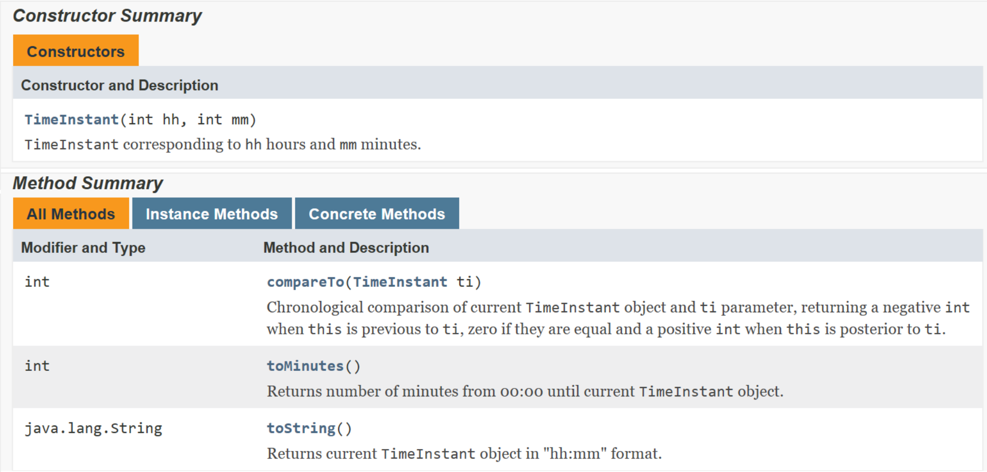
IIP First Partial (ETSInf)
November 6th, 2017. Time: 1 hour and 30 minutes.
Nota: The exam is marked up to 10 points, but its specific weight in the final grade of IIP is **2,4 points**

NAME:	GROUP:

1. 6 points A datatype class `RadioProgram` is going to be defined in order to represent a radio program. Each radio program has associated a title, a start time, an end time (both of the same day and being start time previous to end time), and a type of program, that could be a magazine, a musical program, or a news program. To represent the start and end times, the `TimeInstant` class is available, with the functionality that is partially shown in the documentation below.



Constructor Summary

Constructors

Constructor and Description
`TimeInstant(int hh, int mm)` `TimeInstant` corresponding to hh hours and mm minutes.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`int`	`compareTo(TimeInstant ti)` Chronological comparison of current `TimeInstant` object and `ti` parameter, returning a negative `int` when `this` is previous to `ti`, zero if they are equal and a positive `int` when `this` is posterior to `ti`.
`int`	`toMinutes()` Returns number of minutes from 00:00 until current `TimeInstant` object.
`java.lang.String`	`toString()` Returns current `TimeInstant` object in "hh:mm" format.

You must: implement the `RadioProgram` class, considering that it is in the same directory than the `TimeInstant` class, with the following attributes and methods:

a) (0.25 points) Integer public class and constant attributtes:
   - `MAGAZINE`, with value 0, that represents the magazine program type.
   - `MUSIC`, with value 1, that represents the musical program type.
   - `NEWS`, with value 2, that represents the news program type.

   These constants must be used whenever required (in the classes `RadioProgram` and `RadioManager`).

b) (0.5 points) Private instance attributes: `type` (`int`), `title` (`String`), `startTime` (`TimeInstant`) and `endTime` (`TimeInstant`).

c) (1.25 points) A general constructor such that, given the program type, its title, its start hours and minutes, and its duration in minutes, initialises all the instance attributes; you can suppose that all parameters have correct values.

d) (1 point) An `equals` method, that overrides that of the `Object` class, that checks if two radio programs are the same, i.e., if they have the same type and title.

e) (1 point) A `toString` method, that overrides that of the `Object` class, which, using a `switch` (**mandatory**) to convert to text the type of program, returns the `String` result with a format similar to that in the following examples:

```
06:30 - Radio 1 Breakfast show (Magazine)
10:00 - Clara Amfo (Music)
12:45 - Newsbeat (News)
```

f) (2 points) Two radio programs (aired in the same day) are considered as sorted in the program schedule according to the following criteria:
   - The program that starts earlier is previous.
   - With the same start time, the program that finishes earlier is previous.
   - With the same start and end time, news programs are previous than music program, and those are previous to magazines.
   - When time and type is the same, the order is indifferent.

   Implement a `compareTo` method that, given a `RadioProgram p` as parameter that is aired the same day than `this`, returns an `int` result that is negative when `this` is previous to `p` in the schedule, positive when `p` is previous, and `0` if they have indifferent order.

