# **Programming Assignment Unit 5**

Computer Science, University of the People

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

Instructor, Noman Shihadeh

December 21, 2023

## Course Enrollment and Grade Management System

For this assignment, we were asked to write a program that manages students and courses and the connection between them. We need to build the solution using OOP and encapsulation principles.

1. Students
   1. List Students
   2. Add Students
   3. Edit Students
   4. View Total Grade
2. Courses
   1. List Courses
   2. Add Courses
   3. Edit Courses
3. Enroll a student in a course
4. Set a grade for a given student in a given course.

Source Code:

// CourseManagement.java

package Unit5;

import java.util.ArrayList;

import java.util.Scanner;

/\*\*

 \* The CourseManagement class is the main class of the Hogwarts Course

 \* Management System.

 \* It provides a menu for the user to interact with the system.

 \*/

public class CourseManagement {

    private static ArrayList<Course> courses = new ArrayList<>();

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

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

    /\*\*

     \* The main method of the CourseManagement class.

     \* Initializes the system, displays a welcome message, and processes user

     \* commands.

     \*

     \* @param args command line arguments

     \*/

    public static void main(String[] args) {

        initializeSystem();

        displayWelcomeMessage();

        processUserCommands();

    }

    /\*\*

     \* Displays a welcome message to the user.

     \*/

    private static void displayWelcomeMessage() {

        hr();

        System.out.println("| Welcome to the Hogwarts Course Management System. |");

        System.out.println("| Please select an option below.                    |");

        System.out.println("|    - Remember, bureaucracy is the real magic -    |");

    }

    /\*\*

     \* Displays the menu and processes user selection.

     \*/

    private static void processUserCommands() {

        boolean exit = false;

        while (!exit) {

            displayMenu();

            String choice = scanner.next();

            hr();

            switch (choice.toUpperCase()) {

                case "1":

                    addCourse();

                    break;

                case "2":

                    addStudent();

                    break;

                case "3":

                    enrollStudent();

                    break;

                case "4":

                    setGrade();

                    break;

                case "5":

                    calculateFinalGrades();

                    break;

                case "6":

                    listAllStudents();

                    break;

                case "7":

                    listAllCourses();

                    break;

                case "8":

                    Student.editStudentInfo(scanner, students);

                    break;

                case "9":

                    Course.editCourseInfo(scanner, courses);

                    break;

                case "E":

                    exit = true;

                    break;

                default:

                    System.out.println("Invalid choice. Try again, muggle!");

            }

        }

        System.out.println("Thank you for using the Hogwarts Course Management System.");

        System.out.println("Have a magical day!");

    }

    /\*\*

     \* Displays the menu.

     \*/

    private static void displayMenu() {

        hr();

        System.out.println("Magical System Menu:");

        System.out.println("1. Add a new magic course");

        System.out.println("2. Add a new Hogwarts student");

        System.out.println("3. Enroll a student in a magic course");

        System.out.println("4. Set a grade for magical studies");

        System.out.println("5. View OWLs for a student");

        System.out.println("6. List all students");

        System.out.println("7. List all courses");

        System.out.println("8. Edit a student's information");

        System.out.println("9. Edit a course's information");

        System.out.println("E. Exit");

        System.out.print("Enter your choice: ");

    }

    /\*\*

     \* Initializes the system with some courses and students.

     \*/

    private static void initializeSystem() {

        courses.add(new Course("POT101", "Potions", 5));

        courses.add(new Course("DADA101", "Defense Against the Dark Arts", 5));

        courses.add(new Course("CHRM101", "Charms", 5));

        courses.add(new Course("HERB101", "Herbology", 5));

        students.add(new Student("Harry Potter", "001"));

        students.add(new Student("Hermione Granger", "002"));

        students.add(new Student("Ron Weasley", "003"));

        students.add(new Student("Draco Malfoy", "004"));

        // Enroll all demo students in all demo courses

        for (Course course : courses) {

            for (Student student : students) {

                student.enrollInCourse(course);

            }

        }

    }

    /\*\*

     \* Adds a new course to the system.

