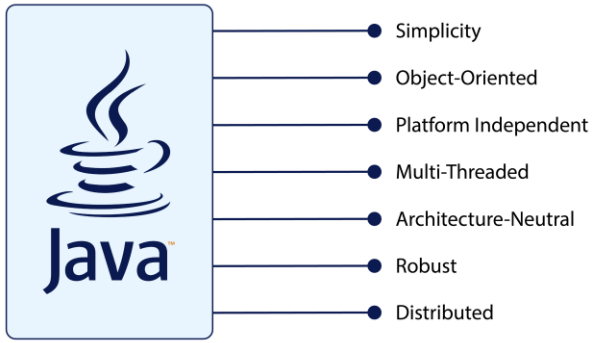
1.Árbol: Representa una estructura jerárquica donde cada elemento tiene un padre y cero o más hijos.

1.Grafo: Representa una colección de nodos y aristas que conectan esos nodos.



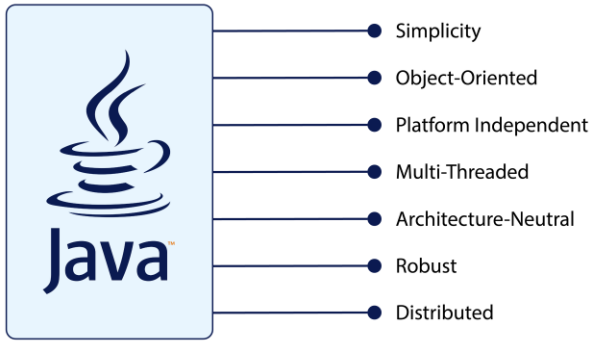
TADs

Los TADs son fundamentales porque permiten abstraer la complejidad de las estructuras de datos y facilitan su uso en el desarrollo de software. Al proporcionar operaciones bien definidas y ocultar los detalles internos, los TADs fomentan el diseño modular, la reutilización de código y la eficiencia en la programación.



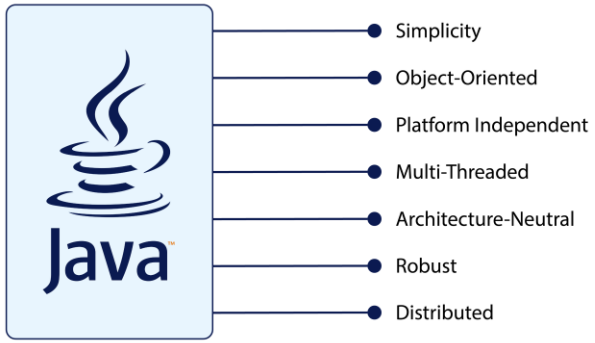
TADs

```java
package com.mycompany.ejemplotads;

/**
 *
 * @author Harol
 */
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class EjemploTADs {

    public static void main(String[] args) {
        // Ejemplo de una Lista
        ArrayList<String> lista = new ArrayList<>();
        lista.add(e: "Elemento 1");
        lista.add(e: "Elemento 2");
        lista.add(e: "Elemento 3");

        System.out.println(x: "Lista:");
        for (String elemento : lista) {
            System.out.println(x: elemento);
        }

        // Ejemplo de una Pila
        Stack<Integer> pila = new Stack<>();
        pila.push(item: 10);
        pila.push(item: 20);
        pila.push(item: 30);

        System.out.println(x: "\nPila:");
        while (!pila.isEmpty()) {
            System.out.println(x: pila.pop());
        }
```



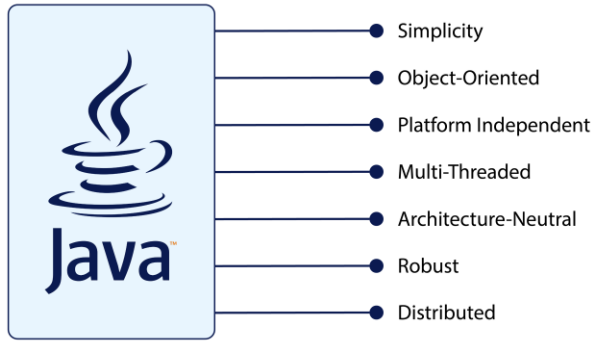
TADs

```
        // Ejemplo de una Cola
        Queue<Double> cola = new LinkedList<>();
        cola.offer( e: 1.1);
        cola.offer( e: 2.2);
        cola.offer( e: 3.3);

        System.out.println( x: "\nCola:");
        while (!cola.isEmpty()) {
            System.out.println( x: cola.poll());
        }
    }
}
```



