Experiment- 5

Student Name: Kamal Mehta	**UID:** 22BET10097
Branch: B.E - IT	**Section/Group:** 22BET-701/A
Semester: 6th	**Date of Performance:** 18-02-25
Subject Name: PBLJ Lab	**Subject Code:** 22ITH-359

Problem 1

1. **Aim:** To develop a Java program that calculates the sum of a list of integers using autoboxing and unboxing, and demonstrates the use of wrapper classes for parsing strings into their respective types.

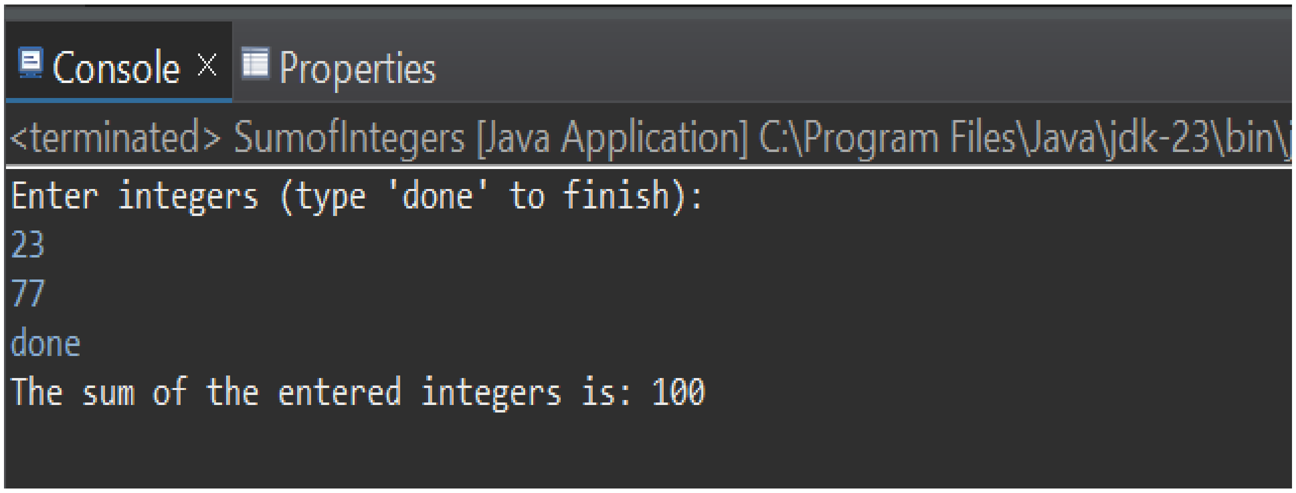
2. **Objective:**
   - To calculate the sum of a list of integers using autoboxing and unboxing.
   - Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).

3. **Code:**

```
package main;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class SumofIntegers {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      List<Integer> integerList = new ArrayList<>();

      System.out.println("Enter integers (type 'done' to finish):");
      while (true) {
         String input = scanner.nextLine();
```

```
            if (input.equalsIgnoreCase("done")) {
                break;
            }
            try {
                Integer number = Integer.parseInt(input);
                integerList.add(number);
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid integer.");
            }        }
        int sum = calculateSum(integerList);
        System.out.println("The sum of the entered integers is: " + sum);
        scanner.close();
    }
    private static int calculateSum(List<Integer> integers) {
        int sum = 0;
        for (Integer num : integers) {
            sum += num;        }
        return sum;
    }
}
```

4. Output:



```
<terminated> SumofIntegers [Java Application] C:\Program Files\Java\jdk-23\bin\j
Enter integers (type 'done' to finish):
23
77
done
The sum of the entered integers is: 100
```

