**Student Management System**

The Student Grade Management System is a console-based application designed to efficiently manage student grades. This application allows users to perform various operations such as adding, viewing, updating, and deleting student records. Each student record includes a name, subject, and grade.

**Table of Content**

|  |  |
| --- | --- |
| Title | Page No |
| Introduction | 2 |
| Features | 3 |
| Objective | 4 |
| Code | 5-19 |
| Result | 19-21 |

**Introduction**

In educational institutions, maintaining and managing student grades is a critical task. The Student Grade Management System is a console-based application designed to streamline this process, providing a user-friendly interface for handling student grade data efficiently and accurately. This system caters to the needs of educators and administrators by offering a comprehensive solution for adding, viewing, updating, and deleting student records, ensuring that grade management is both effective and straightforward.

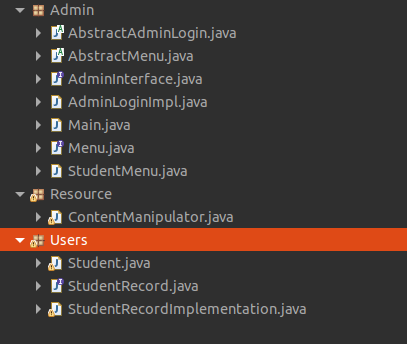
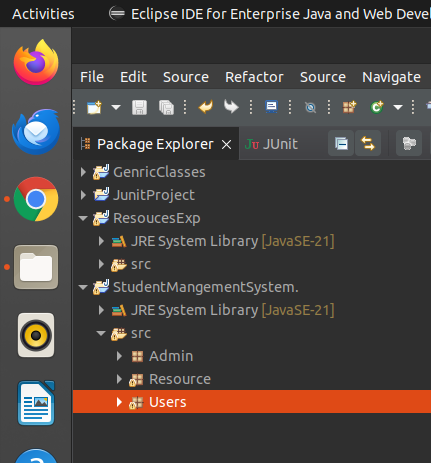
Feature

