Experiment - 5

Name: Akash **UID: 22BET10018**
Branch: BE-IT **Section: 22BET_IOT_703/A**
Semester: 6th **DOP: 21/02/25**
Subject: Project Based Learning in Java **Subject Code: 22ITH-359**

Problem 1:

1. **Aim:** Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).

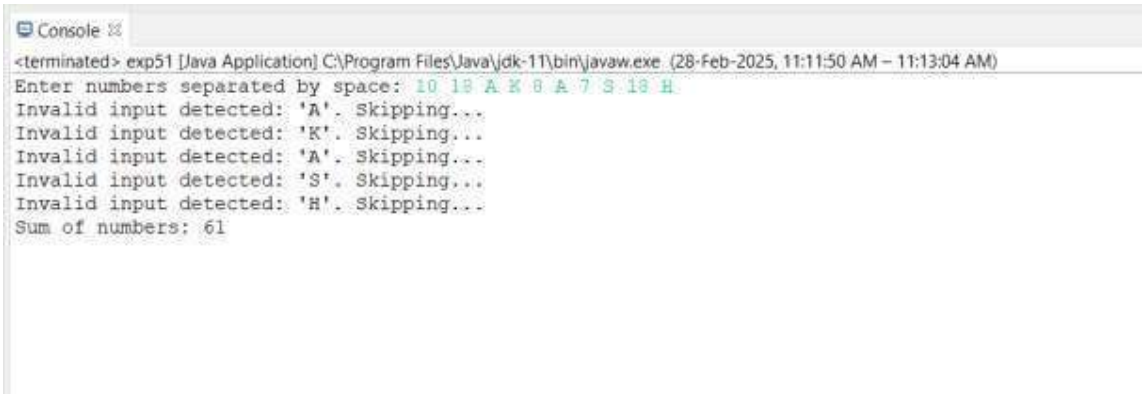
2. **Objective:**
- Understand and implement autoboxing and unboxing in Java.
- Parse strings to integer values using Integer.parseInt() and handle invalid inputs.
- Use Java Collections (e.g., List and ArrayList) to store and process integer values.
- Develop a method to calculate the sum of a list of integers using streams.

3. **Code:**

```
import java.util.*;

public class exp51 { //22BET10018_AKASH
    public static int calculateSum(List<Integer> numbers) {
        return numbers.stream().mapToInt(Integer::intValue).sum();
    }

    public static List<Integer> parseStringToIntegers(List<String> strNumbers) {
        List<Integer> numbers = new ArrayList<>();
        for (String str : strNumbers) {
            try {
                numbers.add(Integer.parseInt(str));
            } catch (NumberFormatException e) {
                System.out.println("Invalid input detected: '" + str + "'. Skipping...");
            }
        }
        return numbers;
    }
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter numbers separated by space: ");
        String input = scanner.nextLine();
        scanner.close();
        List<String> strNumbers = Arrays.asList(input.trim().split("\\s+"));
        List<Integer> numbers = parseStringToIntegers(strNumbers);

        if (numbers.isEmpty()) {
            System.out.println("No valid numbers entered.");
        } else {
            System.out.println("Sum of numbers: " + calculateSum(numbers));
        }
    }
}
```

4. Output:



5. Learning Outcomes:

- Comprehend the process of autoboxing and unboxing between primitive data types and their wrapper classes.
- Learn how to convert strings into integers using Integer.parseInt().
- Use Java collections to store wrapper objects effectively.
- Traverse collections using enhanced for-loops.

Problem 2:

1. **Aim:** Create a Java program to serialize and deserialize a Student object. The program should: Serialize a Student object (containing id, name, and GPA) and save it to a file. Deserialize the object from the file and display the student details. Handle FileNotFoundException, IOException, and ClassNotFoundException using exception handling.

