DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment-5

Name: Aarukh
Branch: BE-CSE
Semester: 6th
Subject Name: Project Based Learning in Java

UID:22BCS50104
Section/Group: 643/B
D.Performance:27/02/2025
Subject Code: 22CSH-359

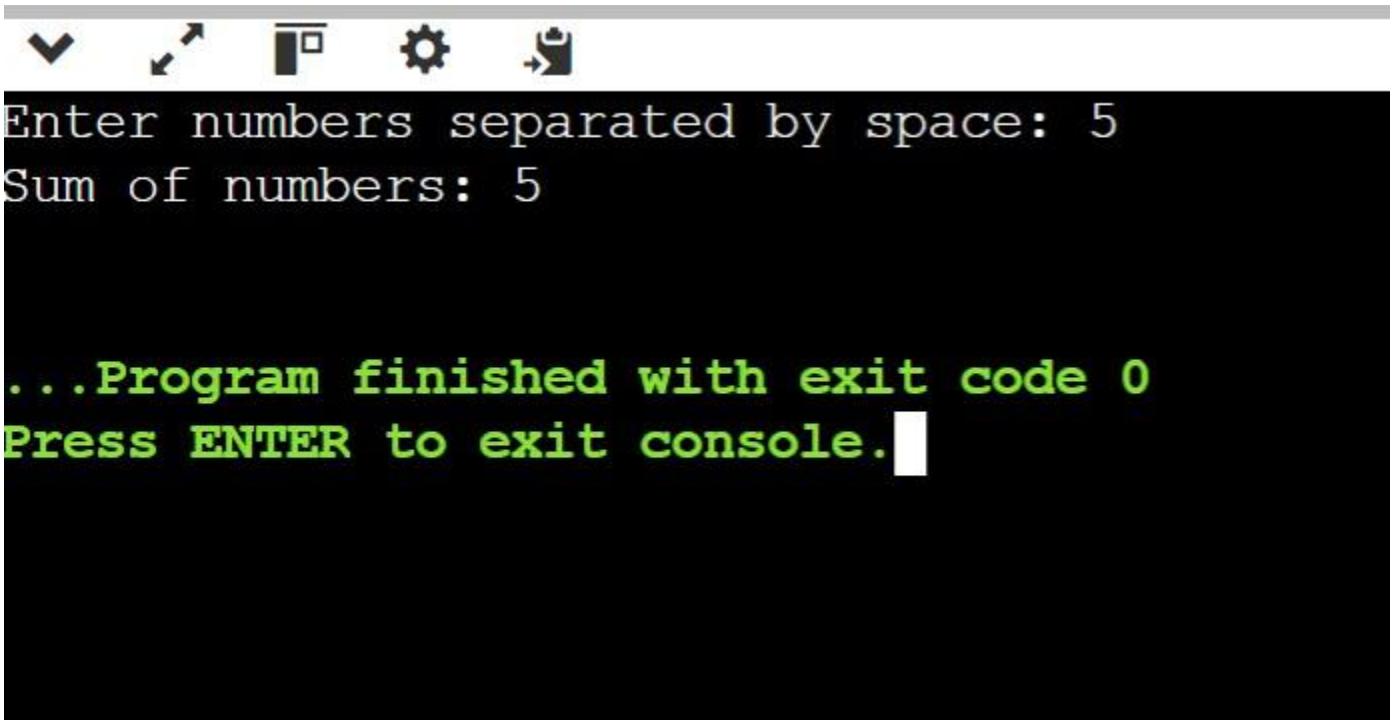
1. **Aim :** Develop Java programs using autoboxing, serialization, file handling, and efficient data processing and management.

2. **Easy Level:**
   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()).

3. **Implementation/Code:**

```
import java.util.*;

public class AutoboxingExample {
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (int num : numbers) {  // Unboxing happens here
            sum += num;
        }
        return sum;
    }

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

        System.out.print("Enter numbers separated by space: ");
        String[] inputs = scanner.nextLine().split(" ");

        for (String input : inputs) {
            numbers.add(Integer.parseInt(input)); // Autoboxing from int to Integer
        }

        System.out.println("Sum of numbers: " + calculateSum(numbers));
        scanner.close();
    }
}
```

4. OUTPUT:



Output for Easy Level:

Medium Level:

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.

