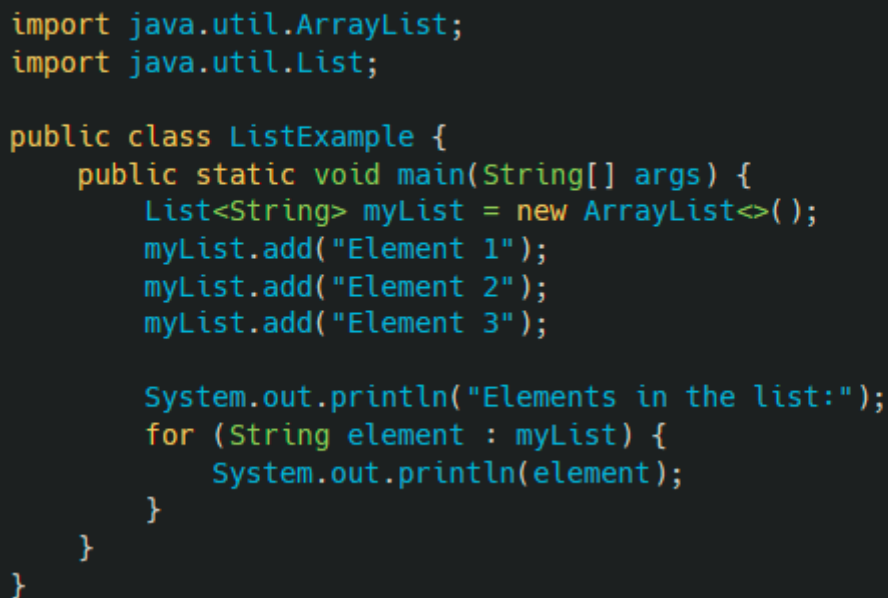


Java Collections Framework: Una Investigación Exhaustiva

El Framework de Colecciones de Java es una parte integral del lenguaje Java y proporciona implementaciones eficientes de estructuras de datos comúnmente utilizadas. Estas estructuras de datos se agrupan en interfaces y clases bajo el paquete `java.util`. A continuación, se presenta una investigación detallada sobre varias interfaces y clases fundamentales dentro del Framework de Colecciones.

List

- Definición: La interfaz `List` representa una colección ordenada que permite elementos duplicados. Proporciona métodos para acceder, insertar, actualizar y eliminar elementos de la lista.



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

public class ListExample {
    public static void main(String[] args) {
        List<String> myList = new ArrayList<>();
        myList.add("Element 1");
        myList.add("Element 2");
        myList.add("Element 3");

        System.out.println("Elements in the list:");
        for (String element : myList) {
            System.out.println(element);
        }
    }
}
```

- Implementaciones notables: `ArrayList`, `LinkedList`, `Vector`.

Set

- Definición: La interfaz `Set` representa una colección que no permite elementos duplicados. Garantiza que no hay elementos repetidos y no define un orden específico.

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

public class SetExample {
    public static void main(String[] args) {
        Set<String> mySet = new HashSet<>();
        mySet.add("Element 1");
        mySet.add("Element 2");
        mySet.add("Element 1"); // Duplicates are not allowed

        System.out.println("Elements in the set:");
        for (String element : mySet) {
            System.out.println(element);
        }
    }
}
```

- Implementaciones notables: HashSet, LinkedHashSet, TreeSet.

SortedSet

- Definición: La interfaz SortedSet extiende la interfaz Set para proporcionar un conjunto ordenado de elementos. Los elementos se ordenan según su orden natural o mediante un comparador proporcionado al crear el conjunto.

```
import java.util.TreeSet;

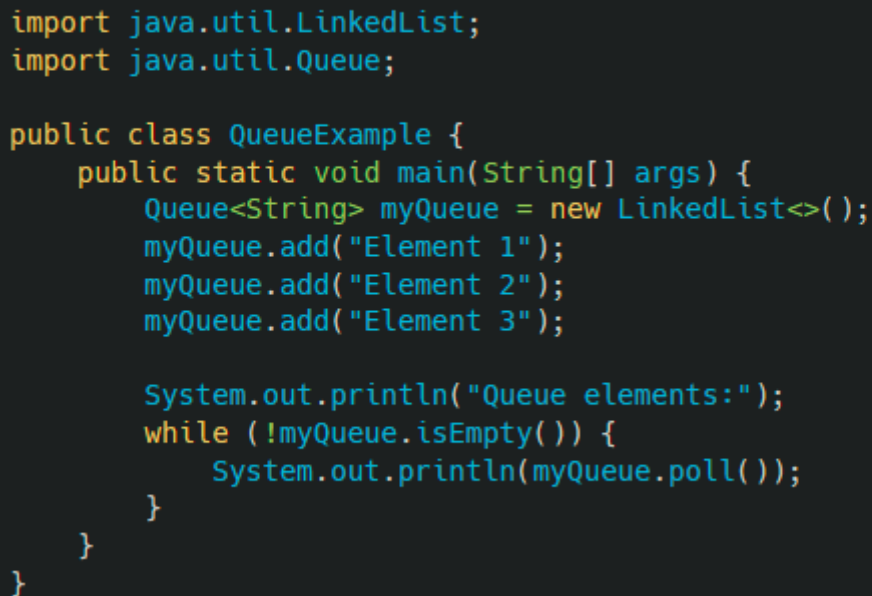
public class SortedSetExample {
    public static void main(String[] args) {
        TreeSet<String> sortedSet = new TreeSet<>();
        sortedSet.add("Banana");
        sortedSet.add("Apple");
        sortedSet.add("Orange");

        System.out.println("SortedSet elements:");
        for (String element : sortedSet) {
            System.out.println(element);
        }
    }
}
```

- Implementaciones notables: TreeSet.

