Assignment 3: Object Introspection (10%)

Fall 2021 – CPSC 501

Due at 23:59, Nov. 20 on D2L

Assignment policy:

- This is an **individual** assignment, so the work you hand in must be your own. Any external sources used must be properly cited (see below).
- Extensions will not be granted to individual students. Requests on behalf of the entire class will only be considered if made more than 24h before the original deadline.
- Some tips to avoid plagiarism in your programming assignments (taken from previous offerings of the course):

  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. When you upload your project to GitLab, make sure your repository is set to **private**. Any repository set to public or internal is visible to other students in the class, and counts as academic misconduct.
  3. 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.
  4. 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.
  5. 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.
  6. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task. Note that this still applies to the current offering.
  7. 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.

Instructions:

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 perform introspection on the argument obj, printing what it finds to standard output. You should find and display the following information about the object:

1. The name of the declaring class
2. The name of the immediate superclass*
    - Always explore the superclass immediately and recursively, even if recursive is false
3. The name of each interface the class implements*
    - Always explore all interfaces immediately and recursively, even if recursive is false
4. The constructors the class declares. For each constructor, give the following information:
    - Name
    - Modifiers
    - Parameter types
    - Exceptions thrown
5. The methods the class declares. For each method, give the following information:
    - Name
    - Modifiers
    - Parameter types
    - Return type
    - Exceptions thrown
6. The fields the class declares. For each field, give the following information:
    - Name
    - Type
    - Modifiers

- Current value
  - If the field is a primitive data type, simply print the value
  - If the field is an object reference and recursive is set to false, then simply print out the reference value directly. This will consist of the name of the object's class, plus the object's identity hash code (e.g. java.lang.Object@7d4991ad)
  - 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 all superclasses and interfaces declared. You should progress all the way up the hierarchy to Object. You will notice that you may end up visiting certain classes (such as Object) multiple times – this is expected. Each time you encounter a superclass or interface, perform the complete recursion. This will result in potentially printing all of the information for a class multiple times.

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 the values of all entries. You can assume that array fields will be limited to **one dimension**.

Infinite recursion: It is possible to define objects that end up in infinite recursion if the recursive method argument is enabled. However, you do not need to design your code to detect or escape circular class references. The driver program for evaluating your assignment will not test objects with this circular reference behaviour.

Formatting: Please indent each class recursed into by one tab of depth, and indicate clearly whenever you enter a new class. It is also helpful to indicate which class you are listing the current fields, methods and constructors for with a header that indicates the current class. The appropriate header would then be displayed each time you enter or leave a recursion level. You can judge what looks best in terms of output formatting, but make sure it is easy for the TAs to read and grade.

Refactoring: At some point in the development of your assignment code, you need to perform two distinct refactorings. Any refactorings from the course list or from Fowler's textbook are accepted. Write up these refactorings in a similar format to assignment 1 and include them in your report.

Other requirements:

- You should have descriptive output. For example, you should not use toString() for Field/Method or Class to get information. You will need to use the reflective API methods discussed in class to pull out the required information and print more descriptive explanatory lines.

- When printing modifiers, you must convert the returned integer information into descriptive text information such as public/private/final/static/transient/etc.
- A Driver program is provided on D2L that creates objects to inspect and then invokes your inspect method on each. This driver will output eight different script*.txt files, each corresponding to a different object.
- During the marking process, your TA will compile and run your code to verify that everything works.
- Remember to use version control and refactoring as discussed, as part of your coding process.

As in the previous assignments, you will be using **GitLab** to maintain version control and to share your final project with the TAs. Your assignment should be kept in a GitLab repository titled CPSC_501_A3. As you develop your code, make sure to use proper version control practices, making regular commits with descriptive messages.

Report: Create a written PDF report that describes your two refactorings in a similar format to assignment 1. Remember to include the name of the refactoring, reasons for making it, and before-and-after code excerpts. The justification can be brief - just a sentence or two is sufficient.

The report should also include **directions for the TA to access your GitLab project**. This is how they will be able to access your code and commit history, so double-check this works correctly before submitting. Make sure to indicate in the report whether you decided to implement the bonus part of the assignment. You may also include any information (known bugs, etc.) that you think will be useful to the TAs when grading.

Bonus - dynamic loading (up to 10%): 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) method against a new instance of the class indicated as the second command line argument. Recursion behaviour is 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.

Submission instructions:

Upload your written report as a **PDF file** to the Assignment 3 dropbox on D2L by 23:59 on November 20th. Make sure you **add your own TA to your GitLab project with reporter access** using their email given on D2L. The TA will use the instructions in your report to access your GitLab project, through which they will grade your submission.

Rubric (100 pts total):

- Version control: Used Git/GitLab properly, making multiple small commits with informative messages (5 pts)
- Refactoring: Performed two refactorings to improve the code structure, which are clearly written up in the report. (5+5 pts)
- Introspection: Program correctly displays the following information.
  - Class: full name (2 pts)
  - Superclass: full name (2 pts)
  - Interfaces: full names (2 pts)
  - Fields: name, modifiers, type (6 pts)
  - Field data: value or reference or null (6 pts)
  - Constructors: modifiers, parameter types, exceptions thrown (10 pts)
  - Methods: name, modifiers, parameter types, return type, exceptions thrown (10 pts)
  - Super recursion: inspects interfaces and parent classes (10 pts)
  - Arrays: handles 1-dimensional arrays with required info (10 pts)
  - Formatting: tabbing, spacing, clarity of description (6 pts)
  - Recursion: handles recursive = true by performing introspection on all sub-objects (16 pts)
- Logistics: Clear, working instructions on how to access GitLab project, submitted as a PDF report. Program can be run from the command line using the specified instruction (5 pts)
- Bonus: Solution uses reflection to dynamically load classes as described (10 pts)

---

author: jcleahy@ucalgary.ca