# Experiment 5

**Student Name:** Garv Kumar **UID:** 22BET10103
**Branch:** BE-IT **Section/Group:** 22BET_IOT-702/A
**Semester:** 6th **Date of Performance:** 05/03/2025
**Subject Name:** PBLJ Lab **Subject Code:** 22ITH-359

## Problem: 1

### 1. Aim:

To implement a Java program that calculates the sum of a list of integers using autoboxing and unboxing.

### 2. Objective:

1. To understand the concept of autoboxing and unboxing in Java.
2. To convert a list of strings into integer values using Integer.parseInt().
3. To perform arithmetic operations using autoboxed values.
4. To enhance user interaction by accepting input dynamically.

### 3. Implementation/Code:

```
import java.util.*;

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

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

        for (String num : input) {
            numbers.add(Integer.parseInt(num));  // Autoboxing
        }
```

```
        int sum = 0;
        for (Integer num : numbers) {
            sum += num;  // Unboxing
        }

        System.out.println("Sum: " + sum);
        scanner.close();
    }
}
```

## 4. Output:

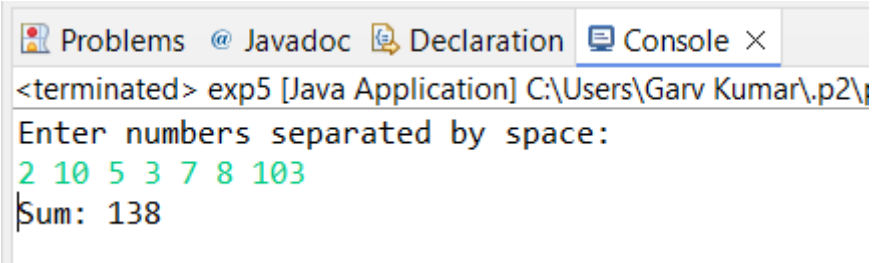


***Fig: Sum using Autoboxing and Unboxing.***

## 5. Learning Outcomes:

1. Gained knowledge of autoboxing and unboxing in Java.
2. Understood how to convert string input into integer values.
3. Learned to perform arithmetic operations using wrapper classes.
4. Improved handling of user inputs dynamically.
5. Developed skills in working with Java collections like ArrayList.

# Problem: 2

## 1. Aim:

To create a Java program that demonstrates object serialization and deserialization using the Serializable interface.

## 2. Objective:

1. To understand the process of serializing and deserializing objects in Java.
2. To implement the Serializable interface in a Java class.
3. To write and read objects from a file using ObjectOutputStream and ObjectInputStream.
4. To handle exceptions that may arise during serialization and deserialization.

## 3. Implementation/Code:

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

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    String name;
    int id;
    double grade;

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

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

public class StudentSerialization {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Student Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Student ID: ");
        int id = scanner.nextInt();
        System.out.print("Enter Grade: ");
        double grade = scanner.nextDouble();

        Student student = new Student(name, id, grade);
        String filename = "student.ser";

        // Serialize
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(filename))) {
            oos.writeObject(student);
            System.out.println("Student object serialized.");
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Deserialize
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(filename))) {
            Student deserializedStudent = (Student) ois.readObject();
            System.out.println("\nDeserialized Student:");
            deserializedStudent.display();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }

        scanner.close();
    }
}
```

## 4. Output:

```
Problems  @ Javadoc  Declaration  Console ×
<terminated> exp5 [Java Application] C:\Users\Garv Kumar\.p2\p
Enter Student Name: Garv
Enter Student ID: 103
Enter Grade: 89
Student object serialized.

Deserialized Student:
Student Name: Garv
Student ID: 103
Grade: 89.0
```

***Fig: Serialization and Deserialization of a Student Object.***

## 5. Learning Outcomes:

1. Understood the concept of object serialization and deserialization.
2. Implemented the Serializable interface in Java.
3. Learned how to write and read objects using file streams.
4. Explored exception handling related to file operations.
5. Developed an understanding of how objects can be stored persistently with user input.

# Problem: 3

## 1. Aim:

To develop a menu-based Java application for managing employee details using file handling and serialization.

## 2. Objective:

1. To create a menu-driven program with options for adding and displaying employee details.
2. To use serialization to store and retrieve employee records from a file.
3. To implement file handling techniques in Java.
4. To improve user interaction through an efficient menu system.

## 3. Implementation/Code:

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

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

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

    public void display() {
        System.out.println("\nEmployee ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Designation: " + designation);
        System.out.println("Salary: $" + salary);
    }
}
```

```
public class EmployeeManagement {
    private static final String FILE_NAME = "employees.dat";

    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("Choose an option: ");

            int choice;
            try {
                choice = scanner.nextInt();
            } catch (InputMismatchException e) {
                System.out.println("Invalid input! Please enter a number.");
                scanner.nextLine(); // Consume invalid input
                continue;
            }

            switch (choice) {
                case 1:
                    addEmployee(scanner);
                    break;
                case 2:
                    displayEmployees();
                    break;
                case 3:
                    System.out.println("Exiting...");
                    scanner.close();
                    System.exit(0);
                default:
                    System.out.println("Invalid choice! Please select again.");
            }
        }
    }

    private static void addEmployee(Scanner scanner) {
```

```
        System.out.print("Enter Employee Name: ");
        scanner.nextLine(); // Consume leftover newline
        String name = scanner.nextLine();

        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume leftover newline

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

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

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

        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME, true))) {
            oos.writeObject(emp);
            System.out.println("Employee added successfully.");
        } catch (IOException e) {
            System.out.println("Error saving employee data: " + e.getMessage());
        }
    }

    private static void displayEmployees() {
        File file = new File(FILE_NAME);
        if (!file.exists()) {
            System.out.println("No employee records found.");
            return;
        }

        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            while (true) {
                try {
                    Employee emp = (Employee) ois.readObject();
                    emp.display();
                } catch (EOFException e) {
                    break; // End of file reached
                }
```

```
            }
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error reading employee data: " + e.getMessage());
        }
    }
}
```

## 4. Output:

```
<terminated> EmployeeManagement [Java Application] C:\Users\Garv Kumar\

Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee Name: Garv
Enter Employee ID: 103
Enter Designation: Coder
Enter Salary: 20000
Employee added successfully.

Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 2

Employee ID: 103
Name: Garv
Designation: Coder
Salary: $20000.0

Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 3
Exiting...
```

***Fig: Menu-Based Employee Management Application.***

## 5. Learning Outcomes:

1. Learned to design a menu-based application with multiple options.
2. Implemented file handling techniques to store and retrieve employee records.
3. Understood serialization and deserialization in a real-world scenario.
4. Improved user interaction through structured input and output handling.
5. Gained experience in object-oriented programming with file handling.