LAB # 02

ArrayList and Vector in JAVA

OBJECTIVE: To implement ArrayList and Vector.

ArrayList

An ArrayList is like an array that resizes to fit its contents. Thus, it differs from array:
i) Size does not need to be specified at creation time.
ii) Grows in size as items added.
iii) ArrayLists, however, come with a selection of powerful methods whereas arrays consist of length instance variable. When a list is created, it is initially empty.

Constructors

new ArrayList( )	- no need to pass in size (10)
new ArrayList(int)	- initial capacity

Creating ArrayList:

```
ArrayList<String> alist=new ArrayList<String>();


ArrayList<Integer> list=new ArrayList<Integer>();
```

Methods

- To access an element in the ArrayList, use the **get()** method and refer to the index number.
- To modify an element, use the **set()** method and refer to the index number.
- To remove an element, use the **remove()** method and refer to the index number.
- To remove all the elements in the ArrayList, use the **clear()** method.
- To find out how many elements an ArrayList have, use the **size()** method.

Sample Program#1

```
import java.util.*;
class JavaExample{
  public static void main(String args[]){
    ArrayList<String> alist=new ArrayList<String>();
    alist.add("Steve");
    alist.add("Tim");
    alist.add("Lucy");
    alist.add("Pat");
    alist.add("Angela");
    alist.add("Tom");
```

```
      //displaying elements
      System.out.println(alist);

      //Adding "Steve" at the fourth position
      alist.add(3, "Steve");

      //displaying elements
      System.out.println(alist);
   }
}
```

Performance Analysis of ArrayList

ArrayList uses the Array data structure and it should be used where more search operations are required. Here, we are going to discuss the affects on the performance of insert, search, and delete operation of ArrayList. Below is an example of different operations using ArrayList;

Sample program#2

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class Example {
  public static void main(String[] args) {
    List<Integer> arrayList = new ArrayList<>();

/*Insert at last in Arraylist*/
    arrayList.add(1);
    arrayList.add(111);
    System.out.println(arrayList);

/*Insert at given index in Arraylist*/
    arrayList.add(1, 11);
    arrayList.add(3, 1111);
    System.out.println(arrayList);

    /*Search by value in ArrayList(searching 111 value)*/
    for(int i=0; i < arrayList.size(); i++) {
      if(arrayList.get(i).equals(111)){
        System.out.println("Value found at index: "+i); /*
      }
    }

/*Get value by index in ArrayList*/
    value = arrayList.get(2);
    System.out.println(value);
/* Remove by value in ArrayList(remove 111)*/
    isDeleted = arrayList.remove(new Integer(111));
    System.out.println(isDeleted);

    /*Remove by index in ArrayList*/
    value = arrayList.get(2);
    System.out.println("Removed value: "+value);
  }
}
```

Insert Value at Last Index
When we insert a value at last index, ArrayList has to;

- Check whether the underlying array is already full or not.
- If the array is full then it copies the data from the old array to new array(size double than an old array).
- Then add the value at the last index.

It has time complexity O(1), but due to the added steps of creating a new array, its worst-case complexity reaches to order of N.

Insert Value at Given Index
The time complexity will be O(N), due to the added steps of creating a new array in ArrayList, and copying the existing values to the new index.

Search by Value
When we search any value in ArrayList, we have to iterate through all elements. This operation has O(N) time complexity.
Here we need to notice that array stores all values in a continuous memory location. Iterating through continuous memory location is more performance efficient.

Get Element by Index
When we get an element by Index then ArrayList is a clear winner.
ArrayList can give you any element in O(1) complexity as the array has random access property. You can access any index directly without iterating through the whole array.

Remove by Value
It is similar to adding value at a given index. To remove an element by value in ArrayList we need to iterate through each element to reach that index and then remove that value. This operation is of O(N) complexity.
In ArrayList, we need to shift all elements after the index of the removed value to fill the gap created.

Remove by Index
To remove by index, ArrayList find that index using random access in O(1) complexity, but after removing the element, shifting the rest of the elements causes overall O(N) time complexity.

<u>Sorting a Collection using ArrayList</u>

We generally use Collections.sort() method to sort a simple array list. However if the ArrayList is of custom object type then in such case you have two options for sorting- **comparable** and **comparator** interfaces.

Sample Program #3

```
import java.util.*;

public class Student implements Comparable {
    private String studentname;
    private int rollno;
    private int studentage;

    public Student(int rollno, String studentname, int studentage) {
        this.rollno = rollno;
        this.studentname = studentname;
        this.studentage = studentage;
    }

