Exercise 1 (Inheritance, method overriding, access rights, super keyword)

We have the following code:

```
class Point {
   private double x;
   private double y;

   public Point(double x, double y) {
      this.x = x;
      this.y = y;
   }

   public String toString() {
      return "(" + this.x + "," + this.y + ")";
   }
}
```

1. Declare a class `PointName`, a subclass of `Point`, allowing manipulation of points defined by their coordinates and a name (of type `char`).
2. Declare a constructor that initializes the attributes of an object created from the `PointNom` class.
3. We want to define a method `move(double dx, double dy)` in the `PointName` class. Make the necessary changes for the declaration of this method, then declare it.
4. Override the `toString()` method in `PointName` to display a string in the form: `name(x,y)`. Example: `A(5,10)`. (The overridden `toString()` method should make use of the `toString()` method from the superclass `Point`.).
5. Create a `T ;,est` class with no attributes and a single `main()` method:

   - Cretae an object of type `PointName`, providing a character as the name and two double values as the coordinates (`'A'`, `5`, `10`).
   - Display the initial state of the object using the `toString()` method.
   - Call the `move(double dx, double dy)` method to change the coordinates of the point (move by `(dx=2, dy=3)`).
   - Display the updated state of the object after the movement.

Exercise 2 (Abstraction, type casting)

Given the following code:

```
public class Shape {
   public abstract double getArea();

   public boolean isLargerTha(Shape s){
      return this. getArea () > s.getArea ();
   }
}
```

```
public class ShapeName extends Shape {
   private char name;
   public ShapeName (char name) {
      this. name = name;
   }
}
```

```
public class Rectangle extends Shape {
   private double length;
   private double weidth;

   public Rectangle(double length, double weidth){
      this.length = length;
      this.weidth = weidth;
   }
}
```

```
public class TestShape {
   public static void main(String[] args) {
      Shape s = new Shape();
      ShapeName sn = new ShapeName('A');
      Rectangle r1 = new Rectangle(5,10);
      Rectangle r2 = new Rectangle(6,8);
      System.out.println(r1.isLargerThan(r2));
   }
}
```

- Identify and correct the errors in the above code.
- The underlined statements are correct. Explain why.

Exercise 3 (Method overriding, method lookup, polymorphism)

Given the following code:

```
class Person {
    public void displayClass() {
        System.out.print("I am a Person");
    }
}
```

```
class Student extends Person {
    private String fieldOfStudy;
    public Student(String fieldOfStudy) {
        this.fieldOfStudy = fieldOfStudy;
    }


    public void displayFieldOfStudy() {
        System.out.print(", my field is: " + this.fieldOfStudy);
    }

    @Override
    public void displayClass() {
        System.out.print("I am a Student");
        displayFieldOfStudy();
    }
}
```

```
class Employee extends Person {}
```

```
class StudentRepresentative extends Student {
    public StudentRepresentative(String fieldOfStudy) {
        super(fieldOfStudy);
    }
}
```

```
public class MethodOverridingTest {
    public static void main(String[] args) {
        Person p = new Person();
        p.displayClass();
        System.out.println();

        Employee e = new Employee();
        e.displayClass();
        System.out.println();

        Student s = new Student("Computer Science");
        s.displayClass();
        System.out.println();

        StudentRepresentativer = new StudentRepresentative ("Math");
        r.displayClass();
        System.out.println();
    }
}
```

```java
public class PolymorphismTest {
    public static void main(String[] args) {
        Person p = new Person();
        Student s = new Student("Computer Science");

        Person p1 = new Student("Math");
        p1.displayClass();
        p1.displayFieldOfStudy();
        Student s1 = new Person();
        s = p;
        s = (Student) p1;
        p = s;

        Object o = p;
    }
}
```

1. What will be displayed when running the class `MethodOverridingTest`? Justify your answers.
2. In the class `PolymorphismTest`, are the underlined statements correct? Justify your answer and suggest corrections if possible.
3. What will the `PolymorphismTest` class display after correction/removal of erroneous statements? Justify your answer.

Exercise 4: Inheritance and Interfaces – Course Evaluation System

A **course** is characterized by the following attributes:

- `title` (String) – the name of the course
- `credit` (int) – the number of credits
- `coefficient` (int) – the weight in the average
- `examGrade` (double) – the final exam grade

The class `Course` implements an interface called `Evaluable`.
The `Evaluable` interface declares two abstract methods:

```
double calculateAverage();
int calculateCreditsEarned();
```

The class `Course` has three subclasses:

1. **LectureCourse** – course with lectures only
2. **LectureTutorialCourse** – course with lectures and tutorials
3. **LectureTutorialPracticalCourse** – course with lectures, tutorials, and practicals

Rules for Calculating Averages :

- **LectureCourse :**
  `average = examGrade`
- **LectureTutorialCourse**
  Includes a `tutorialGrade` (double)
  Constant: `COEF_TUTORIAL = 0.33`

  ```
  average = examGrade * (1 - COEF_TUTORIAL) + tutorialGrade * COEF_TUTORIAL
  ```

- **LectureTutorialPracticalCourse**
  Includes `tutorialGrade` and `practicalGrade` (both doubles)
  Constants: `COEF_TUTORIAL = 0.2`, `COEF_PRACTICAL = 0.2`

  ```
  average = examGrade * (1 - COEF_TUTORIAL - COEF_PRACTICAL) +
            tutorialGrade * COEF_TUTORIAL +
            practicalGrade * COEF_PRACTICAL
  ```

Rule for Credit Acquisition

If the calculated **average ≥ 10**, the full course **credit is awarded**.
Otherwise, the earned credit is **0**.

Required Tasks

1. Define the interface `Evaluable` and the class hierarchy:

    - `Course` (base class)
    - `LectureCourse`, `LectureTutorialCourse`, and `LectureTutorialPracticalCourse` (subclasses)

2. In each class:

    - Create a constructor with parameters
    - Declare **getter** and **setter** methods for all attributes

3. Override the `toString()` method in the superclass and each subclass.
   The output format should resemble:

```
-------------------------------------
Course Title: Object-Oriented Programming
Credits: 5
Coefficient: 3
Exam Grade: 10.0
Tutorial Grade: 10.0
Practical Grade: 10.0
Average: 10.0
Credits Earned: 5
```

4. Write a class `MainProgram` with no attributes and a `main()` method that:

    - Declares a list of `Course` objects (use `ArrayList<Course>`)
    - Fills the list with courses from the 4rd semester of the 2nd year in Computer Science
    - Displays the details of each course using the overridden `toString()` method