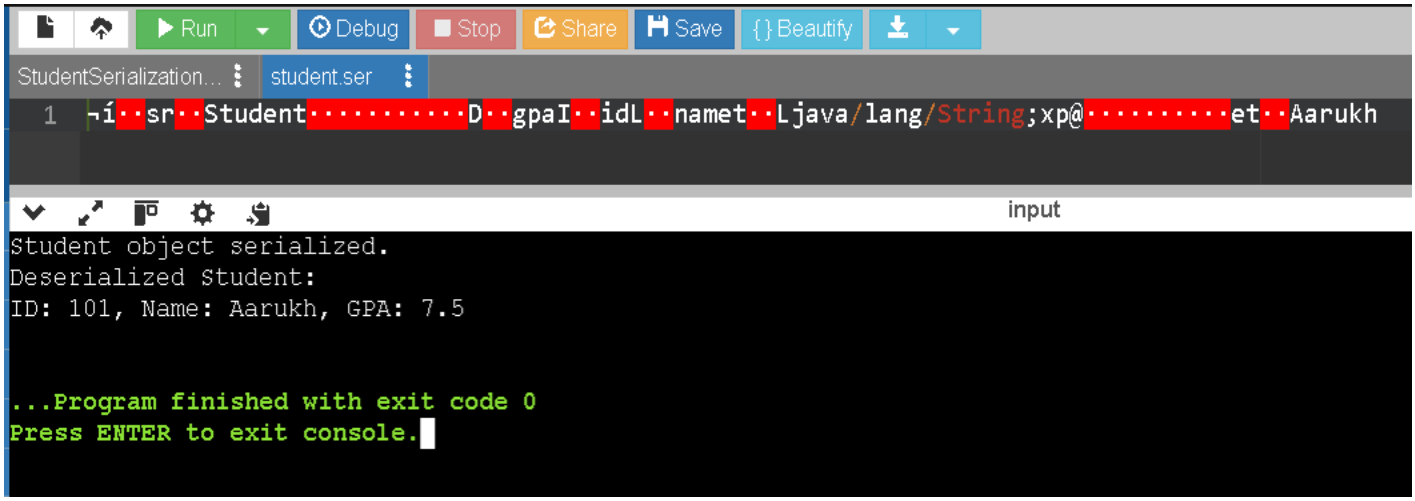
Handle FileNotFoundException, IOException, and ClassNotFoundException using exception handling.

5. Code:

```
import java.io.*;

// Serializable class
class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    int id;
    String name;
    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("ID: " + id + ", Name: " + name + ", GPA: " + gpa);
    }
}

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

    // Serialize object
    public static void serializeStudent(Student student) {
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {
            out.writeObject(student);
            System.out.println("Student object serialized.");
        } catch (IOException e) {
            e.printStackTrace();
```

```
        }
    }

    // Deserialize object
    public static Student deserializeStudent() {
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
            return (Student) in.readObject();
        } catch (FileNotFoundException e) {
            System.out.println("File not found.");
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        return null;
    }

    public static void main(String[] args) {
        Student student = new Student(101, "Alice", 3.8);
        serializeStudent(student);

        Student deserializedStudent = deserializeStudent();
        if (deserializedStudent != null) {
            System.out.println("Deserialized Student:");
            deserializedStudent.display();
        }
    }
}
```

6. Output:



7. Hard Level:

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.

8. Code:-

```
import java.io.*;
import java.util.*;

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

    public Employee(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 EmployeeManagement {
    private static final String FILE_NAME = "employees.dat";

    // Add employee to file
    public static void addEmployee(Employee emp) {
        List<Employee> employees = loadEmployees();
        employees.add(emp);
        saveEmployees(employees);
        System.out.println("Employee added successfully.");
    }

    // Display all employees
    public static void displayEmployees() {
        List<Employee> employees = loadEmployees();
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
            return;
        }
        for (Employee emp : employees) {
            System.out.println(emp);
```

```
    }
}

// Save employee list to file
private static void saveEmployees(List<Employee> employees) {
    try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {
        out.writeObject(employees);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// Load employee list from file
@SuppressWarnings("unchecked")
private static List<Employee> loadEmployees() {
    try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        return (List<Employee>) in.readObject();
    } catch (FileNotFoundException e) {
        return new ArrayList<>(); // Return empty list if file does not exist
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
        return new ArrayList<>();
    }
}

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

    while (true) {
        System.out.println("\n1. Add Employee\n2. Display All Employees\n3. Exit");
        System.out.print("Enter choice: ");
```

```
        choice = scanner.nextInt();
        scanner.nextLine();  // Consume newline

        switch (choice) {
            case 1:
                System.out.print("Enter Employee ID: ");
                int id = scanner.nextInt();
                scanner.nextLine();  // Consume newline
                System.out.print("Enter Name: ");
                String name = scanner.nextLine();
                System.out.print("Enter Designation: ");
                String designation = scanner.nextLine();
                System.out.print("Enter Salary: ");
                double salary = scanner.nextDouble();

                addEmployee(new Employee(id, name, designation, salary));
                break;

            case 2:
                displayEmployees();
                break;

            case 3:
                System.out.println("Exiting program.");
                scanner.close();
                return;

            default:
                System.out.println("Invalid choice. Try again.");
        }
    }
}
```

}

9. Output:-

```
1. Add Employee
2. Display All Employees
3. Exit
Enter choice: 1
Enter Employee ID: 101
Enter Name: Aarukh khan
Enter Designation: khan
Enter Salary: 50000
Employee added successfully.

1. Add Employee
2. Display All Employees
3. Exit
Enter choice: 2
ID: 101, Name: Aarukh khan, Designation: khan, Salary: 50000.0

1. Add Employee
2. Display All Employees
3. Exit
Enter choice:
```

10. Learning outcomes:

1. **Collections in Java:** Learn ArrayList, HashMap, and Collection interfaces for efficient data storage and retrieval.

2. **CRUD Operations:** Implement basic operations like Add, Update, Remove, and Search using Java collections.

3. **Multithreading & Synchronization:** Use synchronized and ReentrantLock to handle concurrent access and prevent race conditions.
4. **Thread Priorities:** Assign priorities (MAX_PRIORITY, NORM_PRIORITY) to ensure important tasks (e.g., VIP bookings) execute first.