@Override
    public int compareTo(Student comparestu) {
        int compareage=((Student)comparestu).getStudentage();

        return this.studentage-compareage;
    }

    @Override
    public String toString() {
        return "[ rollno=" + rollno + ", name=" + studentname + ", age=" + studentage +
"]";
    }
}

public class ArrayListSorting  {

  public static void main(String args[]){
    ArrayList<Student> arraylist = new ArrayList<Student>();
    arraylist.add(new Student(221, "Rameez", 25));
    arraylist.add(new Student(242, "Zain", 26));
    arraylist.add(new Student(211, "Aijaz", 30));

    Collections.sort(arraylist);

    for(Student str: arraylist){
            System.out.println(str);
    }
  }
}
```

Vector

Vector implements a dynamic array. It is similar to ArrayList, but with two differences:

- Vector is synchronized.
- Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

Creating Vector:

```
Vector<Integer> vector = new Vector <>();
Vector<String> vector = new Vector <>();
```

Sample Program#4

```
import java.util.*;
public class VectorExample {
      public static void main(String args[]) {
        //Create a vector
        Vector<String> vec = new Vector<String>();

        //Adding elements using add() method of List
        vec.add("Tiger");
        vec.add("Lion");
        vec.add("Dog");
        vec.add("Elephant");

        System.out.println("Elements are: "+vec);
      }
}
```

equals( ) & hashCode( ) methods

- Java.lang.object has two very important methods defined:
  - public boolean equals(Object obj)
  - public int hashCode()
- Java equals() and hashCode() methods are present in Object class. So every java class gets the default implementation of equals() and hashCode().
- Java hashCode() and equals() method are used in Hash table based implementations in java for storing and retrieving data.

Sample Program#5

```
import java.io.*;

class Example {
   public String name;
   public int id;

   Example(String name, int id){

      this.name = name;
      this.id = id;
   }

   @Override
   public boolean equals(Object obj){

   // checking if both the object references are
   // referring to the same object.
   if(this == obj)
         return true;

if(obj == null || obj.getClass()!= this.getClass())
         return false;

      // type casting of the argument.
      Example eg = (Example) obj;

      // comparing the state of argument with
      // the state of 'this' Object.
      return (eg.name == this.name && eg.id == this.id);
   }

   @Override
   public int hashCode()
   {

      // We are returning the Eg_id
      // as a hashcode value.
      return this.id;  }
}

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

      // creating the Objects of Example class.
      Example e1 = new Example("aa", 1);
      Example e2 = new Example("aa", 1);

      // comparing above created Objects.
```

```
        if(e1.hashCode() == e2.hashCode()) {

            if(e1.equals(e2))
                System.out.println("Both Objects are equal. ");
            else
                System.out.println("Both Objects are not equal. ");
        }
        else
        System.out.println("Both Objects are not equal. ");
    }
}
```

Lab Tasks

1. Write a program that initializes Vector with 10 integers in it. Display all the integers and sum of these integers.

2. Create a ArrayList of string. Write a menu driven program which:
    a. Displays all the elements
    b. Displays the largest String

3. Create a Arraylist storing Employee details including Emp_id, Emp_Name, Emp_gender, Year_of_Joining (you can also add more attributes including these). Then sort the employees according to their joining year using Comparator and Comparable interfaces.

4. Write a program that initializes Vector with 10 integers in it.

- Display all the integers
- Sum of these integers.
- Find Maximum Element in Vector

5. Find the k-th smallest element in a sorted ArrayList

6. Write a program to merge two ArrayLists into one.

Home Tasks

1. Create a Vector storing integer objects as an input.
    a. Sort the vector
    b. Display largest number
    c. Display smallest number

2. Write a java program which takes user input and gives hashcode value of those inputs using hashCode () method.

3. **Scenario based**

Create a java project, suppose you work for a company that needs to manage a list of employees. Each employee has a unique combination of a name and an ID. Your goal is to ensure that you can track employees effectively and avoid duplicate entries in your system.

Requirements

a. Employee Class: You need to create an Employee class that includes:

- name: The employee's name (String).
- id: The employee's unique identifier (int).
- Override the hashCode() and equals() methods to ensure that two employees are considered equal if they have the same name and id.

b. Employee Management: You will use a HashSet to store employee records. This will help you avoid duplicate entries.

c. Operations: Implement operations to:

- Add new employees to the record.
- Check if an employee already exists in the records.
- Display all employees.

4.Create a Color class that has red, green, and blue values. Two colors are considered equal if their RGB values are the same