CPSC 501: Advanced Programming Techniques

Assignment 2: Reflective Object Inspector

Collaboration

Discussing the assignment requirements with others is a reasonable thing to do, and an excellent way to learn. However, the work you hand-in must ultimately be your work. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:

   ```
   # the following code is from
   https://www.quackit.com/python/tutorial/python_hello_world.cfm.
   ```

   Use the complete URL so that the marker can check the source.

2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. However, you may still get a low grade if you submit code that is not primarily developed by yourself.
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's console, then this code is not yours.
4. Collaborative coding is strictly prohibited. Your assignment submission must be strictly your code. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. You can not use (even with citation) another student's code.
5. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).
6. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize.

Late Penalty

Late assignments will not be accepted.

Goal

Experience and develop understanding of reflection.

Technology

Java, Git, Gitlab, Reflection

Specifics

Java 8, gitlab.cpsc.ucalgary.ca

Description

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 **obj**, 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 super-class immediately and recursively (even if recursive=false)*
3. The **name** of each **interface** the class implements
    a. *Always explore interfaces immediately and recursively (even if recursive=false)*
4. The **constructors** the class declares. For each, also find the following:
    a. The **name**
    b. The **exceptions** thrown
    c. The **parameter** types
    d. The **modifiers**
5. The **methods** the class declares. For each, also find the following:
    a. The **name**
    b. The **exceptions** thrown
    c. The **parameter** and **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**

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**).
ii. If the field is an object reference, and recursive is set to true, then immediately recurse on the object.

You must always traverse the inheritance hierarchy to find all the same information about each super-class(es) and interface(super-interfaces) declared. You should progress all the way up the hierarchy. *You will notice you will end up visiting certain classes multiple times and this is expected (eg. Object is the hierarchical super-class of every regular class).*

Arrays

Be sure you can also handle any array object you might encounter. This could be either the starting object or from a field. In addition to the regular name, type, and modifiers you must print out its component type, length, and all its entry values. **Array fields will be limited to one dimension.**

Infinite Recursion

There will be certain objects that will end up in infinite recursion if the recursive method argument is enabled. You do not have design your code to escape circular class references. **The driver program for evaluating your assignment will not test with objects with this circular reference 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 immediately 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. Report some of your refactorings in the project's readme.md. This usage doesn't have to be robust but is designed to encourage continual usage of version-controlled programming development.

Formatting

Please indent each class recursed into by 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. It is recommended to 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.

Submit the following using the Assignment 2 Dropbox in D2L:

1. The complete source of your Inspector class (in a file called Inspector.java) and your unit testing code.
2. An electronic copy of each ***script<no>.txt*** file from the provided driver program.
3. Directions/access to the **gitlab.cpsc.ucalgary.ca** project so that your TA can access your project and read your report.

Bonus:

Create and submit your own, more advanced, driver program **DriverBonus.java**. 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, and (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 a new instance of the class indicated as the second command line argument. Recursion behaviour is of course 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.

Grading

Class	Full name	1	
Superclass	Full name	1	
Interfaces	Full name	1	
Methods	Parameter types, modifiers, name, return types, exceptions	5	
Constructors	Parameter types, modifiers, exceptions	5	
Fields	Type, modifiers, name	3	
Field data	Values, references, null	2	
Super recursion	Interfaces and parent classes	5	
Arrays	Handles 1D arrays with required info	5	
Formatting	Tabbing, spacing, clarity of description	2	
recursive=true		8	
Version Control		4	
Tests/Refactoring		4	
Design quality		4	
Total		**50**	
Bonus	Dynamic loading	5	

Letter	**A+**	**A**	**A-**	**B+**	**B**	**B-**	**C+**	**C**	**C-**	**D+**	**D**	**F**
Points	**47.5**	**45**	**42.5**	**40**	**37.5**	**35**	**32.5**	**30**	**27.5**	**25**	**22.5**	**<22.5**