2. Objective:

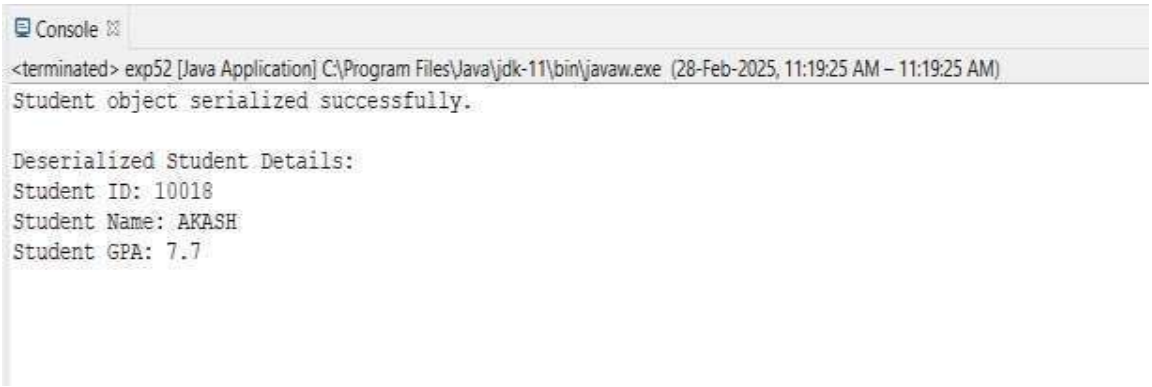
- Learn the concept and implementation of object serialization and deserialization in Java.
- Implement the Serializable interface to enable object persistence.
- Use ObjectOutputStream and ObjectInputStream for file-based object storage and retrieval.
- Handle common exceptions such as FileNotFoundException, IOException, and ClassNotFoundException during file operations.

3. Code:

```
import java.io.*;
class Student implements Serializable { //22BET10018_AKASH
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private double gpa;

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

    public void display() {
        System.out.println("Student ID: " + id);
        System.out.println("Student Name: " + name);
        System.out.println("Student GPA: " + gpa);
    }
}

public class exp52 {
    private static final String FILE_NAME = "student.ser";

    public static void serializeStudent(Student student) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {
            oos.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found. " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Error: Unable to serialize object. " + e.getMessage());
        }
    }
```

```
    public static void deserializeStudent() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
            Student student = (Student) ois.readObject();
            System.out.println("\nDeserialized Student Details:");
            student.display();
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found. Run serialization first. " + e.getMessage());
        } catch (IOException e) {


            System.out.println("Error: Unable to deserialize object. " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.out.println("Error: Student class not found. " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        Student student = new Student(10018, "AKASH", 7.7);
        serializeStudent(student);
        deserializeStudent();
    }
}
```

4. Output:



5. Learning Outcomes:

- Gain knowledge of Java serialization and deserialization processes.
- Implement the Serializable interface in Java classes.
- Apply try-with-resources for efficient stream management.
- Manage exceptions including FileNotFoundException, IOException, and ClassNotFoundException.

Problem 3:

1. **Aim:** Create a menu-based Java application with the following options. 1.Add an Employee 2. Display All 3. Exit If option 1 is selected, the application should gather details of the employee like employee name, employee id, designation and salary and store it in a file. If option 2 is selected, the application should display all the employee details. If option 3 is selected the application should exit.

2. Objective:

- Create a menu-driven application for managing employee records.
- Collect and store employee details (ID, name, designation, salary) using object serialization.
- Retrieve and display all stored employee records from a file.
- Implement robust exception handling for file operations and user input errors.

