Assignment #2 : Maven - Composite projects.

A maven project can be composed by multiple sub-projects, called "modules". Modules can be nested, creating a tree-based structure that follows the principles of the composite pattern (http://en.wikipedia.org/wiki/Composite_pattern). Modules are used to better organize an application in sub-components that can be redistributed independently. At the same time, configuration inheritance allows children POM-s to inherit part of the configuration from their ancestors (we will NOT use this feature explicitly in this exercise). When a parent POM is built, all descendants are also built, following an order that preserves the mutual dependencies (notice that circular dependencies result in a compilation error).

So, one could have the following structure:

```
root
        node
                leaf1
                leaf2
        leaf3
```

root and *node* are folders that contain a pom.xml file each. The POM file will contain a local, unique G.A.V., but its "packaging" will be set to "pom" rather than "jar". Moreover, the POM will contain a "modules" element, with one sub-element for each child. The value of each module element is the name of the child's folder, which needs not be the same as the child's artifactId, although it is very often the case. Leaf modules, instead, are regular maven projects – that is, they contain a regular pom.xml and the *src*/... and *target* folders.
Finally, each module except for the root contains a reference to its immediate parent.

See also Chapter 6 of the recommended reading on Maven (http://books.sonatype.com/mvnex-book/reference/public-book.html)

So, in our example:

root/pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.foo</groupId>
  <artifactId>root</artifactId>
  <packaging>pom</packaging>  <!-- container -->
  <version>0.1</version>

  <modules>    <!-- children -->
    <module>node</module>
    <module>leaf3</module>
  </modules>
</project>
```

root/node/pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.foo</groupId>
    <artifactId>parent</artifactId>
    <version>0.1</version>
  </parent>

  <groupId>org.foo</groupId>
  <artifactId>node</artifactId>
  <version>0.1</version>
  <packaging>pom</packaging>

  <modules>
    <module>leaf1</module>
    <module>leaf2</module>
  </modules>
</project>
```

root/node/leaf1/pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.foo</groupId>
    <artifactId>node</artifactId>
    <version>0.1</version>
  </parent>

  <groupId>org.foo</groupId>
  <artifactId>leaf1</artifactId>
  <version>0.1</version>
  <packaging>jar</packaging>
</project>
```

The exercise consists of the following: create a modular maven project with two sub-modules, one implementing a “data model” and one implementing the “application” itself. The separation of the data model from the actual logic is beneficial and makes the data model potentially reusable. However, the application uses the data model, so an explicit dependency must be stated.

In particular, read and follow the following steps/requirements carefully:

1. Create a parent maven project.
    1. Use “edu.asu.bmi.cda” as groupId
    2. Use your last name + “-1” as artifactId. For example, my artifactId would be “sottara-1”
    3. Use 1.0-SNAPSHOT as version
    4. Remember to configure the packaging and the sub-modules appropriately
2. Create a first sub-module
    1. Use “datamodel” as artifactId. GroupId and version same as parent
    2. Link the sub-model to the parent
    3. Create a Patient class under the package edu.asu.bmi.cda.data with three fields (“id”, “name” and “age”). Add methods (constructors, getters/setters, etc...) as you see fit.
3. Create a second sub-module
    1. Use “example” as artifactId. GroupId and version same as parent
    2. Link the sub-model to the parent
    3. Add the first sub-module as a dependency
    4. Create a “main” application in the edu.asu.bmi.cda.example package. The application is a Java class that imports the edu.asu.bmi.cda.data.Patient class and has a main method that instantiates a Patient and prints it to the console.
        1. The Patient's name must be the same as the student's
        2. Notice that, should you remove the dependency, the module would not compile (depending on the IDE, you may have to update your project)

Build the parent application invoking “mvn clean install” from the parent directory. You should see a “BUILD SUCCESSFUL” message showing that the 3 modules have been built.

Possible points : 100

Instructions : Submit a zip file containing the root folder and all its subfolders. The assignment is individual, but collaboration is not prohibited.