Creating a student handbook using Java serialization with ArrayList and Stream API is a great way to demonstrate object persistence, collections, and functional programming in Java. Here's a step-by-step example of how you can implement it:

**1. Define the Student Class**

This class will represent a student, and it will implement Serializable so it can be serialized into a file.

import java.io.Serializable;

public class Student implements Serializable {

private String name;

private int age;

private String major;

public Student(String name, int age, String major) {

this.name = name;

this.age = age;

this.major = major;

}

// Getters and setters

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getMajor() {

return major;

}

public void setMajor(String major) {

this.major = major;

}

@Override

public String toString() {

return "Student{name='" + name + "', age=" + age + ", major='" + major + "'}";

}

}

**2. Create the StudentHandbook Class**

The StudentHandbook class will use ArrayList to store student objects and provide methods to serialize and deserialize the list. We will also utilize the Stream API for operations such as filtering students by age or major.

import java.io.\*;

import java.util.\*;

import java.util.stream.Collectors;

public class StudentHandbook {

private ArrayList<Student> students;

public StudentHandbook() {

this.students = new ArrayList<>();

}

public void addStudent(Student student) {

students.add(student);

}

public void saveToFile(String filename) {

try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filename))) {

out.writeObject(students);

System.out.println("Student handbook saved to file.");

} catch (IOException e) {

e.printStackTrace();

}

}

public void loadFromFile(String filename) {

try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename))) {

students = (ArrayList<Student>) in.readObject();

System.out.println("Student handbook loaded from file.");

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

public void printStudents() {

students.forEach(System.out::println);

}

// Using Stream API to filter students by age

public List<Student> getStudentsByAge(int age) {

return students.stream()

.filter(student -> student.getAge() == age)

.collect(Collectors.toList());

}

// Using Stream API to filter students by major

public List<Student> getStudentsByMajor(String major) {

return students.stream()

.filter(student -> student.getMajor().equalsIgnoreCase(major))

.collect(Collectors.toList());

}

// Using Stream API to get average age

public double getAverageAge() {

return students.stream()

.mapToInt(Student::getAge)

.average()

.orElse(0);

}

}

**3. Demonstrate Usage in a Main Class**

Here is an example of how to use the StudentHandbook class to add, serialize, deserialize, and filter student data:

public class Main {

public static void main(String[] args) {

// Create the student handbook

StudentHandbook handbook = new StudentHandbook();

// Add students

handbook.addStudent(new Student("Alice", 22, "Computer Science"));

handbook.addStudent(new Student("Bob", 23, "Mathematics"));

handbook.addStudent(new Student("Charlie", 22, "Engineering"));

handbook.addStudent(new Student("Diana", 21, "Biology"));

// Print all students

System.out.println("All Students:");

handbook.printStudents();

// Serialize students to a file

handbook.saveToFile("students.dat");

// Create a new handbook and load from the file

StudentHandbook loadedHandbook = new StudentHandbook();

loadedHandbook.loadFromFile("students.dat");

// Print loaded students

System.out.println("\nLoaded Students:");

loadedHandbook.printStudents();

// Filter students by age using Stream API

List<Student> age22Students = loadedHandbook.getStudentsByAge(22);

System.out.println("\nStudents aged 22:");

age22Students.forEach(System.out::println);

// Filter students by major using Stream API

List<Student> csStudents = loadedHandbook.getStudentsByMajor("Computer Science");

System.out.println("\nStudents in Computer Science:");

csStudents.forEach(System.out::println);

// Get the average age of students

double averageAge = loadedHandbook.getAverageAge();

System.out.println("\nAverage age of students: " + averageAge);

}

}

**Explanation:**

1. **Student Class**: Represents each student with a name, age, and major. It implements Serializable so it can be written to and read from a file.
2. **StudentHandbook Class**:
   * Holds an ArrayList of students.
   * Provides methods to add, serialize, and deserialize students.
   * Uses Stream API for filtering and calculating student data.
3. **Main Class**: Demonstrates how to use the StudentHandbook to add students, serialize and deserialize data, and perform operations with Stream API.

**Running the Program**

When you run the program, it will:

1. Add some students to the handbook.
2. Serialize the list of students to a file named students.dat.
3. Deserialize the students from the file into a new StudentHandbook instance.
4. Print the list of students, filter them by age, major, and compute the average age using the Stream API.

**Output Example:**

All Students:

Student{name='Alice', age=22, major='Computer Science'}

Student{name='Bob', age=23, major='Mathematics'}

Student{name='Charlie', age=22, major='Engineering'}

Student{name='Diana', age=21, major='Biology'}

Student handbook saved to file.

Loaded Students:

Student{name='Alice', age=22, major='Computer Science'}

Student{name='Bob', age=23, major='Mathematics'}

Student{name='Charlie', age=22, major='Engineering'}

Student{name='Diana', age=21, major='Biology'}

Students aged 22:

Student{name='Alice', age=22, major='Computer Science'}

Student{name='Charlie', age=22, major='Engineering'}

Students in Computer Science:

Student{name='Alice', age=22, major='Computer Science'}

Average age of students: 22.0