
Politécnico
Internacional

Politécnico Internacional

Java

ESTRUCTURA DE DATOS

Politécnico Internacional

Java

“Celebra cada pequeña victoria.”

AGENDA

- Conceptos Java
- BetPlay
- Conclusiones

LISTAS ABSTRACTAS

Una "lista abstracta" se refiere a un concepto en programación y ciencias de la computación. Es una estructura de datos que representa una **colección ordenada de elementos en la que cada elemento tiene una posición relativa**. Una "lista abstracta" no se refiere a una implementación concreta en un lenguaje de programación específico, sino que proporciona una descripción de las **operaciones que se pueden realizar en una lista** y sus propiedades generales.

LISTAS ABSTRACTAS

Las operaciones básicas que se pueden realizar en una lista abstracta incluyen:

1. **Inserción**: Agregar un nuevo elemento a la lista en una posición específica.
2. **Eliminación**: Retirar un elemento de la lista en una posición específica.
3. **Acceso**: Obtener el elemento en una posición particular.
4. **Búsqueda**: Encontrar la posición de un elemento específico en la lista.
5. **Longitud**: Obtener la cantidad total de elementos en la lista.

LISTAS ABSTRACTAS

Las listas abstractas son útiles porque **proporcionan una manera de estructurar y manipular conjuntos de datos de manera eficiente**. Pueden utilizarse para implementar una variedad de estructuras de datos más específicas, como arreglos, listas enlazadas, pilas y colas. La elección de la implementación concreta dependerá de los requisitos particulares del problema y las operaciones que necesiten realizarse con mayor frecuencia.

LISTAS ABSTRACTAS

```java
// Definición de la interfaz para la Lista Abstracta
interface ListaAbstracta<E> {
    void agregar(E elemento);  // Método para agregar un elemento a la lista
    E obtener(int indice);     // Método para obtener un elemento en un índice específico
    int tamaño();              // Método para obtener el tamaño de la lista
}

// Implementación de la Lista Abstracta usando un arreglo
class ListaArreglo<E> implements ListaAbstracta<E> {
    private Object[] elementos; // Arreglo para almacenar los elementos
    private int tamaño;         // Variable para rastrear el tamaño actual

    public ListaArreglo(int capacidadInicial) {
        elementos = new Object[capacidadInicial];
        tamaño = 0;
    }

    @Override
    public void agregar(E elemento) {
        // Verificar si es necesario aumentar la capacidad del arreglo
        if (tamaño == elementos.length) {
            Object[] nuevoArreglo = new Object[elementos.length * 2];
            System.arraycopy(src:elementos, srcPos: 0, dest:nuevoArreglo, destPos:0, length: tamaño);
            elementos = nuevoArreglo;
        }

        // Agregar el elemento al final de la lista
        elementos[tamaño] = elemento;
        tamaño++;
    }

    @Override
    public E obtener(int indice) {
        // Verificar si el índice está dentro de los límites
        if (indice < 0 || indice >= tamaño) {
            throw new IndexOutOfBoundsException(s: "Índice fuera de los límites");
        }
```

LISTAS ABSTRACTAS

```
        // Obtener el elemento en el índice especificado
        @SuppressWarnings("unchecked")
        E elemento = (E) elementos[indice];
        return elemento;
    }

    @Override
    public int tamaño() {
        return tamaño;
    }
}

public class EjemploListaAbstracta {
    public static void main(String[] args) {
        // Crear una lista abstracta implementada con arreglo
        ListaAbstracta<String> lista = new ListaArreglo<>(capacidadInicial: 5);

        // Agregar elementos a la lista
        lista.agregar(elemento: "Goku");
        lista.agregar(elemento: "Vegeta");
        lista.agregar(elemento: "Trunks");

        // Obtener y mostrar elementos
        System.out.println(x: "|Politecnico Internacional | Ejemplo Lista Abstracta|");
        System.out.println("Personaje en índice 1: " + lista.obtener(indice: 1));
        System.out.println("Tamaño de la lista de personajes: " + lista.tamaño());
    }
}
```