Queue

- Definición: La interfaz Queue representa una cola, siguiendo la política de tipo FIFO (First-In-First-Out). Los elementos se agregan al final y se eliminan desde el principio de la cola.



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

public class QueueExample {
    public static void main(String[] args) {
        Queue<String> myQueue = new LinkedList<>();
        myQueue.add("Element 1");
        myQueue.add("Element 2");
        myQueue.add("Element 3");

        System.out.println("Queue elements:");
        while (!myQueue.isEmpty()) {
            System.out.println(myQueue.poll());
        }
    }
}
```

- Implementaciones notables: LinkedList, PriorityQueue.

Deque

- Definición: La interfaz Deque extiende Queue para representar una cola doble-ended, lo que significa que los elementos se pueden agregar o quitar desde ambos extremos de la cola.

```
import java.util.ArrayDeque;
import java.util.Deque;

public class DequeExample {
    public static void main(String[] args) {
        Deque<String> myDeque = new ArrayDeque<>();
        myDeque.add("Element 1");
        myDeque.addFirst("Element 2");
        myDeque.addLast("Element 3");

        System.out.println("Deque elements (from first to last):");
        while (!myDeque.isEmpty()) {
            System.out.println(myDeque.pollFirst());
        }
    }
}
```

- Implementaciones notables: ArrayDeque, LinkedList.

Map

- Definición: La interfaz Map representa una colección de pares clave-valor, donde cada clave está asociada a un único valor. No puede contener claves duplicadas.

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

public class MapExample {
    public static void main(String[] args) {
        Map<String, Integer> myMap = new HashMap<>();
        myMap.put("One", 1);
        myMap.put("Two", 2);
        myMap.put("Three", 3);

        System.out.println("Values in the map:");
        for (Map.Entry<String, Integer> entry : myMap.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}
```

- Implementaciones notables: HashMap, LinkedHashMap, TreeMap.

SortedMap

- Definición: La interface SortedMap extiende la interfaz Map para proporcionar un mapa ordenado de elementos. Los elementos se ordenan según el orden natural de las claves o mediante un comparador.

A screenshot of a code editor with a dark background and light-colored text. The code is in Java and demonstrates the use of SortedMap. It includes an import statement for TreeMap, a class definition SortedMapExample, and a main method. The main method creates a SortedMap, adds three entries with keys "One", "Two", and "Three", and then iterates over the keys to print them in sorted order. The code is as follows:

```
import java.util.TreeMap;

public class SortedMapExample {
    public static void main(String[] args) {
        TreeMap<String, Integer> sortedMap = new TreeMap<>();
        sortedMap.put("One", 1);
        sortedMap.put("Three", 3);
        sortedMap.put("Two", 2);

        System.out.println("SortedMap elements:");
        for (String key : sortedMap.keySet()) {
            System.out.println(key + ": " + sortedMap.get(key));
        }
    }
}
```

- Implementaciones notables: TreeMap.

Iterator

- Definición: La interfaz Iterator proporciona un mecanismo para recorrer colecciones de elementos de manera secuencial. Permite el acceso a los elementos sin exponer la estructura interna de la colección.

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

public class IteratorExample {
    public static void main(String[] args) {
        List<String> myList = new ArrayList<>();
        myList.add("Element 1");
        myList.add("Element 2");
        myList.add("Element 3");

        Iterator<String> iterator = myList.iterator();
        System.out.println("Elements using Iterator:");
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

- Implementación estándar: Utilizado en diversas colecciones para permitir el recorrido.

ListIterator

- Definición: La interfaz ListIterator extiende Iterator y proporciona capacidades adicionales para recorrer listas en ambas direcciones y modificar elementos durante el recorrido.

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

public class ListIteratorExample {
    public static void main(String[] args) {
        List<String> myList = new ArrayList<>();
        myList.add("Element 1");
        myList.add("Element 2");
        myList.add("Element 3");

        ListIterator<String> listIterator = myList.listIterator();
        System.out.println("Elements using ListIterator (forward):");
        while (listIterator.hasNext()) {
            System.out.println(listIterator.next());
        }

        System.out.println("Elements using ListIterator (backward):");
        while (listIterator.hasPrevious()) {
            System.out.println(listIterator.previous());
        }
    }
}
```

- Implementación estándar: Se utiliza principalmente con listas, como ArrayList y LinkedList.

El Framework de Colecciones de Java es esencial para el desarrollo de aplicaciones, proporcionando flexibilidad, eficiencia y funcionalidades para el manejo de datos. La elección de la implementación adecuada depende de los requisitos específicos de cada escenario de uso, como la eficiencia en términos de acceso, inserción y eliminación, así como la necesidad de mantener o no un orden específico en la colección.