Experiment 5.1

Student Name: Akshat Srivastava **UID:** 22BCS11740
Branch: BE CSE **Section/Group:** 22BCS_IOT_618_A
Semester: 6th **DoP:** 21/02/2025
Subject Name: PBLJ Lab **Subject Code:** 22CSH-359

1. **Aim:** To develop a Java program that demonstrates autoboxing, unboxing, and parsing of strings into integers using `Integer.parseInt()` to calculate the sum of a list of integers.

2. **Objective:**

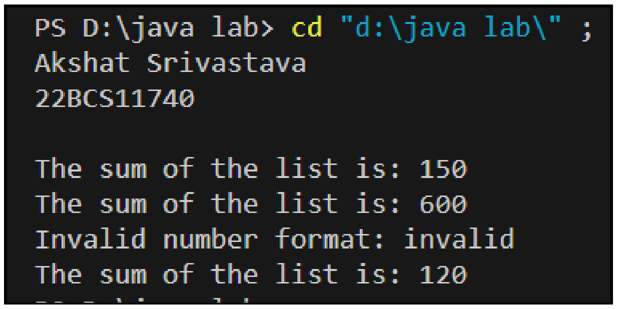
- Implement autoboxing to add integers to a list.
- Use unboxing to retrieve integer values from the list for sum calculation.
- Handle string parsing using `Integer.parseInt()` with exception handling.
- Ensure robustness by skipping invalid numbers during parsing.

3. Implementation/Code:

```
import java.util.*;

public class IntegerSumCalculator {

  public static Integer parseStringToInteger(String str) {

    try {

      return Integer.parseInt(str);

    } catch (NumberFormatException e) {

      System.out.println("Invalid number format: " + str);

      return null;

    }

  }
```

```
public static int calculateSum(List<Integer> numbers) {
    return numbers.stream().mapToInt(Integer::intValue).sum();
}
public static void main(String[] args) {
    List<String> inputs = Arrays.asList("10", "20", "30", "40", "50");
    List<Integer> numbers = new ArrayList<>();
    for (String input : inputs) {
        Integer num = parseStringToInteger(input);
        if (num != null) numbers.add(num);
    }
    System.out.println("The sum of the list is: " + calculateSum(numbers));
    inputs = Arrays.asList("100", "200", "300");
    numbers.clear();
    for (String input : inputs) {
        Integer num = parseStringToInteger(input);
        if (num != null) numbers.add(num);
    }
    System.out.println("The sum of the list is: " + calculateSum(numbers));
    inputs = Arrays.asList("50", "invalid", "70");
    numbers.clear();
    for (String input : inputs) {
        Integer num = parseStringToInteger(input);
        if (num != null) numbers.add(num);
        }
        System.out.println("The sum of the list is: " + calculateSum(numbers));
    }
}
```

4. Output



```
PS D:\java lab> cd "d:\java lab\" ;
Akshat Srivastava
22BCS11740

The sum of the list is: 150
The sum of the list is: 600
Invalid number format: invalid
The sum of the list is: 120
```

5. Learning Outcome:

- Understand and apply autoboxing and unboxing in Java.
- Effectively use wrapper classes and exception handling.
- Parse strings into primitive data types using wrapper class methods.
- Use loops and Java Streams to process collections and calculate sums.

Experiment 5.2

Student Name: Akshat Srivastava **UID:** 22BCS11740
Branch: BE CSE **Section/Group:** 22BCS_IOT_618_A
Semester: 6th **DoP:** 21/02/2025
Subject Name: PBLJ Lab **Subject Code:** 22CSH-359

1. **Aim:** To implement a Java program that serializes and deserializes a `Student` object using `ObjectOutputStream` and `ObjectInputStream` while handling exceptions like `FileNotFoundException`, `IOException`, and `ClassNotFoundException`.

2. **Objective:**

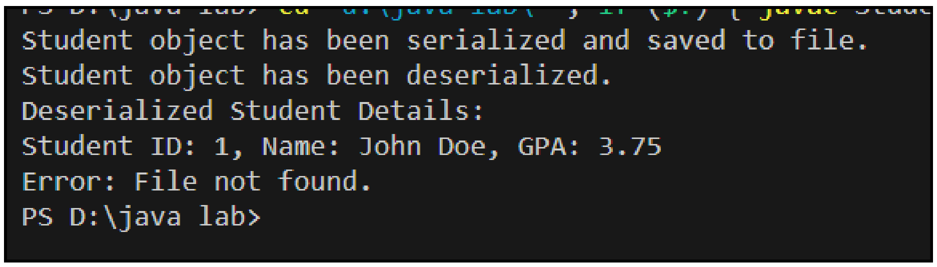
- Create a serializable `Student` class with id, name, and GPA.
- Serialize the object to a file named `student.ser`.
- Deserialize the object from the file and display its details.
- Handle exceptions during serialization and deserialization.