LISTAS ABSTRACTAS

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

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

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

] --- exec:3.1.0:exec (default-cli) @ EjemploListaAbstracta ---
  |Politecnico Internacional | Ejemplo Lista Abstracta|
  Personaje en indice 1: Vegeta
· Tamaño de la lista de personajes: 3
  ------------------------------------------------------------------------
  BUILD SUCCESS
  ------------------------------------------------------------------------
  Total time:  1.116 s
  Finished at: 2023-08-14T06:53:51-05:00
  ------------------------------------------------------------------------
```

DIAGRAMA DE LISTAS SENCILLAS

Un "diagrama de listas sencillas", también conocido como "diagrama de enlaces simples" o "diagrama de listas enlazadas", es una **representación gráfica que ilustra cómo los elementos están organizados y conectados en una estructura de datos de lista enlazada**. En una lista enlazada, cada elemento (nodo) contiene un valor y una referencia (enlace) al siguiente elemento en la lista.

DIAGRAMA DE LISTAS SENCILLAS

Cada nodo en un diagrama de listas sencillas tiene al menos dos partes clave:

1.Valor del Nodo: Este es el dato almacenado en el nodo. Puede ser cualquier tipo de dato, desde números hasta objetos más complejos.

1.Referencia al Siguiente Nodo: Esta es una flecha o línea que conecta el nodo actual con el siguiente nodo en la lista. Esta referencia permite la navegación de un nodo a otro en la lista.

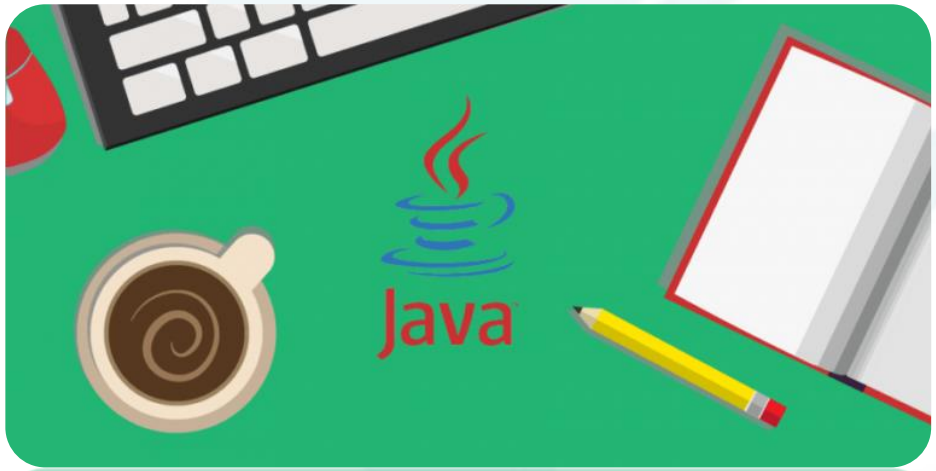


DIAGRAMA DE LISTAS SENCILLAS

