# **Programming Assignment Unit 1**

Computer Science, University of the People

CS 1101-01 Programming Fundamentals - CS 1102-01 - AY2024-T2

Instructor, Noman Shihadeh

December 6, 2023

## Student Record Management System

For this assignment, we were asked to write a program that allows the user to manage student records. The system will allow the user to do the following basic actions:

1. Add a student’s information.
2. Edit a student’s information.
3. View a student's information.
4. Exit the program.

Source Code:

// StudentManagement.java

import java.util.ArrayList;

import java.util.Scanner;

/\*\*

 \* The StudentManagement class manages a list of students and provides

 \* controls to add, update, view, and manage student info.

 \*/

public class StudentManagement {

    private static ArrayList<Student> students = new ArrayList<>();

    private static Scanner scanner = new Scanner(System.in);

    /\*\*

     \* Adds a new student to the list.

     \*

     \* @param student - The student to be added.

     \*/

    public static void addStudent(Student student) {

        students.add(student);

        System.out.println("Student added successfully.");

    }

    /\*\*

     \* Updates the information of an existing student.

     \*

     \* @param ID             - The ID of the student to be updated.

     \* @param updatedStudent - The student object with updated information.

     \*/

    public static void updateStudent(String ID, Student updatedStudent) {

        for (Student student : students) {

            if (student.getID().equals(ID)) {

                student.setName(updatedStudent.getName());

                student.setAge(updatedStudent.getAge());

                student.setGrade(updatedStudent.getGrade());

                System.out.println("Student updated successfully.");

                return;

            }

        }

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

    }

    /\*\*

     \* Displays the details of a specific student.

     \*

     \* @param ID - The ID of the student whose details are to be viewed.

     \*/

    public static void viewStudent(String ID) {

        for (Student student : students) {

            if (student.getID().equals(ID)) {

                System.out.println("Name: " + student.getName());

                System.out.println("ID: " + student.getID());

                System.out.println("Age: " + student.getAge());

                System.out.println("Grade: " + student.getGrade());

                return;

            }

        }

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

    }

    /\*\*

     \* Gets integer input from the user with validation.

     \*

     \* @return A valid integer input.

     \*/

    private static int getIntInput() {

        while (!scanner.hasNextInt()) {

            System.out.println("Invalid input. Please enter a number.");

            scanner.next();

        }

        return scanner.nextInt();

    }

    /\*\*

     \* Creates a Student object based on user input.

     \*

     \* @return A new Student object.

     \*/

    private static Student createStudentFromInput() {

        scanner.nextLine();

        System.out.print("Enter Name: ");

        String name = scanner.nextLine();

        System.out.print("Enter ID: ");

        String ID = scanner.nextLine();

        System.out.print("Enter Age: ");

        int age = getIntInput();

        System.out.print("Enter Grade: ");

        while (!scanner.hasNextDouble()) {

            System.out.println("Invalid input. Please enter a number.");

            scanner.next();

        }

        double grade = scanner.nextDouble();

        return new Student(name, ID, age, grade);

    }

    /\*\*

     \* Handles the process of updating student information.

     \*/

    private static void updateStudentInfo() {

        System.out.print("Enter Student ID to Update: ");

        String updateID = scanner.nextLine();

        // Check if the student with the given ID exists

        boolean studentExists = false;

        for (Student student : students) {

            if (student.getID().equals(updateID)) {

                studentExists = true;

                break;

            }

        }

        // If student exists, proceed with update. Otherwise,

        // notify user and return to menu

        if (studentExists) {

            Student updatedStudent = createStudentFromInput();

            updateStudent(updateID, updatedStudent);

        } else {

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

        }

    }

    /\*\*

     \* The main method to run the application. It displays a menu for managing

     \* students.

     \*/

    public static void main(String[] args) {

        while (true) {

            System.out.println("1. Add New Student\n2. Update Student Information\n3. View Student Details\n4. Exit");

            System.out.print("Choose an option: ");

            int choice = getIntInput();

            switch (choice) {

                case 1:

                    addStudent(createStudentFromInput());

                    break;

                case 2:

                    updateStudentInfo();

                    break;

                case 3:

                    System.out.print("Enter Student ID to View: ");

                    scanner.nextLine();

                    viewStudent(scanner.nextLine());

                    break;

                case 4:

                    System.out.println("Exiting...");

                    System.exit(0);

                default:

                    System.out.println("Invalid option. Please try again.");

            }

        }

    }

}

Student Class:

// Student.java

/\*\*

 \* The Student class represents a student with their personal and academic

 \* details.

