Lab 12

[valid 2020-2021]

Reflection
Create an application to analyze and test java classes.
The application will receive as input java classes and it will display their prototypes and perform the tests specified by the @Test annotation.

The main specifications of the application are:

Compulsory (1p)

- The input will be a .class file, located anywhere in the file system.
- Load the specified class in memory, identifying dynamically its package.
- Using reflection, extract as many information about the class (at least its methods).
- Using reflection, invoke the static methods, with no arguments, annotated with @Test.

Optional (2p)

- The input may be a folder (containing .class files) or a .jar. You must explore it recursively.
- Create the complete prototype, in the same manner as javap tool.
- Identify all public classes annotated with @Test and invoke the methods annotated with @Test, whether static or not.
  If a method requires primitive (at least int) or String arguments, generate mock values for them.
- Print a statistics regarding the tests.

Bonus (2p)

- Consider the case when the input files are .java files and compile the source code before analyzing them. (use Java Compiler, for example).

- Using additional annotations, implement non-functional tests over the methods in order to test their *reliability* and *efficiency*.
- Use a bytecode manipulation and analysis framework, such as ASM, BCEL or Javassist in order to extract the bytecode of the class, perform bytecode instrumentation (inject code in some method) and generate dynamically a class.

Resources

- Packages
- Java Class Loading: The Basics
- Understanding Extension Class Loading
- The Reflection API
- Annotations
- JavaBeans

Objectives

- Understand Java class loading mechanism.
- Learn how to set the CLASSPATH and how to use the system class loader.
- Load classes dynamically.
- Instantiate objects of a class whose name is known only at runtime.
- Use *Reflection API* to inspect or use types at runtime.
- Understand the role of *annotations* in the context of modern programming techniques.