CPSC 501: Advanced Programming Techniques

Assignment 2: Reflective Object Inspector

The goal of this assignment is to create a **reflective object inspector** that does a complete introspection of an object at runtime. The inspector will be implemented in a **Java** class called **Inspector**, and will be invoked using the method:

public void inspect(Object obj, boolean recursive)

This method will introspect on the object specified by the first parameter, printing what it finds to standard output. You should find the following information about the object:

1) The **name** of the declaring **class**
2) The **name** of the immediate **super-class**
    a) **Always** explore the super-class immediately and recursively (regardless of recursive parameter)
3) The **name** of each **interface** the class implements
    a) **Always** explore the super-class immediately and recursively (regardless of recursive parameter)
4) The **constructors** the class declares. For each, also find the following:
    a) The **name**
    b) The **parameter types**
    c) The **modifiers**
5) The **methods** the class declares. For each, also find the following:
    a) The **name**
    b) The **exceptions** thrown
    c) The **parameter types**
    d) The **return type**
    e) The **modifiers**
6) The **fields** the class declares. For each, also find the following:
    a) The **name**
    b) The **type**
    c) The **modifiers**
    d) The **current value** of each **field**.
        i) **If the field is an object reference, and *recursive* is set to false**, then simply print out the "reference value" directly (this will be the name of the object's class plus the object's "identity hash code" ex. java.lang.Object@7d4991ad).

You must always traverse the inheritance hierarchy to find all the same information about each super-class(es) and interface(super-interfaces) declared. (getDeclaredMethods/ getDeclaredFields/ etc. will only give you a list of fields directly declared immediately in a class). **(This is separate from the recursive argument.)** You should progress all the way up the hierarchy. The Object class (Object.class) will return Object.class as its super-class. You can back out of the hierarchy at this point. You will notice you will end up visiting certain classes multiple times and this is expected (Object is the hierarchical super-class of every regular class).

Be sure you can also handle any array you might encounter, printing out its name, component type, length, and all its contents.

There will be certain objects this method will end up in infinite recursion on if the recursive method argument is enabled. You do not have design your code to escape circular class references. The driver program for your assignment will not rely on an example with this circular class reference type behaviour.

Recursive Inspection (recursive = true)
Recursive introspection is an optional method argument that can be enabled for fields introspection.

1) If the inspect method is invoked with *recursive* set to false, then simply find information for the object specified.
2) If it is true, then recursively inspect each field/array object in the same manner of the original object.

Other Requirements

1) You should have descriptive output. For example, you should not use the toString() for Field/Method or Class to get info. You will need to use the provided API methods to pull out the information in the Field/Method/Class/etc. and print more descriptive explanatory lines.
2) When printing modifiers you must convert the returned integer information into descriptive information such as public/private/final/static/transient/etc. for the modifiers.
3) For your submission a driver program will be provided that creates objects to inspect, and then invokes your *inspect* method. This driver will output eight different script<no>.txt files, one for each object.
4) During the marking process, the TA will compile and run your code to verify that everything works.
5) You must also use version control, unit testing, and refactoring throughout this assignment.

Formatting
Please indent each class recursed into one tab (\t) of depth and indicate whenever you enter a new class. It is also helpful to indicate which class you are listing the current fields, methods, constructors for with a header that indicates the current class of the depth level. You may use a single space within a tab level to indicate an entry in a current listing. For example, indenting each method, in the listing of methods, one space.

Bonus: Reflective Dynamic Loading

Create and submit your own, more advanced, driver program. This driver program should take three command line arguments: (1) the name of a class containing the inspect method (2) the name of a class to inspect (3) a boolean for recursive.
This driver program should use reflection to load the class indicated as the first command line argument. This loaded class should then be used to run the **inspect(Object,boolean)** inside against the class indicated as the second command line argument, with recursion indicated by the third command line argument.
This bonus should function even if the class containing the **inspect(Object,boolean) IS NOT** in the project code you have written. The TA should be able to take some other differently named class containing the method and introduce it into the classpath of your code. Through the three command line arguments the TA should be able to use your dynamic loading driver to run **inspect(Object,boolean)** on any object that can be instantiated by a constructor with no arguments.
Make your bonus well-designed to handle invalid argument input, as well as gracefully handling any errors if the reflection fails to find the classes indicated as command line arguments. A bonus that doesn't handle exceptions and bad input will not get full marks.

Submit the following using the Assignment 2 Dropbox in D2L:

1. An electronic copy of your Inspector class (in a file called *Inspector.java*),
2. An electronic copy of each *script<no>.txt* file from the provided driver program.
3. An electronic copy of the final version of your unit test code.
4. An electronic copy of your version control logs. (alternatively making your repository available to your TA so that they can view your version control history in UofC GitLab, GitHub, etc. through web view is also allowed and even recommended)
5. An electronic copy of your refactorings (in a Word, PDF, or text file called *refactorings*).

Advanced Programming Techniques
Assignment 2 Grading

Student:____

Introspection

Name of declaring class	2	______
Name of superclass	2	______
Names of interfaces	2	______
Methods	10	______
Constructors (name, parameters, modifiers)	6	______
Fields (name, type, modifiers)	6	______
Field values	2	______
Super-class and interface recursion	4	______
Handles arrays	2	______
Well-formatted output (shown in *script.txt*)	2	______
Recursive Inspection	6	______

Other Requirements

Version control (show log files)	4	______		
Unit Tests	4	______		
Refactoring (described in *refactorings* file)	4	______		
Design Quality	4	______		
Total	**60**	______		_____%
Bonus: Reflective/Dynamic Loading Driver			5%	_____%
Total				_____%
Letter				_____

Letter	**A+**	**A**	**A-**	**B+**	**B**	**B-**	**C+**	**C**	**C-**	**D+**	**D**	**F**
Min. Perc.	**95%**	**90%**	**85%**	**80%**	**75%**	**70%**	**65%**	**60%**	**55%**	**50%**	**45%**	**Below 45%**