Professional Development with Java

Hands-On Session #1

The goal of this 3-hour 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 Ant automation tool by creating some build files, playing with external tasks and developing new ones.
- Clearly emphasize that the first source of information for software developers is Internet's search engines.

Eclipse

The version of Eclipse to use for this session is Eclipse Helios (Eclipse 3.6.1).

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

- Take a look at Eclipse's folder structure using a filesystem browser: What is Eclipse made of? What are the concepts and technologies behind this?

Start Eclipse by running the `eclipse.exe` file (on Windows) or `eclipse` (on Linux). Place the workspace in your personal folder.

- What is the name of the default perspective?
- What other perspectives are available?
- Can you name the default views of the default perspective?
- Take a look at your workspace's folder structure using a filesystem browser: To which Eclipse concept the `.metadata` folder can be related to?

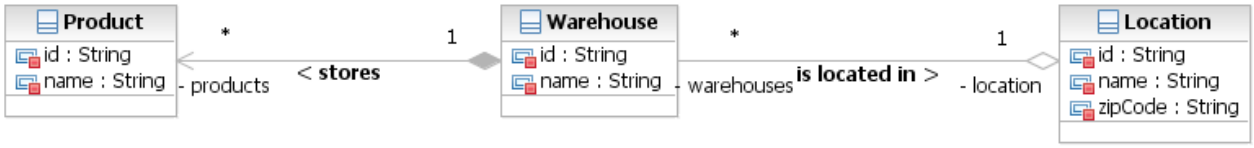
Set the following preferences:

- *Java compiler compliance level* to *1.4*
- *Displayed tab width* to *4*
- Where are these preferences actually stored?
- 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 WarehouseProject, keeping all the default values of the wizard.

- Try to understand the various options that are available, most of them are **very** important.
- Which other kinds of project is the Java Developers edition of Eclipse able to handle?
- Take a look at the project's folder structure using a filesystem 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.



- 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, 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 (classse, 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 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 `Location`, then a new `Warehouse` and, finally, a new `Product`.

- 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? Does the default Eclipse Java projects structure maps to the basic Java application structure?
- Run the application from within Eclipse. What are run configurations and what can be their use?

Import the BuggyWarehouseProject.

- Why does the project is not 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 issue (which is fairly basic). 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.

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. The exercises below are voluntarily poorly describe so that you make a 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).

Install Ant in a convenient place.

Simple exercises:

- Write simple build files which emphasize the behavior of Ant: One will feature target dependencies, another one will intend to introduce circular dependencies, another one will make use of properties to emphasize their immutable character, and a last one will feature task overriding (take a look at the `<import>` element for that).
  Don't hesitate to ask the teacher for more information on what has to be done at this stage.

Create an Ant build file following these guidelines:

- The build file will be used to (1) clean, (2) compile, (3) package and (4) run the applications you've played with in the first part of this hands-on session (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.

Create an 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.
- Once done, ask the teacher for validation.
- Then, code the task and validate it by using it in a new build file.