     \*/

    private static void addCourse() {

        Course newCourse = Course.addNewCourse(scanner);

        courses.add(newCourse);

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

    }

    /\*\*

     \* Adds a new student to the system.

     \*/

    private static void addStudent() {

        Student newStudent = Student.addNewStudent(scanner, students);

        students.add(newStudent);

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

    }

    /\*\*

     \* Enrolls a student in a course.

     \*/

    private static void enrollStudent() {

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

        String studentId = scanner.next();

        System.out.print("Enter course code: ");

        String courseCode = scanner.next();

        Student student = findStudentById(studentId);

        Course course = findCourseByCode(courseCode);

        // Check if the student and course exist

        if (student != null && course != null) {

            String message = student.enrollInCourse(course);

            System.out.println(message);

        } else {

            System.out.println("Invalid student ID or course code.");

        }

    }

    /\*\*

     \* Sets a grade for a student in a course.

     \*/

    private static void setGrade() {

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

        String studentId = scanner.next();

        System.out.print("Enter course code: ");

        String courseCode = scanner.next();

        float grade = Utils.readFloatInRange(scanner, "Enter grade (0.0-100.0): ", 0.0f, 100.0f);

        Student student = findStudentById(studentId);

        Course course = findCourseByCode(courseCode);

        // Check if the student and course exist

        if (student != null && course != null) {

            student.setCourseGrade(course, grade);

            System.out.println("Grade set successfully.");

        } else {

            System.out.println("Invalid student ID or course code.");

        }

    }

    /\*\*

     \* Calculates the final grade for a student.

     \*/

    private static void calculateFinalGrades() {

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

        String studentId = scanner.next();

        Student student = findStudentById(studentId);

        // Check if the student exists

        if (student != null) {

            student.printFinalGrade();

        } else {

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

        }

    }

    /\*\*

     \* Lists all students in the system.

     \*/

    private static void listAllStudents() {

        // Check if there are no students

        if (students.isEmpty()) {

            System.out.println("No students registered.");

        } else {

            // Loop through all students and print their information

            for (Student student : students) {

                student.print();

            }

        }

    }

    /\*\*

     \* Lists all courses in the system.

     \*/

    private static void listAllCourses() {

        // Check if there are no courses

        if (courses.isEmpty()) {

            System.out.println("No courses available.");

        } else {

            // Loop through all courses and print their information

            for (Course course : courses) {

                course.print();

            }

        }

    }

    /\*\*

     \* Finds a student by ID.

     \*

     \* @param id the ID of the student to find

     \* @return the student with the given ID, or null if not found

     \*/

    private static Student findStudentById(String id) {

        // Loop through all students

        for (Student student : students) {

            // If the student's ID matches, return the student

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

                return student;

            }

        }

        return null;

    }

    /\*\*

     \* Finds a course by code.

     \*

     \* @param code the code of the course to find

     \* @return the course with the given code, or null if not found

     \*/

    private static Course findCourseByCode(String code) {

        // Loop through all courses

        for (Course course : courses) {

            // If the course's code matches, return the course

            if (course.getCourseCode().equals(code)) {

                return course;

            }

        }

        return null;

    }

    /\*\*

     \* Prints a horizontal line.

     \*/

    private static void hr() {

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

    }

}

// Course.java

package Unit5;

import java.util.Scanner;

import java.util.ArrayList;

/\*\*

 \* Represents a course offered at the university.

 \* This class includes information about the course code, name, maximum

 \* capacity, and the current number of enrolled students.

 \*/

public class Course {

    private String courseCode;

    private String name;

    private int maxCapacity;

    private int enrolledStudents;

    /\*\*

     \* Constructor for the Course class.

     \* Initializes a new course with the specified code, name, and maximum capacity.