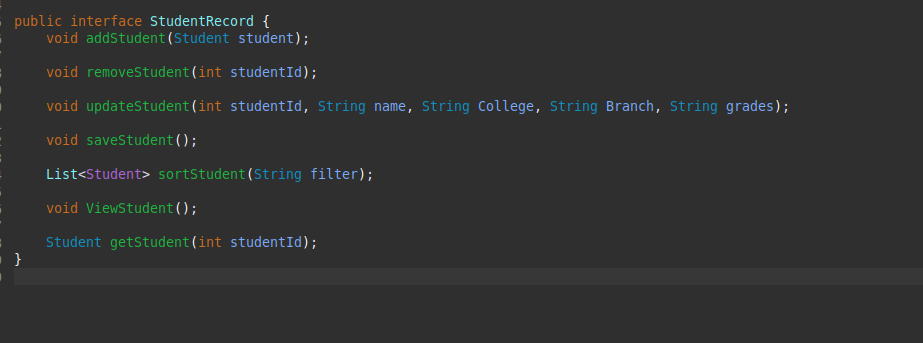
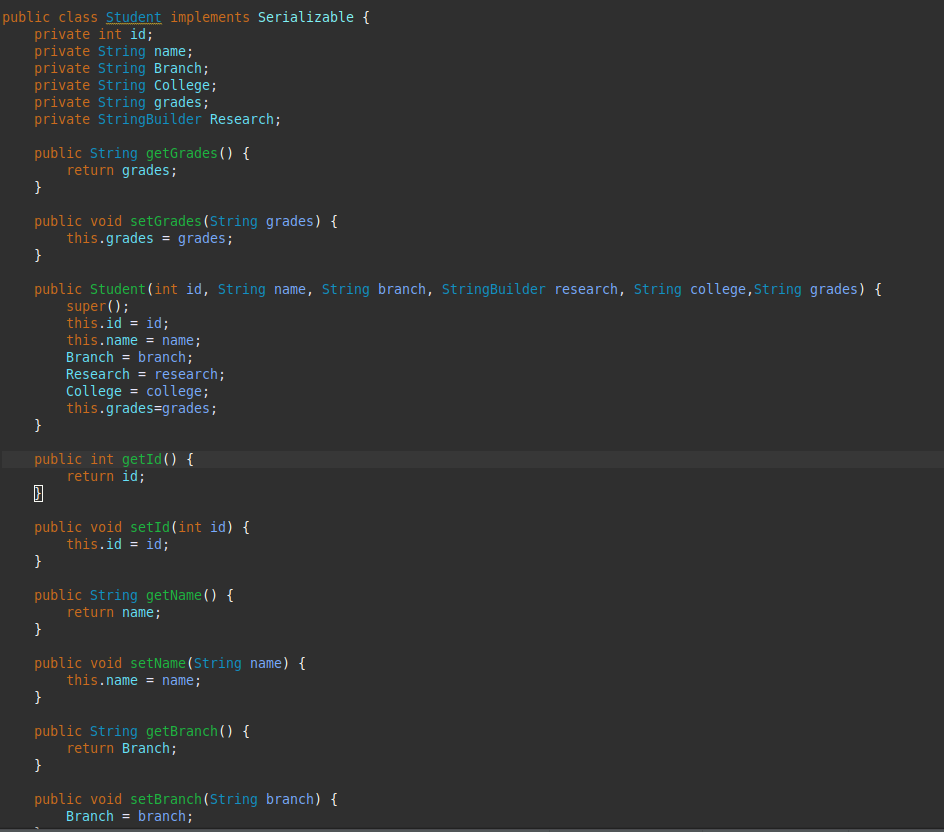
1. **Add Student**: The application allows users to effortlessly add new student records by entering their name, subject, and grade. This feature ensures that new entries can be created quickly and accurately.
2. **View Students**: Users can view a list of all students along with their respective subjects and grades. This feature provides a clear overview of all recorded data, making it easy to review and analyze student performance.
3. **Update Student**: Users can modify the grade of an existing student, ensuring that the records are up-to-date and reflect the most current information.
4. **Delete Student**: This feature enables users to remove a student from the system, ensuring that the database remains relevant and free of obsolete records.
5. **Persist Data**: The system saves student data to a file, allowing the information to be stored permanently. It can also load data from the file when the application is started, ensuring that no data is lost between sessions.
6. **Sort Students**: Using lambda expressions, the application can sort student records by grade. This feature allows users to quickly organize and review student performance in a meaningful order.
7. **Exception Handling**: The system includes robust error handling to manage common issues such as file not found, invalid input, and other runtime exceptions. This ensures the application runs smoothly and provides meaningful feedback when errors occur.

**Objective**

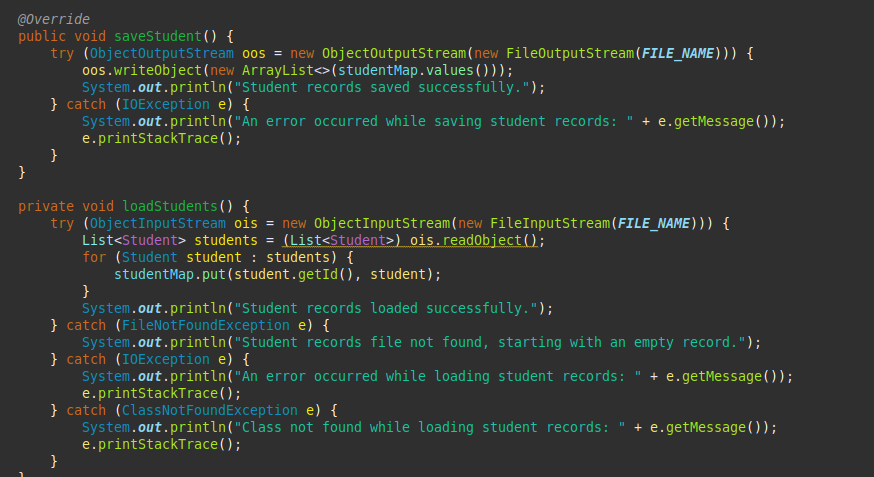
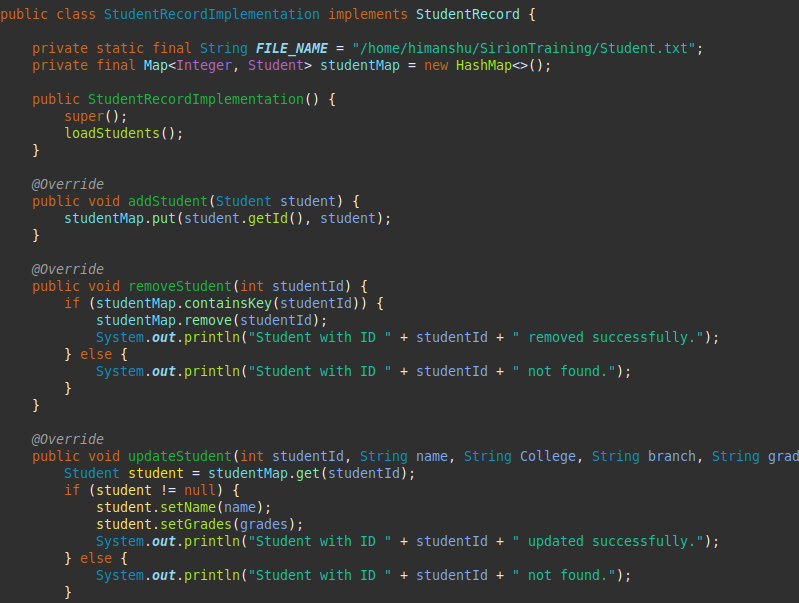
* **Efficiency**: Streamline the process of managing student grades with quick and easy data entry and retrieval.
* **Accuracy**: Ensure that all data is recorded accurately and remains up-to-date.
* **Data Persistence**: Guarantee that student data is securely stored and easily retrievable across different sessions.
* **User-Friendliness**: Provide a simple and intuitive interface that can be easily navigated by users with varying levels of technical expertise.
* **Robustness**: Implement comprehensive error handling to manage potential issues gracefully and maintain application stability.

Code

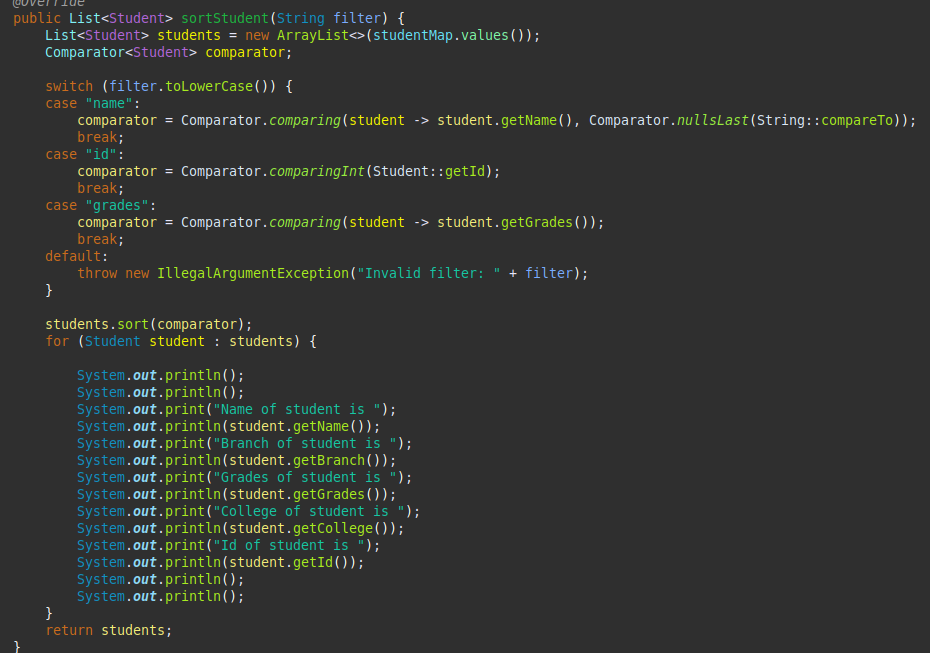


Student Code interface

StudentRecord Implementation



Some other Functionalities



Original code in text

Interface

package Users;

import java.util.\*;

public interface StudentRecord {

void addStudent(Student student);

void removeStudent(int studentId);

void updateStudent(int studentId, String name, String College, String Branch, String grades);

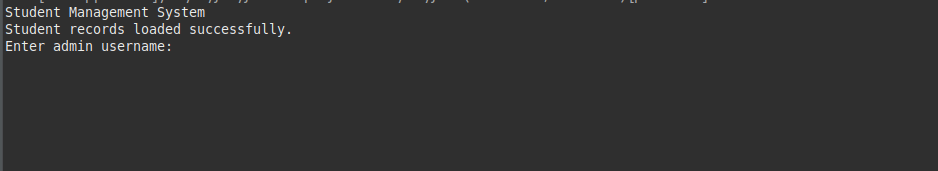
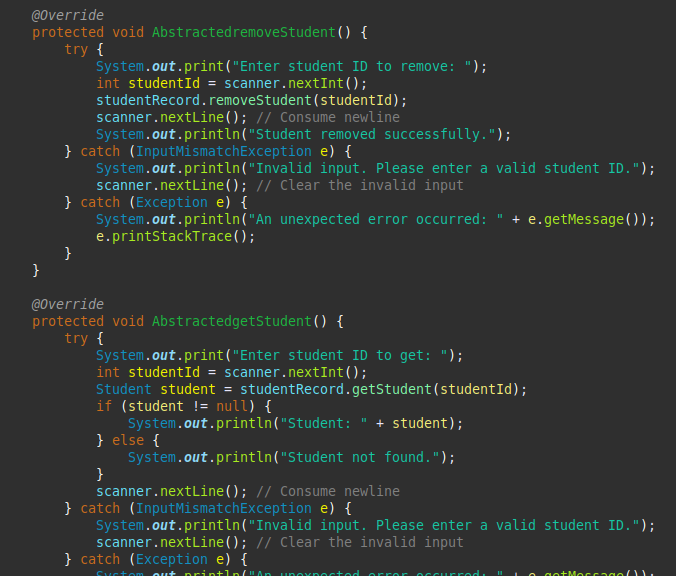
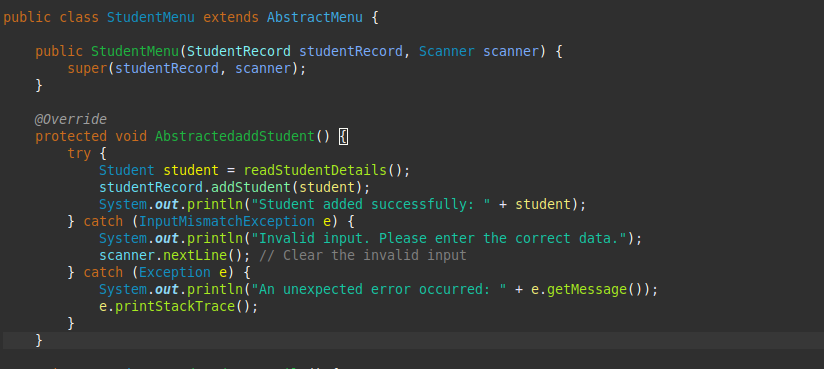
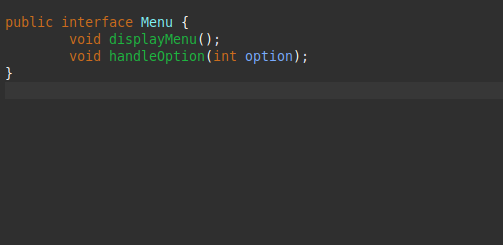
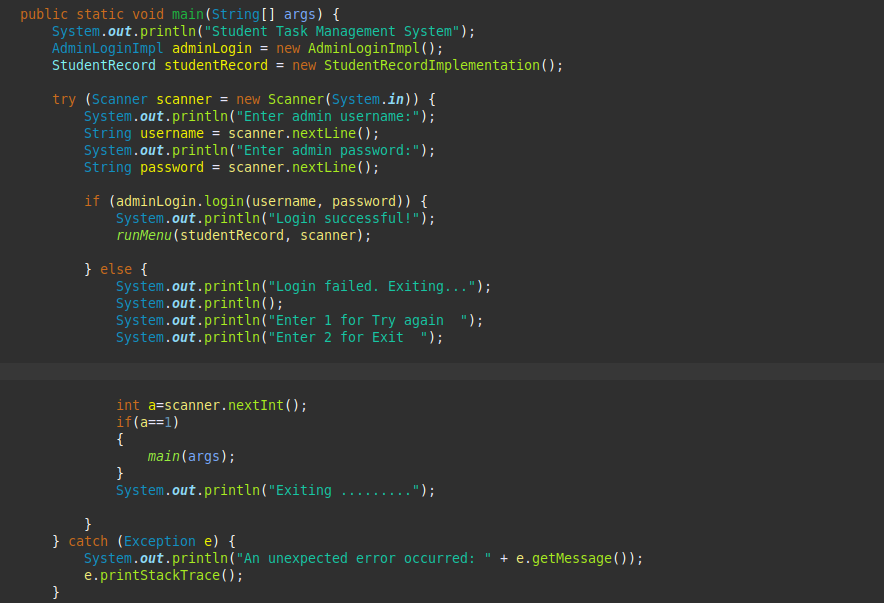
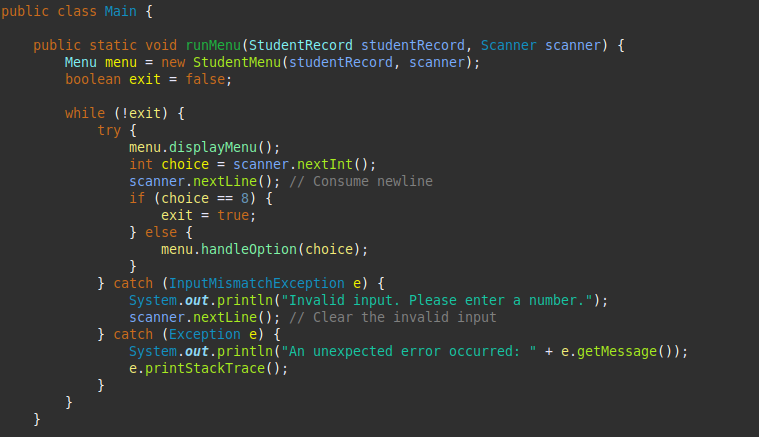
void saveStudent();

List<Student> sortStudent(String filter);

void ViewStudent();

Student getStudent(int studentId);

}



StudentRecord  
package Users;

import java.io.\*;

import java.util.\*;

public class StudentRecordImplementation implements StudentRecord {

private static final String FILE\_NAME = "/home/himanshu/SirionTraining/Student.txt";

private final Map<Integer, Student> studentMap = new HashMap<>();

public StudentRecordImplementation() {

super();

loadStudents();

}

@Override

public void addStudent(Student student) {

studentMap.put(student.getId(), student);

}

@Override

public void removeStudent(int studentId) {

if (studentMap.containsKey(studentId)) {

studentMap.remove(studentId);

System.out.println("Student with ID " + studentId + " removed successfully.");

} else {

System.out.println("Student with ID " + studentId + " not found.");

}

}

@Override

public void updateStudent(int studentId, String name, String College, String branch, String grades) {

Student student = studentMap.get(studentId);

if (student != null) {

student.setName(name);

student.setGrades(grades);

System.out.println("Student with ID " + studentId + " updated successfully.");

} else {

System.out.println("Student with ID " + studentId + " not found.");

}

}

@Override

public Student getStudent(int studentId) {

if (studentMap.containsKey(studentId)) {

Student student = studentMap.get(studentId);

System.out.println("Student with ID " + studentId + " fetched successfully.");

System.out.println();

System.out.println();

System.out.print("Name of student is ");

System.out.println(student.getName());

System.out.print("Branch of student is ");

System.out.println(student.getBranch());

System.out.print("Grades of student is ");

System.out.println(student.getGrades());

System.out.print("College of student is ");

System.out.println(student.getCollege());

System.out.print("Id of student is ");

System.out.println(student.getId());

System.out.println();

System.out.println();

return student;

} else {

System.out.println("Student with ID " + studentId + " not found.");

return null;

}

}

@Override

public void saveStudent() {

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE\_NAME))) {

oos.writeObject(new ArrayList<>(studentMap.values()));

System.out.println("Student records saved successfully.");

} catch (IOException e) {

System.out.println("An error occurred while saving student records: " + e.getMessage());

e.printStackTrace();

}

}

private void loadStudents() {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE\_NAME))) {

List<Student> students = (List<Student>) ois.readObject();

for (Student student : students) {

studentMap.put(student.getId(), student);

}

System.out.println("Student records loaded successfully.");

} catch (FileNotFoundException e) {

System.out.println("Student records file not found, starting with an empty record.");

} catch (IOException e) {

System.out.println("An error occurred while loading student records: " + e.getMessage());

e.printStackTrace();

} catch (ClassNotFoundException e) {

System.out.println("Class not found while loading student records: " + e.getMessage());

e.printStackTrace();

}

}

@Override

public List<Student> sortStudent(String filter) {

List<Student> students = new ArrayList<>(studentMap.values());

Comparator<Student> comparator;

switch (filter.toLowerCase()) {

case "name":

comparator = Comparator.comparing(student -> student.getName(), Comparator.nullsLast(String::compareTo));

break;

case "id":

comparator = Comparator.comparingInt(Student::getId);

break;

case "grades":

comparator = Comparator.comparing(student -> student.getGrades());

break;

default:

throw new IllegalArgumentException("Invalid filter: " + filter);

}

students.sort(comparator);

for (Student student : students) {

System.out.println();

System.out.println();

System.out.print("Name of student is ");

System.out.println(student.getName());

System.out.print("Branch of student is ");

System.out.println(student.getBranch());

System.out.print("Grades of student is ");

System.out.println(student.getGrades());

System.out.print("College of student is ");

System.out.println(student.getCollege());

System.out.print("Id of student is ");

System.out.println(student.getId());

System.out.println();

System.out.println();

}

return students;

}

@Override

public void ViewStudent() {

if (studentMap.isEmpty()) {

System.out.println("No student records available.");

} else {

for (Student student : studentMap.values()) {

System.out.println();

System.out.println();

System.out.print("Name of student is ");

System.out.println(student.getName());

System.out.print("Branch of student is ");

System.out.println(student.getBranch());

System.out.print("Grades of student is ");

System.out.println(student.getGrades());

System.out.print("College of student is ");

System.out.println(student.getCollege());

System.out.print("Id of student is ");

System.out.println(student.getId());

System.out.println();

System.out.println();

}

}

}

}

Result

