Write-Up AP Assignment 2

Sakshat Sachdeva

September 2024

1 Introduction

The following consists of the write-up for the second assignment for the COURSE MANAGEMENT SYSTEM, or CMS for short under the course CSE-201.

2 Generic Classes Feedback System

The first part of the assignment consisted of creating a generic class known as `Feedback` which would be used by the `Student` and `Professor` classes.

The following is the code for the same.

```
public class Feedback<T> {
    private String courseCode;
    private String studentEmail;
    private T feedback;

    public Feedback(String courseCode, String studentEmail, T feedback) {
        this.courseCode = courseCode;
        this.studentEmail = studentEmail;
        this.feedback = feedback;
    }

    public String getCourseCode() {
        return courseCode;
    }

    public String getStudentEmail() {
        return studentEmail;
    }

    public T getFeedback() {
        return feedback;
    }
```

```
    @Override
    public String toString() {
        return "Course: " + courseCode + ", Student: "
        + studentEmail + ", Feedback: " + feedback.toString();
    }
}
```

2.1 Explanation of the Code

- `Feedback<T>` is a generic class where `T` represents the type of feedback. This allows flexibility, as feedback can be of any type (e.g., `String`, `Integer`, or custom objects).
- The `courseCode` and `studentEmail` fields are used to store the course's code and the student's email who is giving the feedback, respectively.
- The constructor `Feedback(String courseCode, String studentEmail, T feedback)` is used to initialize the `courseCode`, `studentEmail`, and `feedback` fields with values passed as arguments when creating an instance of the `Feedback` class.
- The `getCourseCode()` method returns the course code associated with the feedback.
- The `getStudentEmail()` method returns the student's email who submitted the feedback.
- The `getFeedback()` method returns the feedback value of type `T`. This method is flexible because `T` can be any type, depending on how the `Feedback` class is instantiated.
- The `toString()` method provides a string representation of the `Feedback` object, combining the course code, student email, and feedback information in a readable format.

3 Enhanced User Role Management with Object Class

This section outlines the implementation of the `TeachingAssistant` class that inherits from the `Student` class. The purpose of this class is to introduce a new role, `Teaching Assistant (TA)`, with additional functionalities for managing student grades while retaining the core properties of a student. The implementation demonstrates the use of inheritance, maintaining the integrity of the `Student`, `Professor`, and `Administrator` classes.

3.1 Code Implementation

```
import javax.swing.*;

public class TeachingAssistant extends Student {
    public String email;
    public String password;
    public String TACCode;

    public TeachingAssistant(String email, String password,
    String CourseCode) {
        super(email, password);
        this.email = email;
        this.password = password;
        this.TACCode = CourseCode;
    }

    public static void teachingAssistantMenu(TeachingAssistant
    teachingAssistant) {
        JOptionPane.showMessageDialog
        (null, "Welcome Teaching Assistant");
        while (true) {
            int choice = JOptionPane.showOptionDialog(null,
            "Choose which Menu to access", "CMS",
            JOptionPane.DEFAULT_OPTION,
            JOptionPane.INFORMATION_MESSAGE, null,
            new String[]{"Student Menu", "TA Menu", "Exit"}, null);
            if (choice == 0) {
                Student.studentMenu(teachingAssistant);
            } else if (choice == 1) {
                teachingAssistant.teachingAssistantOptions();
            } else if (choice == 2) {
                return;
            } else {
                JOptionPane.showMessageDialog(null, "Invalid choice");
            }
        }
    }

    public void teachingAssistantOptions() {
        int choice = JOptionPane.showOptionDialog(null,
        "Choose the action you want to perform", "CMS",
        JOptionPane.DEFAULT_OPTION,
        JOptionPane.INFORMATION_MESSAGE, null, new String[]
        {"View Student List", "Grade Student", "Exit"}, null);
        if (choice == 0) {
```

```
                for (Course course : User.CourseList) {
                    if (course.CCode.equals(this.TACCode)) {
                        StringBuilder students_list =
                        new StringBuilder("Student List\n");
                        for (String student_email : course.StudentList) {
                            students_list.append("Email: ")
                            .append(student_email).append("\n");
                        }
                        JOptionPane.showMessageDialog(null,
                        students_list.toString());
                        break;
                    }
                }
            } else if (choice == 1) {
                String studentEmail = JOptionPane.showInputDialog(null,
                "Enter the email of the student you want to grade");
                String grade = JOptionPane.showInputDialog(null,
                "Enter the grade of the student");
                for (Student student : User.Students) {
                    if (student.email.equals(studentEmail)) {
                        for (Course course : student.courses) {
                            if (course.CCode.equals(this.TACCode)) {
                                course.Course_grade = grade;
                                JOptionPane.showMessageDialog(null,
                                "Grade updated successfully");
                                return;
                            }
                        }
                    }
                }
            } else if (choice == 2) {
                return;
            } else {
                JOptionPane.showMessageDialog(null, "Invalid choice");
            }
        }

        public static void initaliseAccs() {
            User.TeachingAssistants.add(new TeachingAssistant("ta", "ta", "DSA"));
        }
    }
```

3.2 Explanation of the Code

- `TeachingAssistant` extends the `Student` class, meaning it inherits all the capabilities of the `Student` class and adds additional functionalities such as managing student grades.
- The constructor `TeachingAssistant(String email, String password, String CourseCode)` initializes the `email`, `password`, and `TACCode` fields, while also calling the parent `Student` constructor using `super()` to initialize the inherited `Student` properties.
- The `teachingAssistantMenu(TeachingAssistant teachingAssistant)` method provides a menu system allowing the TA to either access the `Student Menu` (inherited from the `Student` class) or the new `TA Menu`, which offers additional functionalities like viewing and grading students.
- The `teachingAssistantOptions()` method offers the TA specific actions:
  - `View Student List`: Displays a list of students enrolled in the course associated with the TA's `TACCode`.
  - `Grade Student`: Prompts the TA to input a student's email and their grade, which is then updated in the course list.
- The `initaliseAccs()` method creates an initial TA account, adding it to the `User.TeachingAssistants` list. The TA created here has the email `"ta"` and is associated with the course `"DSA"`.