     \*

     \* @param courseCode  the unique code for the course

     \* @param name        the name of the course

     \* @param maxCapacity the maximum number of students that can enroll in the

     \*                    course

     \*/

    public Course(String courseCode, String name, int maxCapacity) {

        this.courseCode = courseCode;

        this.name = name;

        this.maxCapacity = maxCapacity;

        this.enrolledStudents = 0;

    }

    /\*\*

     \* Checks if students can still enroll in the course.

     \*

     \* @return true if the current number of enrolled students is less than the

     \*         maximum capacity, false otherwise

     \*/

    public boolean canEnroll() {

        return enrolledStudents < maxCapacity;

    }

    /\*\*

     \* Increments the number of enrolled students by one, if the course is not full.

     \*/

    public void incrementEnrolledStudents() {

        // Check if the course is not full

        if (canEnroll()) {

            enrolledStudents++;

        } else {

            System.out.println("Course is full, cannot enroll any more students.");

        }

    }

    /\*\*

     \* Edits the course's information.

     \*

     \* @param newName     the new name of the course

     \* @param newCapacity the new maximum capacity of the course

     \*/

    private void editCourseInfo(String newName, int newCapacity) {

        this.name = newName;

        this.maxCapacity = newCapacity;

    }

    /\*\*

     \* Adds a new course to the system.

     \*

     \* @param scanner the scanner to read input from

     \* @return the new course

     \*/

    public static Course addNewCourse(Scanner scanner) {

        System.out.print("Enter course code: ");

        String code = scanner.next();

        scanner.nextLine();

        System.out.print("Enter course name: ");

        String name = scanner.nextLine();

        int capacity = Utils.readPositiveInt(scanner, "Enter max capacity: ");

        return new Course(code, name, capacity);

    }

    public static void editCourseInfo(Scanner scanner, ArrayList<Course> courses) {

        System.out.print("Enter course code: ");

        String courseCode = scanner.next();

        scanner.nextLine();

        Course course = findCourseByCode(courseCode, courses);

        if (course != null) {

            System.out.print("Enter new course name: ");

            String newName = scanner.nextLine();

            int newCapacity = Utils.readPositiveInt(scanner, "Enter new max capacity: ");

            course.editCourseInfo(newName, newCapacity);

            System.out.println("Course information updated successfully.");

        } else {

            System.out.println("Invalid course code.");

        }

    }

    private static Course findCourseByCode(String code, ArrayList<Course> courses) {

        for (Course course : courses) {

            if (course.getCourseCode().equals(code)) {

                return course;

            }

        }

        return null;

    }

    /\*\*

     \* Enrolls a student in the course.

     \*

     \* @param student the student to enroll

     \* @return a message indicating whether the student was enrolled successfully

     \*/

    public String enrollStudent(Student student) {

        // Check if the course is not full

        if (canEnroll()) {

            incrementEnrolledStudents();

            return "Student enrolled in " + this.name + " successfully.";

        }

        return "Cannot enroll in " + this.name + ", course is full.";

    }

    /\*\*

     \* Gets the course code.

     \*

     \* @return the course code.

     \*/

    public String getCourseCode() {

        return courseCode;

    }

    /\*\*

     \* Gets the course name.

     \*

     \* @return the name of the course.

     \*/

    public String getName() {

        return name;

    }

    /\*\*

     \* Gets the maximum capacity of the course.

     \*

     \* @return the maximum capacity.

     \*/

    public int getMaxCapacity() {

        return maxCapacity;

    }

    /\*\*

     \* Gets the current number of enrolled students in the course.

     \*

     \* @return the number of enrolled students.

     \*/

    public int getEnrolledStudents() {

        return enrolledStudents;

    }

    /\*\*

     \* Prints the course info to the console.

     \*/

    public void print() {

        System.out.println("Code: " + courseCode +

                ", Name: " + name +

                ", Capacity: " + maxCapacity +

                ", Enrolled: " + enrolledStudents +

                ", Percentage Full: " + (enrolledStudents \* 100 / maxCapacity) + "%");

    }

}

// EnrolledCourse.java

package Unit5;

import java.util.Date;

/\*\*

 \* Represents a course in which a student is enrolled.