TADs

```
Output - Run (EjemploTADs)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploTADs; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "\
Scanning for projects...

--------------------< com.mycompany:EjemploTADs >---------------------
Building EjemploTADs 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ EjemploTADs ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploTADs\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ EjemploTADs ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ EjemploTADs ---
Lista:
Elemento 1
Elemento 2
Elemento 3

Pila:
30
20
10

Cola:
1.1
2.2
3.3
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.728 s
Finished at: 2023-08-15T14:39:53-05:00
------------------------------------------------------------------------
```



Structs

En Java, no existe un concepto directamente equivalente a los "structs" que se encuentran en algunos otros lenguajes de programación, como C o C++. En C y C++, los "structs" son estructuras de datos que permiten agrupar diferentes tipos de datos bajo un solo nombre y tratarlos como un solo objeto. Sin embargo, en Java, esto se logra mediante la creación de clases.

Structs

En Java, se utilizan clases para definir estructuras de datos personalizadas. Una clase en Java puede tener atributos (variables miembro) que representen diferentes tipos de datos y métodos que definen el comportamiento asociado a esos datos. Las clases en Java permiten encapsular datos y comportamiento en una sola entidad, lo que facilita la programación orientada a objetos y la reutilización de código.

Structs

En Java, el equivalente a los "structs" de otros lenguajes son las clases. Las clases en Java proporcionan una forma más poderosa y flexible de definir estructuras de datos y comportamiento asociado.

Structs

```java
package com.mycompany.structejemplo;

/**
 *
 * @author Harol
 */
public class StructEjemplo {
    public static void main(String[] args) {
        // Creación de una instancia de "struct" en Java
        Persona persona = new Persona(nombre: "Harol Torres", edad: 38, direccion: "Cra 7 # 6 - 10 Bogota  DC");

        // Acceso a los atributos directamente
        System.out.println("Nombre: " + persona.nombre);
        System.out.println("Edad: " + persona.edad);
        System.out.println("Dirección: " + persona.direccion);
    }
}

// Definición de la "clase" que imita un "struct"
class Persona {
    public String nombre;
    public int edad;
    public String direccion;

    public Persona(String nombre, int edad, String direccion) {
        this.nombre = nombre;
        this.edad = edad;
        this.direccion = direccion;
    }
}
```

Structs

```
Output - Run (StructEjemplo)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\StructEjemplo; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "\"C
Scanning for projects...

--------------------< com.mycompany:StructEjemplo >---------------------
Building StructEjemplo 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ StructEjemplo ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\StructEjemplo\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ StructEjemplo ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ StructEjemplo ---
Nombre: Harol Torres
Edad: 38
Dirección: Cra 7 # 6 - 10 Bogota  DC
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.693 s
Finished at: 2023-08-15T14:45:27-05:00
------------------------------------------------------------------------
```

Métodos de Ordenamiento

Los **métodos de ordenamiento son algoritmos utilizados para reorganizar los elementos de una lista o conjunto de datos en un orden específico**, ya sea ascendente o descendente. El objetivo principal de los métodos de ordenamiento es mejorar la eficiencia de búsqueda y acceso a los datos, facilitando la tarea de búsqueda y manipulación de información.

Métodos de Ordenamiento

En Java, los métodos de ordenamiento son esenciales para organizar grandes cantidades de datos en estructuras de datos como arreglos (arrays) o colecciones. Existen diversos algoritmos de ordenamiento, cada uno con su propio enfoque y características de rendimiento.

Métodos de Ordenamiento

1.Ordenamiento de Burbuja: Compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Repite este proceso hasta que la lista esté ordenada.

1.Ordenamiento por Selección: Encuentra el elemento más pequeño y lo coloca en la posición correcta. Repite este proceso para todos los elementos restantes.

1.Ordenamiento por Inserción: Construye una lista ordenada tomando un elemento de la lista no ordenada y colocándolo en su posición correcta dentro de la lista ordenada.

1.Ordenamiento QuickSort: Divide la lista en dos sub-listas alrededor de un pivote y luego ordena recursivamente las sublistas.

Métodos de Ordenamiento

1.Ordenamiento MergeSort: Divide la lista en sublistas más pequeñas, las ordena y luego combina las sublistas ordenadas para obtener la lista final ordenada.

