CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics
List interface

List in Java

➲ A **list in Java** is a collection for storing elements in sequential order. Sequential order means the first element, followed by the second element, followed by the third element, and so on.

A good realtime example of a list is a line of train bogies on a track: To get to the fourth bogie from the first bogie, we have to go through the second and third bogies in that order.

➲ Java list is a sub-interface of the collection interface that is available in java.util package. Sub interface means an interface that extends another interface is called sub interface. Here, the list interface extends the collection interface.

The general declaration of list interface in java is as follows:

```
public interface List<E> extends Collection<E>
```

List in Java

➲ It is an ordered collection where elements are maintained by index positions because it uses an index-based structure. In the list interface, the order is retained in which we add elements. We will also get the same sequence while retrieving elements.

➲ It is used to store a collection of elements where duplicate elements are allowed.

➲ List interface in java has four concrete subclasses. They are ArrayList, LinkedList, Vector, and Stack. These four subclasses implements the list interface.

ArrayList and LinkedList are widely used in Java programs to create a list. The Vector class is deprecated since JDK 5.

Features of List Interface in Java

1. The list allows storing duplicate elements in Java. JVM differentiates duplicate elements by using ‘index’. Index refers to the position of a certain object in an array. It always starts at zero. So, JVM will differentiate between both elements in the list based on their numeral position of the index. Therefore, the index is very useful and plays an important role to differentiate objects.

2. In the list, we can add an element at any position.

3. It maintains insertion order. i.e. List can preserve the insertion order by using the index.

4. It allows for storing many null elements.

5. Java list uses a resizable array for its implementation. Resizable means we can increase or decrease the size of the array.

6. Except for LinkedList, ArrayList, and Vector is an indexed-based structure.

7. It provides a special Iterator called a ListIterator that allows accessing the elements in the forward direction using hasNext() and next() methods. In the reverse direction, it accesses elements using hasPrevious() and previous() methods. We can add, remove elements of the collection, and can also replace the existing elements with the new element using ListIterator.

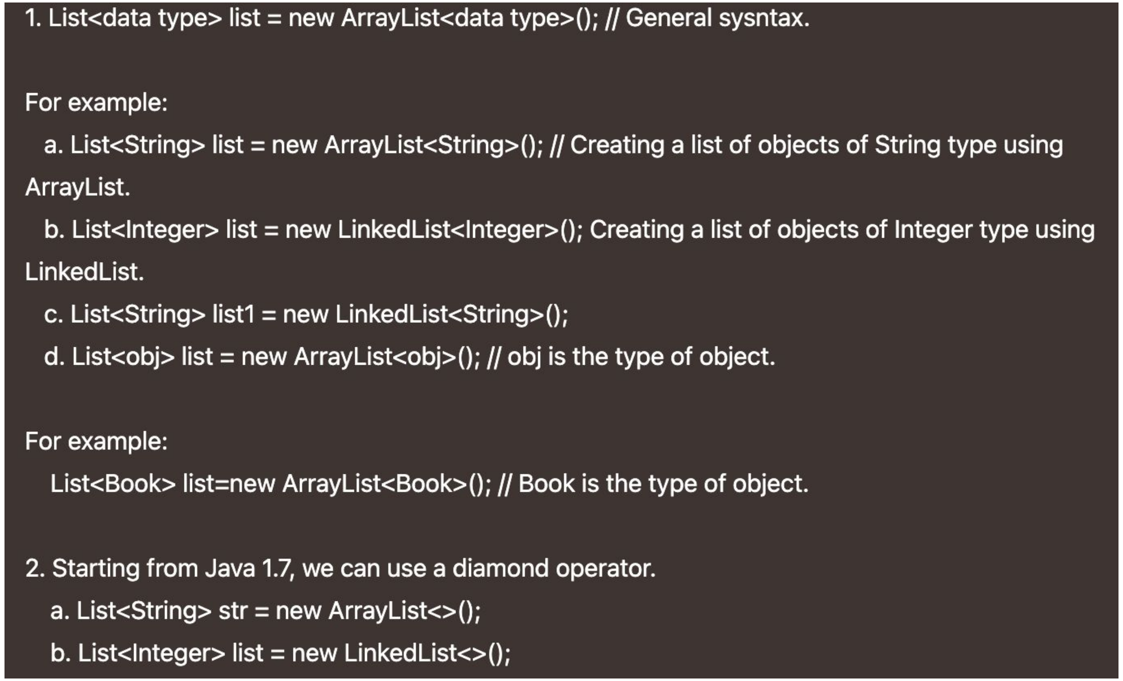
How to create a List in Java?

