Practical Work 3
Radar view model

In this practical work, you will implement some of the elements of the data model of an air traffic controller's radar view, i.e. the information that will be used by the Graphical User Interface (*GUI*) to represent the graphical objects (aircraft, background map, control sectors...). Most of the elements presented here will be reusable in your project. **duration: 2h**

Objectives:

- Keep learning about *NetBeans* (adding existing source files, adding libraries, packages),
- Implement simple classes with UML specification provided,
- Use a third party *API* and data files,
- Use interfaces, collections, genericity.

Question 1: Project settings

➔ Launch NetBeans and create a *Java* project without a main class (uncheck the "*Create Application Class*" box), configured for *Java 1.8 Platform* (you will need *JavaFX* `Property` seen during the last tutorial).

➔ Create a package `fr.enac.sita.visuradar.model`.

➔ Download the interface `IPoint` from the e-campus space and save the file directly in the folder corresponding to the package (for example: *.../src/enac/iatsed/radarview/model*).

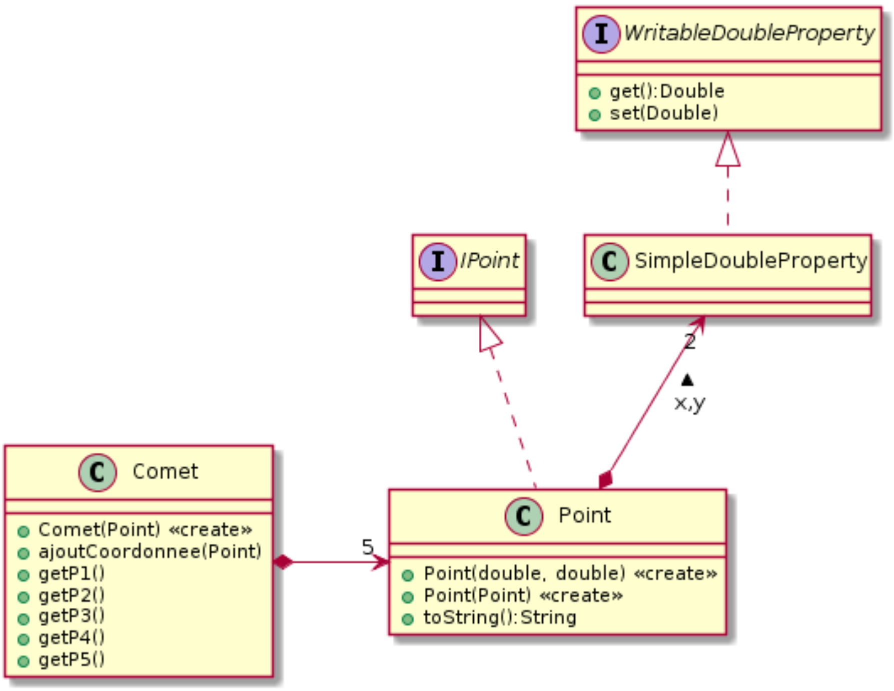
➔ There is a deliberate omission that causes an error. Using the suggestions in the IDE, make the correction so that this interface is well defined in the package where you were asked to place it.

Question 2: Aircraft comet

The comet of a traffic contains 5 points which represent its last 5 known positions. The class `SimpleDoubleProperty` allows to store (see tutorial 3) a value of type double and offers advanced binding and notification services (which will be used later for visualization with *JavaFX*).

➔ Write the class `Point` by carefully reading the following information:
The `DoubleProperty` type declared in the interface `IPoint` is a super-type of `SimpleDoubleProperty`.
For the second constructor you have to do some composition (the properties of the point passed in parameter must not be assigned to the properties of the point to be constructed, their value must be copied).

➔ Write the class `Comet`, knowing that the constructor of this class takes as parameter the coordinates of the initial position of the aircraft. Be careful to keep the composition property when implementing the method `addCoordinates()` (it'll be mandatory to allow the implementation of *JavaFX* bindings for your project).



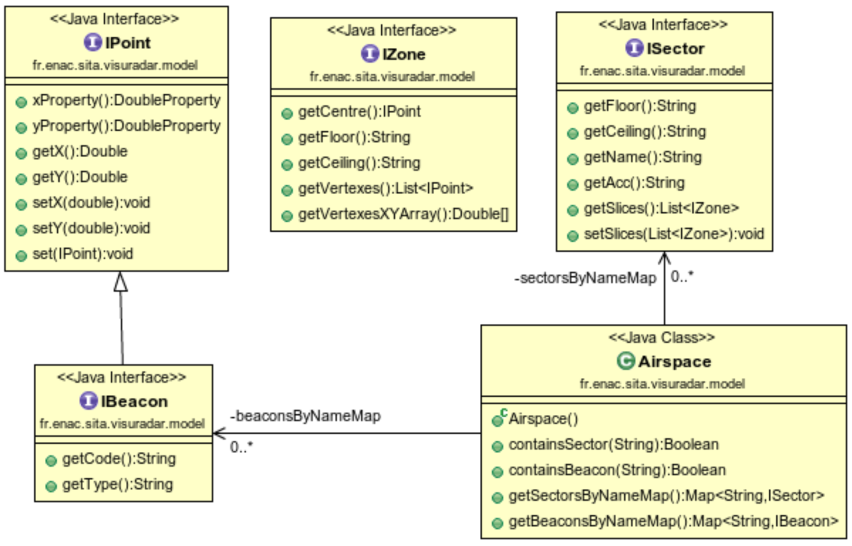
Question 3: Beacons, air traffic control zones and sectors

A beacon is a point to which we add a code (its name, in text format, for example "TOU" or "AMOLO"), and a type (which can have two values: "published" or "unpublished").