 \* This class extends the Course class by adding a grade and the date of

 \* enrollment.

 \*/

public class EnrolledCourse extends Course {

    private float grade;

    private Date dateEnrolled;

    /\*\*

     \* Constructor for EnrolledCourse class.

     \* Initializes an enrolled course based on a given Course object.

     \* Sets the initial grade to 0.0 and the enrollment date to the current date.

     \*

     \* @param course The Course object to base the EnrolledCourse on

     \*/

    public EnrolledCourse(Course course) {

        super(course.getCourseCode(), course.getName(), course.getMaxCapacity());

        this.grade = 0.0f;

        this.dateEnrolled = new Date();

    }

    /\*\*

     \* Gets the grade of the student in this course.

     \*

     \* @return The numerical grade

     \*/

    public float getGrade() {

        return grade;

    }

    /\*\*

     \* Sets the grade of the student for this course.

     \*

     \* @param grade The numerical grade to set

     \*/

    public void setGrade(float grade) {

        // Check if the grade is within the valid range

        if (grade >= 0.0f && grade <= 100.0f) {

            this.grade = grade;

        } else {

            System.out.println("Invalid grade. Grade must be between 0.0 and 100.0.");

        }

    }

    /\*\*

     \* Gets the date when the student enrolled in the course.

     \*

     \* @return The date of enrollment

     \*/

    public Date getDateEnrolled() {

        return dateEnrolled;

    }

    /\*\*

     \* Prints the details of the enrolled course including the grade.

     \*/

    public void printEnrolledCourseDetails() {

        System.out.println("Course: " + getName() +

                ", Code: " + getCourseCode() +

                ", Enrolled On: " + dateEnrolled +

                ", Grade: " + grade);

    }

}

// Course.java

package Unit5;

import java.util.ArrayList;

import java.util.Scanner;

/\*\*

 \* The Student class represents a student.

 \* It stores the student's personal info and the courses they are enrolled in,

 \* along with the grades for each course.

 \*/

public class Student {

    private String name;

    private String ID;

    private ArrayList<EnrolledCourse> enrolledCourses;

    /\*\*

     \* Constructor for the Student class.

     \*

     \* @param name the name of the student

     \* @param ID   the unique ID of the student

     \*/

    public Student(String name, String ID) {

        this.name = name;

        this.ID = ID;

        this.enrolledCourses = new ArrayList<>();

    }

    /\*\*

     \* Enrolls the student in a course.

     \* Adds a new EnrolledCourse object to the enrolledCourses list if possible.

     \*

     \* @param course the course to enroll in

     \* @return a message indicating the result of the enrollment attempt

     \*/

    public String enrollInCourse(Course course) {

        // Check if student is already enrolled in the course

        for (EnrolledCourse enrolledCourse : enrolledCourses) {

            if (enrolledCourse.getCourseCode().equals(course.getCourseCode())) {

                return "Student already enrolled in " + course.getName() + ".";

            }

        }

        String enrollmentMessage = course.enrollStudent(this);

        // If the enrollment was successful, add the course to the student's list of

        // enrolled courses

        if (enrollmentMessage.contains("successfully")) {

            EnrolledCourse enrolledCourse = new EnrolledCourse(course);

            enrolledCourses.add(enrolledCourse);

        }

        return enrollmentMessage;

    }

    /\*\*

     \* Sets a grade to the student for a specific course.

     \*

     \* @param course the course to set the grade to

     \* @param grade  the grade to set

     \*/

    public void setCourseGrade(Course course, float grade) {

        // Loop through the student's enrolled courses

        for (EnrolledCourse enrolledCourse : enrolledCourses) {

            // If the student is enrolled in the specified course, set the grade

            if (enrolledCourse.getCourseCode().equals(course.getCourseCode())) {

                enrolledCourse.setGrade(grade);

                return;

            }

        }

        System.out.println("Student not enrolled in " + course.getName() + ".");

    }

    /\*\*

     \* Adds a new student to the system.