To create a list in java, we can use one of its two concrete subclasses: ArrayList, and LinkedList. We will use ArrayList to create a list and test methods provided by list interface in the program section.

```
List p = new ArrayList();

List q = new LinkedList();

List r = new Vector();

List s = new Stack();
```

How to create Generic List Object in Java

After the introduction of Generic in Java 1.5, we can restrict the type of object that can be stored in the list. The general syntax for creating a list of objects with a generic type parameter is as follows:



```
1. List<data type> list = new ArrayList<data type>(); // General sysntax.

For example:
  a. List<String> list = new ArrayList<String>(); // Creating a list of objects of String type using
ArrayList.
  b. List<Integer> list = new LinkedList<Integer>(); Creating a list of objects of Integer type using
LinkedList.
  c. List<String> list1 = new LinkedList<String>();
  d. List<obj> list = new ArrayList<obj>(); // obj is the type of object.

For example:
   List<Book> list=new ArrayList<Book>(); // Book is the type of object.

2. Starting from Java 1.7, we can use a diamond operator.
   a. List<String> str = new ArrayList<>();
   b. List<Integer> list = new LinkedList<>();
```

Java List Initialization

After creating a list, we need to initialize the list by adding elements to it. There are three methods to initialize the list. They are as follows:

- Using Arrays.asList
- Using Normal way
- Using Anonymous Inner class

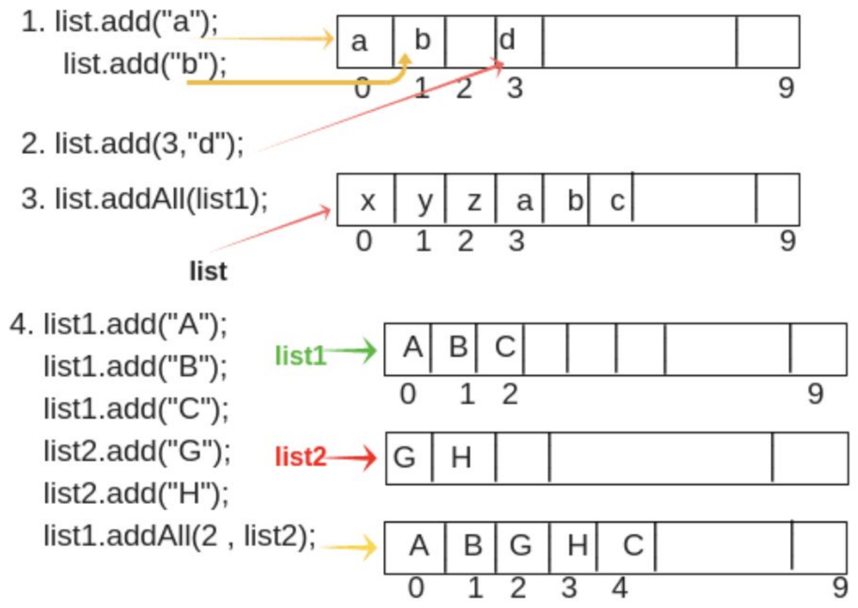
Java List Methods

Java List interface provides 10 specific and useful methods to the out of 15 methods specified by the collection interface that it extends. These methods can be used to initialize a list in java.

They are as follows:

Java List Methods

1. boolean add(Object o): It starts to add the specified element from zero location. If the element is already present at zero location, it will add the next element in one position.



Java List Methods

For example, consider 1 in the above figure.

```
list.add("a"); // Here, list is object reference variable.
list.add("b");                                             //
```

The return type of add() method is boolean and input type is an object.

Java List Methods

2. void add(int index, Object o): This method adds/inserts the specified element at a particular position in the list. For example, suppose we want to add element “d” at 3rd position, we will call add(int index, Object o) method like this:

```
list.add(3,"d"); // It will add element "d" at 3rd position as shown in figure.
```

.

Java List Methods

3. **boolean addAll(Collection c):** Here, Collection c represents a group of elements. This method is used to add/insert a group of elements at the end of the last element.

For example, suppose we want to add three elements x, y, and z at positions 0, 1, and 2 respectively in a list.

```
list.add("x");
list.add("y");
list.add("z");
```

Now we will create a list of another group of elements like this:

```
list1.add("a");
list1.add("b");
list1.add("c");
list.addAll(list1); // It will add group of elements at the end of the last element in the list. The last element is z.
So, after z, it will add list1 as shown in above figure.
```

Java List Methods

4. boolean addAll(int index, Collection c): This method is used to add/insert a group of elements at a particular position in the list and shift the subsequent elements to the right by increasing their indices.

For example, suppose we want to add three elements at positions 1, 2, and 3 respectively in a list using list1 reference variable.

```
list1.add("A");
list1.add("B");
    list1.add("C");
```

Now we will create a list of another group of elements using list2 reference variable.

```
list2.add("G");
    list2.add("H");
```

Assume that we want to add this group of elements at position 2 of the list1. So, we will call addAll(int index, Collection c) method like this:

```
    list1.addAll(2, list2); // You will see that element C is shifted to right at position 4 as shown in the figure.
```

Java List Methods

5. **object remove(int index):** It is used to remove an element at a specified position in the list. For example, consider the above figure.

list1.remove(2); // It will remove the element at 2nd position and element D which is at 3rd position, will be shifted to the left at the 2nd position. The output will be A, B, D.