3. Implementation/Code:

```
import java.io.*;

class Student implements Serializable {

  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 displayStudent() {
        System.out.println("Student ID: " + id + ", Name: " + name + ", GPA: " + gpa);
    }
}
public class StudentSerialization {
    public static void serializeStudent(Student student, String filename) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filename)))
{
            oos.writeObject(student);
            System.out.println("Student object has been serialized and saved to file.");
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found.");
        } catch (IOException e) {
            System.out.println("Error: Unable to serialize object.");
        }
    }
    public static Student deserializeStudent(String filename) {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename))) {
            System.out.println("Student object has been deserialized.");
            return (Student) ois.readObject();
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found.");
        } catch (IOException e) {
            System.out.println("Error: Unable to deserialize object.");
        } catch (ClassNotFoundException e) {
            System.out.println("Error: Class not found.");
        }
        return null;
    }
    public static void main(String[] args) {
        String filename = "student.ser";
        Student student1 = new Student(1, "John Doe", 3.75);
        serializeStudent(student1, filename);
        Student deserializedStudent = deserializeStudent(filename);
        if (deserializedStudent != null) {
            System.out.println("Deserialized Student Details:");
            deserializedStudent.displayStudent();
        }
        deserializeStudent("nonexistent.ser");
    }
}
```

4. Output



```
Student object has been serialized and saved to file.
Student object has been deserialized.
Deserialized Student Details:
Student ID: 1, Name: John Doe, GPA: 3.75
Error: File not found.
PS D:\java lab>
```

5. Learning Outcome:

- Understand Java serialization using `Serializable` interface.
- Use `ObjectOutputStream` and `ObjectInputStream` for object I/O.
- Implement exception handling for file and class-related errors.
- Gain experience with file input/output operations in Java.

Experiment 5.3

Student Name: Akshat Srivastava **UID:** 22BCS11740
Branch: BE CSE **Section/Group:** 22BCS_IOT_618_A
Semester: 6th **DoP:** 21/02/2025
Subject Name: PBLJ Lab **Subject Code:** 22CSH-359

1. **Aim:** To create a menu-based Java application that allows adding employee details, displaying all employees, and exiting the application, with employee data stored and retrieved from a file using serialization and deserialization.

2. **Objective:**

- Design an `Employee` class with name, id, designation, and salary fields.
- Implement a menu with options to add employees, display all employees, and exit the program.
- Store employee data in a file using `ObjectOutputStream` in append mode.
- Retrieve and display employee data using `ObjectInputStream`.
- Handle exceptions related to file input and output operations.