Los diagramas de listas sencillas son útiles para **visualizar la estructura y el flujo de datos en una lista enlazada**. Ayudan a comprender cómo los elementos están conectados entre sí y cómo se puede acceder a los datos a lo largo de la lista. Además, estos diagramas son esenciales para comprender cómo funcionan las operaciones de inserción, eliminación y acceso en una lista enlazada.

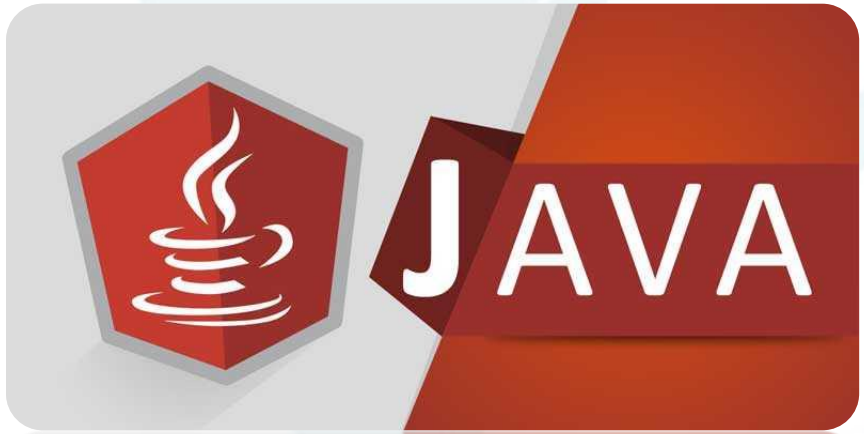


DIAGRAMA DE LISTAS SENCILLAS

Un diagrama de listas sencillas es una herramienta visual que representa gráficamente la organización y las conexiones entre los elementos de una lista enlazada. Ayuda a comprender la estructura y el funcionamiento de las listas enlazadas, que son una forma importante de estructura de datos en programación.

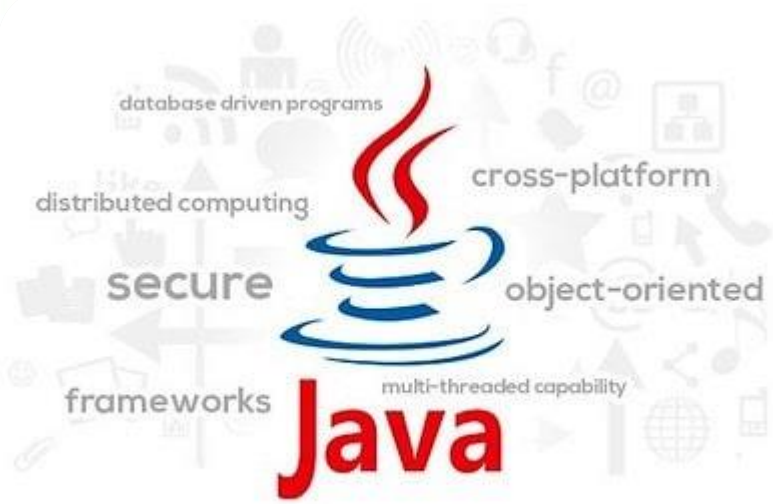
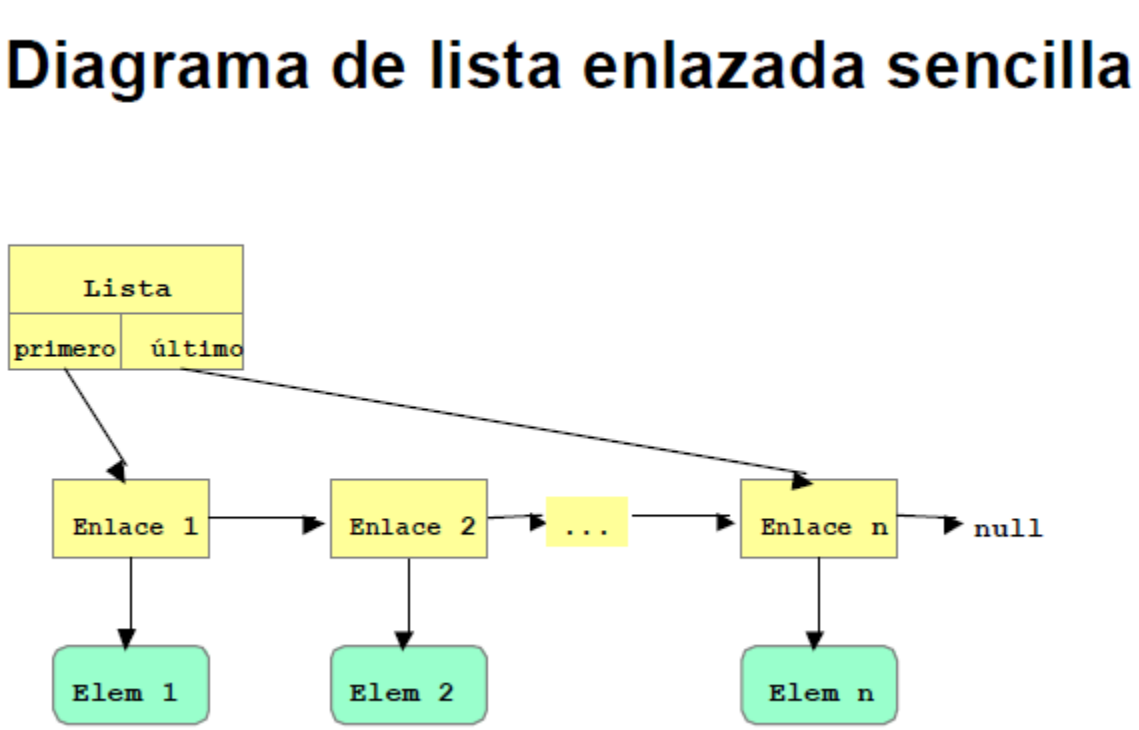


DIAGRAMA DE LISTAS SENCILLAS



Clase de Interna de Enlace

Una **"clase interna de enlace"** es una clase que se define dentro de otra clase en el lenguaje de programación Java. También se le conoce como **"clase interna"** o **"clase anidada".** Esta clase interna generalmente se utiliza para encapsular y representar cierta funcionalidad relacionada con la clase externa. En el contexto de las estructuras de datos como las listas enlazadas, **la clase interna de enlace se utiliza comúnmente para representar los nodos individuales de la lista.**

Clase de Interna de Enlace

En el caso específico de una lista enlazada, una **"clase interna de enlace"** se utiliza para crear los nodos individuales de la lista. Cada nodo contendrá un valor y una referencia (enlace) al siguiente nodo en la secuencia

Clase de Interna de Enlace

