```
// 1. Person (Base Class - for common
attributes)
public class Person {
    private String id;
    private String name;
    private String email;
    private String phoneNumber;

    public Person(String id, String name,
String email, String phoneNumber) {
        // Basic validation in constructor
        if (id == null || id.trim().isEmpty()) {
            throw new
IllegalArgumentException("ID cannot be
null or empty.");
        }
        if (name == null ||
name.trim().isEmpty()) {
            throw new
IllegalArgumentException("Name cannot
be null or empty.");
```

```
        }
        if (email == null || !
email.contains("@")) {
            throw new
IllegalArgumentException("Invalid email
format.");
        }
        if (phoneNumber == null || !
phoneNumber.matches("\\d+")) { // Basic
digit check
            throw new
IllegalArgumentException("Phone number
must contain only digits.");
        }

        this.id = id;
        this.name = name;
        this.email = email;
        this.phoneNumber = phoneNumber;
    }
```

```
    // Getters
    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String getEmail() {
        return email;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    // Setters with validation (demonstrating
encapsulation)
    public void setName(String name) {
        if (name == null ||
```

```
name.trim().isEmpty()) {
            throw new
IllegalArgumentException("Name cannot
be null or empty.");
        }
        this.name = name;
    }

    public void setEmail(String email) {
        if (email == null || !
email.contains("@")) {
            throw new
IllegalArgumentException("Invalid email
format.");
        }
        this.email = email;
    }

    public void setPhoneNumber(String
phoneNumber) {
        if (phoneNumber == null || !
```

```
phoneNumber.matches("\\d+")) {
            throw new
IllegalArgumentException("Phone number
must contain only digits.");
        }
        this.phoneNumber = phoneNumber;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name +
", Email: " + email + ", Phone: " +
phoneNumber;
    }
}

// 2. Course
public class Course {
    private String courseId;
    private String courseName;
    private int credits;
```

```
    public Course(String courseId, String
courseName, int credits) {
        if (courseId == null ||
courseId.trim().isEmpty()) {
            throw new
IllegalArgumentException("Course ID
cannot be null or empty.");
        }
        if (courseName == null ||
courseName.trim().isEmpty()) {
            throw new
IllegalArgumentException("Course name
cannot be null or empty.");
        }
        if (credits <= 0) {
            throw new
IllegalArgumentException("Credits must
be positive.");
        }
```

```
    this.courseId = courseId;
    this.courseName = courseName;
    this.credits = credits;
}

// Getters
public String getCourseId() {
    return courseId;
}

public String getCourseName() {
    return courseName;
}

public int getCredits() {
    return credits;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
```

```
        if (o == null || getClass() !=
o.getClass()) return false;
        Course course = (Course) o;
        return
courseId.equals(course.courseId);
    }

    @Override
    public int hashCode() {
        return courseId.hashCode();
    }

    @Override
    public String toString() {
        return "Course ID: " + courseId + ",
Name: " + courseName + ", Credits: " +
credits;
    }
}

// 3. Grade
```

```
public class Grade {
    private Course course;
    private double score; // Can be
percentage or a point score

    public Grade(Course course, double
score) {
        if (course == null) {
            throw new
IllegalArgumentException("Course cannot
be null.");
        }
        if (score < 0 || score > 100) { //
Assuming score is out of 100
            throw new
IllegalArgumentException("Score must be
between 0 and 100.");
        }
        this.course = course;
        this.score = score;
    }
```

```
    // Getters
    public Course getCourse() {
        return course;
    }

    public double getScore() {
        return score;
    }

    @Override
    public String toString() {
        return "Course: " +
course.getCourseName() + ", Score: " +
String.format("%.2f", score);
    }
}

// 4. Student (inherits from Person)
import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.List;
import java.util.Map;

public class Student extends Person {
    private String studentId; // Could be the same as Person's ID, or a distinct one
    private List<Course> registeredCourses;
    private Map<String, Grade> grades; // Key: Course ID, Value: Grade object

    public Student(String studentId, String name, String email, String phoneNumber) {
        super(studentId, name, email, phoneNumber); // Call superclass constructor
        this.studentId = studentId; // Re-assign for clarity, though inherited from Person
        this.registeredCourses = new ArrayList<>();
        this.grades = new HashMap<>();
```

```
    }

    public String getStudentId() {
        return studentId;
    }

    // Encapsulated method for course
registration
    public boolean
registerForCourse(Course course) {
        if (course == null) {
            System.out.println("Cannot register
for a null course.");
            return false;
        }
        if (!