     \*

     \* @param scanner          The scanner to read input from.

     \* @param existingStudents The list of existing students to check for duplicate

     \*                         IDs.

     \* @return The newly created student.

     \*/

    public static Student addNewStudent(Scanner scanner, ArrayList<Student> existingStudents) {

        System.out.print("Enter student name: ");

        String name = scanner.next();

        scanner.nextLine(); // Consume the rest of the line after reading name

        String id;

        // Loop until a valid ID is entered

        while (true) {

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

            id = scanner.next();

            // Check if the ID is valid and unique

            if (!id.trim().isEmpty() && isUniqueId(id, existingStudents)) {

                break;

            } else {

                System.out.println("Invalid or duplicate ID. Please enter a unique ID.");

            }

        }

        return new Student(name, id);

    }

    public static void editStudentInfo(Scanner scanner, ArrayList<Student> students) {

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

        String studentId = scanner.next();

        Student student = findStudentById(studentId, students);

        if (student != null) {

            System.out.print("Enter new student name: ");

            String newName = scanner.next();

            student.setName(newName);

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

        } else {

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

        }

    }

    private static Student findStudentById(String id, ArrayList<Student> students) {

        for (Student student : students) {

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

                return student;

            }

        }

        return null;

    }

    /\*\*

     \* Checks if a given ID is unique.

     \*

     \* @param id       The ID to check.

     \* @param students The list of existing students to check against.

     \* @return True if the ID is unique, false otherwise.

     \*/

    private static boolean isUniqueId(String id, ArrayList<Student> students) {

        // Loop through the list of existing students

        for (Student student : students) {

            // If the ID matches, return false

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

                return false;

            }

        }

        return true;

    }

    /\*\*

     \* Edits the student's information.

     \*

     \* @param newName the new name of the student

     \* @param newId   the new ID of the student

     \*/

    public void editStudentInfo(String newName, String newId) {

        this.name = newName;

        this.ID = newId;

    }

    /\*\*

     \* Gets the student's name.

     \*

     \* @return the name of the student

     \*/

    public String getName() {

        return name;

    }

    /\*\*

     \* Sets the student's name.

     \*

     \* @param name the new name of the student

     \*/

    public void setName(String name) {

        this.name = name;

    }

    /\*\*

     \* Gets the student's ID.

     \*

     \* @return the ID of the student

     \*/

    public String getID() {

        return ID;

    }

    /\*\*

     \* Sets the student's ID.

     \*

     \* @param ID the new ID of the student

     \*/

    public void setID(String ID) {

        this.ID = ID;

    }

    /\*\*

     \* Calculates and returns the student's GPA.

     \* GPA is computed as the average of grades in enrolled courses.

     \*

     \* @return the GPA of the student

     \*/

    public float getGPA() {

        // If the student is not enrolled in any courses, return 0.0

        if (enrolledCourses.isEmpty()) {

            return 0.0f;

        }

        float totalGrade = 0.0f;

        // Loop through the student's enrolled courses and add up the grades

        for (EnrolledCourse course : enrolledCourses) {

            // Add the grade to the total

            totalGrade += course.getGrade();

        }

        // Divide the total grade by the number of enrolled courses to get the average

        return totalGrade / enrolledCourses.size();

    }

    /\*\*

     \* Prints the student's info and final grade to the console.

     \*/

    public void printFinalGrade() {

        float gpa = this.getGPA();

        System.out.println("Student: " + name + " (ID: " + ID + ")");

        System.out.println("GPA: " + gpa + ", Grade: " + getGradeAsString(gpa));

    }

    /\*\*

     \* Converts a float grade to a string grade.

     \*

     \* @param grade The grade to convert.

     \* @return The string representation of the grade.