```java
// Definición de la clase para la Lista Enlazada
public class ListaEnlazada {

    // Clase interna para representar los nodos de la lista
    private class Nodo {
        int valor;       // Valor del nodo
        Nodo siguiente;  // Referencia al siguiente nodo

        Nodo(int valor) {
            this.valor = valor;
            this.siguiente = null;
        }
    }

    private Nodo cabeza; // Puntero al primer nodo en la lista

    // Constructor para inicializar la lista enlazada
    public ListaEnlazada() {
        cabeza = null;
    }

    // Método para agregar un valor al principio de la lista
    public void agregarAlPrincipio(int valor) {
        Nodo nuevoNodo = new Nodo(valor);
        nuevoNodo.siguiente = cabeza;
        cabeza = nuevoNodo;
    }

    // Método para imprimir los valores de la lista
    public void imprimirLista() {
        Nodo actual = cabeza;
        while (actual != null) {
            System.out.print(actual.valor + " ");
            actual = actual.siguiente;
        }
        System.out.println();
    }
```

Clase de Interna de Enlace

```
    public static void main(String[] args) {
        // Crear una lista enlazada
        ListaEnlazada lista = new ListaEnlazada();

        // Agregar valores al principio de la lista
        lista.agregarAlPrincipio(valor: 3);
        lista.agregarAlPrincipio(valor: 2);
        lista.agregarAlPrincipio(valor: 1);

        // Imprimir la lista
        System.out.print(s: "Lista enlazada: ");
        lista.imprimirLista();
    }
}
```

Clase de Interna de Enlace

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

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

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

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

--- exec:3.1.0:exec (default-cli) @ ListaEnlazada ---
|Politecnico Internacional | Lista enlazada |Lista enlazada: 1 2 3
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.672 s
Finished at: 2023-08-14T07:33:06-05:00
------------------------------------------------------------------------
```

Listas Genéricas

Una "lista genérica" es una **estructura de datos que puede almacenar elementos de cualquier tipo**, es decir, no está limitada a un tipo de dato específico. En el contexto de la programación orientada a objetos, se utiliza el concepto de "genéricos" para crear estructuras de datos o clases que puedan manejar tipos de datos variables de manera segura.

Listas Genéricas

En Java, por ejemplo, las listas genéricas se implementan utilizando la **interfaz List de la biblioteca estándar** (**java.util.List**). Esta interfaz permite definir listas que pueden contener elementos de cualquier tipo (**clases, tipos primitivos, etc**.) y proporciona mecanismos para asegurar el tipo de datos durante la compilación, lo que reduce los errores en tiempo de ejecución.

Listas Genéricas

Las listas genéricas son útiles porque:

1.Reutilización de código: Una lista genérica puede ser utilizada para almacenar cualquier tipo de dato, lo que significa que no es necesario implementar listas específicas para diferentes tipos de datos.

1.Flexibilidad: Permite manejar colecciones de diferentes tipos de datos de manera homogénea y sin necesidad de conversiones explícitas.

1.Seguridad en tipos de datos: Los genéricos permiten detectar errores de tipo en tiempo de compilación, en lugar de en tiempo de ejecución, lo que conduce a un código más seguro y confiable.