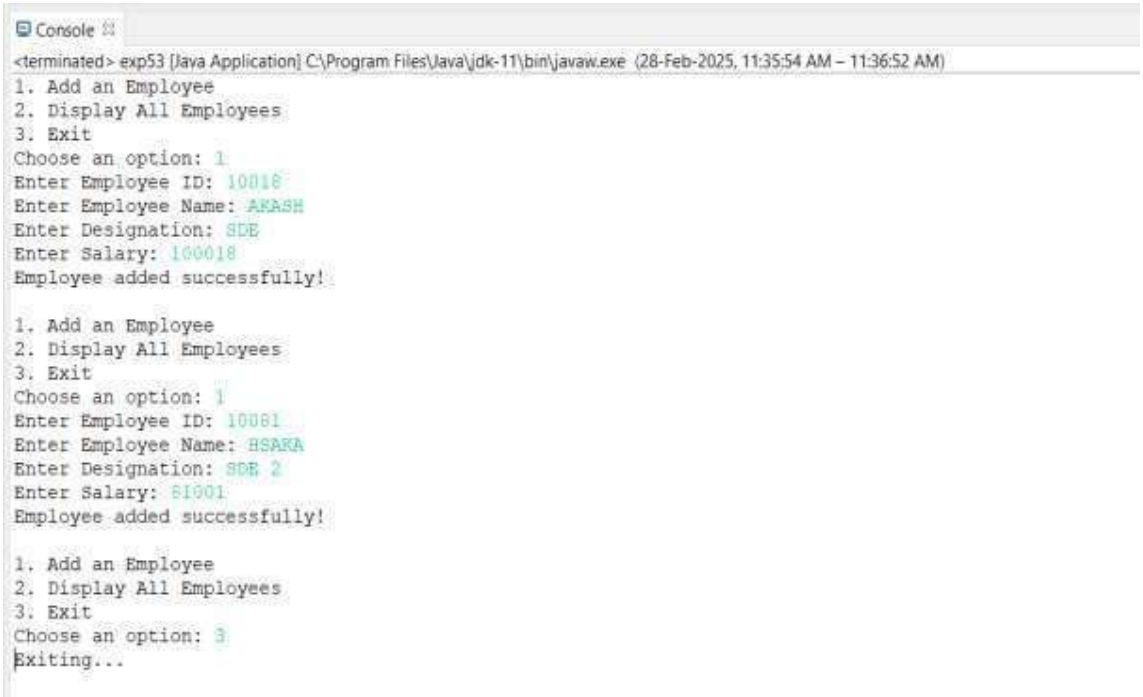
3. Code:

```
//22BET10018_AKASH
import java.io.*;
import java.util.*;

class Employee1 implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private double salary;

    public Employee1(int id, String name, String designation, double salary) {
        this.id = id;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " + salary;
    }
}

public class exp53 {
    private static final String FILE_NAME = "employees.dat";
    private static Scanner scanner = new Scanner(System.in);

    public static void addEmployee() {
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
```

```
        scanner.nextLine(); // Consume newline

        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();

        System.out.print("Enter Designation: ");
        String designation = scanner.nextLine();

        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();

        Employee1 employee = new Employee1(id, name, designation, salary);
        saveEmployeeToFile(employee);
        System.out.println("Employee added successfully!\n");
    }

    public static void saveEmployeeToFile(Employee1 employee) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE_NAME,
true))) {
            oos.writeObject(employee);
        } catch (IOException e) {
            System.out.println("Error: Unable to save employee.");
        }
    }

    public static void displayAllEmployees() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
            System.out.println("\nEmployee Details:");
            while (true) {
                Employee1 emp = (Employee1) ois.readObject();
                System.out.println(emp);
            }
        } catch (EOFException e) {
            // End of file reached
        } catch (FileNotFoundException e) {
            System.out.println("No employee records found.\n");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error reading employee records.\n");
        }
    }

    public static void main(String[] args) {
        while (true) {
            System.out.println("1. Add an Employee");
            System.out.println("2. Display All Employees");
            System.out.println("3. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    addEmployee();
                    break;
```

```
                case 2:
                    displayAllEmployees();
                    break;
                case 3:
                    System.out.println("Exiting...");
                    return;
                default:



                    System.out.println("Invalid choice! Try again.\n");
            }
        }
    }
}
```

4. Output:



5. Learning Outcomes:

- Develop a menu-based console application.
- Gain knowledge of file input/output for reading and writing text files.
- Apply exception handling techniques for file operations.
- Enhance skills in collecting and managing user input using the Scanner class.