JAVA

Fundamental Components of the Java Collections Framework

1. **Containers:**
    - Java offers containers like Lists (ArrayList, LinkedList), Sets (HashSet, TreeSet), Maps (HashMap, TreeMap), and others.
    - These are akin to STL's vectors, lists, maps, and sets. They are used for efficient data storage and manipulation.
2. **Algorithms:**
    - Java provides a range of algorithms, particularly through the Collections and Arrays classes.
    - These include sorting (Collections.sort()), searching (Collections.binarySearch()), and shuffling (Collections.shuffle()), similar to STL's algorithms.
    - Examples :

```java
// Java program to demonstrate working of Collections.
// binarySearch()
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class GFG {
    public static void main(String[] args)
    {
        List<Integer> al = new ArrayList<Integer>();
        al.add(1);
        al.add(2);
        al.add(3);
        al.add(10);
        al.add(20);

        // 10 is present at index 3.
        int index = Collections.binarySearch(al, 10);
        System.out.println(index);

        // 13 is not present. 13 would have been inserted
        // at position 4. So the function returns (-4-1)
        // which is -5.
        index = Collections.binarySearch(al, 13);
        System.out.println(index);
    }
}
```

Output

```
3
-5
```

```java
// Java program to demonstrate working of Collections.sort()
import java.util.*;

public class Collectionsorting
{
    public static void main(String[] args)
    {
        // Create a list of strings
        ArrayList<String> al = new ArrayList<String>();
        al.add("Geeks For Geeks");
        al.add("Friends");
        al.add("Dear");
        al.add("Is");
        al.add("Superb");

        /* Collections.sort method is sorting the
        elements of ArrayList in ascending order. */
        Collections.sort(al);

        // Let us print the sorted list
        System.out.println("List after the use of" +
                           " Collection.sort() :\n" + al);
    }
}
```

Output

```
List after the use of Collection.sort() :
[Dear, Friends, Geeks For Geeks, Is, Superb]
```

3. **Iterators:**
    - Java's iterators, like the Iterator and ListIterator, are used to traverse through collections.
    - They are specifically designed to work with Java's collection types.
4. **Function Objects:**
    - Java utilizes functional interfaces, especially with the advent of lambdas in Java 8, to pass behaviour to algorithms.
    - These are similar to C++'s functions but leverage Java's lambda expressions and method references.

Arrays vs. ArrayLists in Java

Aspect	Arrays	ArrayLists
Size Flexibility	Fixed size; the size is determined at the time of creation and cannot change.	Dynamic size; ArrayLists can grow and shrink at runtime as needed.
Initialization	Can be initialized with a fixed set of elements or predefined size.	Supports dynamic initialization; elements can be added or removed dynamically.
Functions	Limited built-in functionality; requires manual effort for resizing and advanced operations.	Extensive built-in functions for adding, removing, searching, and resizing, among others.

1.Pair in Java

- In Java, while there isn't a direct built-in equivalent to C++'s std::pairin the standard Java libraries up until Java 7, the concept of pairing two values together is often implemented using classes from the standard library itself.
- An alternative to using the Pair is the SimpleEntry class from the java.util.AbstractMap package, available as part of Java's standard library.

AbstractMap.SimpleEntry Class - Introduction

- The SimpleEntry class functions similarly to the std::pair in C++.
- It typically stores two objects or values together, such as a key and a value in a map.
- This class is handy in scenarios where you need to return two values from a method or when you want to maintain a relationship between two related objects without creating a separate class.

Key Features and Methods of AbstractMap.SimpleEntry

- **Creation:**
  - To create an instance of SimpleEntry, you use its constructor, passing the two values you want to pair.
  - Syntax ,

    AbstractMap.SimpleEntry<KeyType, ValueType> variableName = new AbstractMap.SimpleEntry<>(key, value);
- **Accessing Values:**
  - **getKey()** : Returns the first element of the entry, typically used as the key.
  - **getValue()** : Returns the second element of the entry, typically used as the value.
- **Equality Check:**
  - **equals()** : Compares two SimpleEntry objects based on the values they store.
- **String Representation:**
  - **toString()** : Returns a string representation of the entry.
- **Hash Code:**
  - **hashCode()** : Generates a hash code for the entry, useful in collections that use hashing (like HashMap, HashSet).

