Professional Development with Java

Hands-On Session #1

The goal of this hands-on session is to:

- Discover and practice the Eclipse IDE by using it to create a small set of classes.
- Discover and practice the reflection API.
- Discover and practice the Ant automation tool by creating some build files, playing with external tasks and developing new ones.

You're expected to send by e-mail, at last one week after the end of the session, your work. The subject of the e-mail must start with `[2F2-2013-TP1]`. Details on what you've done will be provided through simple text files (no Word, no LibreOffice, etc.) with a 100-character line length. Don't include binaries (`*.class` files).

Eclipse

The version of Eclipse to use for this session is Eclipse 4.2.

Install Eclipse by unzipping the corresponding package in a convenient place.

- Take a look at Eclipse's folder structure using a file-system browser: What is Eclipse made of? What are the underlying concepts?

Start Eclipse by running the `eclipse` file. Place the workspace in your personal folder.

- What is the name of the default perspective?
- What other perspectives are available? What is the purpose of each one?
- Name the default views of the default perspective.
- Take a look at your workspace's folder structure using a file-system browser: To which Eclipse concept the `.metadata` folder can be related to? Inspect the files it contains.

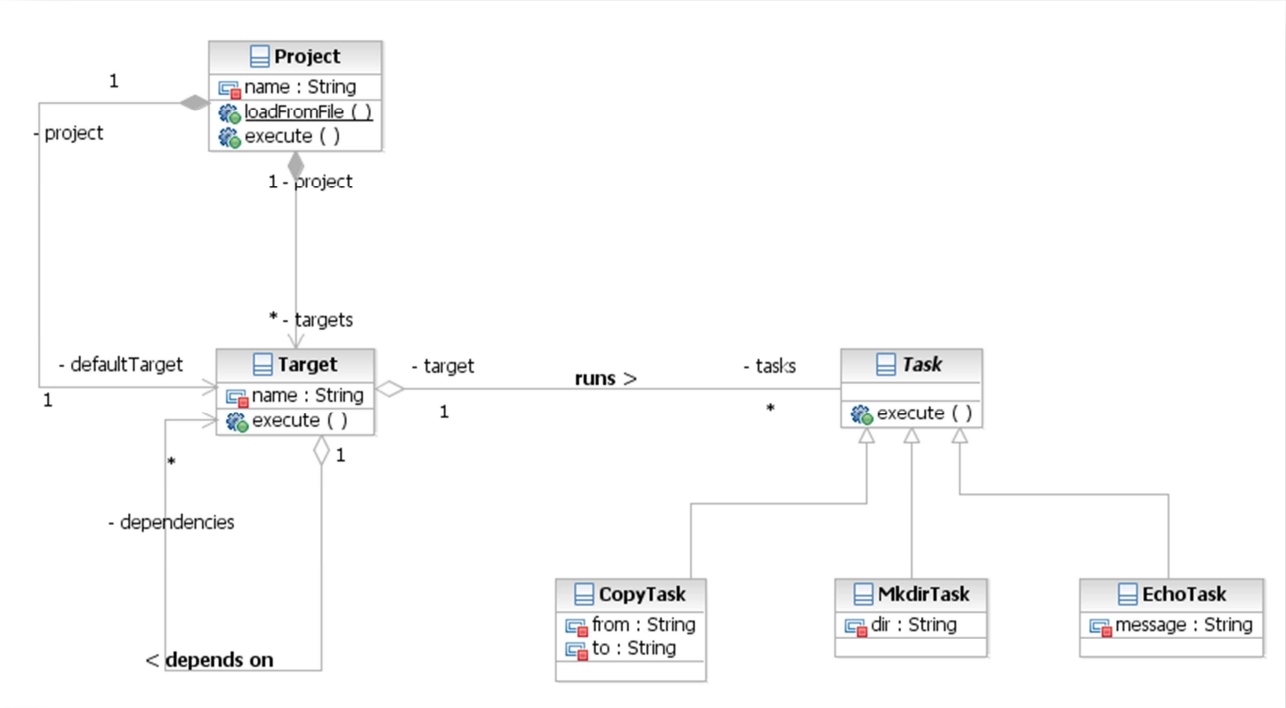
Set the following preferences:

- *Java compiler compliance level* to *1.6*
  NB: Although this preference can be set at the workspace level, new projects may override this setting at the project level. So, when creating a new project, don't forget to tweak its preferences/properties.
- *Displayed tab width* to *4*
- Where are these preferences actually stored?
- Carefully take a look at the other existing preferences and try to identify the most important ones for everyday development.

Create a new Java project called `MyAnt`, keeping all the default values of the wizard.

- Try to understand the various options that are available, most of them are **very** important.
- What other types of projects is the Java Developers edition of Eclipse able to handle?
- Take a look at the project's folder structure using a file-system browser: Does this folder structure corresponds to what's displayed in the *Package Explorer* view? Why?
- Open the *Navigator* view: What does this view display?
- Take a look at the `.project` and `.classpath` files: Given their name and their content, what is each of these files about? To which Java notion is the `.classpath` file related to? Note that these files are **critical**.