- ➔ Write the interface `IBeacon` which contains two methods that return the associated code and type respectively.
- ➔ Download the interfaces `IZone` and `ISector` from e-campus.

An airspace contains beacons and sectors, which can be stored in two associative arrays (the keys are the names associated with the beacons and sectors, the values are the objects, represented by their respective interfaces).

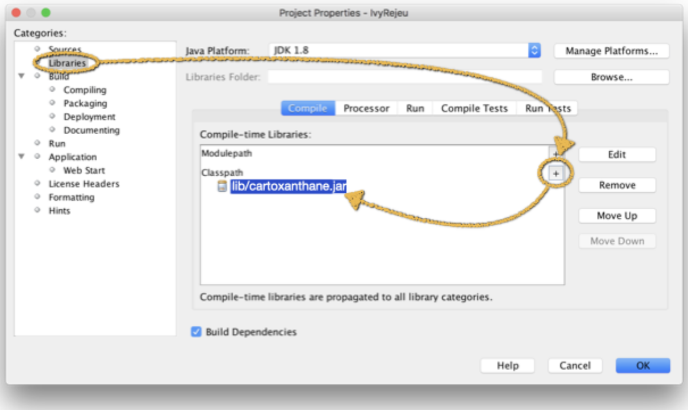
- ➔ Write the class `Airspace`. For now, you will provide a default constructor that initializes the attributes with an empty collection.
- ➔ Add a method that returns a list of beacons of type "published":
  `public List<IBeacon> getPublishedBeacons()`



Question 4: xml data reading system

The file "*cartoxanthane.jar*" available on e-campus is a library of compiled and compressed *Java* code (*JAR* format). It will allow you to read airspace data stored in xml files. To use it, you have to import it in your project:

- Add a folder named "*lib*" in your project folder and save the library in this folder,
- Right-click on the project name then select the item "*Properties*",
- In the left column select the category "*Libraries*",
- In the tab "*Compile*" press the button + at the right of "*classpath*", choose the option "*Add JAR/Folder*",
- Select the library *"cartoxanthane.jar"*. Make sure that the library is referenced with its relative path ("reference as relative path" option), which will not break your project if you move it later.


Project Properties - IvyRejeu

Categories: Sources, Libraries, Build (Compiling, Packaging, Deployment, Documenting), Run, Application (Web Start), License Headers, Formatting, Hints

Java Platform: JDK 1.8

Libraries Folder:

Compile | Processor | Run | Compile Tests | Run Tests

Compile-time Libraries:

Modulepath

Classpath

lib/cartoxanthane.jar

Compile-time libraries are propagated to all library categories.

☑ Build Dependencies

Adding local libraries to your project allows you to neatly group the libraries needed to run your project, which can then be exported and deployed more easily on other machines later. If for some reason you prefer to reuse the same library in several projects, it is better to store it elsewhere on your hard disk (outside a project) or to use a build tool like *Maven* or *Gradle*, which will retrieve the right version of the library from an internet repository.

Now let's go back to the library `cartoxanthane`:

The interface `ICartographyManager` available with this subject specifies methods to load french sectors, beacons and basemap definitions from description files, regardless of the cartography and the file formats. *Xanthane* is one of these formats used by french civil aviation authorities. The library `cartoxanthane` contains a class `CartographyManagerXanthane` which is an implementation of the interface `ICartographyManager` allowing to load sectors, beacons and basemap definitions from xml files thanks to a library called *JAXB* (integrated to standard *Java* until it's version 8). Let's retrieve them:

- ➔ Download the archive "*data.zip*" and unzip it. The archive gathers xml files containing parameters of the application (file "*param.xml*") and data of the French airspace (folder "*data_maps*"). Place the file "*param.xml" and the* folder "*data_maps*" at the root of your project. Also download the interface `ICartographyManager` and add it to your package `model`.
- ➔ Add a new constructor in the class `Airspace`, that allows to build the associative arrays from the list of beacons and sectors retrieved from an object that implements the interface `ICartographyManager`:
  `public Airspace(ICartographyManager cartographyManager)`

Question 5: Tests

The airspace description files contain lat/lon coordinates. As soon as the files are read, these coordinates are converted into a format that can be displayed directly on the screen. We will talk more about this format when you know how to display them in a *GUI*. For the moment you will test our data by simply displaying them in the console:

- ➔ Add a package named `test` (a normal package in your source packages, not in the test packages like during the last practical work with *JUnit*).
- ➔ Write a class that has a method `main()`, which builds your objects and displays the data. You may redefine some `toString()` methods to check the functioning of your objects.

By the way, you have noticed that up to now you have worked on a set of interfaces and have only implemented some of them. Yet your program works. This is due to the fact that the library `cartoxanthane` offers realizations of all these interfaces (classes `CartographyManagerXanthane`, `BeaconXanthane`, `SectorXanthane`…) that are compatible with both the specification of the corresponding interface (`ICartographyManager`, `IBeacon`, `ISector`…) and the structure of the airspace description files (*Xanthane* format). This way of doing things, common in Object Oriented Programming, allows you to easily change the data format without impacting your whole code.

For your project, we will spare you the laborious realization of these classes related to the data format, in order to allow you to concentrate on the more interesting aspects of the work remaining to be done, which we will approach progressively during the next sessions: communication with an air traffic simulator (*PW4*), programming of graphics and interaction (*Rich Graphical Interface Programming* course)…