# IIP First Partial - ETSInf

## November 9th, 2016. Time: 1 hour and 30 minutes.

**Note:** The exam is marked up to 10 points, but its specific weight in the final grade of IIP is 2.4 points

1. 1 point You have available the `RealPoint` class, that defines a point in a bidimensional real space by using the attributes `x` and `y` (which represent abscissa and ordenate, respectively), with a functionality that is partially shown in the following documentation:

**Constructor Summary**

Constructors

| Constructor and Description |
|---|
| `RealPoint()`<br>Creates a `RealPoint` with coordinates `(0.0,0.0)` |
| `RealPoint(double abs, double ord)`<br>Creates a `RealPoint` with coordinates `(abs,ord)` |

**Method Summary**

Methods

| Modifier and Type | Method and Description |
|---|---|
| `double` | `distance(RealPoint p)`<br>Returns distance from current `RealPoint` to `p` |
| `boolean` | `equals(java.lang.Object o)`<br>Returns `true` when `o` is a `RealPoint` with the same coordinates that the current `RealPoint` and returns `false` otherwise |
| `java.lang.String` | `toString()`<br>Returns a `String` with the data of the current `RealPoint` with format `(x, y)` |

**You must:** Implement the `distance` method such that returns the distance between the current `RealPoint` and another `RealPoint p` given as parameter, **rounding** the result to **four** decimal digits. Remember that distance between two points $(a_1, b_1)$ and $(a_2, b_2)$ is calculated as $\sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}$

**Solution:**

```
public double distance(RealPoint p) {
    double abs = this.x - p.x;
    double ord = this.y - p.y;
    double dist = Math.sqrt(abs * abs + ord * ord);
    return Math.round(dist * 10000) / 10000.0;
    // Or: return Math.round(dist * Math.pow(10, 4)) / Math.pow(10, 4);
}
```

2. 6.5 points By using the previously described `RealPoint` class, it is desired to represent the buildings at the Vera campus of the UPV. For this mission, a datatype class called `Building` is to be defined; this class would keep physical data on the building (GPS coordinates, identity code in a plane), as well as the use assigned to the building (type of use and name of the entity that uses the building).

**You must**: implement the `Building` class with the following attributes and methods:

a) (0.5 points) Integer public class and constant attributtes:

- `DEPARTAMENT`, with value 0, that represents a building for a department
- `SCHOOL`, with value 1, that represents a building for teaching activities, such as classroom buildings, schools, or labs
- `SERVICES`, with value 2, that represents buildings with other activities, such as bars, administrative offices, etc.

These constants must be used whenever required (in the classes `Building` and `BuildingManager`

b) (0.5 points) Private instance attributes `code` (`String`), `entity` (`String`), `type` (`int`) and `coordinates` (`RealPoint`).

c) (1.5 points) Two constructor methods:

- A general constructor with all needed parameters (one of them of `RealPoint` type) in order to initialise all the instance attributes; you can suppose that all parameters have correct values
- A default constructor (without parameters) that creates a departamental building, used by the entity `DSIC`, with code `1F`, and coordinates `(39.4625, -0.3472)`

d) (0.5 points) A consultor (`get`) and a modifier (`set`) method for the attribute `coordinates`; you can suppose in the modifier that the parameter has a correct value

e) (1 point) An `equals` method, that overrides that of the `Object` class, that checks if two buildings are the same taking into account only the building data, not the use data; i.e., if they have the same code and coordinates. Notice that a building can be used by two different entities (e.g., building `1G` is used by `ETSINF` and `DISCA`)

f) (1 point) A `toString` method, that overrides that of the `Object` class, which, using a `switch` (**mandatory**), returns the `String` result with a format similar to that in the following examples (for the GPS coordinates you must employ the `toString` method of the `RealPoint` class):

```
Departamental building 1F (DSIC), GPS: (39.4625, -0.3472)
Departamental building 1G (DISCA), GPS: (39.4826, -0.3470)
Teaching building 1G (ETSINF), GPS: (39.4826, -0.3470)
Services building 3N (BBAA bar), GPS: (39.4841, -0.3443)
```

g) (1.5 points) A method `closestToRectorate` that, given a `Building` reference `e` as parameter:

- If current building `this` is closer to the rectorate coordinates `(39.4823, -0.3457)` than building `e`, returns `-1`
- If current building `this` is further to the rectorate than building `e`, returns `1`
- If the two distances are the same then:
  - If buildings are of different type, service buildings are supposed to be closer to rectorate than teaching buildings, and these last than departamental buildings; thus, `-1` or `1` must be returned accoding the case. For example, if building `this` is `DISCA` and building `e` is `ETSINF` (departamental and teaching buildings respectively, with the same distance to rectorate), the method must return `1` indicating that `ETSINF` is closer to rectorate than `DISCA`; in case that `this` is `ETSINF` and `e` is `DISCA`, it must return `-1`
  - If the two buildings have the same type, it must return `0`

**Solution:**

```
public class Building {
    public static final int DEPARTAMENT = 0;
    public static final int SCHOOL = 1;
    public static final int SERVICES = 2;

    private String code, entity;
    private int type;
    private RealPoint coordinates;
```

```
    public Building(String c, String e, int t, RealPoint p) {
        code = c;
        entity = e;
        type = t;
        coordinates = p;
    }

    public Building() {
        this("1F", "DSIC", DEPARTAMENT, new RealPoint(39.4625, -0.3472));
    }

    public RealPoint getCoordinates() { return coordinates; }

    public void setCoordinates(RealPoint p) { coordinates = p; }

    public boolean equals(Object o) {
        return o instanceof Building
            && code.equals(((Building) o).code)
            && coordinates.equals(((Building) o).coordinates);
    }

    public String toString() {
        String res = "";
        switch (type) {
            case DEPARTAMENT:
                res += "Departamental "; break;
            case SCHOOL:
                res += "Teaching "; break;
            case SERVICES:
                res += "Services "; break;
        }
        res += "building " + code + " (" + entity + "), GPS: " + coordinates;
        return res;
    }

    public int closestToRectorate(Building e) {
        RealPoint rectorate = new RealPoint(39.4823, -0.3457);
        double distThis = coordinates.distance(rectorate);
        double distE = e.coordinates.distance(rectorate);
        int result = 0;
        if (distThis < distE) { result = -1; }
        else if (distThis > distE) { result = 1; }
        else if (type < e.type) { result = 1; }
        else if (type > e.type) { result = -1; }
        return result;
    }
}
```

3. 2.5 points **You must**: implement the program class `BuildingManager` with a `main` method that executes the following actions:

   a) (1 point) Create an object `e1` of the `Building` datatype by using the general constructor, in order to represent a teaching building employed by `ETSINF` entity, with code `1G` and coordinates `(39.4826, -0.3470)`

   b) (0.5 points) Create an object `e1` of the `Building` datatype by using the default constructor

c) (1 point) Call the `closestToRectorate` in order to compare `e1` and `e2` and, after that, show on the screen which is the closest building to rectorate after writing `"The closest building to rectorate is "`, or, if they are at the same distance, the message `"Both bulding are at the same distance to rectorate"`

**Solution:**

```
public class BuildingManager {
    public static void main(String[] args) {
        Building e1 = new Building("1G", "ETSINF", Building.SCHOOL,
            new RealPoint(39.4826, -0.3470));
        Building e2 = new Building();

        int comp = e1.closestToRectorate(e2);
        String res = "The closest building to rectorate is ";
        if (comp == -1) { res += e1; }
        else if (comp == 1) { res += e2; }
        else { res = "Both bulding are at the same distance to rectorate"; }
        System.out.println(res);
    }
}
```