**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**
**CS F213**
**LAB-08**

**AGENDA** **29/10/2020** **TIME: 02 Hours**

1. **Anonymous inner classes**
2. **List, Arraylist, Iterator, ListIterator and Linked Lists**
3. **Comparable and Comparator Interface**

## 1. Anonymous Inner Classes

In some cases, you might find yourself writing an inner class and then using that class in just a single line of your program. For cases like this, you have the option of using an anonymous inner class. An Anonymous class is created with a variation of the 'new' operator that has the form

```
new  superclass-or-interface (parameter-list) {
        methods-and-variables
}
```

This constructor defines a new class, without giving it a name, and it simultaneously creates an object that belongs to that class. This form of new operator can be anywhere where general new could be used. The intention of this expression is to create: a new object belonging to a class that is the same as superclass-or-interface but with extra methods-and-variables added. The effect is to create a uniquely customized object, just as the point in the program where you need it.

**Note**: If an interface is used as the base, the anonymous class must implement the interface by defining all methods that are declared in the interface and the parameter-list must be empty

**1.1Example**:

```
public class Movie {
     //Declaring an interface inside the class
     Interface Bookable {
          public void printTicket();
          public void giveTicket(String movie);
     }
     public void BookTheTicket() {
          //Writing an InnerClass that implements the above
          interface
          class EnglishMovie implements Bookable {
               String name ;
               Public void printTicket()
               {eTicket("BlindDate");}
               public void giveTicket(String movie) {
                    name = movie;
                    System.out.println("You have booked
                         + for the movie " + name);
               }
          }//end of class EnglishMovie
```

```
          // creating an object for the inner class
          Bookable hollywood = new EnglishMovie();

          // anonymous innerclass which is basing
          the interface..
          Bookable hindiMovie = newBookable() {
               public void printTicket() {
                    giveTicket("Bachna Ae Haseeno");
               }
               public void giveTicket(String movie) {
                    String name = movie;
                    System.out.println("You have booked
                         + for     the movie "+ name);
               }
          };

          hollywood.giveTicket();
          hindiMovie.giveTicket();
     }//end of method BookTheTicket

     public static void main(String[] args) {
             Movie easyMovie = new Movie();
             easyMovie.BookTheTicket();
     }

} //end of class Movie
```

In the above example, the EnglishMovie is inner class and for the instance hindiMovie we created Anonymous inner class.

**1.2 Exercise:**

```
public class Anonymous{
     public Circle getCircle(int radius) {
     // Write one line statement that returns object of Circle
       // by writing Anonymous inner class.
     }
     public static void main(String[] args) {
          Anonymous p = new Anonymous();
          Circle w = p.getCircle(10);
          // The output here should give correct value of area
          // of the circle.
          System.out.println(w.area());
     }
}
class Circle {
     private int rad;
     public Circle(int radius) { rad = radius; }
     public double area() { return rad*rad; }
}
```

## 2.List Interface

A List is an ordered Collection (sometimes called a *sequence*). The user of this interface has precise control over where in the list each element is inserted. In addition to the operations inherited from Collection, the ListInterface includes operations for the following:

- Positional access — manipulates elements based on their numerical position in the list
- Search — searches for a specified object in the list and returns its numerical position
- Iteration — extends Iterator semantics to take advantage of the list's sequential nature
- Range-view — performs arbitrary *range operations* on the list.

There are two implementations:

- LinkedList gives faster insertions and deletions
- ArrayList gives faster random access

## 3. ArrayList

It is resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null.

ArrayList supports dynamic arrays that can grow as needed unlike standard Java arrays, which are of a fixed length.

In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list.

**3.1 Example**: Following example shows the use of ArrayList and its methods.

```
import java.util.ArrayList;

publicclass ArrayListDemo {
     public static void main(String[] args) {
          //Creating a new ArrayList
          ArrayList arlTest = new ArrayList();
          //Size of arrayList
          System.out.println("Size of ArrayList at creation: "
                                              +arlTest.size());
          //Lets add some elements to it
          arlTest.add("B");
          arlTest.add("I");
          arlTest.add("T");
          arlTest.add("S");
          //Recheck the size after adding elements
          System.out.println("Size of ArrayList after adding
                                    elements: "+arlTest.size());
          //Display all contents of ArrayList
          System.out.println("List of all elements: "
                                                  + arlTest);
```

```
            //Remove some elements from the list
            arlTest.remove("B");
            System.out.println("See contents after removing one
                                        + element: " + arlTest);
            //Remove element by index
            arlTest.remove(2);
            System.out.println("See contents after removing
                                 + element by index: " + arlTest);
            //Check size after removing elements
            System.out.println("Size of arrayList after removing
                                    elements: " + arlTest.size());
            System.out.println("List of all elements after
                                + removing elements: " + arlTest);
            //Check if the list contains "T"
            System.out.println(arlTest.contains("T"));
      }
}
```

Run the above code and observe the output.

