Date:

ADDITIONAL EXPERIMENT-1

Title: "Student Marks Management System using Collections, Generics, Lambda Expressions & Streams API." Aim: Write a Java program to manage a list of students and their marks using **Collections** and **Generics.**

Source Code:

```
import java.util.*; import java.util.stream.Collectors;

// Generic Class for Student class Student<T> {     private T rollNo;     private String name;     private int
marks1, marks2, marks3;

   public Student(T rollNo, String name, int marks1, int marks2, int marks3) {         this.rollNo = rollNo;
this.name = name;         this.marks1 = marks1;         this.marks2 = marks2;         this.marks3 = marks3;

   }

   public T getRollNo() {         return rollNo;

   }

   public String getName() {

      return name;

   }

   public int getTotalMarks() {         return marks1 + marks2 + marks3;

   }

   @Override

   public String toString() {

      return rollNo + " " + name + " - Total: " + getTotalMarks();

   }

}

public class StudentMarksManagement {

 public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

      List<Student<Integer>> students = new ArrayList<>();

      System.out.print("Enter number of students: ");         int n = sc.nextInt();

      for (int i = 0; i < n; i++) {

         System.out.println("\nEnter details for Student " + (i + 1));

         System.out.print("Enter Roll No: ");

                int rollNo = sc.nextInt();

                sc.nextLine(); // consume newline

                System.out.print("Enter Name: ");
```

```
           String name = sc.nextLine();

           System.out.print("Enter Marks in 3 subjects: ");

                   int m1 = sc.nextInt();

                   int m2 = sc.nextInt();

                   int m3 = sc.nextInt();

           students.add(new Student<>(rollNo, name, m1, m2, m3));

        }

        // Sort by Name using Lambda         students.sort((s1, s2) ->
s1.getName().compareToIgnoreCase(s2.getName()));

        System.out.println("\n--- All Students Sorted by Name ---");

                   students.forEach(System.out::println);

        // Filter students with total marks > 200 using Streams

        List<Student<Integer>> above200 = students.stream()

              .filter(s -> s.getTotalMarks() > 200)

              .collect(Collectors.toList());

                    System.out.println("\n--- Students with Total Marks > 200 ---");
                   above200.forEach(System.out::println);

        // Find topper using Streams         students.stream()

              .max(Comparator.comparingInt(Student::getTotalMarks))

              .ifPresent(s -> System.out.println("\nTopper: " + s.getName() + " with " + s.getTotalMarks() + "
marks"));

        // Average marks using Streams         double avg = students.stream()

              .mapToInt(Student::getTotalMarks)

              .average()

              .orElse(0.0);

        System.out.println("Average Marks: " + avg);

        sc.close();

    }

}
```

Test Case

Input:

Enter number of students: 3

Enter Roll No: 101

Enter Name: Amit

```
Enter Marks in 3 subjects: 85 90 80

Enter Roll No: 102

Enter Name: John

Enter Marks in 3 subjects: 70 65 75

Enter Roll No: 103

Enter Name: Zara

Enter Marks in 3 subjects: 95 85 90
```

Output:

```
--- All Students Sorted by Name ---

            101 Amit - Total: 255

            102 John - Total: 210

            103 Zara - Total: 270

--- Students with Total Marks > 200 --101 Amit - 255 103 Zara – 270

Topper: Zara with 270 marks

Average Marks: 245.0
```

Date:

ADDITIONAL EXPERIMENT-2

Title: "Library Book Management System using Collections, Generics, Lambda Expressions & Streams API"

Aim: Write a Java program to manage a library's book collection using **Collections** and **Generics**.

Source Code:

```
import java.util.*; import java.util.stream.Collectors;

// Generic Class for Book class Book<T> {

            private T bookId;

            private String title;

            private String author;

            private double price;

  public Book(T bookId, String title, String author, double price) {

            this.bookId = bookId;

            this.title = title;

            this.author = author;

            this.price = price;

  }

  public T getBookId() {

             return bookId;

  }

  public String getTitle() {

    return title;

  }

  public String getAuthor() {

    return author;

  }

  public double getPrice() {

    return price;    }

  @Override

  public String toString() {

    return bookId + " " + title + " - Rs. " + price;

  }
```

```
}
public class LibraryBookManagement {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
        List<Book<String>> books = new ArrayList<>();
        System.out.print("Enter number of books: ");
        int n = sc.nextInt();
          sc.nextLine(); // consume newline
        for (int i = 0; i < n; i++) {
            System.out.println("\nEnter details for Book " + (i + 1));
            System.out.print("Enter Book ID: ");
            String id = sc.nextLine();
            System.out.print("Enter Title: ");
                    String title = sc.nextLine();
            System.out.print("Enter Author: ");
                    String author = sc.nextLine();
            System.out.print("Enter Price: ");
                    double price = sc.nextDouble();
                    sc.nextLine(); // consume newline
            books.add(new Book<>(id, title, author, price));
        }
        // Sort by Price using Lambda
          books.sort((b1, b2) -> Double.compare(b1.getPrice(), b2.getPrice()));
        System.out.println("\n--- All Books Sorted by Price ---");
                    books.forEach(System.out::println);
        // Filter books with price > 500 using Streams
        List<Book<String>> expensiveBooks = books.stream()
                .filter(b -> b.getPrice() > 500)
                .collect(Collectors.toList());
        System.out.println("\n--- Books with Price > 500 ---");
                    expensiveBooks.forEach(System.out::println);
        // Find most expensive book         books.stream()
                .max(Comparator.comparingDouble(Book::getPrice))
            .ifPresent(b -> System.out.println("\nMost Expensive Book: " + b.getTitle() + " - Rs. " +
b.getPrice()));
        // Calculate average price
                double avgPrice = books.stream()
                 .mapToDouble(Book::getPrice)
                 .average()
                 .orElse(0.0);
        System.out.println("Average Price: " + avgPrice);
        sc.close();
    }
}
```

Test Case

Input:

```
Enter number of books: 3
Enter Book ID: B101
Enter Title: Java Programming
Enter Author: James Gosling
Enter Price: 450
Enter Book ID: B102
Enter Title: Data Structures
Enter Author: Robert Lafore
Enter Price: 550
Enter Book ID: B103
Enter Title: Effective Java
Enter Author: Joshua Bloch
Enter Price: 800
```

Output:

```
--- All Books Sorted by Price --B101 Java Programming - Rs. 450
B102 Data Structures - Rs. 550
B103 Effective Java - Rs. 800 --- Books with Price > 500 --B102 Data Structures - Rs. 550
B103 Effective Java - Rs. 800
Most Expensive Book: Effective Java - Rs. 800
Average Price: 600.0
```