# Experiment 5.1

**Student Name: Sarthak Rana** **UID: 22BCS16222**
**Branch: CSE** **Section:642/B**
**Semester: 6th** **DOP:24/02/25**
**Subject: Java** **Subject Code: 22CSH-359**

**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()).

**Objective:** The goal of this experiment is to demonstrate autoboxing and unboxing in Java while implementing a simple program that reads a list of integers from the user, handles invalid inputs, and calculates their sum.

## CODE:

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

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

        System.out.println("Enter numbers (type 'done' to finish):");
        while (scanner.hasNext()) {
            String input = scanner.next();
            if (input.equalsIgnoreCase("done")) {
                break;
            }
            try {
                numbers.add(parseInteger(input)); // Autoboxing happens here
            } catch (NumberFormatException e) {
                System.out.println("Invalid number format: " + input);
            }
        }

        scanner.close();

        // Calculate the sum
        int sum = calculateSum(numbers);

        // Display the result
        System.out.println("The sum of the numbers is: " + sum);
    }

    // Method to parse string into Integer
    public static Integer parseInteger(String str) {
        return Integer.parseInt(str); // Returns an Integer object (autoboxing)
```

```
    }

    // Method to calculate the sum
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            sum += num; // Unboxing happens here
        }
        return sum;
    }
```

**OUTPUT:**

```
Enter numbers (type 'done' to finish):
10 20 invalid 30 done
Invalid number format: invalid
The sum of the numbers is: 60


...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter numbers (type 'done' to finish):
100 200 300
done
The sum of the numbers is: 600


...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter numbers (type 'done' to finish):
10 20 30 40 50 done
The sum of the numbers is: 150


...Program finished with exit code 0
Press ENTER to exit console.
```

# Experiment 5.2

**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.

**Objective:** The objective of this experiment is to learn and practice object serialization and deserialization in Java while handling common file and class-related exceptions such as FileNotFoundException, IOException, and ClassNotFoundException.

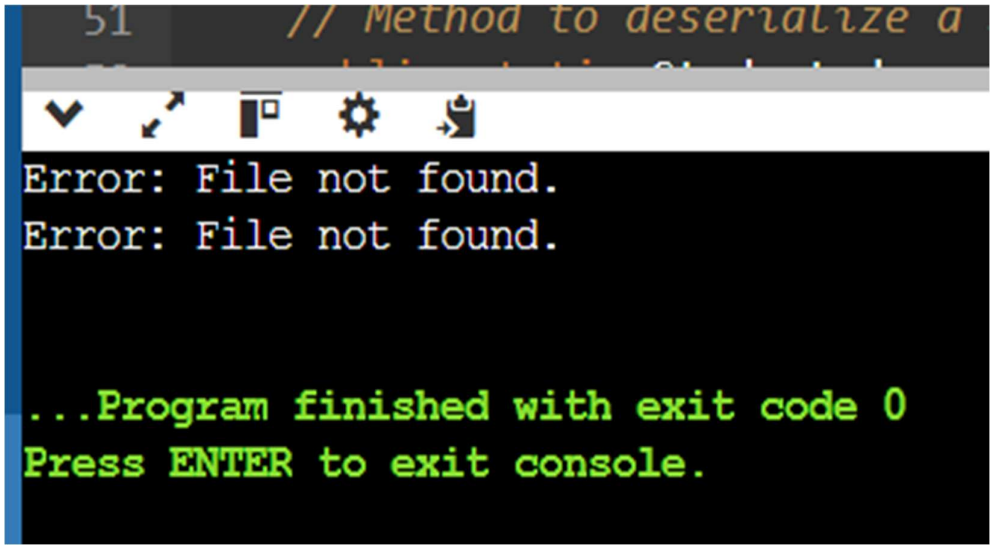
**CODE:**

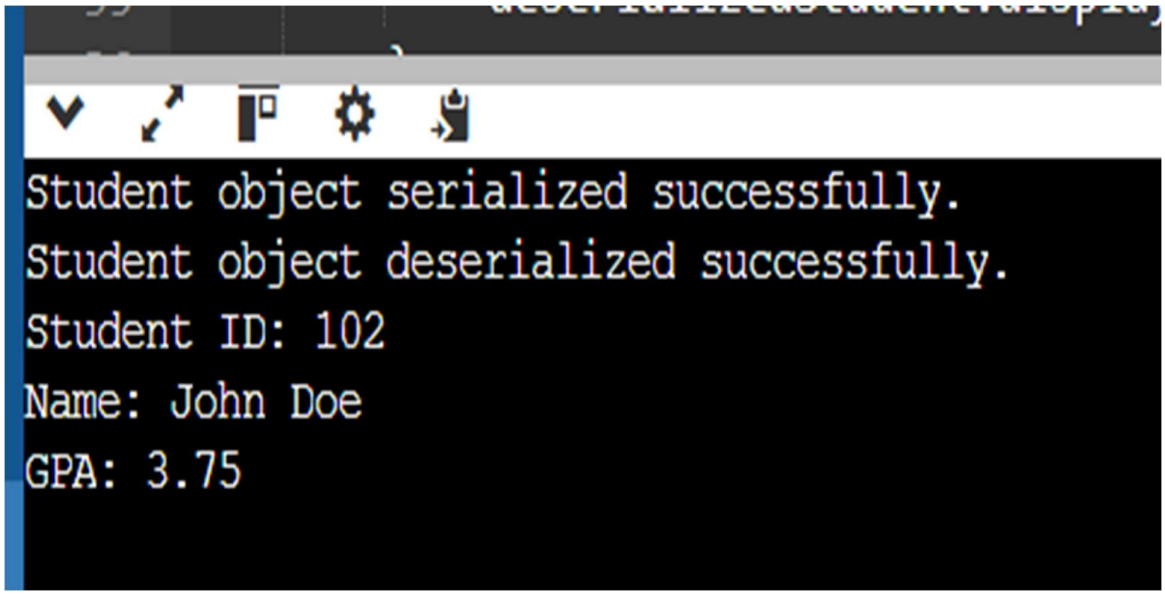
```
import java.io.*;

