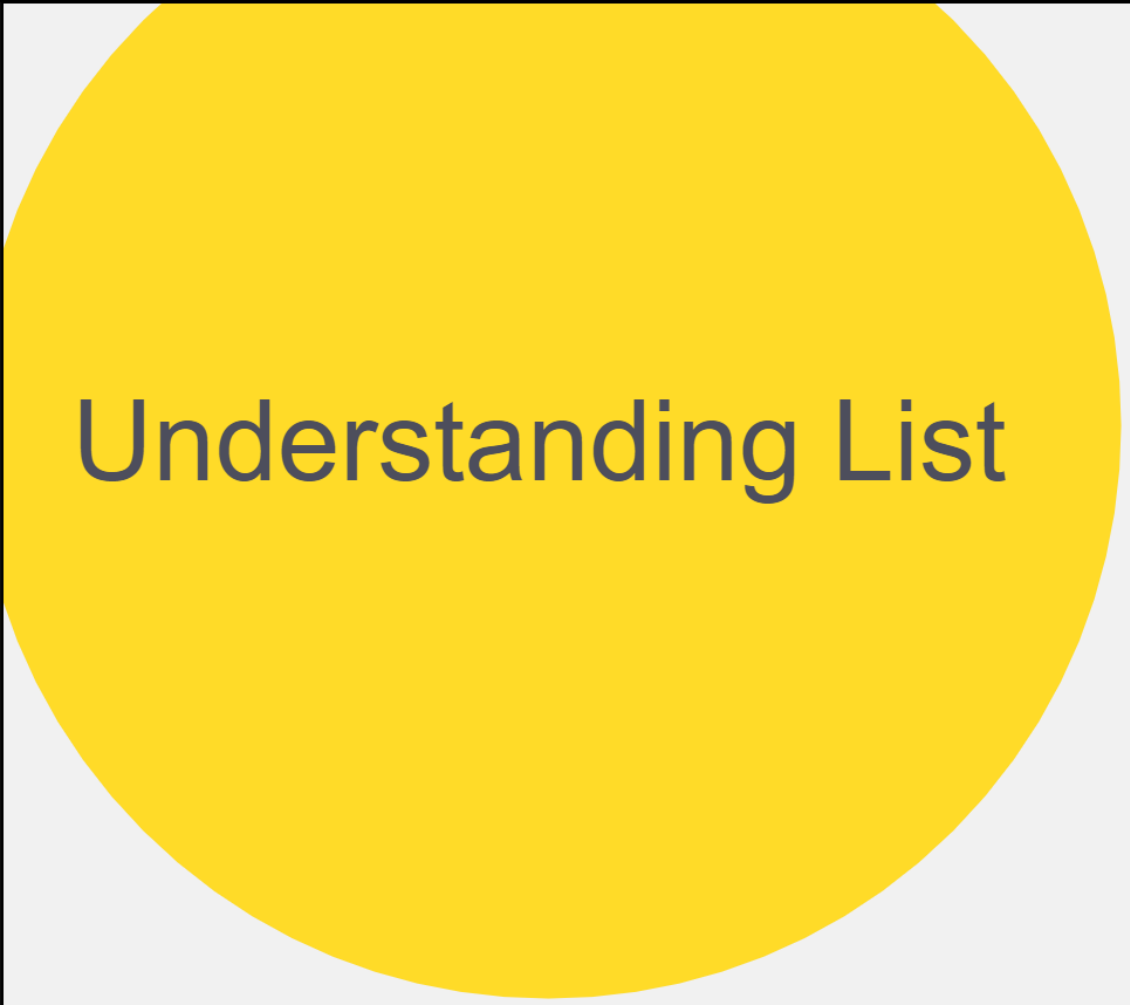
List

Agenda

1. List
2. ArrayList
3. Vector
4. LinkedList

Understanding List



List

- List interface extends from Collection interface
- It stores elements in a sequential manner
- Elements in the list can be accessed or inserted based on their position
- Starts with zero based index
- Can contain duplicate elements
- An Iterator can be used to access the elements of the List

Please refer documentation and note down the important methods available in List interface

The ArrayList Class

- ArrayList class implements List interface
- It supports dynamic array that can grow dynamically
- Standard arrays are of fixed size. After arrays are created they cannot grow or shrink
- It provides more powerful insertion and search mechanisms than arrays
- Gives faster Iteration and fast random access
- Ordered Collection (by index), but not Sorted

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(0, new Integer(42));
int total = list.get(0).intValue();
```

Refer documentation for the various ways in which an ArrayList can be created and the various methods available in ArrayList

Example

Let's Check the power of ArrayList with an example:

```
import java.util.*;
public class ArrayListTest {
public static void main(String[] args) {
List<String> test = new ArrayList<String>();
String s = "hi";
test.add("string");
test.add(s);
test.add(s+s);
System.out.print(test.size());
System.out.print(test.contains(42));
System.out.print(test.contains("hihi"));
test.remove("hi");
System.out.print(test.size());
} }
```

which produces:

```
3 false true 2
```

Iterator

- **Iterator is an object that enables you to traverse through a collection**
- **Can be used to remove elements from the collection selectively, if desired**

```
public interface Iterator<E>

 {
 boolean hasNext();
 E next();
 void remove();
 }
```

```
ArrayList<Integer> ai=new ArrayList<Integer>();
Iterator i=ai.iterator();
while (i.hasNext())
System.out.println(i.next());
```

Iterator

- Java provides 2 interfaces that define the methods by which you can access each element of a collection : enumeration & iterators. Enumeration is a legacy interface and is considered obsolete for new code. It is now superceded by the iterator interface.
- The iterator() method returns an iterator to a collection. It is very similar to an Enumeration, but differs in the two respects:
- Iterator allows the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.
- The first point is important: There was *no* safe way to remove elements from a collection while traversing it with an Enumeration. The semantics of this operation were ill defined, and differed from implementation to implementation.
- **boolean hasNext() -** Returns true if there are more elementsObject
- **next() -** Returns next element. Throws NoSuchElementException if there is no next element.
- **void remove() -** Removes current element. Throws IllegalStateException if an attempt is made to call remove() that is not preceded by a call to next()

ListIterator

- Used for obtaining a iterator for collections that implement List
- ListIterator gives us the ability to access the collection in either forward or backward direction
- Has both next() and previous() method to access the next and previous element in the List

Example

```
class ListIteratorExample {
   public static void main(String[] args) {
     ArrayList aList = new ArrayList();
     //Add elements to ArrayList object
    aList.add("1");
    aList.add("2");
    aList.add("3");
   ListIterator listIterator = aList.listIterator();
     System.out.println("Previous Index is : " + listIterator.previousIndex());
    System.out.println("Next Index is : " + listIterator.nextIndex());
       //advance current position by one using next method
    listIterator.next();
       System.out.println("After increasing current position by one element : ");
    System.out.println("Previous Index is : " + listIterator.previousIndex());
    System.out.println("Next Index is : " + listIterator.nextIndex());
    }}
```

Advantage of Iterator over for-each method

- **for-each construct can also be used for iterating through the Collection**
- **Use Iterator instead of the for-each construct when you need to:**
  - Remove the current element
  - The for-each construct hides the iterator, so you cannot call remove
  - Iterate over multiple collections in parallel

```
for(Object o : oa) {
Fruit d2 = (Fruit)o;
System.out.println(d2.name); }
```

Enhanced for loop

- **Iterating over collections looks cluttered**

```
void printAll(Collection<emp> e) {
 for (Iterator<emp> i = e.iterator(); i.hasNext(); )
 System.out.println(i.next()); } }
```

- **Using enhanced for loop we can do the same thing as**

```
void printAll(Collection<emp> e) {
 for (emp t: e) )
 System.out.println(t); }}
```

- **The loop above reads as “for each emp t in e.”**

Linked List

- **Implements the List and also the Queue interface**
- **Some Useful Methods**
  - void addFirst(Object x)
  - void addLast(Object x)
  - Object getFirst()
  - Object getLast()
  - Object removeFirst()
  - Object removeLast()

The Vector Class

- The java.util.Vector class implements a growable array of Objects
- Same as ArrayList, but Vector methods are synchronized for thread safety
- New java.util.Vector is implemented from List Interface
- Creation of a Vector

```
Vector v1 = new Vector(); // allows old or new
                             methods
List v2 = new Vector(); // allows only the new
                           (List) methods.
```

Points to Ponder

- Use List Collection classes if the order in which the element added matters
- Use List when you want to perform insert, delete and update operations based on particular positions in the list
- ArrayList, LinkedList and Vector all of them implement List interface
- LinkedList provides a better performance over ArrayList in insertion and deletion operation.
- In case of frequent insertion and deletion operation the choice can be LinkedList than ArrayList
- Search operations are faster in ArrayList
- Both ArrayList and LinkedList are not synchronized
- Vector is synchronized
- If thread safety is not important, then we should choose either ArrayList or LinkedList

Quiz

1. **Which of the following class is synchronized?**
   a. ArrayList
   b. Vector
   c. LinkedList
   d. All of the above
2. **In which of the following classes position based operations can be performed?**
   a. ArrayList
   b. LinkedList
   c. Vector
   d. All of the above

Summary

- **In this module, you have learnt**
  - How to work with
    - ArrayList
    - LinkedList
    - Vector

Thank You