Fig 1: Output for Problem 1

Problem 2

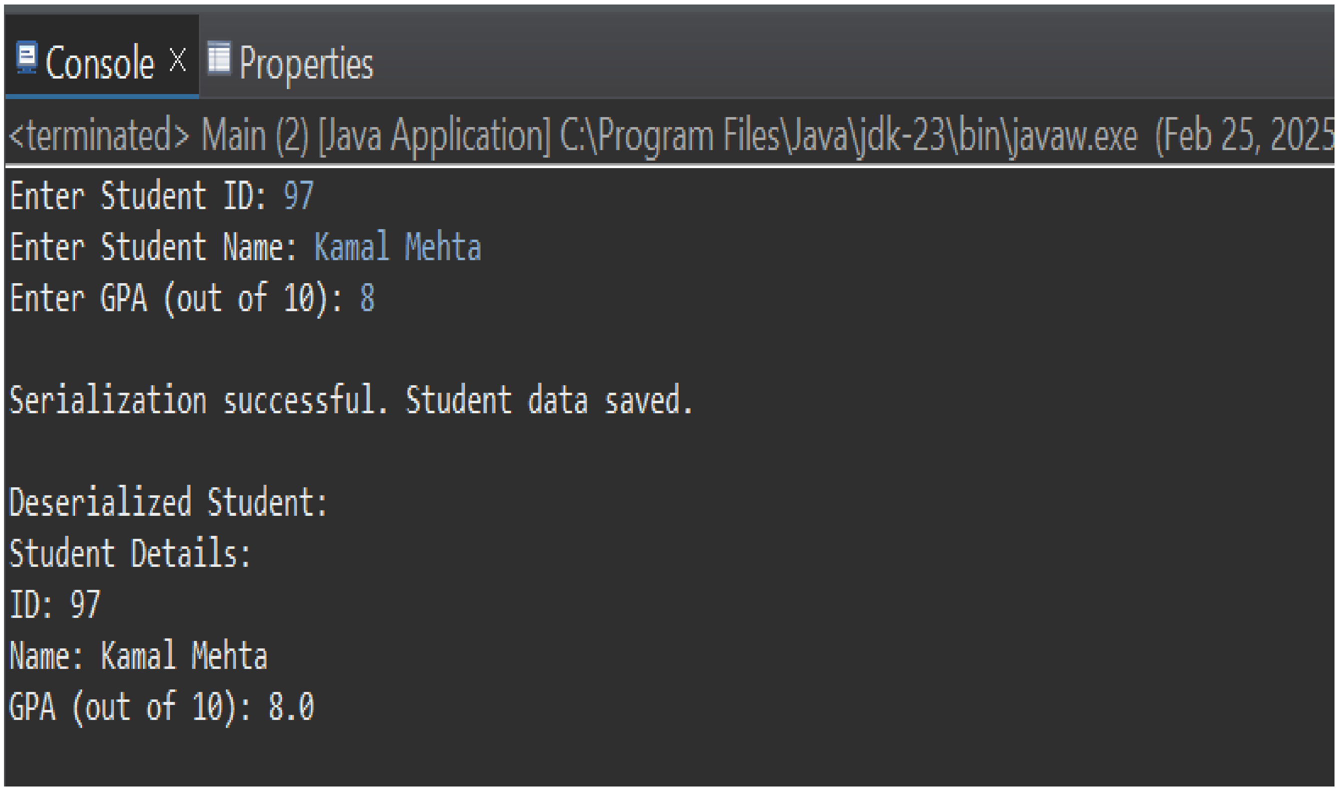
1. **Aim:** Create a Java program to serialize and deserialize a Student object.

2. **Objective:**
   - To 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.

3. **Code:**

```
package Main;

import java.io.*;
import java.util.Scanner;

class Student implements Serializable {
   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;
   }
   @Override
   public String toString() {
      return "Student Details:\n" +
            "ID: " + id + "\n" +
               "Name: " + name + "\n" +
               "GPA (out of 10): " + gpa;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            System.out.print("Enter Student ID: ");
            int id = scanner.nextInt()
            scanner.nextLine();

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

            System.out.print("Enter GPA (out of 10): ");
            double gpa = scanner.nextDouble();

            Student student = new Student(id, name, gpa);

            try (ObjectOutputStream oos =
                 new ObjectOutputStream(new FileOutputStream("student_data.ser"))) {

                oos.writeObject(student);
                System.out.println("\nSerialization successful. Student data saved.");

            } catch (IOException e) {
                System.err.println("Error during serialization: " + e.getMessage());
            }

            try (ObjectInputStream ois =
                 new ObjectInputStream(new FileInputStream("student_data.ser"))) {

                Student deserializedStudent = (Student) ois.readObject();
                System.out.println("\nDeserialized Student:\n" + deserializedStudent);
            } catch (ClassNotFoundException | IOException e) {
                System.err.println("Error during deserialization: " + e.getMessage());
            }
        } finally {
            scanner.close();
        }
    }
}
```

4. Output:



```
Console X   Properties
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe  (Feb 25, 2025
Enter Student ID: 97
Enter Student Name: Kamal Mehta
Enter GPA (out of 10): 8

Serialization successful. Student data saved.

Deserialized Student:
Student Details:
ID: 97
Name: Kamal Mehta
GPA (out of 10): 8.0
```

Fig 2: Output for Problem 2

Problem 3

1. **Aim:** To develop a menu-based Java application that manages employee records, demonstrating file handling, data storage, and retrieval.