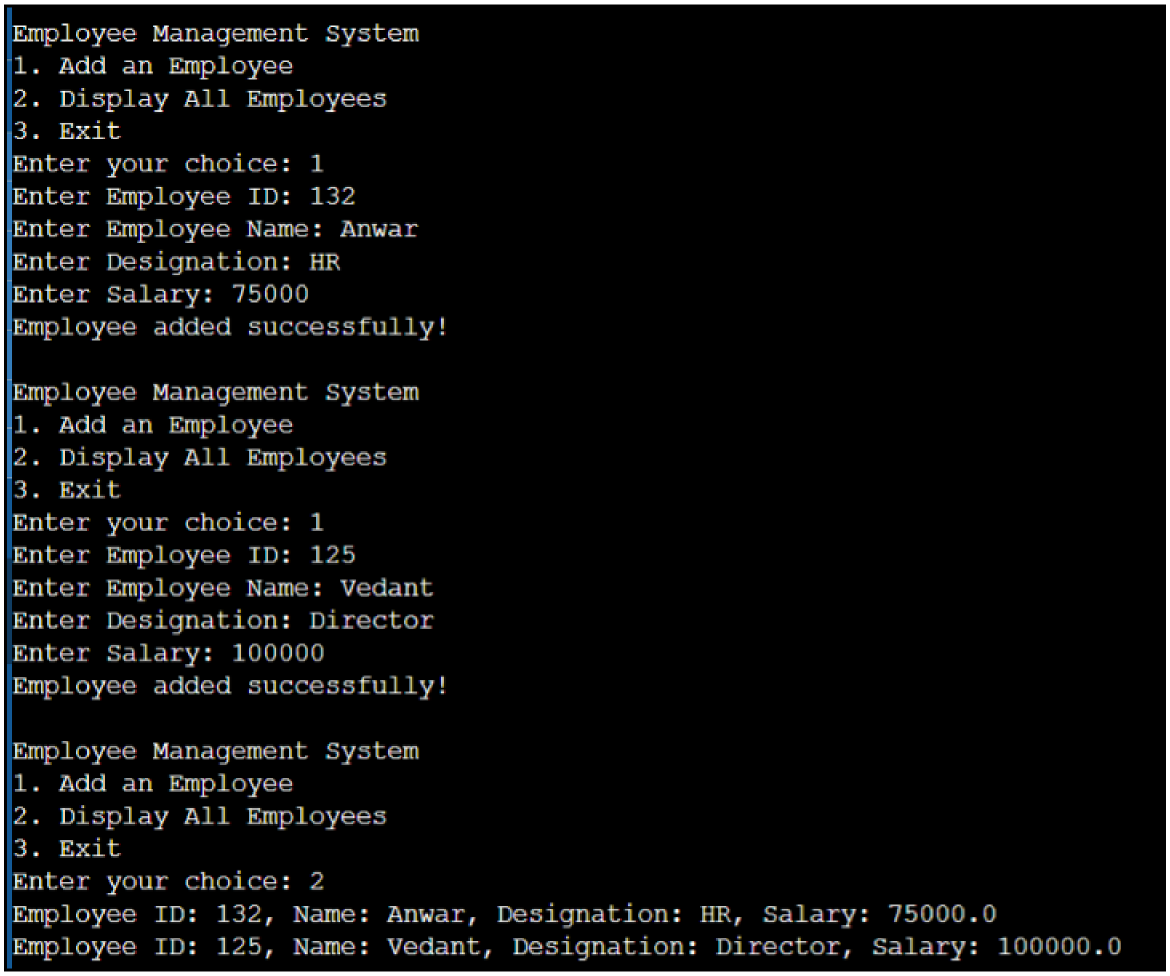
3. **Implementation/Code:**

```
import java.io.*;
import java.util.*;
class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private 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 "Employee ID: " + id + ", Name: " + name + ", Designation: " + designation
+ ", Salary: " + salary;
 }
}

public class EmployeeManagementSystem {
 private static final String FILE_NAME = "employees.ser";
 private static List<Employee> employees = new ArrayList<>();

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

  Employee employee = new Employee(id, name, designation, salary);
  employees.add(employee);
  saveEmployees();
  System.out.println("Employee added successfully!");
 }

 public static void displayAllEmployees() {
  loadEmployees();
  if (employees.isEmpty()) {
   System.out.println("No employees found.");
  } else {
   for (Employee employee : employees) {
    System.out.println(employee);
   }
  }
 }
 private static void saveEmployees() {
  try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
   oos.writeObject(employees);
  } catch (IOException e) {
   System.err.println("Error saving employees: " + e.getMessage());
  }
 }

 @SuppressWarnings("unchecked")
 private static void loadEmployees() {
  try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
   employees = (List<Employee>) ois.readObject();
  } catch (FileNotFoundException e) {
   employees = new ArrayList<>();
  } catch (IOException | ClassNotFoundException e) {
   System.err.println("Error loading employees: " + e.getMessage());
  }
 }

 public static void main(String[] args) {
  Scanner scanner = new Scanner(System.in);
  while (true) {
   System.out.println("\nEmployee Management System");
   System.out.println("1. Add an Employee");
   System.out.println("2. Display All Employees");
   System.out.println("3. Exit");
   System.out.print("Enter your choice: ");
   int choice = scanner.nextInt();
   scanner.nextLine();

   switch (choice) {
   case 1:
    addEmployee();
    break;
   case 2:
    displayAllEmployees();
    break;
   case 3:
    System.out.println("Exiting...");
```

```
            return;
          default:
            System.out.println("Invalid choice! Please try again.");
          }
        }
      }
    }
```

4. Output



```
Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 132
Enter Employee Name: Anwar
Enter Designation: HR
Enter Salary: 75000
Employee added successfully!

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 125
Enter Employee Name: Vedant
Enter Designation: Director
Enter Salary: 100000
Employee added successfully!

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 2
Employee ID: 132, Name: Anwar, Designation: HR, Salary: 75000.0
Employee ID: 125, Name: Vedant, Designation: Director, Salary: 100000.0
```

5. Learning Outcome:

• Understand file handling and serialization in Java to store and retrieve objects persistently.
• Learn how to implement a menu-driven console application using loops and conditional statements.
• Gain experience in object-oriented programming (OOP) by defining and managing Employee objects.