// Student class implementing Serializable
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;
    }

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

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

```
public static void main(String[] args) {
    Student student = new Student(102, "John Doe", 3.75);

    // Serialize the Student object
    serializeStudent(student);

    // Deserialize the Student object
    Student deserializedStudent = deserializeStudent();
    if (deserializedStudent != null) {
        deserializedStudent.displayStudent();
    }
}

// Method to serialize a Student object
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.");
    } catch (IOException e) {
        System.out.println("Error: IO Exception occurred while serializing.");
    }
}

// Method to deserialize a Student object
public static Student deserializeStudent() {
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        System.out.println("Student object deserialized successfully.");
        return (Student) ois.readObject();
    } catch (FileNotFoundException e) {
        System.out.println("Error: File not found.");
    } catch (IOException e) {
        System.out.println("Error: IO Exception occurred while deserializing.");
    } catch (ClassNotFoundException e) {
        System.out.println("Error: Class not found.");
```

```
        }
        return null;
    }
}
```

**OUTPUT:**



# Experiment 5.3

**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.

**Objective:** The objective of this experiment is to develop a Java application that allows users to add and display employee details. The application should store the employee information, such as name, ID, designation, and salary, in a file for persistence. It should also provide a menu-based system for user interaction, enabling users to either add new employees, view existing employee records, or exit the application. The program should handle file operations effectively and ensure proper exception handling to deal with potential errors, such as input and file-related issues. The experiment aims to help users understand file handling, exception management, and the creation of interactive systems in Java.

## CODE:

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

class Employee {
    private String name;
    private int id;
    private String designation;
    private double salary;

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

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

```
    public String toFileFormat() {
        return id + "," + name + "," + designation + "," + salary;
    }
}

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

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Add an Employee");
            System.out.println("2. Display All Employees");
            System.out.println("3. Exit");
            System.out.print("Please select an option: ");
            int option = scanner.nextInt();
            scanner.nextLine();  // Consume newline character

            switch (option) {
                case 1:
                    addEmployee(scanner);
                    break;
                case 2:
                    displayAllEmployees();
                    break;
                case 3:
                    System.out.println("Exiting the program...");
                    scanner.close();
                    System.exit(0);
                    break;
                default:
                    System.out.println("Invalid option. Please try again.");
            }
        }
    }
```

```
    // Method to add an employee
    public static void addEmployee(Scanner scanner) {
        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();  // Consume newline character
        System.out.print("Enter Designation: ");
        String designation = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();

        Employee employee = new Employee(name, id, designation, salary);

        // Write the employee details to the file
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_NAME, true))) {
            writer.write(employee.toFileFormat());
            writer.newLine();
            System.out.println("Employee added successfully!");
        } catch (IOException e) {
            System.out.println("Error while saving the employee details.");
        }
    }

    // Method to display all employees
    public static void displayAllEmployees() {
        try (BufferedReader reader = new BufferedReader(new FileReader(FILE_NAME))) {
            String line;
            System.out.println("\nEmployee Details:");
            while ((line = reader.readLine()) != null) {
                String[] details = line.split(",");
                System.out.println("Employee ID: " + details[0] + ", Name: " + details[1] + ", Designation: " +
details[2] + ", Salary: " + details[3]);
            }
        } catch (IOException e) {
            System.out.println("Error while reading the employee details.");
        }
```

```
    }
}
```

**OUTPUT:**

```
Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Please select an option: 1
Enter Employee Name: Sarthak Rana
Enter Employee ID: 101
Enter Designation: software engineer
Enter Salary: 100000
Employee added successfully.
```

```
Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Please select an option: 2

Employee Details:
Employee ID: 101, Name: Sarthak Rana , Designation: Software Engineer, Salary: 100000.0
Employee ID: 101, Name: Sarthak Rana , Designation: Software Engineer, Salary: 100000.0
Employee ID: 102, Name: Rana , Designation: Tester, Salary: 150000.0
```