SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

NAME: Anish Sharma
ROLL NO: I011
SAP ID: 60003220045
BRANCH: Information Technology
BATCH: 1

EXPERIMENT NO. 01

CO/LO: CO1- Modify the behaviour of methods, classes, and interfaces at runtime.

AIM / OBJECTIVE: To implement different collection types

AIM: Implementation of Stack and Queues using Arrays.

DESCRIPTION OF EXPERIMENT:

Collections in Java

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

What is Collection in Java: A Collection represents a single unit of objects, i.e., a group.

What is a framework in Java

- o It provides readymade architecture.
- o It represents a set of classes and interfaces.
- o It is optional.

What is Collection framework

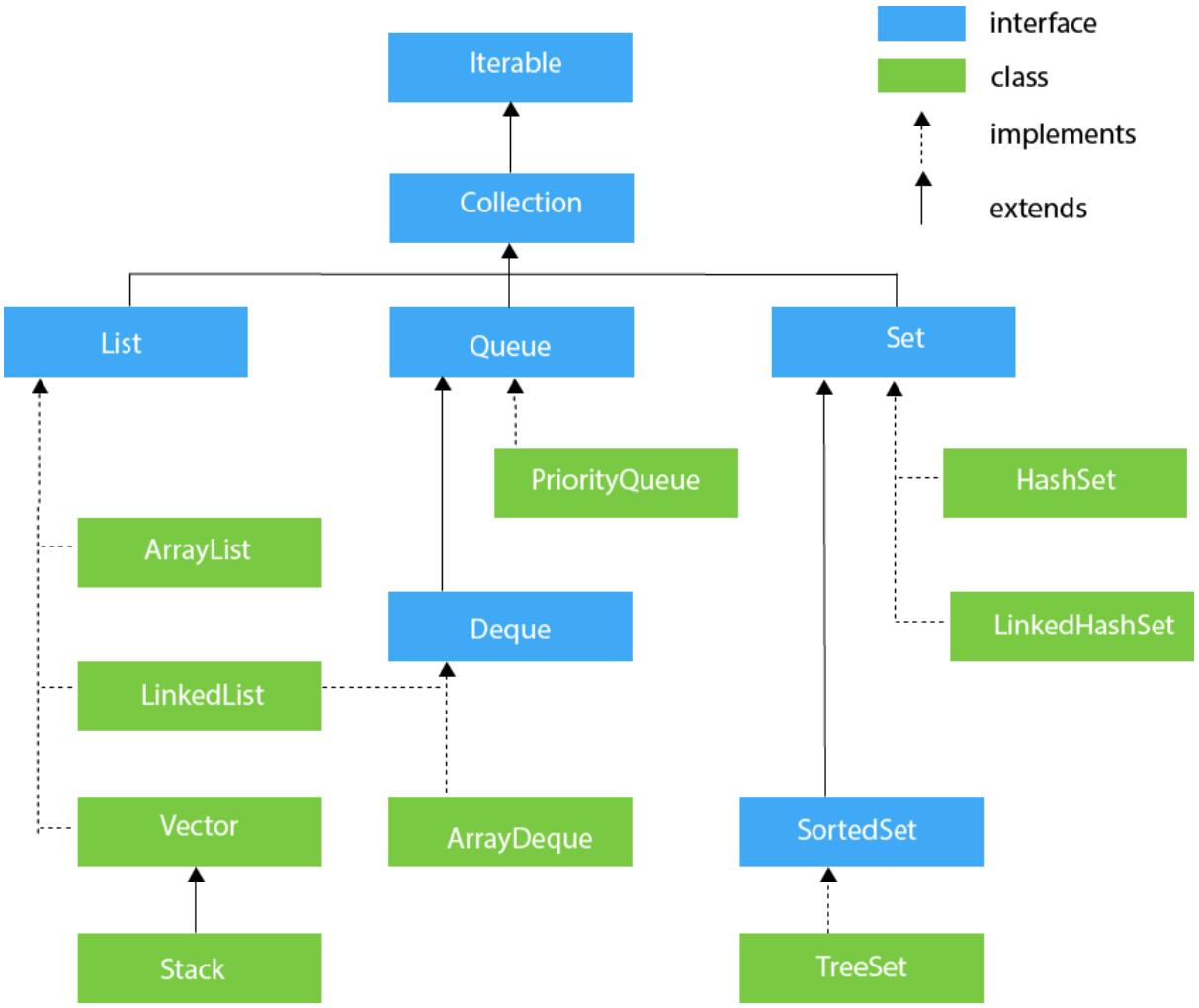
The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

2. Algorithm

Hierarchy of Collection Framework



COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

PROCEDURE / ALGORITHM:

1) Write a Java program to create a vector that stores names of 10 employees, later add 2 new employees, then remove eighth employee from vector, copy the vector to another vector, display the values and size of new vector

Code:

```
import java.util.*;


public class vector {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		Vector<String> v1 = new Vector<String>();
		// Vector<String> v2 = new Vector<String>();
		System.out.println("Enter names of 10 employees:");
		String s = "";
		for (int i = 0; i < 10; i++) {
			s = sc.nextLine();
		    v1.add(s);
		 }
		 v1.add("e11");
		v1.add("e12");
		System.out.println("two employees are added");
		v1.remove(7);
		System.out.println("employee 8 is deleted");
		Object v2 = v1.clone();
		System.out.println("vector is copied");
	System.out.println(v2);
    }
}
```

OUTPUT:

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
Enter names of 10 employees:
e1
e2
e3
e4
e5
e6
e7
e8
e9
e10
two employees are added
employee 8 is deleted
vector is copied
[e1, e2, e3, e4, e5, e6, e7, e9, e10, e11, e12]
```

2) In this problem we have given you three classes in the editor:

· Student class
· Rockstar class
· Hacker class

In the main method, we populated an ArrayList with several instances of these classes. count method calculates how many instances of each type is present in the ArrayList. (Use instanceof)

Sample Input

Student
Student
Rockstar
Student
Hacker

Sample Output

3 1

Code:

```
import java.util.*;
class Student {}
class Rockstar {}
class Hacker {}

public class P2 {
    public static void main(String[] args) {
        ArrayList<Object> people = new ArrayList<Object>();
        people.add(new Student());
        people.add(new Student());
        people.add(new Hacker());
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
        people.add(new Student());
        people.add(new Rockstar());
        count(people);
    }

    public static void count(ArrayList<Object> people) {
        int studentCount = 0;
        int rockstarCount = 0;
        int hackerCount = 0;
        for (Object person : people) {
            if (person instanceof Student) {
                studentCount++;
            } else if (person instanceof Rockstar) {
                rockstarCount++;
            } else if (person instanceof Hacker) {
                hackerCount++;
            }
        }
        System.out.println("Student count: " + studentCount);
        System.out.println("Rockstar count: " + rockstarCount);
        System.out.println("Hacker count: " + hackerCount);
    }
}
```

OUTPUT:

```
Student count: 3
Rockstar count: 1
Hacker count: 1
```

3) 3. You are given n pairs of strings. Two pairs (a,b) and (c,d) are identical if a=c and b=d. That also implies (a,b) is not same as (b,a). After taking each pair as input, you need to print a number of unique pairs you currently have. Print n lines. In the ith line, print number of unique pairs you have after taking ith pair as input.

Sample Input:
5
Ayush Dhruv
Ayush Advait
Advait Shivam

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

Shivam Dhruv
Ayush Dhruv
Sample Output:

CODE:

```
import java.util.*;

public class P1 {
    public static void main(String[] args) {
        HashSet<String> hs = new HashSet<String>();
        System.out.println("enter number of strings");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int count = -1;
        for (int i = 0; i <= n; i++) {
            if (hs.add(sc.nextLine())) {
                count++;
            }
            System.out.println("unique strings:" + count);
        }
        System.out.println("total number of unique elements:" + hs.size());
        System.out.println(hs);
    }
}
```

OUTPUT:

```
enter number of strings
5
unique strings:0
Ayush Dhruv
unique strings:1
Ayush Advait
unique strings:2
Advait Shivam
unique strings:3
Shivam Dhruv
unique strings:4
Ayush Dhruv
unique strings:4
total number of unique elements:5
[, Ayush Advait, Shivam Dhruv, Advait Shivam, Ayush Dhruv]
```

4) WAP to convert an infix expression to postfix and evaluate the same.
[input in the form of 5*7+8-3]

Code:

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
import java.util.*;

public class infix {
    private static boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/';
    }

    private static int getPrecedence(char operator) {
        switch (operator) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            default:
                return 0;
        }
    }

    public static String infixToPostfix(String infix) {
        StringBuilder postfix = new StringBuilder();
        Stack<Character> operatorStack = new Stack<>();
        for (char c : infix.toCharArray()) {
            if (Character.isDigit(c)) {
                postfix.append(c);
            } else if (c == '(') {
                operatorStack.push(c);
            } else if (c == ')') {
                while (!operatorStack.isEmpty() && operatorStack.peek() !=
'(') {
                    postfix.append(operatorStack.pop());
                }
                operatorStack.pop(); // Pop '('
            } else if (isOperator(c)) {
                while (!operatorStack.isEmpty() && getPrecedence(c) <=
getPrecedence(operatorStack.peek())) {
                    postfix.append(operatorStack.pop());
                }
                operatorStack.push(c);
            }
        }
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
        while (!operatorStack.isEmpty()) {
            postfix.append(operatorStack.pop());
        }
        return postfix.toString();
    }

    public static int evaluatePostfix(String postfix) {
        Stack<Integer> operandStack = new Stack<>();
        for (char c : postfix.toCharArray()) {
            if (Character.isDigit(c)) {
                operandStack.push(c - '0');
            } else if (isOperator(c)) {
                int operand2 = operandStack.pop();
                int operand1 = operandStack.pop();
                int result = performOperation(c, operand1, operand2);
                operandStack.push(result);
            }
        }
        return operandStack.pop();
    }

    private static int performOperation(char operator, int operand1, int
operand2) {
        switch (operator) {
            case '+':
                return operand1 + operand2;
            case '-':
                return operand1 - operand2;
            case '*':
                return operand1 * operand2;
            case '/':
                if (operand2 == 0) {
                    throw new ArithmeticException("Division by zero");
                }
                return operand1 / operand2;
            default:
                throw new IllegalArgumentException("Invalid operator");
        }
    }
    public static void main(String[] args) {
        String infixExpression = "5*7+8-3";
        String postfixExpression = infixToPostfix(infixExpression);
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
        System.out.println("Postfix expression:" + postfixExpression);
        int result = evaluatePostfix(postfixExpression);
        System.out.println("Result: " + result);
    }
}
```

OUTPUT:

```
Postfix expression:57*8+3-
Result: 40
```

5) Write a java program that maintains a separate linked list denoting marks of students from two divisions. Perform the following operations on Linked List:
a. Merge both the list
b. Sort
c. Minimum and maximum
d. Split failed and passed
e. Count number of students getting above 90 marks
CODE:

```
import java.util.*;

class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class LinkedList {
    Node head;

    public LinkedList() {
        this.head = null;
    }

    public void insert(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
        } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
    }

    public void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public void merge(LinkedList otherList) {
        Node current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = otherList.head;
    }

    public void sort() {
        if (head == null || head.next == null) {
            return;
        }
        Node current = head;
        while (current != null) {
            Node index = current.next;
            while (index != null) {
                if (current.data > index.data) {
                    int temp = current.data;
                    current.data = index.data;
                    index.data = temp;
                }
                index = index.next;
            }
            current = current.next;
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
        }
    }

    public int getMinimum() {
        if (head == null) {
            return -1;
        }
        int min = head.data;
        Node current = head.next;
        while (current != null) {
            if (current.data < min) {
                min = current.data;
            }
            current = current.next;
        }
        return min;
    }

    public int getMaximum() {
        if (head == null) {
            return -1;
        }
        int max = head.data;
        Node current = head.next;
        while (current != null) {
            if (current.data > max) {
                max = current.data;
            }
            current = current.next;
        }
        return max;
    }

    public LinkedList splitPassedAndFailed(int passThreshold) {
        LinkedList passedList = new LinkedList();
        Node current = head;
        while (current != null) {
            if (current.data >= passThreshold) {
                passedList.insert(current.data);
            }
            current = current.next;
        }
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
        current = head;
        while (current != null) {
            if (current.data < passThreshold) {
                current.data = -1;
            }
            current = current.next;
        }
        return passedList;
    }

    public int countAboveThreshold(int threshold) {
        int count = 0;
        Node current = head;
        while (current != null) {
            if (current.data > threshold) {
                count++;
            }
            current = current.next;
        }
        return count;
    }
}

public class P4 {
    public static void main(String[] args) {
        LinkedList division1 = new LinkedList();
        division1.insert(85);
        division1.insert(95);
        division1.insert(70);
        division1.insert(60);
        division1.insert(80);
        LinkedList division2 = new LinkedList();
        division2.insert(75);
        division2.insert(90);
        division2.insert(65);
        division2.insert(55);
        division2.insert(100);
        System.out.println("Division 1 marks:");
        division1.display();
        System.out.println("Division 2 marks:");
        division2.display();
        division1.merge(division2);
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
        System.out.println("Merged marks:");
        division1.display();
        division1.sort();
        System.out.println("Sorted marks:");
        division1.display();
        int minMark = division1.getMinimum();
        int maxMark = division1.getMaximum();
        System.out.println("Minimum Mark: " + minMark);
        System.out.println("Maximum Mark: " + maxMark);
        int passThreshold = 60;
        LinkedList passedList = division1.splitPassedAndFailed(passThreshold);
        System.out.println("Passed Marks:");
        passedList.display();
        System.out.println("Failed Marks:");
        division1.display();
        int above90Count = division1.countAboveThreshold(90);
        System.out.println("Number of students getting above 90 marks: " +
                above90Count);
    }
}
```

OUTPUT:

```
Division 1 marks:
85 95 70 60 80
Division 2 marks:
75 90 65 55 100
Merged marks:
85 95 70 60 80 75 90 65 55 100
Sorted marks:
55 60 65 70 75 80 85 90 95 100
Minimum Mark: 55
Maximum Mark: 100
Passed Marks:
60 65 70 75 80 85 90 95 100
Failed Marks:
-1 60 65 70 75 80 85 90 95 100
Number of students getting above 90 marks: 2
```

6) There are a number of students in a school who wait to be served. Two types of events, ENTER and SERVED, can take place which are described below. ENTER: A student with some priority enters the queue to be served. SERVED: The student with the highest priority is served (removed) from the queue. A unique id is assigned to each student entering the queue. The queue serves the students based on the following criteria (priority criteria): The student having the highest Cumulative Grade Point Average (CGPA) is served first. Any

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

students having the same CGPA will be served by name in ascending case-sensitive alphabetical order. Any students having the same CGPA and name will be served in ascending order of the id. Create the following two classes: The Student class should implement: The constructor Student(int id, String name, double cgpa). The method int getID() to return the id of the student. The method String getName() to return the name of the student. The method double getCGPA() to return the CGPA of the student. The Priorities class should implement the method List getStudents(List events) to process all the given events and return all the students yet to be served in the priority order.

CODE:

```
import java.util.*;

class Student {
    private int id;
    private String name;
    private double cgpa;

    public Student(int id, String name, double cgpa) {
        this.id = id;
        this.name = name;
        this.cgpa = cgpa;
    }

    public int getID() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getCGPA() {
        return cgpa;
    }
}

class Priorities {
    public List<Student> getStudents(List<String> events) {
        PriorityQueue<Student> priorityQueue = new PriorityQueue<>(
                (a, b) -> {
                    if (a.getCGPA() != b.getCGPA()) {
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
                        return Double.compare(b.getCGPA(), a.getCGPA()); //
Higher CGPA first
                    } else if (!a.getName().equals(b.getName())) {
                        return a.getName().compareTo(b.getName()); //
Alphabetical order
                    } else {
                        return Integer.compare(a.getID(), b.getID()); // Lower
ID first
                    }
                });
        for (String event : events) {
            String[] parts = event.split(" ");
            if (parts[0].equals("ENTER")) {
                String name = parts[1];
                double cgpa = Double.parseDouble(parts[2]);
                int id = Integer.parseInt(parts[3]);
                Student student = new Student(id, name, cgpa);
                priorityQueue.add(student);
            } else if (parts[0].equals("SERVED")) {
                priorityQueue.poll(); // Remove the highest priority student
            }
        }
        List<Student> students = new ArrayList<>(priorityQueue.size());
        while (!priorityQueue.isEmpty()) {
            students.add(priorityQueue.poll());
        }
        return students;
    }
}

public class P5 {
    public static void main(String[] args) {
        Priorities priorities = new Priorities();
        List<String> events = Arrays.asList(
                "ENTER John 3.75 50",
                "ENTER Mark 3.5 45",
                "ENTER Steve 3.75 35",
                "SERVED",
                "SERVED",
                "ENTER Andy 3.85 55",
                "ENTER Bob 3.65 44",
                "SERVED");
```

COURSE CODE: DJS22ITL306 **DATE:29/9/2023**
COURSE NAME: Programing Laboratory 1 (Advanced Java) **CLASS: I1-Batch1**

```
        List<Student> students = priorities.getStudents(events);
        for (Student student : students) {
            System.out.println(student.getName());
        }
    }
}
```

OUTPUT:

```
Bob
Mark
```

OBSERVATION:

1. **What do you mean by Collection? List few interfaces and classes of collection.**

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Interfaces:

1 Collection: This is the root interface of the Java Collections Framework. It defines the basic methods and behavior that all collections should have. Subinterfaces include List, Set, and Queue.

2. List: This interface represents an ordered collection of elements where duplicate elements are allowed. Some common implementations are ArrayList, LinkedList, and Vector.

3. Set: This interface represents an unordered collection of unique elements. Duplicate elements are not allowed. Some common implementations are HashSet, LinkedHashSet, and TreeSet.

4. Queue: This interface represents a collection designed for holding elements prior to processing. It typically follows the FIFO (First-In-First-Out) order. Some common implementations are LinkedList and PriorityQueue.

5. Map: Although not a subtype of Collection, Map is an important part of the Java Collections Framework. It represents a collection of key-value pairs where each key is

associated with a unique value. Some common implementations are HashMap, LinkedHashMap, and TreeMap.

Classes:

1. ArrayList: An implementation of the List interface that uses a dynamic array to store elements. It is resizable and allows fast random access.

2. LinkedList: Another implementation of the List interface that uses a doubly-linked list to store elements. It is efficient for insertions and deletions but slower for random access.

3. HashSet: An implementation of the Set interface that uses a hash table to store elements. It provides constant-time average complexity for basic operations like add, remove, and contains.

4. TreeSet: An implementation of the Set interface that stores elements in a sorted order using a red-black tree. It maintains elements in ascending order.

5. HashMap: An implementation of the Map interface that uses a hash table to store key-value pairs. It provides efficient key-value lookups and is not ordered.

2. What is the need of collections? How it helps to represent any data structure?

Collections are a fundamental concept in computer programming, and they are used to store, retrieve, manipulate, and communicate aggregate data. They represent data items that form a natural group. For example, a poker hand (a collection of cards), a mail folder (a collection of letters), or a telephone directory (a mapping of names to phone numbers) can all be represented as collections.
In the context of data structures, collections provide an architecture to store and manipulate a group of objects. They can achieve all the operations that you perform on data such as searching, sorting, insertion, manipulation, and deletion.
Linear data structures like arrays, stacks, queues, linked lists where data elements are arranged sequentially.
Non-linear data structures like trees and graphs where data elements are not placed sequentially.
In Java specifically, a separate framework named the “Collection Framework” has been defined which holds all the collection classes and interfaces in it. This framework provides a standard way to handle a group of objects so that they can be accessed and manipulated efficiently.
So, the need for collections arises from the requirement to efficiently store, access, and manipulate data in a structured manner. They provide a flexible way to represent real-world entities as data structures in our programs. Collections are an essential part of any programming language and mastering them can greatly enhance your problem-solving skills.

CONCLUSION: I learned the implementation of Collection in JAVA and its its application via performing the experiments.