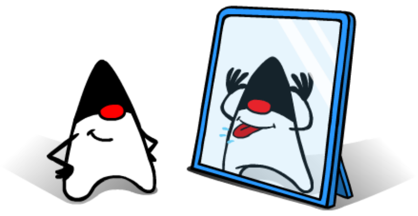
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

B.Sc. Software Development – Advanced Object-Oriented Design Principles & Patterns (2016)

ASSIGNMENT DESCRIPTION & SCHEDULE

Measuring Stability Using the Reflection API



Note: This assignment will constitute 50% of the total marks for this module.

Overview

One of the most powerful features of the Java language is the ability to dynamically inspect types at run-time. An application, through the various forms of inheritance and composition forms an **object graph** inside a JVM, the structure of which is a realisation of a UML class diagram. Specifically, the object graph is a digraph (directed graph) where the edges correspond to the dependencies between classes. As a graph data structure, the standard graph algorithms, such as depth / breadth first search, topological sorting and layout algorithms can all be applied to an object graph if necessary.

An object graph can be dynamically inspected using the *Java Reflection API*, which is available from the package *java.util.reflect*. The Reflection API allows a class to be dynamically queried about it structure, including its class signature, constructors, instance variables, class fields, methods and return types. In addition, the parameters to constructors and methods can also be determined. This ***metadata*** provides a full description of all the classes and objects within the JVM and provides a mechanism for the analysis of the structure of an application and the degree to which the principles of cohesion and coupling have been upheld.

Basic Requirements

You are required to create a Java application that uses reflection to analyse an arbitrary Java Application Archive (JAR) and calculates the positional stability of each of the component classes in its object graph. Recall that the **Positional Stability** *(I)* of a type can be measured by counting the number of dependencies that enter and leave that type:

$$I = \frac{Ce}{Ca + Ce}$$

- **Afferent Couplings (Ca):** the number of edges incident on a type, i.e. the number of types with a dependency on another type.

- **Efferent Couplings (Ce):** the number of edges emanating from a type, i.e. the number of types that a given type is dependent upon.

The positional stability (I) is a metric in the range [0..1], with zero indicating a maximally stable class, i.e. a class that can be depended on by many other classes. Although difficult to change, such a type is both responsible, independent and ideally either an abstract class or an interface. A positional stability of one implies a maximally instable class that is both irresponsible and Dependent.

Your application should process a JAR archive provided by a user, compute the positional stability of each class and present the results in a tabular structure, i.e. a Swing GUI with a JTable or something similar. A stub Swing application is available on Moodle to help you get started. Your application should ***ignore any standard classes*** belonging to the basic Java SDK / JRE libraries. Note that 30% of the marks for the assignment are reserved for additional features.

You are also required to provide a ***UML diagram*** of your design and to ***JavaDoc*** your code.

You should consider adding some or all of the following to your application ***after you have met the basic requirements***:

- Applying ***design patterns*** to your application where apposite, either verbatim or as cocktails and composites.
- Colouring GUI components as the user interacts with them, i.e. push changes to the view from the model (the object graph).
- Create your own customised GUI component (implement your own control, e.g. a stability meter).
- Visualise the object graph using a layout algorithm, e.g. force-directed layout.
- Implement other appropriate metrics, such coupling, cohesion and encapsulation measures.
- Include an OODB (db4o). Do not include any other DB type or marks will be deducted…

Note that the whole point of this assignment is for you to demonstrate an understanding of the principles of object-oriented design by using abstraction, encapsulation, composition, inheritance and polymorphism WELL throughout the application. Please pay particular attention to how your application must be packaged and submitted. Marks will be deducted if you deviate (even slightly…) from the requirements. Finally, as 4th year software students, you should appreciate that, if your code does not compile you cannot pass the assignment…

Deployment and Submission

- ***The project must be submitted by midnight on Tuesday 10th January 2017.*** The project must be submitted as a Zip archive ***(not a rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the "Measuring Stability Using the Reflection API (50%) Assignment Upload" heading on Moodle.
- The name of the Zip archive should be <*id*>.zip where <*id*> is your GMIT student number.
- The Zip archive should have the following structure (do NOT submit the assignment as an Eclipse project):

Marks	Category
src	A directory that contains the packaged source code for your application.

README.txt	A text file outlining the main features of your application, with a set of instructions for running the programme. The URL of the GitHub repo for the project should also be provided.
design.png	A UML diagram of your API design. Your UML diagram should only show the relationships between the key classes in your design. Do not show methods or variables in your class diagram.
docs	A directory containing the JavaDocs for your application.

Marking Scheme

Marks for the project will be applied using the following criteria:

Marks	**Category**
(30%)	Robustness
(10%)	Cohesion
(10%)	Coupling
(10%)	JavaDocs and UML Diagram
(10%)	Packaging & Distribution (GitHub and Moodle)
(30%)	Documented (and relevant) extras.

You should treat this assignment as a project specification. Any deviation from the requirements will result in a loss of marks. Each of the categories above will be scored using the following criteria:

• 0–30%	Not delivering on basic expectation
• 40-50%	Meeting basic expectation
• 60–70%	Tending to exceed expectation
• 80-90%	Exceeding expectations
• 90-100%	Exemplary

Dynamic Class Introspection

The contents of a Java Application Archive can be read as follows using instances of the classes *java.util.jar.JarInputStream* and *java.util.jar.JarEntry* :

```
JarInputStream in  = new JarInputStream(new FileInputStream(new File("mylib.jar")));
JarEntry next = in.getNextJarEntry();
while (next != null) {
  if (next.getName().endsWith(".class")) {
    String name = next.getName().replaceAll("/", "\\.");
    name = name.replaceAll(".class", "");
    if (!name.contains("$")) name.substring(0, name.length() - ".class".length());
    System.out.println(name);
  }
  next = in.getNextJarEntry();
}
```

Classes can be dynamically loaded (by name) through invoking the Java class loader. Note that if the class is not found, i.e. not on the CLASSPATH, a *ClassNotFoundException* will be thrown:

```
Class cls = Class.forName("ie.gmit.sw.Stuff");
```

Once an initial class is loaded (of type ***Class***), every other class in an application can be processed in a similar way as an object graph is a ***recursive*** structure. Consider the following types of processing that can be performed on a ***Class*** (an example is available on Moodle):

```
Package pack = cls.getPackage(); //Get the package
boolean iface = cls.isInterface(); //Is it an interface?
```

```
Class[] interfaces = cls.getInterfaces(); //Get the set of interface it implements
Constructor[] cons = cls.getConstructors(); //Get the set of constructors
Class[] params = cons[i].getParameterTypes(); //Get the parameters
Field[] fields = cls.getFields(); //Get the fields / attributes
Method[] methods = cls.getMethods(); //Get the set of methods
Class c =  methods[i].getReturnType(); //Get a method return type
Class[] params = methods[i].getParameterTypes(); //Get method parameters
```