2. **Objective:**

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

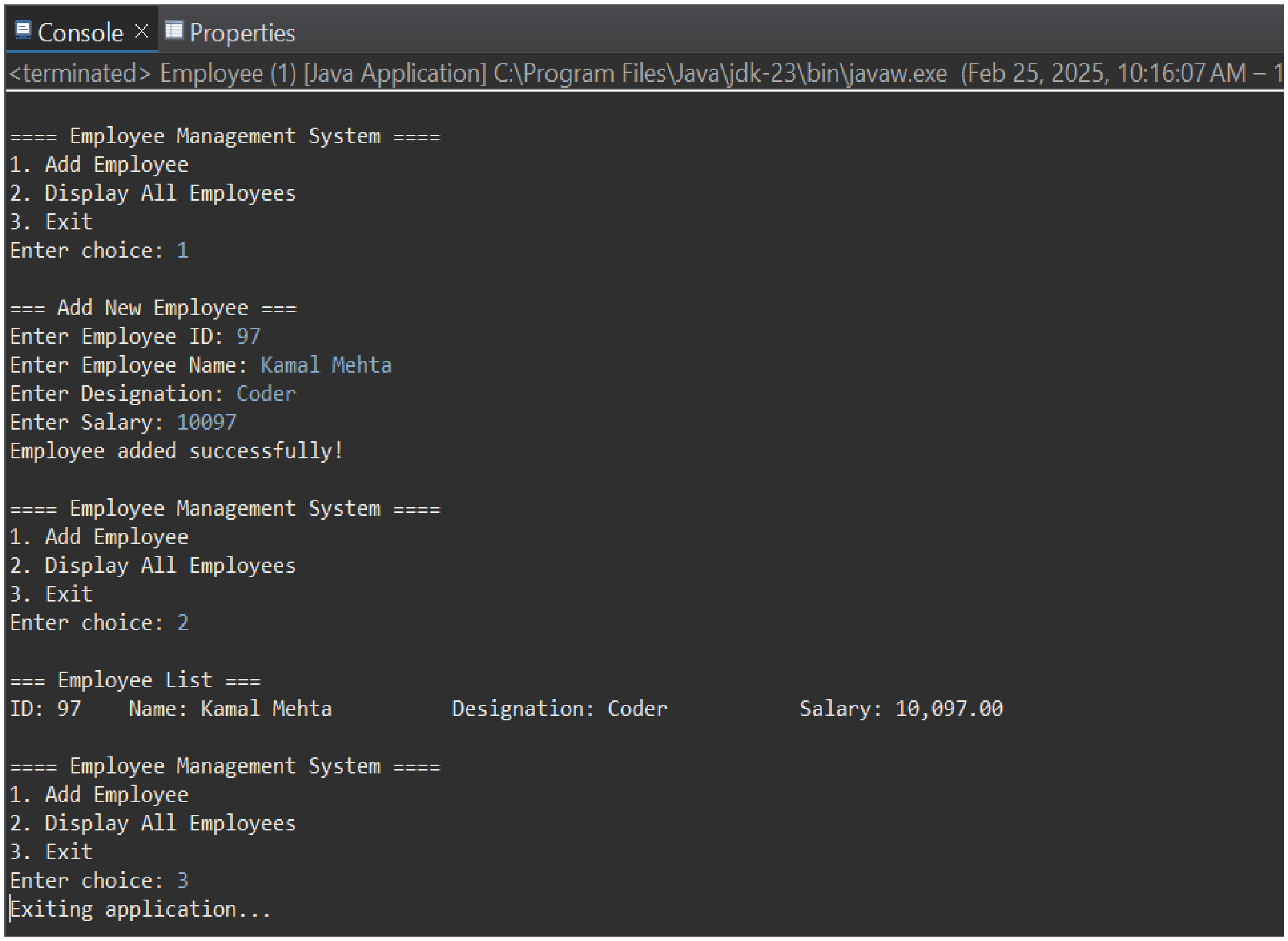
3. **Code:**

```
package main;

import java.io.*;
import java.util.Scanner;

public class Employee {
    private static final String FILE_NAME = "employees.txt";

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

        do {
            printMenu();
            choice = getIntInput(scanner, "Enter choice: ");

            switch(choice) {
```

```
            case 1:
                addEmployee(scanner);
                break;
            case 2:
                displayEmployees();
                break;
            case 3:
                System.out.println("Exiting application...");
                break;
            default:
                System.out.println("Invalid choice! Please try again.");
        }
    } while(choice != 3);

    scanner.close();
}

private static void printMenu() {
    System.out.println("\n==== Employee Management System ====");
    System.out.println("1. Add Employee");
    System.out.println("2. Display All Employees");
    System.out.println("3. Exit");
}

private static void addEmployee(Scanner scanner) {
    System.out.println("\n=== Add New Employee ===");

    int id = getIntInput(scanner, "Enter Employee ID: ");
    scanner.nextLine(); // Clear buffer
    String name = getStringInput(scanner, "Enter Employee Name: ");
    String designation = getStringInput(scanner, "Enter Designation: ");
    double salary = getDoubleInput(scanner, "Enter Salary: ");

    try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_NAME,
true))) {
        String record = String.format("%d|%s|%s|%.2f", id, name, designation, salary);
```

```
            writer.write(record);
            writer.newLine();
            System.out.println("Employee added successfully!");
        } catch (IOException e) {
            System.out.println("Error saving employee data: " + e.getMessage());
        }
    }
    private static void displayEmployees() {
        System.out.println("\n=== Employee List ===");

        File file = new File(FILE_NAME);
        if(!file.exists()) {
            System.out.println("No employees found in the system.");
            return;
        }

        try (BufferedReader reader = new BufferedReader(new FileReader(FILE_NAME)))
{
            String line;
            while((line = reader.readLine()) != null) {
                String[] parts = line.split("\\|");
                if(parts.length == 4) {
                    System.out.printf("ID: %-5d Name: %-20s Designation: %-15s Salary:
%,.2f%n",
                        Integer.parseInt(parts[0]),
                        parts[1],
                        parts[2],
                        Double.parseDouble(parts[3]));
                }
            }
        }
 catch (IOException e) {
            System.out.println("Error reading employee data: " + e.getMessage());
        } catch (NumberFormatException e) {
            System.out.println("Error parsing data: Invalid number format");
        }
    }
    private static int getIntInput(Scanner scanner, String prompt) {
        while(true) {
            try {
                System.out.print(prompt);
                return scanner.nextInt();
            } catch (Exception e) {
                System.out.println("Invalid input! Please enter a valid integer.");
                scanner.nextLine();
            }
        }
    }
    private static double getDoubleInput(Scanner scanner, String prompt) {
        while(true) {
            try {
                System.out.print(prompt);
                return scanner.nextDouble();
            } catch (Exception e) {
                System.out.println("Invalid input! Please enter a valid number.");
                scanner.nextLine();
            }
        }
    }
    private static String getStringInput(Scanner scanner, String prompt) {
        System.out.print(prompt);
        return scanner.nextLine().trim();
    }
}
```