Listas Genéricas

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

public class EjemploListaGenerica {
    public static void main(String[] args) {
        // Crear una lista genérica para almacenar enteros
        List<Integer> listaDeEnteros = new ArrayList<>();

        // Agregar elementos a la lista
        listaDeEnteros.add(e: 10);
        listaDeEnteros.add(e: 20);
        listaDeEnteros.add(e: 30);

        // Imprimir los elementos de la lista
        System.out.println(x: "|Politecnico Internacional | Ejemplo Lista Generica|");
        for (int elemento : listaDeEnteros) {
            System.out.println(x: elemento);
        }
    }
}
```

Listas Genéricas

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

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

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

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

--- exec:3.1.0:exec (default-cli) @ EjemploListaGenerica ---
|Politecnico Internacional | Ejemplo Lista Generica|
10
20
30
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.709 s
Finished at: 2023-08-14T07:54:08-05:00
------------------------------------------------------------------------
```

Listas Genéricas

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

public class EjemploListaGenericaDiversa {
    public static void main(String[] args) {
        // Crear una lista genérica para almacenar diferentes tipos de datos
        List<Object> listaDiversa = new ArrayList<>();

        // Agregar diferentes tipos de elementos a la lista
        listaDiversa.add(e: "Hola");      // Cadena de caracteres
        listaDiversa.add(e: 42);          // Entero
        listaDiversa.add(e: 3.14);        // Punto flotante (double)
        listaDiversa.add(e: true);        // Valor booleano

        // Imprimir los elementos de la lista
        System.out.println(x: "|Politecnico Internacional | Ejemplo Lista Generica Diversa|");
        for (Object elemento : listaDiversa) {
            System.out.println(x: elemento);
        }
    }
}
```

Listas Genéricas

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

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

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

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

--- exec:3.1.0:exec (default-cli) @ EjemploListaGenericaDiversa ---
|Politecnico Internacional | Ejemplo Lista Generica Diversa|
Hola
42
3.14
true
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.671 s
Finished at: 2023-08-14T07:55:08-05:00
------------------------------------------------------------------------
```

Listas Tipadas

Una "**lista tipada**" es una estructura de datos que permite almacenar elementos de un tipo de dato específico. En otras palabras, es una lista en la que todos los elementos deben ser del mismo tipo. En el contexto de la programación, se utiliza el término "**tipado**" para referirse a la especificación de un tipo de dato particular para una variable, parámetro, o en este caso, para los elementos de una lista.

Listas Tipadas

Las listas tipadas son muy comunes en lenguajes de programación modernos, como Java, C#, Python (en su forma de listas) y otros. La principal ventaja de las listas tipadas es que proporcionan seguridad en el tipo de datos durante la compilación, lo que significa que cualquier intento de agregar un elemento de un tipo no compatible generará un error en tiempo de compilación en lugar de en tiempo de ejecución.

Listas Tipadas

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

public class EjemploListaTipada {
    public static void main(String[] args) {
        // Crear una lista tipada para almacenar cadenas de caracteres
        List<String> listaDeCadenas = new ArrayList<>();

        // Agregar elementos a la lista
        listaDeCadenas.add(e: "Harol");
        listaDeCadenas.add(e: "Torres");
        listaDeCadenas.add(e: "Harol.Torres@poli.edu.co");
        listaDeCadenas.add(e: "14 de Agosto de 2023");

        // No se permitiría agregar un entero en una lista tipada de cadenas
        // listaDeCadenas.add(42); // Esto causaría un error de compilación

        // Imprimir los elementos de la lista
         System.out.println(x: "|Politecnico Internacional | Ejemplo Lista Tipada|");
        for (String cadena : listaDeCadenas) {
            System.out.println(x: cadena);
        }
    }
}
```

Listas Tipadas

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

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

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

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

--- exec:3.1.0:exec (default-cli) @ EjemploListaTipada ---
|Politecnico Internacional | Ejemplo Lista Tipada|
Harol
Torres
Harol.Torres@poli.edu.co
14 de Agosto de 2023
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.658 s
Finished at: 2023-08-14T08:17:07-05:00
------------------------------------------------------------------------
```

Listas Posición Ordinal

Una **"lista posición ordinal"** es una **estructura de datos en la que cada elemento está asociado a una posición numérica o índice que refleja su orden en la secuencia**. Básicamente, es una colección de elementos en la que cada elemento tiene un número que indica su posición en relación con los otros elementos. En otras palabras, es similar a una lista normal, pero se enfoca en el orden de los elementos en lugar de en sus valores.

