

Tirana Metropolitan University

Faculty Of Computer Science And IT

**Object Oriented Programming  
Project Title: Student Management System**

**Submitted by: Ester Xhavara Submitted to: Msc.Evis Plaku**

**Klesia Kaca Msc.Megi Tartari**

1. **Abstract:**

The Student Management System is a Java application that demonstrates an understanding of Object-Oriented Programming principles. Through the application of abstraction, encapsulation, inheritance, and polymorphism, the codebase is structured around the Person and Student classes, offering a hierarchical representation of individuals and students. Notably, the system includes file manipulation and exception handling, adeptly reading and writing student data to CSV files with a focus on real-world scenarios. It is designed to manage and organize student information efficiently.

A highlight is the implementation of generic collections, where maps efficiently manage student data, and constant lists enhance code readability. The system's user interaction component stands out with a console interface, incorporating a Scanner for input validation, ensuring a seamless and error-resistant experience. Overall, the Student Management System serves as a comprehensive showcase of acquired skills, blending sophisticated OOP concepts, effective file handling, and a user-friendly interface for managing student information.

1. **Key Functionalities:**

**Student Information Management:**

Create and store student profiles with essential details.

Manage course-related information, including grades and absences.

Calculate and display average grades and total absences for each student.

File Handling and Exception Handling:

Read and write student data to and from CSV files.

Handle exceptions gracefully during file operations to ensure program robustness.

**Generic Collections:**

Utilize generic collections, such as maps, to efficiently manage student data.

Demonstrate the use of lists to organize and display information.

**User Interaction:**

Implement a console-based user interface for ease of interaction.

Employ exception handling to ensure a smooth user experience.

1. **Methodology/Technical Parts:**

Object-Oriented Programming Paradigm:

**Abstraction and Encapsulation:**

The Student and Person classes demonstrate abstraction by encapsulating student and person details.

Accessor methods (getCourses, getAbsences, getGrades) provide controlled access to class attributes.

**Inheritance and Polymorphism:**

The Student class extends the Person class, showcasing inheritance.

Polymorphism is evident in the overridden methods calculateAverageGrade and displayDetails.

File Manipulation and Exception Handling:

**Reading and Writing CSV Files:**

The loadStudents and saveStudents methods in StudentManager illustrate file reading and writing.

Exception handling is applied to address potential errors during file operations.

Generic Collections:

**Maps for Student Data:**

Maps (absences and grades) are employed to efficiently organize and retrieve student information.

The use of COURSES as a constant list enhances code readability.

**Lists for Displaying Data:**

Lists are utilized to present student details, courses, and average grades in a user-friendly manner.

User Interaction and Console Interface:

**Interactive Console Menu:**

The main method provides a user-friendly console menu using a Scanner for input.

Input validation and exception handling ensure a smooth user experience.

1. **Summary:**

The Student Management System project effectively applies key OOP principles, file manipulation techniques, and generic collections. The code is well-structured, modular, and exhibits good coding practices.

By integrating these concepts, the application provides a robust and practical solution for managing student information. The emphasis on user interaction and exception handling enhances the overall usability and reliability of the system.

** User Input Validation**

**The code employs a Scanner to validate user input in the main menu, specifically expecting integer input with nextInt().**

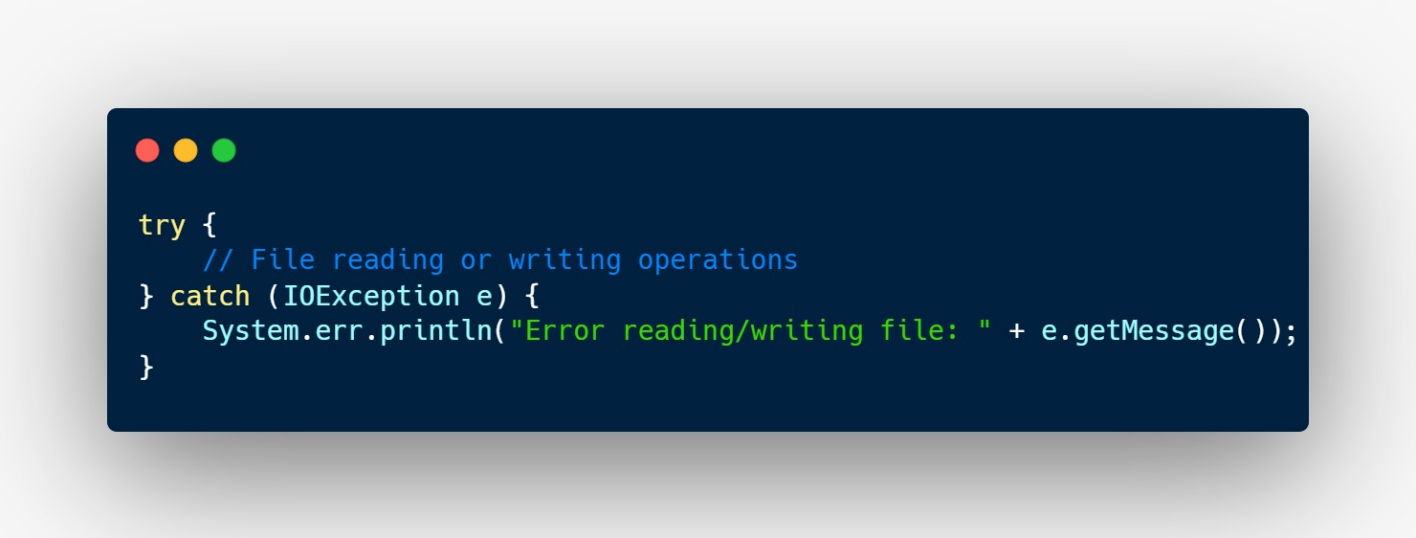
**If the user enters a non-integer value, the program gracefully catches InputMismatchException, prints an error message, and proceeds to the next iteration of the loop.**

**This approach safeguards a smooth and error-resistant user experience, crucial for maintaining the application's reliability.**

**Polymorphism in Displaying Details**

**This snippet showcases polymorphism via method overriding.**

**In the Student class, the displayDetails method is specialized compared to its superclass (Person), incorporating details such as courses, grades, and absences. This flexibility and code readability result from using the same method name in both classes.**

 **Exception Handling in File Operations**

**In managing file operations, the code implements a robust exception handling mechanism.**

**It utilizes try-catch blocks, particularly catching IOException, a general exception for input/output issues. This strategy ensures the program gracefully handles potential errors during file operations, such as file not found or permission issues.**

**By doing so, the code prevents unexpected crashes and provides users with meaningful error messages, contributing to the overall resilience and user-friendliness of the application.**