## 4. Iterator

Each of the collection classes provides an iterator( ) method that returns an iterator to the start of the collection. By using this iterator object, you can access each element in the collection, one element at a time. In general, to use an iterator to cycle through the contents of a collection, follow these steps:

1. Obtain an iterator to the start of the collection by calling the collection's iterator( )
2. Set up a loop that makes a call to hasNext( ). Have the loop iterate as long as hasNext( ) returns true.
3. Within the loop, obtain each element by calling next( ).

## 5. ListIterator

An iterator for lists that allows the programmer to traverse the list in either direction, modifies the list during iteration, and obtains the iterator's current position in the list. A ListIterator has no current element; its *cursor position* always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next().

An iterator for a list of length n has n+1 possible cursor positions, as illustrated by the carets (^) below:

```
                  Element(0)  Element(1)  Element(2) ... Element(n-1)
cursor positions: ^           ^           ^                ^
```

**5.1 Example**: Iterator and ListIterator

(1) Student.java

```
public class Student {
     private String name;
     private String gender;
     private int age;

     public Student(String name,String gender,int age){
          this.name=name;
          this.gender=gender;
          this.age=age;
     }

     public String getName(){
          return name;
     }
     public String getGender(){
          return gender;
     }

     publicvoid setName(String name){
          this.name=name;
     }
     public String toString(){
          return name+"   "+gender+" "+age;
     }
}
```

(2) TestStudentList.java

```
// Demonstrate Iterator and ListIterator
import java.util.*;

public class TestStudentList {
     public static void main(String args[]) {
          // create an array list

          ArrayList studentList = new ArrayList();
          // add elements to the array list
          studentList.add(new Student("Ramesh","Male",18));
          studentList.add(new Student("Reeta","Female",19));
          studentList.add(new Student("Seema","Female",18));
          studentList.add(new Student("Suresh","Male",20));

          System.out.println("Original contents of +
                                                  studentList:");
          Iterator itr = studentList.iterator();
```

```
            while(itr.hasNext()) {
                Object element = itr.next();
                System.out.print(element +"\n");
            }

            System.out.println();

            // modify objects being iterated
            ListIterator litr = studentList.listIterator();

            while(litr.hasNext()) {
                 Student element = (Student)litr.next();
                 if(element.getGender().equals("Male")){
                      element.setName("Mr."+element.getName());
                 }
                 else{
                      element.setName("Miss."+element.getName());
                 }
                 litr.set(element);
            }

            System.out.println("Modified contents of
                                                + studentList: ");
            itr = studentList.iterator();

            while(itr.hasNext()) {
                Object element = itr.next();
                System.out.print(element + "\n");
            }

            System.out.println();
            // now, display the list backwards
            System.out.println("Modified list backwards: ");

            while(litr.hasPrevious()) {
                Object element = litr.previous();
                System.out.print(element + "\n");
            }

            System.out.println();
      }
}
```

**Exercise:**

The L&L Bank can handle up to 30 customers who have savings accounts. Design and implement a program that manages the accounts. Keep track of key information and allow each customer to make deposits and withdrawals. Produce appropriate error messages for invalid transactions. Do this practice problem using ArrayList and Iterator.