Listas Posición Ordinal

Las listas de posición ordinal **son útiles en situaciones donde el orden de los elementos es relevante y es necesario realizar operaciones basadas en esa secuencia**. Se utilizan comúnmente para llevar un seguimiento del orden en que se ingresaron o procesaron los elementos.

Listas Posición Ordinal

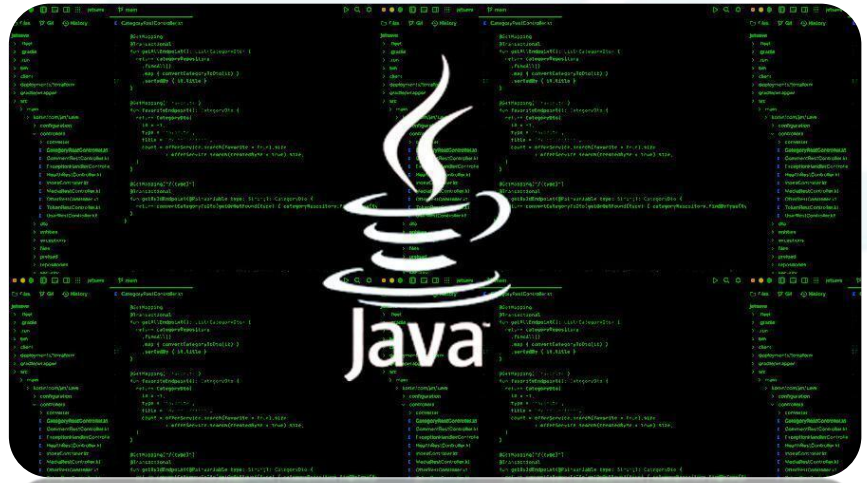
```java
import java.util.ArrayList;
import java.util.List;

public class ListaPosicionOrdinal {
    public static void main(String[] args) {
        // Crear una lista de nombres en posición ordinal
        List<String> listaNombres = new ArrayList<>();

        // Agregar nombres a la lista
        listaNombres.add(e: "Goku");
        listaNombres.add(e: "Vegeta");
        listaNombres.add(e: "Trunks");

        // Imprimir la lista de nombres junto con su posición ordinal
        System.out.println(x: "|Politecnico Internacional | Ejemplo Lista Posicion Ordinal|");
        for (int i = 0; i < listaNombres.size(); i++) {
            String nombre = listaNombres.get(index: i);
            int posicion = i + 1; // Sumar 1 para que las posiciones comiencen desde 1
            System.out.println("Posición " + posicion + ": " + nombre);
        }
    }
}
```

