Exercise 1:

1- Implement the following class that represents Students and contains the following attributes:
   - studentID: student id.
   - nbCourses: number of courses taken by student in last semester.
   - totalPoints: total points for student in last semester.
   - totalCredits: total number of hours taken by student in last semester.
   - fail: true if a student failed in at least one course otherwise false.
   - GPA: the student GPA in last semester.
   - money: student reward.

2- Using the class Student, write the class TestStudent that processes the following tasks:
   - Ask the user of the program to enter number of students to process.
   - For each student, read the ID and nbCourses from the user of the program.
   - For each course, read course credits and student grade in this course (A, B, C, D and F).
   - For each student, calculate totalCredits, totalPoints, GPA and money.
   - For each student, print his ID, GPA and money.
   - At the end of the program, print
     - ID and GPA of the student who has highest GPA.
     - How many students failed in the last semester?

Hints:
The student totalPoints:

<table>
<thead>
<tr>
<th>Grade</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points for one hour</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

The student GPA is totalPoints / totalCredits.
The student money is
   - 800.0 SR - if GPA less than or equal 2.0
   - 1500.0 SR – if GPA more than or equal 4.5 and totalPoints not less than 12 hour.
   - Otherwise, 1000.0 SR.
Exercise 2:

1- Implement the following class that represents Call and contains the following attributes:
   - day: the call start day between 1-7 (1 = Saturday ..... 7= Friday).
   - startHour: the start hour of the call between 1-24.
   - startMin: the start minute of the call between 0-59.
   - endHour: the end hour of the call between 1-24.
   - endMin: the end minute of the call between 0-59.
   - duration: the call duration in minutes.
   - cost: the cost of the call.

The call cost calculated by 35 cents for one minute except
   - Any call starts in Thursday or Friday calculated by 45 cents for one minute.
   - Any call starts from 12:00 (HH:MM) until 17:59 in the days from Saturday until Wednesday calculated by 50 cents for one minute.

2- Using the class Call, write the class TestCall that will show and run the following menu until user enters 5:

Choose:
1 > Enter new Call information.
2 > Print average cost of all calls.
3 > Print the total duration of all calls.
4 > Print the maximum duration of a call on Thursday or Friday.
5 > Exit.

Descriptions for each option:
1- Read only the day, start hour, start minute and duration of the call. Then print the call cost and call finish time [HH:MM].
2- Print the average cost of all calls have been read.
3- Print the total duration in minutes of all calls have been read.
4- Print the maximum duration of a call on Thursday or Friday [HH:MM], if any.
5- Exit from the program.

NOTE:
You must check the information entered by the user.
For example:
Date = 10, start Hour = 66, start Min = 77, duration = -13
Error: Bad call information
Exercise 3: (Homework)

1- Implement the following class that represents Circle.

**Geometry:**
We specify a circle in the plane by its center \((x, y)\) and its radius \(r\).

A. Circle \(i\) **intersects** circle \(j\) if the Euclidean distance between their centers is less than or equal to the sum of their radii. We consider two nested circles (one circle is completely contained inside the other) to intersect.

B. Circle \(i\) **contains** circle \(j\) if the Euclidean distance between their centers is less than or equal to the radius of circle \(i\) minus the radius of circle \(j\).

\[
\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \leq (r_i + r_j)
\]

\[
\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} > (r_i + r_j)
\]

\[
\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \leq (r_i - r_j)
\]

\[
\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} > (r_i - r_j)
\]

2- Using the class Circle, write the TestCircle class that processes the following tasks:
   a. Create a circle and read its attributes from keyboard.
   b. Ask the user to enter the information of many circles; the reading will stop if the user enters -1 for radius.
   c. For each circle display, Is it intersecting the first circle? Is it containing the first circle?
   d. At the end of the program, display how many circles intersect the first circle? Also, how many circles contain the first circle?

**Note: (Reducing the conditions) examples**
If circle \(i\) contains circle \(j\) then you don’t need to check the intersection,
Result: \([i\text{ contains }j]\).
If circle \(i\) doesn’t intersect circle \(j\) then you don’t need to check the containment,
Result: \([i\text{ does not intersect }j]\).