7. Program to demonstrate the difference between StringBuffer and StringBuilder.

Aim

To write a Java program to demonstrate the difference between StringBuffer and StringBuilder classes.

Objective

To understand and demonstrate the difference between **StringBuffer** and **StringBuilder** in Java based on **thread safety** and **performance**.

Theory

- **StringBuffer** → Thread-safe (synchronized), slower
- **StringBuilder** → Not thread-safe, faster
- Both are **mutable** (can be changed without creating new objects.
-

Algorithm

1. Create a StringBuffer object.
2. Append strings and display the result.
3. Create a StringBuilder object.
4. Append strings and display the result.
5. Compare behavior and output.

Java Program

```
class StringBufferVsStringBuilder {
    public static void main(String[] args) {
        // StringBuffer example
        StringBuffer sbuf = new StringBuffer("Hello");
        sbuf.append(" World");
        System.out.println("StringBuffer Output: " + sbuf);
        // StringBuilder example
        StringBuilder sbui = new StringBuilder("Hello");
        sbui.append(" World");
        System.out.println("StringBuilder Output: " + sbui);
    }
}
```

Output

```
StringBuffer Output: Hello World
StringBuilder Output: Hello World
```

Conclusion

- Use **StringBuffer** in multi-threaded environments
- Use **StringBuilder** for better performance in single-threaded programs

8. Program to implement Collection Classes

Aim

To write a Java program to implement the ArrayList collection class.

Objective

To implement and understand Java Collection classes:

- ArrayList
- LinkedList
- Vector

8. (a) ArrayList Implementation

Aim

To write a Java program to implement the ArrayList collection class.

Objective

To understand dynamic array behavior using the ArrayList class in Java.

Algorithm

1. Create an ArrayList.
2. Add elements.
3. Display elements.
4. Remove an element.
5. Display updated list.

Java Program

```
import java.util.ArrayList;

class ArrayListDemo {
    public static void main(String[] args) {

        ArrayList<String> list = new ArrayList<>();

        list.add("Java");
        list.add("Python");
        list.add("C++");

        System.out.println("ArrayList Elements: " + list);

        list.remove("Python");
        System.out.println("After Removal: " + list);
    }
}
```

Output

```
ArrayList Elements: [Java, Python, C++]
After Removal: [Java, C++]
```

9. (b) LinkedList Implementation

Aim

To write a Java program to implement the LinkedList collection class.

Objective

To understand linked list operations using the LinkedList class in Java.

Algorithm

1. Create a LinkedList.
2. Add elements.
3. Add element at beginning.
4. Remove an element.
5. Display the list.

Java Program

```
import java.util.LinkedList;

class LinkedListDemo {
    public static void main(String[] args) {

        LinkedList<Integer> list = new LinkedList<>();

        list.add(10);
        list.add(20);
        list.add(30);


 list.addFirst(5);

        System.out.println("LinkedList Elements: " + list);

        list.removeLast();
        System.out.println("After Removal: " + list);


  }
}
```

Output

```
LinkedList Elements: [5, 10, 20, 30]

After Removal: [5, 10, 20]
```

10. (c) Vector Implementation

Aim

To write a Java program to implement the Vector collection class.

Objective

To understand synchronized dynamic arrays using the Vector class in Java.

Algorithm

1. Create a Vector.
2. Add elements.
3. Display elements.
4. Remove element.
5. Display updated vector.

Java Program

```
import java.util.Vector;

class VectorDemo {
   public static void main(String[] args) {

      Vector<String> v = new Vector<>();

      v.add("Apple");
      v.add("Banana");
      v.add("Mango");

      System.out.println("Vector Elements: " + v);


  v.remove("Banana");
      System.out.println("After Removal: " + v);
   }
}
```

Output

```
Vector Elements: [Apple, Banana, Mango]
After Removal: [Apple, Mango]
```

Comparison Summary

Feature	ArrayList	LinkedList	Vector
Thread-safe	No	No	Yes
Performance	Fast	Slower	Slowest
Data Structure	Dynamic Array	Doubly Linked List	Dynamic Array
Legacy	No	No	Yes