Listas Posición Ordinal

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

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

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

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

--- exec:3.1.0:exec (default-cli) @ ListaPosicionOrdinal ---
|Politecnico Internacional | Ejemplo Lista Posicion Ordinal|
Posición 1: Goku
Posición 2: Vegeta
Posición 3: Trunks
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.689 s
Finished at: 2023-08-14T08:24:11-05:00
------------------------------------------------------------------------
```

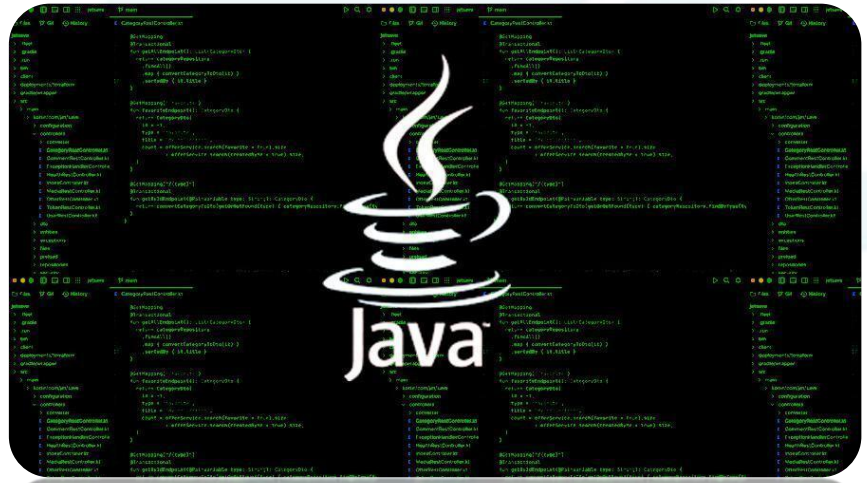
Métodos de Iteracción

Los **"métodos de iteración"** son aquellos métodos que permiten recorrer y acceder a los elementos de una colección de datos, como listas, conjuntos, mapas u otras estructuras de datos similares. **Estos métodos son utilizados para navegar a través de los elementos contenidos en una colección y realizar operaciones en cada uno de ellos**. En esencia, los métodos de iteración proporcionan una forma conveniente de acceder y procesar los elementos de una colección uno por uno.



Métodos de Iteracción

Los métodos de iteración son esenciales para muchas tareas en programación, ya que permiten realizar acciones como búsqueda, filtrado, transformación o cualquier otra operación que implique interactuar con cada elemento de una colección. Simplifican el proceso de recorrer una colección sin requerir bucles explícitos y manejan internamente la lógica de navegación a través de los elementos.

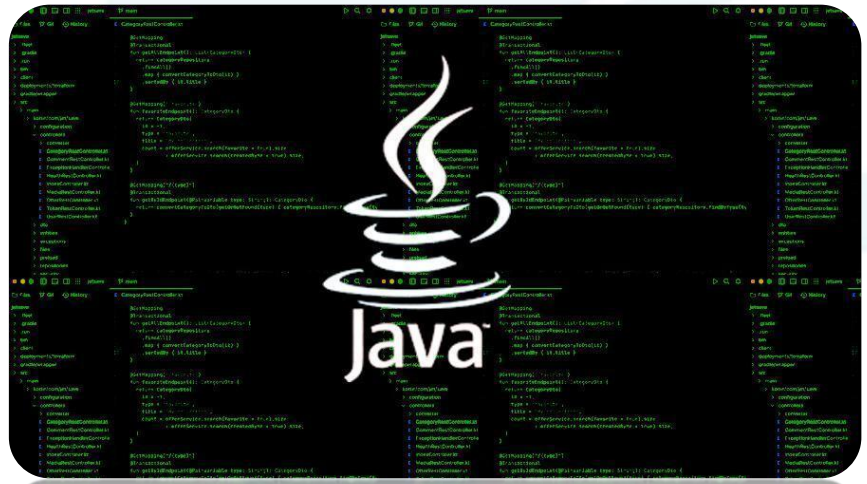


Métodos de Iteracción

Algunos ejemplos comunes de métodos de iteración en diferentes lenguajes de programación incluyen:

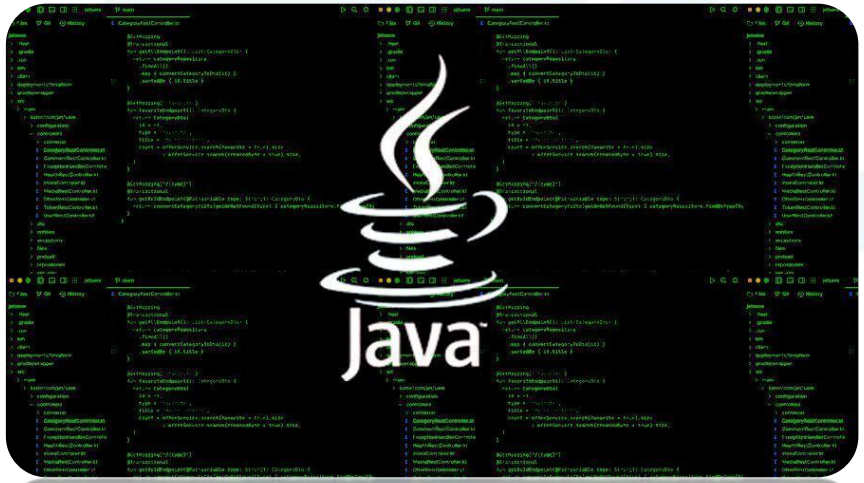
Java: forEach, iterator, Stream API
Python: for-in, map, filter
JavaScript: forEach, map, filter

Estos métodos de iteración permiten realizar acciones en cada elemento de una colección de manera más compacta y legible que implementar bucles de iteración tradicionales.



Métodos de Iteracción

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

public class EjemploMetodosIteracion {
    public static void main(String[] args) {
        // Crear una lista de números
        List<Integer> numeros = new ArrayList<>();
        numeros.add(e: 1);
        numeros.add(e: 2);
        numeros.add(e: 3);
        numeros.add(e: 4);
        numeros.add(e: 5);

        // Utilizar el método forEach para imprimir el doble de cada número
        System.out.println(x: "|Politecnico Internacional | Ejemplo Metodos de Iteraccion ForEach|");
        System.out.println(x: "Números originales y su doble:");
        numeros.forEach(numero -> {
            int doble = numero * 2;
            System.out.println(numero + " -> " + doble);
        });
    }
}
```