1.Ordenamiento HeapSort: Construye un árbol binario de montículo (heap) a partir de los elementos y luego extrae los elementos en orden.

Métodos de Ordenamiento

Los métodos de ordenamiento son fundamentales para la programación y el análisis de algoritmos, ya que pueden afectar significativamente el rendimiento y la eficiencia de las operaciones sobre datos ordenados. Dependiendo de la cantidad de datos y la naturaleza de los datos, ciertos métodos de ordenamiento pueden ser más eficientes que otros

Métodos de Ordenamiento

```java
public class EjemploOrdenamientoBurbuja {

    public static void main(String[] args) {
        int[] arreglo = {5, 2, 8, 1, 3};

        System.out.println(x: "Arreglo original:");
        imprimirArreglo(arr:arreglo);

        ordenarBurbuja(arr:arreglo);

        System.out.println(x: "\nArreglo ordenado:");
        imprimirArreglo(arr:arreglo);
    }

    // Implementación del método de ordenamiento Burbuja
    public static void ordenarBurbuja(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    // Intercambiar elementos si están en el orden incorrecto
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    // Método auxiliar para imprimir un arreglo
    public static void imprimirArreglo(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

Métodos de Ordenamiento

```
Output - Run (EjemploOrdenamientoBurbuja)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploOrdenamientoBurbuja; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "\"C:\\
Scanning for projects...

--------------< com.mycompany:EjemploOrdenamientoBurbuja >--------------
Building EjemploOrdenamientoBurbuja 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ EjemploOrdenamientoBurbuja ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploOrdenamientoBurbuja\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ EjemploOrdenamientoBurbuja ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ EjemploOrdenamientoBurbuja ---
Arreglo original:
5 2 8 1 3

Arreglo ordenado:
1 2 3 5 8
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.664 s
Finished at: 2023-08-15T14:52:50-05:00
------------------------------------------------------------------------
```

Métodos de Ordenamiento

```java
public class EjemploOrdenamientoSeleccion {

    public static void main(String[] args) {
        int[] arreglo = {5, 2, 8, 1, 3};

        System.out.println( "Arreglo original:");
        imprimirArreglo( arr: arreglo);

        ordenarSeleccion( arr: arreglo);

        System.out.println( "\nArreglo ordenado:");
        imprimirArreglo( arr: arreglo);
    }

    // Implementación del método de ordenamiento por selección
    public static void ordenarSeleccion(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int indiceMinimo = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[indiceMinimo]) {
                    indiceMinimo = j;
                }
            }
            // Intercambiar el elemento actual con el elemento mínimo
            int temp = arr[i];
            arr[i] = arr[indiceMinimo];
            arr[indiceMinimo] = temp;
        }
    }

    // Método auxiliar para imprimir un arreglo
    public static void imprimirArreglo(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

Métodos de Ordenamiento

```
Output - Run (EjemploOrdenamientoSeleccion)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploOrdenamientoSeleccion; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "\"C:\
Scanning for projects...

-------------< com.mycompany:EjemploOrdenamientoSeleccion >-------------
Building EjemploOrdenamientoSeleccion 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ EjemploOrdenamientoSeleccion ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploOrdenamientoSeleccion\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ EjemploOrdenamientoSeleccion ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ EjemploOrdenamientoSeleccion ---
Arreglo original:
5 2 8 1 3

Arreglo ordenado:
1 2 3 5 8
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.685 s
Finished at: 2023-08-15T14:54:56-05:00
------------------------------------------------------------------------
```

Métodos de Búsqueda

Los métodos de búsqueda son algoritmos utilizados para encontrar la posición o el valor de un elemento específico dentro de una estructura de datos, como un arreglo o una colección. Estos métodos permiten identificar si un elemento particular existe en la colección y, en caso afirmativo, su posición o valor. Los métodos de búsqueda son fundamentales para acceder eficientemente a los datos y realizar operaciones de recuperación.

Métodos de Búsqueda

En Java, los métodos de búsqueda son importantes para encontrar elementos en una variedad de estructuras de datos.

Métodos de Búsqueda

1.Búsqueda Lineal (Secuencial): Recorre los elementos uno por uno en orden hasta encontrar el elemento deseado. Es simple pero puede ser ineficiente para conjuntos de datos grandes.

1.Búsqueda Binaria: Funciona en arreglos ordenados. Divide repetidamente el conjunto de datos en dos mitades y descarta una mitad en cada paso, hasta que el elemento deseado sea encontrado.

