Softwarekonstruktion - Exercise 9

9 OSGi

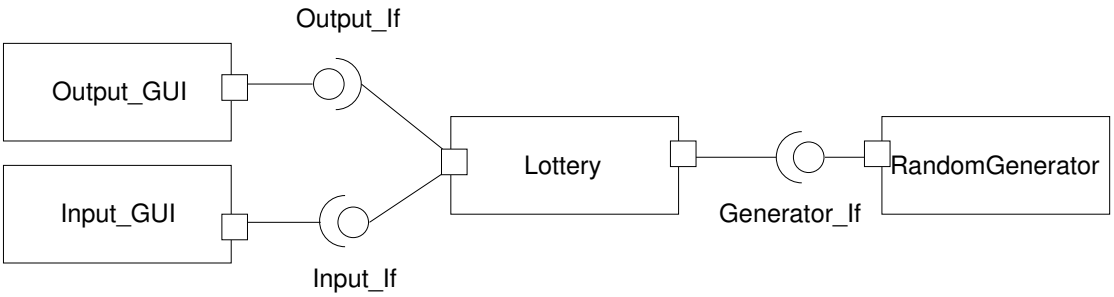
This exercise should be solved until **Wednesday (23:59 latest), January 5th, 2011**. You have to submit your solution to your tutor by email:

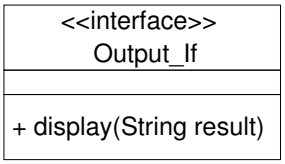
Holger Schmidt: holger.schmidt [at] cs.tu-dortmund.de
Gregor Kotainy: gregor.kotainy [at] tu-dortmund.de

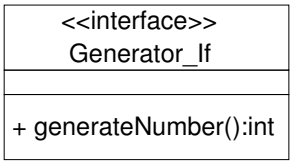
You have to work in groups of two or three persons. Only one person per group has to submit a group's solution. State the names and matriculation numbers of the group members in your email and as a comment in each of your source code files.

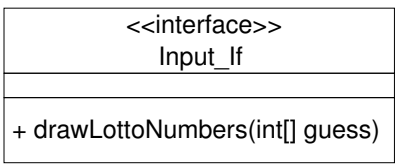
9.1 Implementation

Implement an OSGi application realising the slightly adjusted lottery simulation:









You can use the simple Java components implementation of the lottery simulation as a basis for your implementation: `http://inky.cs.tu-dortmund.de/main2/jj/teaching/ws10/swk/uebungszettel/lottery.zip`.

1. Create the following bundles in Eclipse:

OutputGUIBundle
- META–INF
  - MANIFEST.MF
- outputguibundle
  - Activator.java
- outputguibundle.implementation
  - OutputGUI.java
- outputguibundle.interfaces
  - Output_If.java

InputGUIBundle
- META–INF
  - MANIFEST.MF
- inputguibundle
  - Activator.java
  - InputGUI.java

LotteryBundle
- META–INF
  - MANIFEST.MF
- lotterybundle
  - Activator.java
- lotterybundle.implementation
  - Lottery.java
- lotterybundle.interfaces
  - Input_If.java

GeneratorBundle
- META–INF
  - MANIFEST.MF
- generatorbundle
  - Activator.java
- generatorbundle.implementation
  - RandomGenerator.java
- generatorbundle.interfaces
  - Generator_If.java

To create a bundle in Eclipse, you have to create a new *Plug-in Project*:
File → *New* → *Project ...* → *Plug-in Development* → *Plug-in Project*
As *Target Platform* choose *an OSGi framework: standard*. The packages of a bundle are stored in `src`.

2. For each bundle, add information about provided and required interfaces into the `MANIFEST.MF`:
   - provided interface: *Runtime* → *Exported Packages* → *Add ...* → choose the package that contains the interface
   - required interface: *Dependencies* → *Imported Packages* → *Add ...* → choose the package that contains the interface

3. Every bundle that provides an interface should register its implementation of the provided interface at the *Service Registry* as soon as the bundle is started. Implement the `Activator` classes accordingly. Pass the `BundleContext` to the implementations via their constructor.

4. Instead of connecting the components via their `connectTo`-methods, a component that wants to use a method of a required interface has to request the appropriate service from the *Service Registry* to use it. Do not forget to release the service after using it. `InputGUI` should implement a method `startLottery()` that asks the user for its guess and starts the drawing of the lotto numbers afterwards. This method should be called when the bundle `InputGUIBundle` is started.

5. To deploy the bundles as JARs the export function of Eclipse can be used:
   File → *Export ...* → *Plug-in Development* → *Deployable plug-ins and fragements*
   As *Destination Directory* choose `osgi`.

6. Create a new *Run Configuration*:
   Run → *Run Configurations ...* → *OSGi Framework* → *New*
   Deselect all bundles except the bundle `org.eclipse.osgi`. Select the tab *Arguments* and insert `-console -clean` to *Program arguments*.

7. By means of this new run configuration the OSGi framework can be started. The lifecycle of bundles can be managed by a *Management Agent* called *Equinox Console*. Use the following comands:

   - `ss`: prints a list of currently installed bundles along with their state
   - `install file:/osgi/plugins/`*name-of-the-bundle*:
     installs the specified bundle
   - `start` *id*: starts the specified bundle

   Keep in mind that that the bundles cannot be started in any order because of their dependencies.