Solution:

```
public class RadioProgram {
    public static final int MAGAZINE = 0;
    public static final int MUSIC = 1;
    public static final int NEWS = 2;

    private int type;
    private String title;
    private TimeInstant startTime;
    private TimeInstant endTime;

    public RadioProgram(int typ, String tit, int h, int m, int time) {
        type = typ;
        title = tit;
        startTime = new TimeInstant(h, m);
        int end = startTime.toMinutes() + time;
        endTime = new TimeInstant(end / 60, end % 60);
    }

    public boolean equals(Object o) {
        return o instanceof RadioProgram
            && type == ((RadioProgram) o).type
            && title.equals(((RadioProgram) o).title);
    }

    public String toString() {
        String res = startTime + " - " + title + " (";
        switch (type) {
            case MAGAZINE:
                res += "Magazine)";
                break;
            case MUSIC:
                res += "Music)";
                break;
            case NEWS:
                res += "News)";
        }
        return res;
    }

    public int compareTo(RadioProgram p) {
        int res = startTime.compareTo(p.startTime);
        if (res == 0) {
            res = endTime.compareTo(p.endTime);
            if (res == 0) { res = p.type - type; }
        }
        return res;
    }
}
```

2. 2 points **You must:** implement the `RadioManager` program class, in the same directory as `RadioProgram` class, with a `main` method that executes the following actions:

   a) (0.25 points) Create an object `p1` of the `RadioProgram` datatype, to represent a magazine radio program of title `Radio 1 Breakfast show`, that starts at 6:30 and has a duration of 210 minutes.

   b) (0.25 points) Create an object `p2` of the `RadioProgram` datatype, to represent a musical radio program, with title `Clara Amfo`, that starts at 10:00 and has a duration of 165 minutes.

   c) (1.5 points) Compare `p1` with `p2` by using the `compareTo` method and, according its result, shows the programs on the screen according to its order in the schedule.

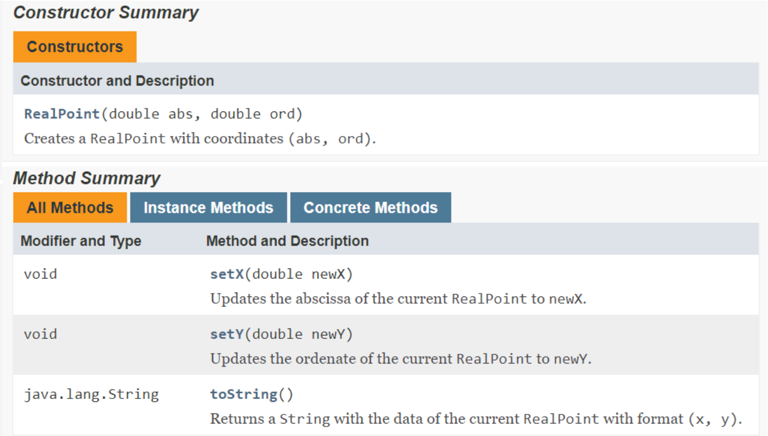
Solution:

```
public class RadioManager {
    public static void main(String[] args) {
        RadioProgram p1 = new RadioProgram(RadioProgram.MAGAZINE, "Radio 1 Breakfast show",
                                             6, 30, 210);

        RadioProgram p2 = new RadioProgram(RadioProgram.MUSIC, "Clara Amfo",
                                             10, 0, 165);

        if (p1.compareTo(p2) < 0) { System.out.println(p1 + "\n" + p2); }
        else { System.out.println(p2 + "\n" + p1); }
    }
}
```

3. 2 points You have available the RealPoint class, that defines a point in a bidimensional real space (by using two attributes which represent abscissa and ordenate), with a functionality that is partially shown in the following documentation:



Constructor Summary

Constructors

Constructor and Description
RealPoint(double abs, double ord) Creates a RealPoint with coordinates (abs, ord).

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	setX(double newX) Updates the abscissa of the current RealPoint to newX.
void	setY(double newY) Updates the ordenate of the current RealPoint to newY.
java.lang.String	toString() Returns a String with the data of the current RealPoint with format (x, y).

Given the following program class:

```
public class Exercise3 {
    public static void main(String[] args) {
        RealPoint p = new RealPoint(1.0, -1.0);
        double x = 1.0, y = -1.0;
        System.out.println("Before changeCoord: x = " + x + ", y = " + y
                           + ", p = " + p);

        changeCoord(x, y, p);
        System.out.println("After changeCoord once: x = " + x + ", y = " + y
                           + ", p = " + p);

        changeCoord(x, y, p);
        System.out.println("After changeCoord twice: x = " + x + ", y = " + y
                           + ", p = " + p);
    }

    public static void changeCoord(double x, double y, RealPoint p) {
        p.setX(y);
        p.setY(x);
    }
}
```

You must: Complete what is shown on the screen when it is executed.

```
Before changeCoord: x = 1.0, y = -1.0, p = (1.0, -1.0)

After changeCoord once: x = 1.0, y = -1.0, p = (-1.0, 1.0)

After changeCoord twice: x = 1.0, y = -1.0, p = (-1.0, 1.0)
```