1.Búsqueda de Hash: Utiliza una función de hash para calcular una posición en una estructura de datos como una tabla hash. Es eficiente para búsquedas si la función de hash está bien diseñada.

1.Búsqueda en Árboles: Utiliza la estructura de un árbol, como un árbol binario de búsqueda, para buscar elementos de manera eficiente a través de comparaciones.

1.Búsqueda Interpolada: Funciona en arreglos ordenados con valores uniformemente distribuidos. Calcula la posición estimada del elemento basada en su valor y realiza búsquedas más rápidas que la búsqueda binaria en algunos casos.

Métodos de Búsqueda

Los métodos de búsqueda son cruciales para la eficiencia y la optimización de los algoritmos y programas. Permiten acceder rápidamente a los datos relevantes y tomar decisiones informadas sobre cómo manejar o presentar la información. Los métodos de búsqueda son esenciales en la programación y el análisis de algoritmos para crear soluciones eficientes y efectivas

Métodos de Búsqueda

```java
package com.mycompany.ejemplometodosbusqueda;

/**
 *
 * @author Harol
 */
public class EjemploMetodosBusqueda {

    public static void main(String[] args) {
        int[] arreglo = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

        int elementoBuscado = 11;

        int posicionLineal = busquedaLineal(arr:arreglo, objetivo:elementoBuscado);
        int posicionBinaria = busquedaBinaria(arr:arreglo, objetivo:elementoBuscado);
        int posicionHash = busquedaHash(arr:arreglo, objetivo:elementoBuscado);

        if (posicionLineal != -1) {
            System.out.println("Búsqueda lineal: El elemento " + elementoBuscado + " está en la posición " + posicionLineal);
        } else {
            System.out.println("Búsqueda lineal: El elemento " + elementoBuscado + " no se encontró");
        }

        if (posicionBinaria != -1) {
            System.out.println("Búsqueda binaria: El elemento " + elementoBuscado + " está en la posición " + posicionBinaria);
        } else {
            System.out.println("Búsqueda binaria: El elemento " + elementoBuscado + " no se encontró");
        }

        if (posicionHash != -1) {
            System.out.println("Búsqueda de hash: El elemento " + elementoBuscado + " está en la posición " + posicionHash);
        } else {
            System.out.println("Búsqueda de hash: El elemento " + elementoBuscado + " no se encontró");
        }
    }
```

Métodos de Búsqueda

```java
// Implementación de búsqueda binaria
public static int busquedaBinaria(int[] arr, int objetivo) {
    int izquierda = 0;
    int derecha = arr.length - 1;
    while (izquierda <= derecha) {
        int medio = izquierda + (derecha - izquierda) / 2;
        if (arr[medio] == objetivo) {
            return medio;
        }
        if (arr[medio] < objetivo) {
            izquierda = medio + 1;
        } else {
            derecha = medio - 1;
        }
    }
    return -1; // No se encontró el elemento
}

// Implementación de búsqueda de hash (solo para fines ilustrativos)
public static int busquedaHash(int[] arr, int objetivo) {
    // En este ejemplo simple, asumimos que el arreglo es una tabla hash
    // y que la función hash devuelve directamente el índice
    int indice = objetivo % arr.length;
    if (arr[indice] == objetivo) {
        return indice;
    } else {
        return -1; // No se encontró el elemento
    }
}
```

Métodos de Búsqueda

```
Output - Run (EjemploMetodosBusqueda)
cd C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploMetodosBusqueda; "JAVA_HOME=C:\\Program Files\\Java\\jdk-20" cmd /c "\"C:\\P
Scanning for projects...

----------------< com.mycompany:EjemploMetodosBusqueda >----------------
Building EjemploMetodosBusqueda 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ EjemploMetodosBusqueda ---
skip non existing resourceDirectory C:\Users\Harol\OneDrive\Documentos\NetBeansProjects\EjemploMetodosBusqueda\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ EjemploMetodosBusqueda ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ EjemploMetodosBusqueda ---
Búsqueda lineal: El elemento 11 está en la posición 5
Búsqueda binaria: El elemento 11 está en la posición 5
Búsqueda de hash: El elemento 11 no se encontró
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.692 s
Finished at: 2023-08-15T15:29:02-05:00
------------------------------------------------------------------------
```

¿Preguntas?

CONCLUSIONES