Example :

```
import java.util.AbstractMap.SimpleEntry;
public class Main {
   public static void main(String[] args) {
      SimpleEntry<Integer, String> entry = new SimpleEntry<>(1, "One");
      // Accessing the values
      Integer key = entry.getKey();
      String value = entry.getValue();
      // Printing the values
      System.out.println("Key: " + key + ", Value: " + value); // key -> 1 and value -> "One"
      // Equality Check
      SimpleEntry<Integer, String> entry2 = new SimpleEntry<>(1, "One");
      System.out.println("Is entry equal to entry2? " + entry.equals(entry2)); // Two entries are equals
-> true
      // String Representation
      System.out.println("String representation: " + entry.toString()); // String -> "1=One"
      // Hash Code
      System.out.println("Hash code: " + entry.hashCode()); // Hash Code of entry is 79431
   }
}
```

Explanation:

- **Creating a SimpleEntry:**
    - A SimpleEntry is instantiated with two values: an integer 1 and a string "One". The integer acts as a key, and the string is the value.
- **Accessing the Values:**
    - The getKey() method retrieves the key (1 in this case), and getValue() retrieves the associated value ("One").
- **Checking Equality:**
    - Another SimpleEntry object is created and compared to the first.
      The equals() method checks if both entries have the same key and value.
- **String Representation:**
    - The toString() method provides a string representation of the entry, typically displaying its contents.
- **Hash Code:**
    - The hashCode() method generates a hash code for the entry, which is useful for hashing algorithms in data structures like HashMap.

2.ArrayList in Java

- ArrayLists are part of the Java Collections Framework and provide a way to use dynamically sized arrays.
- Unlike regular arrays, they offer more flexibility and a variety of useful built-in methods.

Key Features and Methods of ArrayList

1. **Creation:**
    - ArrayLists can be created and initialized in various ways.
    - **Syntax**: **ArrayList<Type> arrayListName = new ArrayList<>();**
2. **Adding Elements:**
    - **add(element)**: Adds an element to the ArrayList.
    - **add(index, element)**: Inserts an element at the specified index.
3. **Accessing Elements:**
    - **get(index)**: Returns the element at the specified index.
4. **Modifying Elements:**
    - **set(index, element)**
5. **Size and Capacity:**: Updates the element at the specified index.
    - **size()**: Returns the number of elements in the ArrayList.
6. **Removing Elements:**
    - **remove(index):** Removes the element at the specified index.
    - **remove(Object)**: Removes the first occurrence of the specified element.
7. **Other Useful Methods:**
    - **clear()**: Removes all elements from the ArrayList.
    - **isEmpty()**: Returns trueif the ArrayList is empty.

Example :

```
import java.util.ArrayList;
public class Main {
   public static void main(String[] args) {
      // Creating an ArrayList
      ArrayList<Integer> list = new ArrayList<>();
      // Adding elements
      list.add(10);
      list.add(20);
      list.add(30);
      // Accessing elements
      System.out.println("Element at index 1: " + list.get(1)); // 20
      // Modifying elements
      list.set(1, 25);
      // Iterating over ArrayList
      System.out.println("Elements in list:");
      for (int elem : list) {
         System.out.println(elem);
      }
      // Size of ArrayList
      System.out.println("Size of list: " + list.size());
      // Removing elements
      list.remove(Integer.valueOf(10));
      list.remove(0); // Removes element at index 0
      // Check if ArrayList is empty
      System.out.println("Is list empty? " + list.isEmpty());
      // Clearing the ArrayList
      list.clear();
   }
}
```

- In this example, an ArrayList of integers is created, and elements are added, accessed, modified, and removed.
- The ArrayList class offers a variety of methods for managing a dynamic array, making it a versatile data structure in Java.

3. ArrayList of SimpleEntry(Pair) in Java

- It's important to note that AbstractMap.SimpleEntry can be used in Java to create a pair of two values.
- You can use Java's ArrayList to create a dynamic list of these entries.

Example:

```
import java.util.AbstractMap.SimpleEntry;
import java.util.ArrayList;
public class Main {
   public static void main(String[] args) {
      // Creating an ArrayList of SimpleEntries
      ArrayList<SimpleEntry<Integer, String>> listOfEntries = new ArrayList<>();
      // Adding entries to the ArrayList
      listOfEntries.add(new SimpleEntry<>(1, "One"));
      listOfEntries.add(new SimpleEntry<>(2, "Two"));
      listOfEntries.add(new SimpleEntry<>(3, "Three"));
      // Iterating over the ArrayList and accessing elements of the SimpleEntry
      for (SimpleEntry<Integer, String> entry : listOfEntries) {
         Integer key = entry.getKey(); // First element of the entry
         String value = entry.getValue(); // Second element of the entry
         System.out.println("Key: " + key + ", Value: " + value);
      }
   }
}
```

In this example:

- An ArrayList of SimpleEntry<Integer, String>is created.
- Entries are added to the list using listOfEntries.add(new SimpleEntry<>(key, value));
- The list is iterated, and the key and value of each entry are accessed using entry.getKey() and entry.getValue().

This approach is analogous to having a vector of pairs in C++ and is useful in scenarios where you need to maintain a list of related value pairs, such as key-value pairs.

Note :

Using AbstractMap.SimpleEntry requires no additional dependencies since it's part of the standard Java library, making it a convenient choice for projects where minimal external dependencies are desired.