     \*/

    private static String getGradeAsString(float grade) {

        if (grade >= 90.0f) {

            return "A";

        } else if (grade >= 80.0f) {

            return "B";

        } else if (grade >= 70.0f) {

            return "C";

        } else if (grade >= 60.0f) {

            return "D";

        } else {

            return "F";

        }

    }

    /\*\*

     \* Prints the student's information to the console.

     \*/

    public void print() {

        System.out.println("ID: " + ID + ", Name: " + name);

        // Check if the student is not enrolled in any courses

        if (enrolledCourses.isEmpty()) {

            System.out.println("Not enrolled in any courses (yet).");

            return;

        }

        System.out.println("Enrolled Courses:");

        // Loop through the student's enrolled courses and print them

        for (EnrolledCourse course : enrolledCourses) {

            System.out.println("Enrolled in: " + course.getName() + ", Grade: " + course.getGrade());

        }

    }

}

// Utils.java

package Unit5;

import java.util.Scanner;

/\*\*

 \* The Utils class provides utility methods for the Course Management System.

 \*/

public final class Utils {

    /\*\*

     \* Reads a positive integer from the console.

     \*

     \* @param scanner The scanner to read input from.

     \* @param prompt  The prompt to display to the user.

     \* @return A positive integer input by the user.

     \*/

    public static int readPositiveInt(Scanner scanner, String prompt) {

        int number;

        while (true) {

            System.out.print(prompt);

            // Check if the next token is an integer

            if (scanner.hasNextInt()) {

                number = scanner.nextInt();

                // Check if the integer is positive

                if (number > 0)

                    return number;

                else

                    System.out.println("Value must be a positive integer.");

            } else {

                System.out.println("Invalid input. Please enter an integer.");

                scanner.next();

            }

        }

    }

    /\*\*

     \* Reads a float value within a specified range from the console.

     \*

     \* @param scanner The scanner to read input from.

     \* @param prompt  The prompt to display to the user.

     \* @param min     The minimum acceptable value.

     \* @param max     The maximum acceptable value.

     \* @return A float value input by the user within the specified range.

     \*/

    public static float readFloatInRange(Scanner scanner, String prompt, float min, float max) {

        float number;

        // Loop until a valid input is entered

        while (true) {

            System.out.print(prompt);

            if (scanner.hasNextFloat()) {

                number = scanner.nextFloat();

                // Check if the number is within the specified range

                if (number >= min && number <= max)

                    return number;

                else

                    System.out.println("Value must be between " + min + " and " + max + ".");

            } else {

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

                scanner.next();

            }

        }

    }

}

Output:

-----------------------------------------------------

| Welcome to the Hogwarts Course Management System. |

| Please select an option below. |

| - Remember, bureaucracy is the real magic - |

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: 1

-----------------------------------------------------

Enter course code: MG101

Enter course name: Magic For Beginners

Enter max capacity: 6

Course added successfully.

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: 2

-----------------------------------------------------

Enter student name: Simon Stirling

Enter student ID: 1234

Student added successfully.

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: 3

-----------------------------------------------------

Enter student ID: 1234

Enter course code: MG101

Student enrolled in Magic For Beginners successfully.

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: 4

-----------------------------------------------------

Enter student ID: 1234

Enter course code: MG101

Enter grade (0.0-100.0): 99.9

Grade set successfully.

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: 5

-----------------------------------------------------

Enter student ID: 1234

Student: Simon (ID: 1234)

GPA: 99.9, Grade: A

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: 6

-----------------------------------------------------

ID: 001, Name: Harry Potter

Enrolled Courses:

Enrolled in: Potions, Grade: 0.0

Enrolled in: Defense Against the Dark Arts, Grade: 0.0

Enrolled in: Charms, Grade: 0.0

Enrolled in: Herbology, Grade: 0.0

ID: 002, Name: Hermione Granger

Enrolled Courses:

Enrolled in: Potions, Grade: 0.0

Enrolled in: Defense Against the Dark Arts, Grade: 0.0

Enrolled in: Charms, Grade: 0.0

Enrolled in: Herbology, Grade: 0.0

ID: 003, Name: Ron Weasley

Enrolled Courses:

Enrolled in: Potions, Grade: 0.0

Enrolled in: Defense Against the Dark Arts, Grade: 0.0

Enrolled in: Charms, Grade: 0.0

Enrolled in: Herbology, Grade: 0.0

ID: 004, Name: Draco Malfoy

Enrolled Courses:

Enrolled in: Potions, Grade: 0.0

Enrolled in: Defense Against the Dark Arts, Grade: 0.0

Enrolled in: Charms, Grade: 0.0

Enrolled in: Herbology, Grade: 0.0

ID: 1234, Name: Simon

Enrolled Courses:

Enrolled in: Magic For Beginners, Grade: 99.9

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: 7

-----------------------------------------------------

Code: POT101, Name: Potions, Capacity: 5, Enrolled: 4, Percentage Full: 80%

Code: DADA101, Name: Defense Against the Dark Arts, Capacity: 5, Enrolled: 4, Percentage Full: 80%

Code: CHRM101, Name: Charms, Capacity: 5, Enrolled: 4, Percentage Full: 80%

Code: HERB101, Name: Herbology, Capacity: 5, Enrolled: 4, Percentage Full: 80%

Code: MG101, Name: Magic For Beginners, Capacity: 6, Enrolled: 1, Percentage Full: 16%

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: 8

-----------------------------------------------------

Enter student ID: 1234

Enter new student name: Simon B.Stirling

Student information updated successfully.

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: -----------------------------------------------------

Invalid choice. Try again, muggle!

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: 9

-----------------------------------------------------

Enter course code: MG101

Enter new course name: Magic Begins

Enter new max capacity: 7

Course information updated successfully.

-----------------------------------------------------

Magical System Menu:

1. Add a new magic course

2. Add a new Hogwarts student

3. Enroll a student in a magic course

4. Set a grade for magical studies

5. View OWLs for a student

6. List all students

7. List all courses

8. Edit a student's information

9. Edit a course's information

E. Exit

Enter your choice: E

-----------------------------------------------------

Thank you for using the Hogwarts Course Management System.

Have a magical day!

The main parts of the code:

1. The “main” method is our program entry point, it is also where we build our demo stock data to be passed into the different methods and then we print the results to the user interface.
2. The “calculateAveragePrice” method takes in a single parameter called “prices” and contains a float array. The method then iterates over the array while for each iteration it accumulates the value in the array’s current location into a helper variable. After the loop exits, we return the helper variable as a return value.
3. The “findMaximumPrice” method takes in a single parameter called “prices” and contains a float array. After initializing and creating a local helper variable to the value located in the first position of the array. Next, we iterate over the array while for each item in the array, we check to see it is a value that is greater than the value currently found in our helper variable. If there is, we replace the value in the helper variable with the current array value. Finally, when we exit the loop, we return the value found in the helper variable since it now should hold the largest value found in the array.
4. The “countOccurrences” method takes in two parameters the first, is called “prices” and contains a float array. The second is called “targetPrice” and contains a float. In this method, we first create and initialize a local helper variable which will contain the incrementor that will represent and count the number of times the value “targetPrice” was found in the array. We next iterate through the array each time checking if the current value in the array is equal to the value found in “targetPrice” If yes, we increment the helper variable if not, we continue with no action. At the end of the loop, we return the value in the helper variable that represents the number of times we encountered the value in the array.
5. The “computeCumulativeSum” method takes in a single parameter called “prices” and contains an ArrayList of floats. The first thing we do is to create and initialize a new local variable of ArrayList that we will use and populate with the new values from the calculations we will perform in this method. We will also need another helper variable to represent a running summary of the values as we iterate the list. We now iterate over the list each time adding the current list value to the helper variable and then adding the calculated value into the new local list variable. At the end of the loop, we return the local list value which will contain a running cumulative sum of the list that was passed in the parameters.

## 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: