DEPARTMENTOF

COMPUTERSCIENCE&ENGINEERING

Discover. Learn. Empower.

Experiment 5

Student Name: Vanshul kale **UID:**22BCS14133

Branch: BE-CSE **Section/Group:IOT**-641/**B**

Semester:6th **DateofPerformance:5**/02/202**5**

Subject Name: Project Based Learning in Java with Lab **SubjectCode**:22CSH-359

1. **Aim:.** Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

2. **Objective:** The objective of developing Java programs using core concepts like data structures, collections, and multithreading is to create efficient, scalable, and responsive applications. These concepts are essential for solving problems that require efficient data management, concurrency handling, and the ability to process large amounts of data concurrently.

3. **Implementation / Code:**

```
import java.io.*;

import java.util.*;

 class Employee implements Serializable {

   private String name;

   private transient double salary; // Transient field won't be serialized

  public Employee(String name, double salary) {

      this.name = name;

      this.salary = salary;

   }

  public String getName() {

      return name;

   }
```

```
    public double getSalary() {
        return salary;
    }
@Override
    public String toString() {
        return "Employee[name=" + name + ", salary=" + salary + "]";
    }
}
 public class CompleteJavaExample {
  public static void main(String[] args) {
        // Autoboxing Example
        System.out.println("Autoboxing Example:");
        ArrayList<Integer> list = new ArrayList<>();
        for (int i = 1; i <= 5; i++) {
            list.add(i); // Autoboxing occurs here
        }
    System.out.println("List with autoboxed integers: " + list);
    // Unboxing Example
        int sum = 0;
        for (Integer num : list) {
            sum += num; // Unboxing occurs here
        }
```

```
System.out.println("Sum of numbers: " + sum);
    System.out.println();

    // File Handling Example
    String filename = "employee_data.txt";
    System.out.println("File Handling Example:");

    // Writing to a file
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
        writer.write("Employee List\n");
        writer.write("---------------\n");
        writer.write("Name, Salary\n");
        writer.write("Alice, 75000.00\n");
        writer.write("Bob, 65000.50\n");
        writer.write("Charlie, 70000.25\n");
        System.out.println("Data written to file.");
    } catch (IOException e) {
        e.printStackTrace();
    }

    // Reading from a file
    try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
```

```
            String line;

            System.out.println("Reading data from file:");

            while ((line = reader.readLine()) != null) {

                System.out.println(line);

            }

        } catch (IOException e) {

            e.printStackTrace();

        }


        // Creating Employee objects

        System.out.println();

        System.out.println("Creating Employee objects for serialization:");

        Employee emp1 = new Employee("Alice", 75000.00);

        Employee emp2 = new Employee("Bob", 65000.50);

        Employee emp3 = new Employee("Charlie", 70000.25);


        // Serialize objects to file

        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("employees.ser"))) {

            oos.writeObject(emp1);

            oos.writeObject(emp2);

            oos.writeObject(emp3);

            System.out.println("Employee objects serialized to file.");
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Deserialize objects from file
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("employees.ser"))) {
            Employee deserializedEmp1 = (Employee) ois.readObject();
            Employee deserializedEmp2 = (Employee) ois.readObject();
            Employee deserializedEmp3 = (Employee) ois.readObject();

            System.out.println("Employee objects deserialized:");
            System.out.println(deserializedEmp1);
            System.out.println(deserializedEmp2);
            System.out.println(deserializedEmp3);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }

        // Efficient Data Processing Example
        System.out.println();
        System.out.println("Efficient Data Processing Example:");
```

```
        // Using HashMap for efficient data storage
        Map<Integer, Employee> employees = new HashMap<>();
        employees.put(1, emp1);
        employees.put(2, emp2);
        employees.put(3, emp3);

        // Efficient data processing: filtering employees with salary > 70000
        List<Employee> highEarners = new ArrayList<>();
        for (Employee emp : employees.values()) {
            if (emp.getSalary() > 70000) {
                highEarners.add(emp);
            }
        }

        // Sorting employees by salary in descending order
        highEarners.sort((e1, e2) -> Double.compare(e2.getSalary(), e1.getSalary()));

        // Print sorted high earners
        System.out.println("High earners sorted by salary:");
        for (Employee emp : highEarners) {
            System.out.println(emp);
        }
    }
```

}

4. **OUTPUT**:

```
Autoboxing Example:
List with autoboxed integers: [1, 2, 3, 4, 5]
Sum of numbers: 15

File Handling Example:
Data written to file.
Reading data from file:
Employee List
---------------
Name, Salary
Alice, 75000.00
Bob, 65000.50
Charlie, 70000.25

Creating Employee objects for serialization:
Employee objects serialized to file.
Employee objects deserialized:
Employee[name=Alice, salary=0.0]
Employee[name=Bob, salary=0.0]
Employee[name=Charlie, salary=0.0]

Efficient Data Processing Example:
High earners sorted by salary:
Employee[name=Alice, salary=75000.0]
Employee[name=Charlie, salary=70000.25]


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

5. **Learning Outcomes:**

1. Usage of File Handling
2. Usage of Autoboxing and Unboxing
3. Usage of Serialization and Transient Keyword
4. Usage of Object-Oriented Programming (OOP)