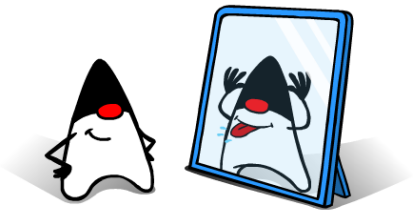
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

B.Sc. Software Development – Advanced Object-Oriented Design Principles & Patterns (2020)
ASSIGNMENT DESCRIPTION & SCHEDULE

Measuring Software Design Quality 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 its 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 one or more design and structural metrics for each of the component classes in its object graph. You are free to use any number of or combination of metrics, including LOCs, SLOC, CP, cyclomatic complexity, positional stability, CK metrics and MOOD metrics.

- Your application should process a JAR archive provided by a user, compute one or more metrics for each class and present the results in a structured format, i.e. a JavaFX GUI or a command line console user-interface.

- The results should be stored in an instance of **MicroStream DB** and have one or more stream-based queries for displaying results (a single query will suffice).
- You are also required to provide a ***UML diagram*** of your design and to ***JavaDoc*** your code.
- ***Do not use modules*** as this will complicate the reflection process with additional responsibilities for handling access rights and dealing with legacy packages.

A stub JavaFX application is available on Moodle (the finished source code from the lab *Building a GUI with Model-View-Controller*) to help you get started, but you are free to discard this entirely and use any other client that you like, including a command line UI or a bootstrapped ***Runner*** class with parameters. Your application should ***ignore any standard classes*** belonging to the basic Java SDK / JRE libraries.

Your objective is to implement the required features and then **extend**, **modify** and **refactor** your code to create an elegant application design. You should aim to accomplish the following in your application:

- Apply the **SOLID** principles throughout your application.
- Group together **cohesive** elements into methods, types and packages.
- **Loosely-couple** together all programme elements to the maximum extent possible.
- Create **a reusable set of abstractions** that are defined by interfaces and base classes.
- **Encapsulate** at method, type and package level.
- Apply creational, structural and behavioural **design patterns** throughout the application where appropriate.

You are free to asset-strip any online resources for text, images and functionality provided that you modify any code used and cite the source both in the README and inline as a code comment above the relevant section of the programme. You are not free to re-use whole components and will only be given marks for work that you have undertaken yourself. Please pay particular attention to how your application must be packaged and submitted and the scoring rubric provided. Marks will only be awarded for features described in the scoring rubric.

Deployment and Delivery

- ***The project must be submitted by midnight on Friday 8th January 2021***. Before submitting the assignment, you should review and test it from a command prompt on ***a different computer*** to the one that you used to program the assignment.
- The project must be submitted as a Zip archive ***(not a 7z, rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the "*Measuring Software Design Quality Using the Reflection API (50%) Assignment Upload*" heading of Moodle. Only the contents of the submitted Zip will be considered. ***Do not add comments to the Moodle assignment upload form.***
- The name of the Zip archive should be <*id*>.zip where <*id*> is your GMIT student number.
- You must use the **package name ie.gmit.sw** for the assignment. **Do not use modules**.
- The Zip archive should have the following structure (do NOT submit the assignment as an Eclipse project):

Component	Category
metrics.jar	A Java archive containing your API and runner class with a ***main()*** method. You can create the JAR file using the following command from inside the "bin" folder of the Eclipse project: **jar –cf metrics.jar *** The application should be executable from a command line as follows: **java –cp ./metrics.jar ie.gmit.sw.Runner**
src	A directory that contains the packaged **source code** for your application.

README.pdf	A **PDF** file detailing the main features of your application in **no more than 300 words**. All **features and their design rationale** must be fully documented. You should consider also including the UML class diagram in this document to help explain your design clearly.
design.png	A **UML class diagram** of your API design. The UML diagram should only show the relationships between the key classes in your design. Do not show private methods or attributes in your class diagram. You can create high quality UML diagrams online at **www.draw.io**.
docs	A directory containing the **JavaDocs** for your application. You can generate JavaDocs with the following command from inside the "*src*" folder of the Eclipse project: javadoc -d [path to javadoc destination directory] ie.gmit.sw Make sure that you read the **JavaDoc** tutorial provided on Moodle and comment your source code correctly using the **JavaDoc** standard.

Marking Scheme

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

Element	Marks	Description
Structure	5	All-or-nothing. The packaging and deployment are both correct. The application launches correctly without any manual intervention from a command line with the command: ***java–cp ./metrics.jar ie.gmit.sw.Runner***
README	10	All **features and their design rationale** are fully documented in 300 words or less. The README should clearly document where and why any design patterns have been used.
UML	10	A UML class diagram that correctly shows all the **important structures and relationships** between types in your design.
JavaDocs	10	All classes are fully commented using the **JavaDoc** standard and generated docs are available in the *docs* directory.
Robustness	25	The application reads in a JAR file, outputs one or more metrics, saves the metrics to an OODB and permits at least one stream based query.
Cohesion	10	There is very high **cohesion** between packages, types and methods.
Coupling	10	The API design promotes **loose coupling** at every level.
Extras	20	Only **relevant extras** that have been fully documented in the README.

You should treat this assignment as a project specification. Any deviation from the requirements will result in some or all marks not being earned for a category. Each of the categories above will be scored using the following criteria:

Range	Expectation
0–39%	Not delivering on basic expectations. Programme does not meet the minimum requirements.
40-60%	Meets basic expectations.
61–70%	Tending to exceed expectations. Additional functionality added.
80-90%	Exceeding expectations. Any mark over 70% requires evidence of independent learning and cross-fertilization of ideas, i.e. your project must implement a set of features using advanced concepts not covered in depth during the module.
90-100%	Exemplary. Your assignment can be used as a teaching aid with little or no editing.

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***:

```
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]
```