 \*/

public class Student {

    // Private instance variables to store student information

    private String name;

    private String ID;

    private int age;

    private double grade;

    /\*\*

     \* Constructor to initialize a new Student object with given details.

     \*

     \* @param name  - The name of the student.

     \* @param ID    - The unique identifier for the student.

     \* @param age   - The age of the student.

     \* @param grade - The grade of the student.

     \*/

    public Student(String name, String ID, int age, double grade) {

        this.name = name;

        this.ID = ID;

        this.age = age;

        this.grade = grade;

    }

    /\*\*

     \* Getter for the student's name.

     \*

     \* @return The name of the student.

     \*/

    public String getName() {

        return name;

    }

    /\*\*

     \* Setter for the student's name.

     \*

     \* @param name - The new name of the student.

     \*/

    public void setName(String name) {

        this.name = name;

    }

    /\*\*

     \* Getter for the student's ID.

     \*

     \* @return The ID of the student.

     \*/

    public String getID() {

        return ID;

    }

    /\*\*

     \* Setter for the student's ID.

     \*

     \* @param ID - The new ID of the student.

     \*/

    public void setID(String ID) {

        this.ID = ID;

    }

    /\*\*

     \* Getter for the student's age.

     \*

     \* @return The age of the student.

     \*/

    public int getAge() {

        return age;

    }

    /\*\*

     \* Setter for the student's age.

     \*

     \* @param age - The new age of the student.

     \*/

    public void setAge(int age) {

        this.age = age;

    }

    /\*\*

     \* Getter for the student's grade.

     \*

     \* @return The grade of the student.

     \*/

    public double getGrade() {

        return grade;

    }

    /\*\*

     \* Setter for the student's grade.

     \*

     \* @param grade - The new grade of the student.

     \*/

    public void setGrade(double grade) {

        this.grade = grade;

    }

}

Output:

1. Add New Student

2. Update Student Information

3. View Student Details

4. Exit

Choose an option: 1

Enter Name: Simon

Enter ID: 1234

Enter Age: 32

Enter Grade: 4.0

Student added successfully.

1. Add New Student

2. Update Student Information

3. View Student Details

4. Exit

Choose an option: 2

Enter Student ID to Update: 1234

Enter Name: Bob

Enter ID: 1234

Enter Age: 33

Enter Grade: 3.9

Student updated successfully.

1. Add New Student

2. Update Student Information

3. View Student Details

4. Exit

Choose an option: 3

Enter Student ID to View: 1234

Name: Bob

ID: 1234

Age: 33

Grade: 3.9

1. Add New Student

2. Update Student Information

3. View Student Details

4. Exit

Choose an option: 4

Exiting...

The main parts of the code:

1. StudentManagment.java
   1. The “main” method is our program entry point, it is also where we show the user the options he has and process his input by calling the appropriate method relevant to his selection.
   2. The “addStudent” method allows the user to add a student to the records by guiding him through the adding process of entering the required values for a student. The data input by the user is validated and then added to our static array storage for later use.
   3. The “updateStudentInfo” method allows the user to enter a student id he wants to edit, if the student ID is found in the records the user is prompted to enter the new information for that student. If the ID is not located in the current records the user is returned to the main menu.
   4. The “viewStudent” method takes the ID given by the user's input and displays the student’s details. If, however, the ID isn’t found in the records a message is displayed.
   5. The “createStudentFromInput” method is a kind of helper method that is used in multiple points in the code. It helps us build a student object based on the input of the user. This includes verifying that the input is valid and correct. Once all the valid user input is entered into the new object instance the reference is returned to the methods caller for continued use.
2. Student.java
   1. This class represents the blueprints for creating a student object.
   2. The constructor takes in the initial values for the object and passes them to the local variables for the object.
   3. Each variable is also wrapped in a getter and setter method that allows safe and encapsulated access to the local variables.

## References

Java Language and Virtual Machine Specifications

<https://docs.oracle.com/javase/specs/index.html>

Introduction to Programming Using Java - Version 9.0, JavaFX Edition

<https://math.hws.edu/javanotes/>

Source Code:

