Experiment5.1

Name:Aastik **UID:22BCS14345**
Branch:CSE **Section:22BCS_KPIT-902/B**
Semester:6th **DOP:03/03/2025**
Subject:PBLJ **SubjectCode:22CSH-359**

---

1. **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()).

2. **Objective:** Demonstrate **autoboxing** and **unboxing** in Java by converting string numbers into Integer objects, storing them in a list, and computing their sum.

3. **Algorithm:**

Step1:InitializetheProgram

1. Starttheprogram.
2. ImportArrayListandListclasses.
3. DefinetheAutoboxingExampleclass.

Step2:ConvertStringArraytoIntegerList

1. DefinethemethodparseStringArrayToIntegers(String[]strings).
2. CreateanemptyArrayList<Integer>.
3. Iteratethroughthestringarray:
   - Converteach stringtoan IntegerusingInteger.parseInt(str).
   - Addtheintegertothelist (**autoboxing**happenshere).
4. Return the listofintegers.

Step3:CalculatetheSum ofIntegers

1. DefinethemethodcalculateSum(List<Integer>numbers).
2. Initializeavariablesumto0.
3. Iteratethroughthelist:
   - Extracteachinteger(**unboxing**happenshere).
   - Add itto sum.
4. Returnthetotal sum.

Step4:ExecuteMain Function

1. Definemain(String[]args).
2. Createastringarraywithnumericvalues.
3. CallparseStringArrayToIntegers()toconvertitintoalistofintegers.
4. CallcalculateSum()tocomputethesum.
5. Printtheresult.

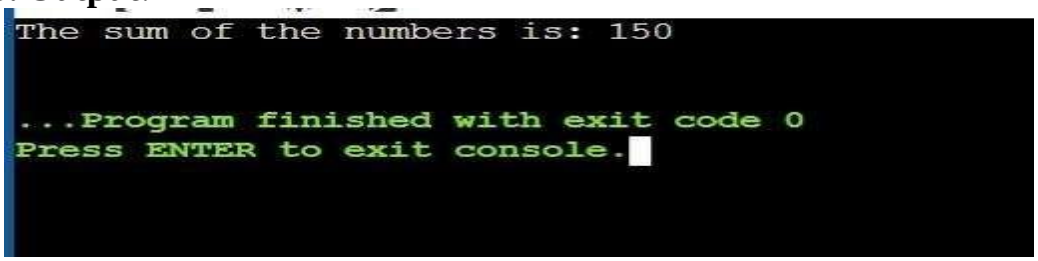
Step5:TerminatetheProgram

1. End theexecution.

4. Code:

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

publicclassAutoboxingExample
{publicstaticvoidmain(String[]args){
        String[] numberStrings = {"10", "20", "30", "40", "50"};

        List<Integer>numbers=parseStringArrayToIntegers(numberStrings);

        int sum = calculateSum(numbers);