registeredCourses.contains(course)) {
            registeredCourses.add(course);
            System.out.println(getName() + "
registered for " +
course.getCourseName());
```

```
            return true;
        } else {
            System.out.println(getName() + " is
already registered for " +
course.getCourseName());
            return false;
        }
    }

    public List<Course>
getRegisteredCourses() {
        return new
ArrayList<>(registeredCourses); // Return a
copy to prevent external modification
    }

    // Encapsulated method for adding/
updating a grade
    public boolean addGrade(Course course,
double score) {
        if (course == null) {
```

```
            System.out.println("Cannot add
grade for a null course.");
            return false;
        }
        if (!
registeredCourses.contains(course)) {
            System.out.println(getName() + " is
not registered for " +
course.getCourseName() + ". Cannot add
grade.");
            return false;
        }
        try {
            Grade grade = new Grade(course,
score);
            grades.put(course.getCourseId(),
grade);
            System.out.println("Grade " + score
+ " added for " + getName() + " in " +
course.getCourseName());
            return true;
```

```
        } catch (IllegalArgumentException e) {
            System.out.println("Error adding
grade for " + getName() + ": " +
e.getMessage());
            return false;
        }
    }

    public Map<String, Grade> getGrades() {
        return new HashMap<>(grades); //
Return a copy to prevent external
modification
    }

    // Method to view grades
    public String viewGrades() {
        if (grades.isEmpty()) {
            return getName() + " has no grades
available yet.";
        }
        StringBuilder sb = new
```

```
StringBuilder("Grades for " + getName() + "
(ID: " + studentId + "):\n");
        for (Grade grade : grades.values()) {
            sb.append("-
").append(grade).append("\n");
        }
        return sb.toString();
    }

    // Method to update personal details
(leveraging Person's setters)
    public void updatePersonalDetails(String
newName, String newEmail, String
newPhoneNumber) {
        boolean updated = false;
        if (newName != null && !
newName.trim().isEmpty()) {
            try {
                setName(newName);
                updated = true;
            } catch (IllegalArgumentException
```

```
e) {
                System.out.println("Error updating
name: " + e.getMessage());
            }
        }
        if (newEmail != null && !
newEmail.trim().isEmpty()) {
            try {
                setEmail(newEmail);
                updated = true;
            } catch (IllegalArgumentException
e) {
                System.out.println("Error updating
email: " + e.getMessage());
            }
        }
        if (newPhoneNumber != null && !