Java List Methods

6. object get(int index): This method is used to return element/object stored at a specified position in the list. The return type of get() method is Object and input type is int[index of List].

For example:

```
list1.get(2); // Output will be 'C'.
```

Java List Methods

7. int indexOf(Object o): It is used to return the index of a particular element of the first occurrence in the list. If the element is not present in the list then it will return -1. It takes as an argument as an element and returns as an integer value of that element as it is index value.

For example, suppose an element ‘A’ is present at position 0 and the same element is also present at position 9 in the list.

```
list.indexOf("A"); // It will return integer value of an element "A" of first occurrence. i.e from zero position, not from
position 9. So the output is 0.
```

Java List Methods

8. **int lastIndexOf(Object o):** It returns the index of the last occurrence of a specified element in the list. If the list does not contain that particular element, it will return -1.

For example:

```
list.lastIndexOf("A"); // Output will be 9.
```

Java List Methods

9. **object set(int index, Object o):** This method replaces the existing element at the specified position in the list with new specified element.

For example:

```
list1.set(2, "Z");
```

Java List Methods

10. ListIterator listIterator(): It returns listIterator of the elements in the list in a proper sequence.

Java List Methods

11. ListIterator listIterator(int): This method returns a listIterator of the elements in the list in proper sequence, starting at the specified position in the list.

List Example Programs

Let’s take different kinds of example programs based on all the above methods to understand better.

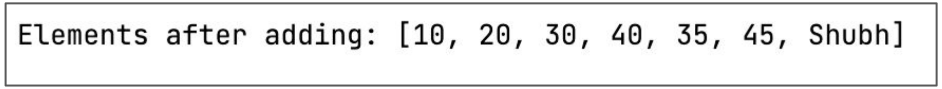
Let’s create an example program where we will add both integer and string elements together. So, we will not use generic in this program.

Example 1: Adding elements

```java
import java.util.ArrayList;
import java.util.List;
public class ListTester1 {
   public static void main(String[] args) {
       // Create a List.
       List al = new ArrayList();  // Here, there is no use of generic. So, no type safety.
       // We can add both integer and string elements.

       // Adding elements using add() method with reference variable al.
       al.add(10);
       al.add(20);
       al.add(30);
       al.add(40);
       al.add("Shubh");

       // Adding element to 4th position.
       al.add(4, 35);

       // Adding element to 5th position.
       al.add(5, 45);

       System.out.println("Elements after adding: " +al);
   }
}
```

Example 1: Adding elements

Output:



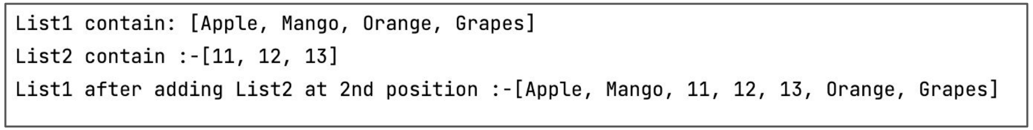
```
Elements after adding: [10, 20, 30, 40, 35, 45, Shubh]
```

Example 2: Adding all elements from collection

```
import java.util.ArrayList;
import java.util.List;
public class ListTester2 {
   public static void main(String[] args) {
       // Create a list of only String type. JVM throws an error if we try any type other than String.
       List<String> al = new ArrayList<String>();
       al.add("Apple");
       al.add("Mango");
       al.add("Orange");
       al.add("Grapes");
       System.out.println("List1 contain: " +al);

       // Create another List2 of String type.
       List<String> al2 = new ArrayList<String>();
       al2.add("11");
       al2.add("12");
       al2.add("13");
       System.out.println("List2 contain :-"+al2);

       // Adding List2 in List1 at 2nd position(i.e index=2) using addAll() method.
       al.addAll(2, al2);
       System.out.println("List1 after adding List2 at 2nd position :-"+al);
   }
}
```

Example 2: Adding all elements from collection

Output:



```
List1 contain: [Apple, Mango, Orange, Grapes]
List2 contain :-[11, 12, 13]
List1 after adding List2 at 2nd position :-[Apple, Mango, 11, 12, 13, Orange, Grapes]
```

Java List vs ArrayList

List is an interface whereas ArrayList is an implementation class that implements the List interface.

When to use List?

There are the following points for using list in java application that should be to keep in mind:

1. List can be used when we want to allow or store duplicate elements.

2. It can be used when we want to store null elements.

3. When we want to preserve my insertion order, we should go for list.

END