StudentManagementSystem

Step1:

Create a Student class

```
public class Student {
    //Private Attributes
    private int id;
    private String name;
    private int age;
    private String course;

    //Constructor
    public Student(int id, String name, int age, String
course) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.course = course;
    }

    //Getters & Setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

```
    public String getCourse() {
        return course;
    }

    public void setCourse(String course) {
        this.course = course;
    }


    //Overriding toString()
    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", course='" + course + '\'' +
                '}';
    }

}
```

Step2:

Create an interface StudentOperations

```
public interface StudentOperations {
    void addStudent(Student student);
    Student getStudent(int id);
    void updateStudent(int id, Student updatedStudent);
    void deleteStudent(int id);
    void displayAllStudents();
}
```

Step3:

Create an implementation class StudentOperationImpl

```
import java.util.HashMap;

public class StudentOperationsImpl implements
StudentOperations{
    // Using hashMap to store student object along with their
ID
    private HashMap<Integer,Student> studentMap = new
HashMap<>();

    @Override
    public void addStudent(Student student) {
        //to add student to a map
        studentMap.put(student.getId(),student);
        System.out.println("Student Added Successfully!!!!!");
    }

    @Override
    public Student getStudent(int id) {
        return studentMap.get(id);
    }


    @Override
    public void updateStudent(int id, Student updatedStudent)
{
        //check if the student exists in a map
        if (studentMap.containsKey(id)){
            studentMap.put(id,updatedStudent);
            System.out.println("Student Updated
Successfully!!!!!");
        } else {
            System.out.println("Student Not Found!!!!");
        }
    }

    @Override
    public void deleteStudent(int id) {
        //To remove the student
        if (studentMap.remove(id) != null){
            System.out.println("Student Deleted
Successfully!!!");
        } else {
            System.out.println("Student Not Found");
        }
```

```
    }

    @Override
    public void displayAllStudents() {
            if (studentMap.isEmpty()){
                System.out.println("No Students in Map");
            } else {
                for (Student student: studentMap.values()){
                    System.out.println(student);
                }
            }
    }
}
```

Step4:

Create a StudentManagementSystem class with main method.

```
import java.util.Scanner;

public class StudentManagementSystem {
    public static void main(String[] args) {
        StudentOperations operations = new
StudentOperationsImpl();
        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("Welcome!!!");
            System.out.println("1. Add Student");
            System.out.println("2. Get Student By ID");
            System.out.println("3. Update Student");
            System.out.println("4. Delete Student");
            System.out.println("5. Display All Students");
            System.out.println("6. Exit! ");
            System.out.println("Enter Choice: ");
            choice = scanner.nextInt();

            switch (choice){

                case 1:
                    System.out.println("Enter ID:");
                    int id = scanner.nextInt();
                    scanner.nextLine();  //To consume new line
```

```
                    System.out.println("Enter Your Name: ");
                    String name = scanner.nextLine();
                    System.out.println("Enter Your Age: ");
                    int age = scanner.nextInt();
                    scanner.nextLine();  //To consume new line
                    System.out.println("Enter Your Course: ");
                    String course = scanner.nextLine();
                    operations.addStudent(new
Student(id,name,age,course));
                    break;

                case 2:
                    //Get student by ID
                    System.out.println("Enter Student ID to
diaplay: ");
                    id = scanner.nextInt();
                    Student student =
operations.getStudent(id);
                    if (student != null){
                        System.out.println(student);
                    }else {
                        System.out.println("Student not
Found!!");
                    }
                    break;

                case 3:
                    System.out.println("Enter Student ID to
update:  ");
                    id = scanner.nextInt();
                    scanner.nextLine();
                    System.out.println("Enter New Name: ");
                    name = scanner.nextLine();
                    System.out.println("Enter New Age: ");
                    age = scanner.nextInt();
                    scanner.nextLine();
                    System.out.println("Enter New Course: ");
                    course = scanner.nextLine();
                    operations.updateStudent(id, new
Student(id, name,age,course));
                    break;

                case 4:
                    System.out.println("Enter Student ID to
delete: ");
                    id = scanner.nextInt();
                    operations.deleteStudent(id);
                    break;

                case 5:
                    System.out.println("List of Students:");
```

```
                    operations.displayAllStudents();
                    break;

                case 6:
                    System.out.println("Thank You!!!");
                    break;


                default:
                    System.out.println("Invalid
Choice!!!!!!!");
            }
        }while(choice != 6);
    }
}
```

LinkedHashMap

A subclass of HashMap

It maintains the insertion order of elements

Characteristics:

- Not Synchronized
- Slower than the HashMap because It maintains the insertion order.

LinkedHashMap (Java SE 21 & JDK 21)

```
package mapExample;

import java.util.LinkedHashMap;

public class LinkedHashMapExample {
    public static void main(String[] args) {
        //Creating LHM
        LinkedHashMap<String,String> linkedHM = new
LinkedHashMap<>();

        //To add elements
        linkedHM.put("Krishna","CEO");
        linkedHM.put("Govind","Developer");
```

```
        linkedHM.put("Gopal","Tester");

        //To retrieve value by key
        System.out.println("Role of Govind: "+
linkedHM.get("Govind"));

        //To get all values
        System.out.println("Values of LHM : "+
linkedHM.values());
    }
}
```

FAQs:

Does LinkedHashMap maintain insertion order of elements?

➔ Yes

Can LinkedHashMap store null keys?

➔ Yes, LHM allows one null key and multiple null values

When to use which map?

Hashtable: When thread-safety is required

HashMap: For faster operations in single threaded environment

LinkedHashMap: When maintaining an insertion order is important.

TreeMap

Features:

- Sorted Order: Keys are maintained in natural order(Ascending)
- Unique Keys: Keys must be unique, if we are attempting to insert duplicate keys it will overwrite the existing value.
- Null Handling:
  - Keys: TreeMap Does not allow null keys
  - Values: It allows multiple null values
- Thread Safety:
  - It is not synchronized.
  - For thread-safety we can wrap it inside Collections.synchronizedMap

synchronizedMap(**Map**<K,V> m)
Returns a synchronized (thread-safe) map backed by the specified map.

What is the difference between Collection & Collections?

TreeMap (Java SE 21 & JDK 21)

Commonly used methods:

Put(K Key, V value)

Get(Object key)

firstKey()

lastKey()

```
package mapExample;

import java.util.TreeMap;

public class TreeMapExample {
    public static void main(String[] args) {
        //Create TreeMap
        TreeMap<Integer,String> treeMap = new TreeMap<>();

        //Adding elements
        treeMap.put(3, "Three");
        treeMap.put(1,"One");
        treeMap.put(4,"Four");
        treeMap.put(2,"Two");
        treeMap.put(1, "Duplicate");

        //Displaying the map
        System.out.println("TreeMap: "+ treeMap);

        //Accessing the element
        System.out.println("Value for key 2nd: "+
treeMap.get(2));

        //to remove the element
        treeMap.remove(2);
        System.out.println("After Removal: "+ treeMap);

        System.out.println("First Key : "+
treeMap.firstKey());
        System.out.println("Last Key: "+ treeMap.lastKey());

    }
}
```

TASK:

What are multiple ways to reverse the order.