newPhoneNumber.trim().isEmpty()) {
            try {

setPhoneNumber(newPhoneNumber);
```

```
                updated = true;
            } catch (IllegalArgumentException
e) {
                System.out.println("Error updating
phone number: " + e.getMessage());
            }
        }
        if (updated) {
            System.out.println("Personal details
for " + studentId + " updated.");
        } else {
            System.out.println("No valid
personal details provided for update or no
changes made for " + studentId + ".");
        }
    }

    @Override
    public String toString() {
        return "Student " + super.toString();
    }
```

```
}

// 5. StudentManagementSystem
(Orchestrator)
import java.util.HashMap;
import java.util.Map;

public class StudentManagementSystem {
    private Map<String, Student> students;
// Key: Student ID
    private Map<String, Course> courses; //
Key: Course ID

    public StudentManagementSystem() {
        this.students = new HashMap<>();
        this.courses = new HashMap<>();
    }

    // --- Student Operations ---
    public Student addStudent(String id,
String name, String email, String
```

```
phoneNumber) {
        if (students.containsKey(id)) {
            System.out.println("Error: Student
with ID " + id + " already exists.");
            return null;
        }
        try {
            Student student = new Student(id,
name, email, phoneNumber);
            students.put(id, student);
            System.out.println("Student " +
name + " (ID: " + id + ") added
successfully.");
            return student;
        } catch (IllegalArgumentException e) {
            System.out.println("Error adding
student: " + e.getMessage());
            return null;
        }
    }
```

```
    public boolean removeStudent(String
studentId) {
        if (students.containsKey(studentId)) {
            students.remove(studentId);
            System.out.println("Student with ID
" + studentId + " removed.");
            return true;
        } else {
            System.out.println("Error: Student
with ID " + studentId + " not found.");
            return false;
        }
    }

    public Student getStudent(String
studentId) {
        return students.get(studentId);
    }

    public void displayAllStudents() {
        if (students.isEmpty()) {
```

```
            System.out.println("No students
registered in the system.");
            return;
        }
        System.out.println("\n--- All Students
---");
        for (Student student :
students.values()) {
            System.out.println(student);
        }
    }

    // --- Course Operations ---
    public Course addCourse(String
courseId, String courseName, int credits) {
        if (courses.containsKey(courseId)) {
            System.out.println("Error: Course
with ID " + courseId + " already exists.");
            return null;
        }
        try {
```

```
            Course course = new
Course(courseId, courseName, credits);
            courses.put(courseId, course);
            System.out.println("Course '" +
courseName + "' (ID: " + courseId + ")
added successfully.");
            return course;
        } catch (IllegalArgumentException e) {
            System.out.println("Error adding
course: " + e.getMessage());
            return null;
        }
    }

    public Course getCourse(String
courseId) {
        return courses.get(courseId);
    }

    public void displayAllCourses() {
        if (courses.isEmpty()) {
```

```
            System.out.println("No courses
available in the system.");
            return;
        }
        System.out.println("\n--- All Courses
---");
        for (Course course : courses.values())
{
            System.out.println(course);
        }
    }

    // --- Registration and Grade Operations
---
    public boolean
registerStudentForCourse(String studentId,
String courseId) {
        Student student =
getStudent(studentId);
        Course course = getCourse(courseId);
```

```
        if (student == null) {
            System.out.println("Error: Student
with ID " + studentId + " not found.");
            return false;
        }
        if (course == null) {
            System.out.println("Error: Course
with ID " + courseId + " not found.");
            return false;
        }

        return
student.registerForCourse(course);
    }

    public boolean assignGrade(String
studentId, String courseId, double score) {
        Student student =
getStudent(studentId);
        Course course = getCourse(courseId);
```

```
        if (student == null) {
            System.out.println("Error: Student
with ID " + studentId + " not found.");
            return false;
        }
        if (course == null) {
            System.out.println("Error: Course
with ID " + courseId + " not found.");
            return false;
        }