        System.out.println("Thesumof thenumbersis:"+sum);
    }
    publicstaticList<Integer>parseStringArrayToIntegers(String[]strings)
        {List<Integer>integerList=newArrayList<>();
        for (String str : strings)
            { integerList.add(Integer.parseInt(str)
            );
        }
        returnintegerList;
    }
    publicstaticintcalculateSum(List<Integer>numbers)
        { intsum =0;
        for(Integernum:numbers)
            { sum+= num;
        }
        returnsum;
    }
}
```

5. Output:



6. LearningOutcomes:

- Understandtheconceptof**autoboxingandunboxing**inJavaandhowprimitivetypesare automatically converted to their wrapper classes and vice versa.
- Learnhowto**convertstringvaluesintoIntegerobjects**usingInteger.parseInt()andstore them in a list.
- Gainexperiencein**workingwithArrayLists**tostoreandmanipulateacollectionof numbers dynamically.
- Developproficiencyin**iteratingthroughcollections**andperformingarithmetic operations like summation.

Experiment5.2

1. Aim: CreateaJavaprogramtoserializeanddeserializeaStudent object. The program should:

- SerializeaStudentobject(containingid, name,andGPA)andsaveittoafile.
- Deserializetheobjectfrom thefileand displaythestudentdetails.
- Handle FileNotFoundException, IOException, and ClassNotFoundException using exception handling.

2. Objective: The objective is to serialize and deserialize a Student object, store and retrieveits id, name, and GPA from a file, and handle exceptions like FileNotFoundException, IOException, and ClassNotFoundException.

3. Algorithm:

Step1:InitializetheProgram

1. Starttheprogram.
2. Importthenecessaryclasses(java.io.*).
3. DefineaStudentclassimplementing Serializable.
4. Declareattributes:
    - id(int)
    - name(String)
    - gpa(double)
5. DefineaconstructortoinitializeStudentobjects.
6. OverridetoString()todisplaystudentdetails.

Step 2: Define the Serialization Method

1. CreateserializeStudent(Studentstudent).
2. Useatry-with-resourcesblocktocreateanObjectOutputStream:
    - OpenaFileOutputStreamtowritetostudent.ser.
    - WritetheStudentobjecttothefileusingwriteObject().
3. Handleexceptions:
    - FileNotFoundException→Printerrormessage.
    - IOException→Printerrormessage.
4. Printasuccessmessageifserializationissuccessful.

Step 3: Define the Deserialization Method

1. CreatedeserializeStudent().
2. Useatry-with-resourcesblocktocreateanObjectInputStream:
    - OpenaFileInputStreamtoreadstudent.ser.
    - ReadtheStudentobjectusingreadObject().
3. Handleexceptions:
    - FileNotFoundException→Printerrormessage.
    - IOException→Printerrormessage.
    - ClassNotFoundException→Printerrormessage.
4. Printthedeserializedstudentdetails.

Step 4: Execute Main Function

1. Definemain(String[]args).
2. CreateaStudentobjectwithsampledata.
3. CallserializeStudent()tosavetheobject.
4. CalldeserializeStudent()toreadanddisplaytheobject.

Step 5: Terminate the Program

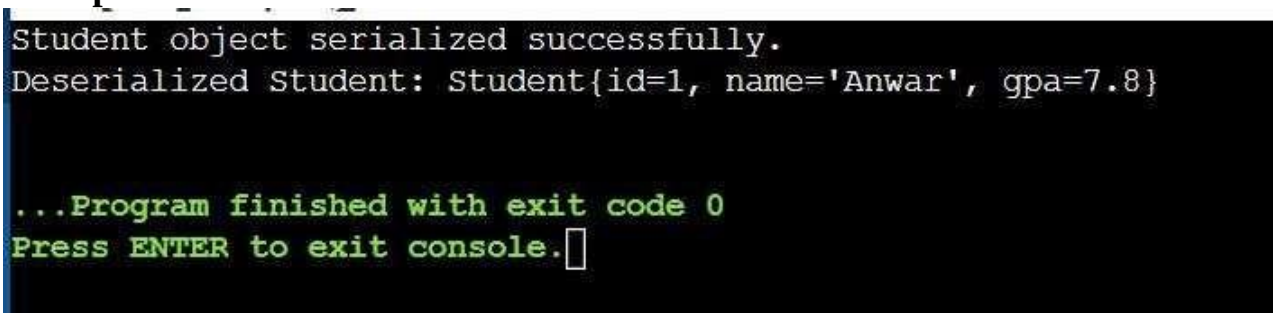
1. Endexecution.

4. ImplementationCode:

```
importjava.io.*;
classStudentimplementsSerializable{
   privatestaticfinallongserialVersionUID=1L;
   private int id;
   privateStringname;
   privatedoublegpa;
   publicStudent(intid,Stringname,doublegpa)
      { this.id = id;
      this.name=name;
      this.gpa = gpa;
   }
   @Override
   publicStringtoString(){
      return"Student{id="+id +", name='"+name+"', gpa=" +gpa +"}";
   }
}
publicclassStudentSerialization{
   privatestaticfinalStringFILE_NAME="student.ser";

   publicstaticvoidmain(String[]args){
      Studentstudent=newStudent(1,"Anwar",7.8);
      serializeStudent(student);
      deserializeStudent();
   }
   publicstaticvoidserializeStudent(Studentstudent){
      try(ObjectOutputStreamoos=newObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
         oos.writeObject(student);
         System.out.println("Studentobjectserializedsuccessfully.");
      }catch(FileNotFoundExceptione)
         {System.err.println("Filenotfound:"+e.getMessage());
      }catch(IOExceptione){
         System.err.println("IOExceptionoccurred:"+e.getMessage());
      }
   }
   publicstaticvoiddeserializeStudent(){
      try(ObjectInputStreamois=newObjectInputStream(newFileInputStream(FILE_NAME)))
{
         Student student = (Student) ois.readObject();
         System.out.println("DeserializedStudent:"+student);
      }catch(FileNotFoundExceptione)
         {System.err.println("Filenotfound:"+e.getMessage());
      }catch(IOExceptione){
         System.err.println("IOExceptionoccurred:"+e.getMessage());
      }catch(ClassNotFoundExceptione){
```

```
            System.err.println("Classnotfound:"+e.getMessage());
        }
    }
}
```

5. Output



6. LearningOutcomes:

- Understandobjectserializationanddeserialization inJava.
- LearnhowtouseObjectOutputStreamandObjectInputStreamforfileoperations.
- Implement exception handling for FileNotFoundException, IOException, and ClassNotFoundException.
- Gainhands-onexperienceinstoringandretrievingobjectsfromafile.
- Developskillsindatapersistenceand filemanagement using Java.

Experiment5.3

1. **Aim:** Createamenu-basedJavaapplicationwiththefollowing options.
   1. AddanEmployee
   2. DisplayAll
   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. **Objective:** The objective is to develop a menu-based Java application that allows users to **add employee details**, **store them in a file**, and **display all stored employee records**, with an option to exit the program.

3. Algorithm:

Step1:InitializetheProgram

1. Starttheprogram.
2. Importjava.util.*andjava.util.concurrent.*forthreadhandling.
3. DefineaclassTicketBookingSystemwith:
   - AList<Boolean>representingseatavailability(trueforavailable,falsefor booked).
   - Asynchronized methodbookSeat(intseatNumber,StringpassengerName)to ensure thread safety.

Step2:ImplementSeatBookingLogic

1. DefinebookSeat(intseatNumber,StringpassengerName):
   - Iftheseatisavailable(true),markitasbooked(false).
   - Printconfirmation:"SeatXbookedsuccessfullyby Y".
   - Ifalreadybooked,print:"SeatXisalready booked."

Step3:DefineBookingThreads

1. CreateaclassPassengerThreadextendingThread:
   - Storepassengername,seatnumber,andbookingsystemreference.
   - Implementrun()methodtocallbookSeat().

Step4:AssignThreadPriorities

1. CreateVIPandRegularpassengerthreads.
2. SethigherpriorityforVIP passengersusingsetPriority(Thread.MAX_PRIORITY).
3. Setdefaultpriorityforregularpassengers.

Step5:HandleUserInput&SimulateBooking

1. Inmain(),createaninstanceof TicketBookingSystem.
2. Acceptnumberofseatsandbookingsfrom theuser.
3. CreatemultiplePassengerThreadinstancesforVIPandregularpassengers.
4. Startall threadsusingstart().

Step6:Synchronization&PreventingDoubleBooking

1. Usethesynchronizedkeywordinbook Seat()toensureonlyonethreadaccessesitata time.
2. Ensurethreadexecutionorderbyassigninghigherpriority toVIP threads.

Step7:DisplayFinalBookingStatus

1. Afterallthreadsfinishexecution,displaythelistofbookedseats.
2. Endtheprogram witha message:"Allbookings completedsuccessfully."

4. ImplementationCode:

importjava.io.*;import

```
java.util.*;
classEmployeeimplementsSerializable{
    privatestaticfinallongserialVersionUID=1L;
    private int id;
    private String name;
    privateStringdesignation;
    private double salary;

    publicEmployee(intid,Stringname,Stringdesignation,doublesalary)
            { this.id = id;
            this.name=name;
            this.designation=designation;
            this.salary = salary;
    }
    @Override
    publicStringtoString(){
            return"EmployeeID:"+id+ ",Name:"+ name+ ",Designation:"+designation
+",Salary:"+salary;
    }
}
publicclassEmployeeManagementSystem{
    private static final String FILE_NAME = "employees.ser";
    privatestaticList<Employee>employees=newArrayList<>();
    public static void addEmployee() {
            Scannerscanner=newScanner(System.in);
            System.out.print("Enter Employee ID: ");
            int id = scanner.nextInt();
            scanner.nextLine();
            System.out.print("EnterEmployeeName:");
            String name = scanner.nextLine();
            System.out.print("Enter Designation: ");
            String designation = scanner.nextLine();
            System.out.print("Enter Salary: ");
            doublesalary=scanner.nextDouble();
            Employeeemployee=newEmployee(id,name,designation,salary);
            employees.add(employee);
            saveEmployees();
            System.out.println("Employeeaddedsuccessfully!");
    }
    publicstaticvoiddisplayAllEmployees()
            { loadEmployees();
            if(employees.isEmpty()){
                    System.out.println("Noemployeesfound.");
            } else {
                    for(Employeeemployee:employees)
                            { System.out.println(employee);
                    }
            }
    }
```

```
    privatestaticvoidsaveEmployees(){
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            oos.writeObject(employees);
        }catch(IOExceptione){
            System.err.println("Errorsavingemployees:"+e.getMessage());
        }
    }

    @SuppressWarnings("unchecked")
    privatestaticvoidloadEmployees(){
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            employees=(List<Employee>)ois.readObject();
        }catch(FileNotFoundExceptione)
            {employees=new
            ArrayList<>();
        }catch(IOException|ClassNotFoundExceptione)
            {System.err.println("Errorloadingemployees:"+e.getMessage());
        }
    }

    publicstaticvoidmain(String[]args){
        Scannerscanner=newScanner(System.in);
        while (true) {
            System.out.println("\nEmployeeManagementSystem");
            System.out.println("1. Add an Employee");
            System.out.println("2. Display All Employees");
            System.out.println("3. Exit");
            System.out.print("Enteryourchoice:"); int
            choice = scanner.nextInt();
            scanner.nextLine();

            switch(choice)
            {case1:
                addEmployee();
                break;
            case2:
                displayAllEmployees();
                break;
            case3:
                System.out.println("Exiting...");
                return;
            default:
                System.out.println("Invalidchoice!Pleasetryagain.");
            }
        }
    }
}
```

5. 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
```

6. LearningOutcomes:

- UnderstandfilehandlingandserializationinJavatostoreandretrieveobjects persistently.
- Learnhowtoimplementamenu-drivenconsoleapplicationusingloopsandconditional statements.
- Gainexperienceinobject-orientedprogramming(OOP)bydefiningandmanaging Employee objects.
- Practiceexceptionhandlingtomanagefile-relatederrorslikeFileNotFoundExceptionand IOException.
- DevelopskillsinlistmanipulationanduserinputhandlingusingArrayListand Scanner.