Métodos de Iteracción

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

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

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

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

--- exec:3.1.0:exec (default-cli) @ EjemploMetodosIteracion ---
|Politecnico Internacional | Ejemplo Metodos de Iteraccion ForEach|
Números originales y su doble:
1 -> 2
2 -> 4
3 -> 6
4 -> 8
5 -> 10
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.674 s
Finished at: 2023-08-14T08:31:44-05:00
------------------------------------------------------------------------
```

Pila (Push / Pop)

Una "pila" (también conocida como "stack" en inglés) es una estructura de datos lineal que sigue el principio "último en entrar, primero en salir" (LIFO, por sus siglas en inglés: "Last In, First Out"). Esto significa que el último elemento agregado a la pila es el primero en ser retirado. Las operaciones básicas en una pila son "push" y "pop".

- **Push**: Agregar un elemento en la parte superior de la pila.

- **Pop**: Retirar el elemento superior de la pila.

Pila (Push / Pop)

Además de estas operaciones básicas, también es común tener una operación para verificar si la pila está vacía y para obtener el elemento en la parte superior sin retirarlo.

Las pilas son ampliamente utilizadas en programación y algoritmos debido a su naturaleza LIFO. Pueden utilizarse para rastrear el estado de llamadas a funciones (control de flujo), para deshacer operaciones, para evaluar expresiones matemáticas, y para resolver problemas relacionados con la reversión de secuencias.

Pila (Push / Pop)

una pila es una estructura de datos que sigue la regla LIFO, donde el último elemento agregado es el primero en ser retirado.

Las operaciones principales son "push" para agregar y "pop" para retirar elementos. Las pilas son ampliamente utilizadas en la programación y en la resolución de problemas donde el orden inverso es relevante.

Pila (Push / Pop)

```java
import java.util.Stack;

public class EjemploPila {
    public static void main(String[] args) {
        // Crear una pila para números enteros
        Stack<Integer> pila = new Stack<>();

        // Agregar elementos a la pila (operación "push")
        pila.push(item: 10);
        pila.push(item: 20);
        pila.push(item: 30);
        System.out.println(x: "|Politecnico Internacional | Ejemplo Pila Push Pop|");
        System.out.println("Pila después de agregar elementos: " + pila);

        // Retirar elementos de la pila (operación "pop")
        int elemento1 = pila.pop();
        int elemento2 = pila.pop();

        System.out.println("Elemento retirado: " + elemento1);
        System.out.println("Elemento retirado: " + elemento2);

        System.out.println("Pila después de retirar elementos: " + pila);
    }
}
```

Pila (Push / Pop)

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

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

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

] --- exec:3.1.0:exec (default-cli) @ EjemploPila ---
  |Politecnico Internacional | Ejemplo Pila Push Pop|
  Pila después de agregar elementos: [10, 20, 30]
  Elemento retirado: 30
  Elemento retirado: 20
- Pila después de retirar elementos: [10]
  ------------------------------------------------------------------------
  BUILD SUCCESS
  ------------------------------------------------------------------------
  Total time:  0.663 s
  Finished at: 2023-08-14T08:47:26-05:00
  ------------------------------------------------------------------------
```

¿Preguntas?

CONCLUSIONES