(A) Create an Account class that tracks individual customer information.

```
public class Account {
     private long acctNumber;
     private double balance;
     private String name;
     /*Complete the Account class by adding proper constructor,
     accessor method and mutator method as required. Override
     toSring() method to display account details. */
     //Write your code here
}
```

(B) Complete the code of Bank class as per the commented instructions

```
public class Bank {

     private ArrayList<Account> accts;
     int maxActive;

     public boolean addAccount (Account newone) {
       /* Write the code for adding new account, return false if
          account can't be created */
     }

     public boolean removeAccount (long acctnum) {
       /* Write the code for removing the account, return false
          if account does not exist */
     }

     public double deposit(long acctnum, double amount) {
      /* Write the code for depositing specified amount to the
         account,return -1 if account does not exist */
     }

     public double withdraw(long acctnum, double amount) {
      /* Write the code for withdrawing specified amount from
         the account,return -1 if insufficient balance or
         account does not exist*/
     }

     //override toString() method to display details of all the
       accounts in bank
}
```

(C) Write a suitable driver class to test the behavior of the methods of above classes.

## 6. LinkedList

The LinkedList class extends AbstractSequentialList and implements the List interface. It provides a linked-list data structure. Implements all optional list operations, and permits all elements (including null).

In addition to implementing the List interface, the LinkedList class provides methods to get, remove and insert an element at the beginning and end of the list. These operations allow linked lists to be used as a stack, queue, or double-ended queue.

The class implements the Deque interface, providing first-in-first-out queue operations for add, poll, along with other stack and deque operations.Also of the operations perform as could be expected for a doubly-linked list.

**6.1 Example:**

```
import java.util.*;

public class ReservationIncharge{

     public static void main(String[] args){

          //getting an instance of ReservationCounter
          ReservationCounter counter =
                        ReservationCounter.InitializeCounter();
          counter.standInLine("Amitab");
          counter.standInLine("amir");
          counter.standInLine("salman");
          counter.standInLine("Tom Cruise");
          for(int position=0;
             position < counter.customersInLine();position++){
               System.out.println("customer at "+(position+1)
                     + " place in line is"
                     + counter.checkTheCustomerAt(position));
          }
          System.out.println("Ticket given to
                                        "+counter.giveTicket());
          counter.leaveTheLine();
          counter.standInLine("Emma Watson");
          System.out.println("Ticket given to
                                        "+counter.giveTicket());
          counter.leaveTheLine();
          System.out.println("customer in position 2 is
                                "+counter.checkTheCustomerAt(1));
          System.out.println("Still there are
                                "+counter.customersInLine()+
                                             " people in line");
     }
}
```

```
class ReservationCounter{

	LinkedList ReservationQueue;

	private ReservationCounter(){
		//initializing the ReservationQueue
		ReservationQueue=new LinkedList();
	}

	//Adding an element to the linkedlist from the end just
	like in a queue
	public void standInLine(String customer){
		ReservationQueue.add((String)customer);
	}

	//default Removal of the element is done from the front i.e
	first element is removed
	public void leaveTheLine(){
		ReservationQueue.remove();
	}

	//demonstration of how the first element of list can be
	accessed.
	public String giveTicket(){
		return (String)ReservationQueue.getFirst();
	}

	//retrieving data from anywhere in the list
	public String checkTheCustomerAt(int position){
		return (String)ReservationQueue.get(position);
	}

	public static ReservationCounter InitializeCounter(){
		returnnew ReservationCounter();
	}

	//size of the list
	public int customersInLine(){
		return ReservationQueue.size();
	}
}
```

Observe the output of the above code.

**6.2 Exercise:**

A stack is a type of data structure – a means of storing information in a computer. When a new object is entered in a stack, it is placed on top of last the previously entered objects. In other words, the stack data structure is just like a stack of cards, papers, stack of books, or any other real world objects you can think of. This method is referred as LIFO (last in first out). Implement Stack using LinkedList.

## 7. Comparable and Comparator Interface

1. Java provides two interfaces to sort objects using data members of the class:
   a. Comparable
   b. Comparator
2. Comparable is meant for objects with natural ordering which means the object itself must know how it is to be ordered. For example, Roll Numbers of students. Whereas, Comparator interface sorting is done through a separate class.
3. Logically, Comparable interface compares "this" reference with the object specified and Comparator in Java compares two different class objects provided.
4. If any class implements Comparable interface, then collection of that object either List or Array can be sorted automatically by using Collections.sort() or Arrays.sort() method and objects will be sorted based on the natural order defined by CompareTo method.

**Note:** For examples, please refer the lecture slides.

Q1. Refer to the class diagram given below:

a. Modify the MovableCircle class to sort the objects based on the `radius` using Comparable interface.
b. Modify the MovablePoint class to compare two Point objects based on their `xSpeed` and `ySpeed` using Comparator interface.