        return student.addGrade(course,
score);
    }

    public String viewStudentGrades(String
studentId) {
        Student student =
getStudent(studentId);
        if (student == null) {
            return "Error: Student with ID " +
```

```
studentId + " not found.";
        }
        return student.viewGrades();
    }

    public void
updateStudentPersonalDetails(String
studentId, String newName, String
newEmail, String newPhoneNumber) {
        Student student =
getStudent(studentId);
        if (student == null) {
            System.out.println("Error: Student
with ID " + studentId + " not found. Cannot
update details.");
            return;
        }

student.updatePersonalDetails(newName,
newEmail, newPhoneNumber);
    }
```

```
}

// 6. Main Application (for demonstration)
public class Main {
    public static void main(String[] args) {
        StudentManagementSystem sms =
new StudentManagementSystem();

        // 1. Register Students
        System.out.println("\n--- Registering
Students ---");
        Student alice =
sms.addStudent("S001", "Alice Smith",
"alice@example.com", "1234567890");
        Student bob =
sms.addStudent("S002", "Bob Johnson",
"bob@example.com", "0987654321");
        sms.addStudent("S001", "Alice Smith",
"alice@example.com", "1234567890"); //
Attempt to add duplicate ID
        sms.addStudent("S003", "Charlie",
```

```
"charlie@.com", "123"); // Invalid email/
phone
        sms.displayAllStudents();

        // 2. Add Courses
        System.out.println("\n--- Adding
Courses ---");
        Course programming =
sms.addCourse("C101", "Introduction to
Programming", 3);
        Course dataStructures =
sms.addCourse("C102", "Data Structures",
4);
        Course databases =
sms.addCourse("C103", "Database
Management", 3);
        sms.addCourse("C101", "Introduction
to Programming", 3); // Attempt to add
duplicate ID
        sms.displayAllCourses();
```

```
        // 3. Register Students for Courses
        System.out.println("\n--- Registering
Students for Courses ---");
        if (alice != null && programming !=
null) {

sms.registerStudentForCourse(alice.getStu
dentId(), programming.getCourseId());
        }
        if (alice != null && dataStructures !=
null) {

sms.registerStudentForCourse(alice.getStu
dentId(), dataStructures.getCourseId());
        }
        if (bob != null && programming != null)
{

sms.registerStudentForCourse(bob.getStu
dentId(), programming.getCourseId());
        }
```

```
        if (bob != null && databases != null) {

sms.registerStudentForCourse(bob.getStu
dentId(), databases.getCourseId());
        }

        // Try to register for a course they are
already in
        if (alice != null && programming !=
null) {

sms.registerStudentForCourse(alice.getStu
dentId(), programming.getCourseId());
        }
        // Try to register non-existent student/
course

sms.registerStudentForCourse("S999",
"C101");

        // 4. Assign Grades
```

```
        System.out.println("\n--- Assigning
Grades ---");
        sms.assignGrade(alice.getStudentId(),
programming.getCourseId(), 85);
        sms.assignGrade(alice.getStudentId(),
dataStructures.getCourseId(), 92);
        sms.assignGrade(bob.getStudentId(),
programming.getCourseId(), 78);
        sms.assignGrade(bob.getStudentId(),
databases.getCourseId(), 90);

        // Try to assign grade for a course
student is not registered in
        sms.assignGrade(alice.getStudentId(),
databases.getCourseId(), 70);
        // Try to assign invalid grade
        sms.assignGrade(alice.getStudentId(),
programming.getCourseId(), 105);

        // 5. View Grades
        System.out.println("\n--- Viewing
```

```
Grades ---");

System.out.println(sms.viewStudentGrades
(alice.getStudentId()));

System.out.println(sms.viewStudentGrades
(bob.getStudentId()));

System.out.println(sms.viewStudentGrades
("S999")); // View grades for non-existent
student

        // 6. Update Personal Details
        System.out.println("\n--- Updating
Personal Details ---");

sms.updateStudentPersonalDetails(alice.g
etStudentId(), null,
"alice.smith@newdomain.com",
"9988776655");
```

```
sms.updateStudentPersonalDetails(bob.getStudentId(), "Robert Johnson", "invalid-email", null); // Invalid email update

sms.updateStudentPersonalDetails(bob.getStudentId(), "Robert Johnson Jr.", null, null); // Valid name update only
        sms.displayAllStudents(); // Show updated details

        // 7. Remove a Student
        System.out.println("\n--- Removing Student ---");

sms.removeStudent(alice.getStudentId());
        sms.displayAllStudents();

        // Try to get details of a removed student
        System.out.println("\n--- Attempting to view removed student's details ---");
```

```
System.out.println(sms.viewStudentGrades
(alice.getStudentId()));
    }
}
```