This implementation demonstrates how inheritance can be effectively used to extend the functionalities of an existing class, in this case, the `Student` class, to create a specialized role with additional responsibilities.

4 Robust Exception Handling in Login and Course Registration

This section covers the implementation of custom exception handling for login failures and course registration scenarios where the course is full.

4.1 Login Exception Handling

We implemented a custom exception called `InvalidLoginE` to handle cases where a user enters invalid login credentials or makes an invalid choice. Below is the code for the login method, which uses the `InvalidLoginE` class for exception handling.

4.1.1 InvalidLoginE Class

```
public class InvalidLoginE extends Exception {
    public InvalidLoginE(String message) {
        super(message);
    }
}
```

4.1.2 Login Method with Exception Handling

The `s_login()` method prompts the user to either create an account or log in. If an invalid choice is made, an `InvalidLoginE` exception is thrown. The login credentials are validated, and if incorrect, the exception is also thrown.

```
public static Student s_login() {
    JOptionPane.showMessageDialog(null, "Please create or
----login to continue\n Choose 1 for creating a new account\n
----Choose 2 for login");
    int choice = Integer.parseInt(JOptionPane.showInputDialog(null,
    "Enter your choice"));

    try {
        if (choice == 1) {
            // Creating a new account
            JOptionPane.showMessageDialog(null, "Please
------------create a new account to continue");
            String email = JOptionPane.showInputDialog(null,
            "Enter your email");
            String password = JOptionPane.showInputDialog(null,
            "Enter your password");
            Student user = new Student(email, password);
            User.Students.add(user);
            JOptionPane.showMessageDialog(null,
            "Account created successfully\nPlease login to continue");
            String email1 = JOptionPane.
            showInputDialog(null, "Enter your email");
            String password1 = JOptionPane.showInputDialog(null,
            "Enter your password");
            return loginUser(email1, password1);
        } else if (choice == 2) {
            // Logging in
            JOptionPane.showMessageDialog(null, "Please login to continue");
            String email = JOptionPane.showInputDialog(null,
            "Enter your email");
            String password = JOptionPane.showInputDialog(null,
            "Enter your password");
```

```
            return loginUser(email, password);
        } else {
            throw new InvalidLoginE("Invalid choice.
------------Please enter 1 or 2.");
        }
    } catch (InvalidLoginE e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
        return null;
    }
}

public static Student loginUser(String email, String password)
throws InvalidLoginE {
    for (Student student : User.Students) {
        if (student.getEmail().equals(email)
        && student.getPassword().equals(password)) {
            JOptionPane.showMessageDialog(null, "Login successful");
            return student;
        }
    }
    throw new InvalidLoginE("Invalid login credentials");
}
```

4.2 Course Registration Exception Handling

In the course registration system, we handle cases where the course is full by throwing a `CourseFullE` exception. If the student exceeds the maximum credit limit or does not meet the prerequisites, appropriate messages are displayed.

4.2.1 CourseFullE Class

```
public class CourseFullE extends RuntimeException {
    public CourseFullE(String message) {
        super(message);
    }
}
```

4.2.2 Course Registration Method with Exception Handling

The following code checks if the course is full, if prerequisites are met, and if the student has not exceeded their credit limit. If the course is full, a `CourseFullE` exception is thrown.

```
public void register_course() {
    JOptionPane.showMessageDialog(null, "Courses available for registration");
    Admin.print_course();
```

```
        String course_name = JOptionPane.showInputDialog(null,
        "Enter the course code for the course you want to register for");

        for (Course course : CourseList) {
            if (course.CCode.equals(course_name)) {
                boolean prerequisitesMet = true;

                for (String prerequisite : course.preRequisites) {
                    boolean hasPrerequisite = false;
                    for (Course completedCourse : completed_courses) {
                        if (completedCourse.CCode.equals(prerequisite)) {
                            hasPrerequisite = true;
                            break;
                        }
                    }
                    if (!hasPrerequisite) {
                        prerequisitesMet = false;
                        break;
                    }
                }

                if (prerequisitesMet) {
                    if (this.Credits < 20 && course.Credits + this.Credits <= 20)
                    {
                        if (course.StudentList.size()
                        >= course.maxCapacity) {
                            throw new CourseFullE("Course " +
                            course.CCode + " is full.");
                        }
                        courses.add(course);
                        course.StudentList.add(this.email);
                        this.Credits += course.Credits;
                    } else {
                        JOptionPane.showMessageDialog(null,
                        "You have exceeded the maximum credit limit");
                    }
                } else {
                    JOptionPane.showMessageDialog(null,
                    "You do not meet the prerequisites for this course");
                }
                break;
            }
        }
    }
```

4.3 Explanation of Exception Handling

- The `s_login()` method ensures that invalid login attempts are handled with the `InvalidLoginE` exception.
- The course registration method throws a `CourseFullE` exception when a student tries to register for a course that is already full.
- Both methods use `try-catch` blocks to manage these exceptions gracefully, allowing the system to continue running without crashing and providing feedback to the user.

5 End.

This concludes the CMS and its write up. Give it a shot and let me know how it is. Any contributions to the same are appreciated!

Creator: Sakshat Sachdeva
Github: `https://github.com/Sak-drago`