Implement the following classes twice: First by hand-coding everything, then by using Eclipse's wizards as much as possible.



- The package has to be `fr.isima.myant`.
- For each attribute of each class, define the corresponding getter and setter.
- Take time to discover the many possibilities offered by the tool to work with the source code, go to the declaration of elements (Ctrl + click), display the hierarchy of the classes, etc.
- Note the time saved by using Eclipse's wizards.

Play with Eclipse's shortcuts and built-in features to rapidly navigate among resources (classes, files, etc.). Particularly, try the Ctrl + Shift + L shortcut to get a list of most of the other Eclipse's shortcuts.

- What is the shortcut used to rapidly open a *resource*?
- What is the shortcut used to rapidly open a *type*?
- What is the shortcut used to build the workspace?
- What is the goal of Ctrl + Shift + o?
- What other shortcuts seem promising for your everyday use?

Add a new class to the project with a `main()` method. This method will instantiate a new `Project`, which will have just one `Target`, which itself will just run the `EchoTask`.

- Build this application (although this is normally automatically done for you).
- Take a look at the project's folder structure: Where are the `*.class` files located?
- Run the application from within Eclipse. What are run configurations and what can be their use? How are they materialized one the file-system?

Import the `BuggyStuff` project.

- Why isn’t the project being successfully compiled? Fix it.
- Try to run the application: It will fail. To guess why, switch to the *Debug* perspective to debug it and try to pinpoint the various issues. Try to discover as much as possible Eclipse's

debug features: Breakpoints, watch expressions, the inspector, the possibility to change attributes in-memory, the step into/step over/step return actions, etc.

Reflection API

The goal is now to enhance the `MyAnt` project so that is becomes pluggable, that is, to be able to use tasks that have been defined externally. Implement it as follow:

- Projects will be defined using a text file which must look like this:

```
# The imports section lists the external tasks to use
# The external tasks must be in the classpath, but NOT in the build path
use fr.isima.myant.MyNewTask1 as mynewtask1
use fr.isima.myant.MyNewTask2 as mynewtask2

# For the moment, the default target acts as the entry point of your project
# No other targets will be defined
default:
echo[message:"Starting..."]
mynewtask1[att1:"value1", att2:"value2"]
mynewtask2[att1:"blabla", att2:"blabla", att3:"blabla"]
echo[message:"Completed!"]
```

- A project will be executed by running the following command: `java -jar MyAnt.jar ~/mybuild.txt`
- You're not here to code a lexer/parser (for that you would use ANTLR by the way), so consider that the files won't have typos, indentation, etc. Use a `BufferedReader` in front of a `FileReader` to read the files. Use the `String.split()` method recursively to parse each line.
- Use the `Class.forName()` method to dynamically load classes (for example to dynamically load `fr.isima.myant.MyNewTask1`).
- Use the reflection API to set the parameters of the invoked tasks.
- The Javadoc is available here: http://docs.oracle.com/javase/6/docs/api/index.html.

Ant

The version of Ant to use for this session is not the one bundled with Eclipse, but the plain 1.8.2 one. Anyway, you may use Eclipse's features to write the build files, build your own tasks, etc. To complete the exercises below, you must make an intensive use of the Ant manual, which you should consider as one of your best friends (along with the person sitting on your left and/or your right). The Ant manual is bundled with Ant.

Install Ant in a convenient place:

- Unzip/untar the Ant archive that was provided to you in a convenient place.
- Add Ant's `bin` folder to the `PATH` environment variable.
- It's not necessary to define the `ANT_HOME` environment variable.

Simple exercises:

- Write simple build files which emphasize the behavior of Ant: One will feature target dependencies (name it `build-targetdep.xml`), another one will intend to introduce circular dependencies (`build-circulardep.xml`), another one will make use of properties to emphasize their immutable character (`build-immut.xml`), and a last one will feature task overriding (take a look at the `<import>` element for that – `build-override.xml`).

Create an Ant build file following these guidelines:

- The build file (`build-myant.xml`) will be used to (1) clean, (2) compile, (3) package and (4) run the MyAnt application you've just build (one target per enumerated action).
- The following will be defined as properties: The Java source code version, the place to get the source code from, the place to store the JAR file, the name of this JAR file, and every other attribute that can help to make the build file as generic as possible. These properties will be defined using properties files.

Enhance the previously defined Ant build file as follow:

- Add an `init` target which will be used to ensure that every required properties are defined before other targets are actually run and which, if it's not the case, assumes default values.
- Add a target which allows working with several projects at the same time.
  You may need a set of external tasks that were presented several times during the course.

Write your own Ant task which will, in just one step, compile, package and run a Java application based on the following indications:

- Before coding the Ant task, clearly define the XML structure of the task: Define the mandatory parameters to be used, the optional ones, etc.
- Then, code the task and validate it by using it in a new build file.