



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 5

Student Name: Jorawar Singh

Branch: BE CSE

Semester: 06

Subject Name: Project Based Learning in Java

UID: 22BCS13118

Section/Group: EPAM-801(B)

Date of Performance: 23-2-25

Subject Code: 22CSH-359

1. Aim-

Easy: 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()).

Medium: 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.

Hard: 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.

2. Procedure-

Easy Level: Sum of Integers

1. Initialize an empty list.
2. Take user inputs until "end" is entered.
3. Convert each input to an integer (autoboxing) and add to the list.
4. Calculate the sum by unboxing each Integer.
5. Display the sum.

Medium Level: Serialization and Deserialization

1. Create a Student class implementing Serializable.
2. Serialize:
 - Create a Student object.
 - Save it to a file using ObjectOutputStream.
3. Deserialize:
 - Read the object from the file using ObjectInputStream.
 - Display the object data.

Hard Level: Employee Management

1. Display a menu:

- Add Employee
 - Display All Employees
 - Exit
2. For Add Employee:
 - Take input for ID, Name, Designation, and Salary.
 - Save it as an Employee object in a list.
 - Serialize the list to a file.
 3. For Display All Employees:
 - Deserialize the list from the file.
 - Display each employee's details.
 4. Exit the program on user choice.

3. Code- EASY:

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

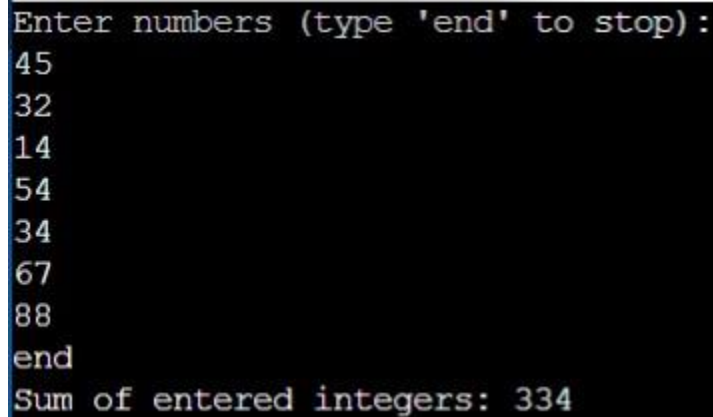
```
public class SumOfIntegers {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter numbers (type 'end' to stop):");

        while (true) {
            String input = sc.next();
            if (input.equalsIgnoreCase("end")) {
                break;
            }
            try {
                // Autoboxing: Converting primitive int to Integer
                numbers.add(Integer.parseInt(input));
            } catch (NumberFormatException e) {
                System.out.println("Invalid input, please enter an integer.");
            }
        }

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

```
        System.out.println("Sum of entered integers: " + sum);  
        sc.close();  
    }  
}
```



```
Enter numbers (type 'end' to stop):  
45  
32  
14  
54  
34  
67  
88  
end  
Sum of entered integers: 334
```

MEDIUM:

```
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;  
    }
```

```
    @Override  
    public String toString() {  
        return "ID: " + id + ", Name: " + name + ", GPA: " + gpa;  
    }  
}
```

```
public class StudentSerialization {
```

```
    // Serialize the Student Object
```

```
    public static void serializeStudent(Student student, String filename) {
```

```
try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(filename))) {
    out.writeObject(student);
    System.out.println("Student serialized successfully!");
} catch (FileNotFoundException e) {
    System.out.println("File not found: " + e.getMessage());
} catch (IOException e) {
    System.out.println("IO Exception: " + e.getMessage());
}
}

// Deserialize the Student Object
public static void deserializeStudent(String filename) {
    try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(filename))) {
        Student student = (Student) in.readObject();
        System.out.println("Deserialized Student: " + student);
    } catch (FileNotFoundException e) {
        System.out.println("File not found: " + e.getMessage());
    } catch (IOException e) {
        System.out.println("IO Exception: " + e.getMessage());
    } catch (ClassNotFoundException e) {
        System.out.println("Class not found: " + e.getMessage());
    }
}

public static void main(String[] args) {
    Student student = new Student(101, "Dipesh", 8.5);
    String filename = "student.ser";

    // Serialize
    serializeStudent(student, filename);

    // Deserialize
    deserializeStudent(filename);
}
```

Student serialized successfully!

Deserialized Student: ID: 101, Name: Dipesh, GPA: 8.5



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

HARD:

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

// Serializable Class
class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    int id;
    String name;
    String 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";

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

```
Employee employee = new Employee(id, name, designation, salary);
ArrayList<Employee> employees = readEmployees();
employees.add(employee);
writeEmployees(employees);

System.out.println("Employee added successfully!");
}

// Method to display all employees
public static void displayAllEmployees() {
    ArrayList<Employee> employees = readEmployees();
    if (employees.isEmpty()) {
        System.out.println("No employees found.");
    } else {
        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }
}

// Method to read employees from the file
public static ArrayList<Employee> readEmployees() {
    ArrayList<Employee> employees = new ArrayList<>();
    try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
        employees = (ArrayList<Employee>) in.readObject();
    } catch (FileNotFoundException e) {
        System.out.println("No existing records found.");
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Error reading employee data: " + e.getMessage());
    }
    return employees;
}

// Method to write employees to the file
public static void writeEmployees(ArrayList<Employee> employees) {
    try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
        out.writeObject(employees);
    } catch (IOException e) {
        System.out.println("Error writing employee data: " + e.getMessage());
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// Main Menu
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    while (true) {
        System.out.println("\n--- Employee Management System ---");
        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 = sc.nextInt();

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

```
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 101
Enter Employee Name: Garisha
Enter Designation: Software Manager
Enter Salary: 1000000
No existing records found.
Employee added successfully!

--- Employee Management System ---
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 105
Enter Employee Name: Yashika
Enter Designation: Director
Enter Salary: 1500000
Employee added successfully!

--- Employee Management System ---
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 2
ID: 101, Name: Garisha, Designation: Software Manager, Salary: 1000000.0
ID: 105, Name: Yashika, Designation: Director, Salary: 1500000.0

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

4. Learning Outcomes-

- **Autoboxing & Unboxing:** Efficiently convert between primitive types and their wrapper classes in Java.
- **Serialization & Deserialization:** Store and retrieve object states using file handling.
- **Object-Oriented Design:** Implement classes with attributes and methods, demonstrating encapsulation.
- **File I/O Operations:** Read from and write to files for persistent data storage.
- **Menu-Driven Programming:** Build interactive console applications with dynamic user input handling.