4. Output:



```
<terminated> Employee (1) [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe  (Feb 25, 2025, 10:16:07 AM – 1

==== Employee Management System ====
1. Add Employee
2. Display All Employees
3. Exit
Enter choice: 1

=== Add New Employee ===
Enter Employee ID: 97
Enter Employee Name: Kamal Mehta
Enter Designation: Coder
Enter Salary: 10097
Employee added successfully!

==== Employee Management System ====
1. Add Employee
2. Display All Employees
3. Exit
Enter choice: 2

=== Employee List ===
ID: 97    Name: Kamal Mehta          Designation: Coder          Salary: 10,097.00

==== Employee Management System ====
1. Add Employee
2. Display All Employees
3. Exit
Enter choice: 3
Exiting application...
```

Fig 3: Output for Problem 3

5. Learning Outcome:

1. **Wrapper Classes and Autoboxing:** Understood and effectively used Java's wrapper classes and the concepts of autoboxing and unboxing to handle primitive data types and objects seamlessly.

2. **Serialization and Deserialization:** Gained proficiency in serializing and deserializing objects for data persistence, enabling the storage and retrieval of object states in Java applications.

3. **Exception Handling:** Developed robust exception handling skills to manage file and I/O-related exceptions, ensuring reliable and error-resistant code.
4. **File Handling:** Learned file operations, including reading and writing data, to efficiently manage data storage and retrieval in Java applications.
5. **Interactive Application Design:** Enhanced ability to design and implement interactive, menu-driven applications that facilitate user interaction and data management.
6. **Data Management:** Learned to gather, store, and retrieve complex data structures using file handling techniques, crucial for real-world applications.
7. **Problem-Solving and Integration:** Improved problem-solving skills and integrate various Java concepts to create efficient and effective software solutions.