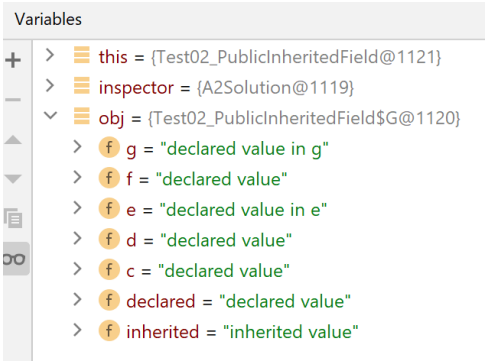
ASSIGNMENT 2

CS 474: Object-Oriented Languages and Environments / Spring 2023



Description

In this assignment, you will write a tool to inspect and modify the fields belonging to a Java object. Your tool could be used to implement an IDE utility, as depicted above. You will use Java reflection to implement this assignment. There are 10 test cases, worth 10 points each: undergraduate students need to get 80 points total for full credit, while graduate students should get all 100 points.

For this assignment, you need to write a class that implements the interface **`ObjectInspector`**, as shown below. You will be provided with startup code that you cannot modify (except as described in this document), and you only have to submit an implementation to the interface described below.

```
interface ObjectInspector {
    Map<String, String> describeObject (Object o);
    void updateObject(Object o, Map<String, Object> fields);
}
```

Method **`describeObject`** takes a single argument, which is an Object. It should return a **map** with one entry for each field in the object, where the **key** is the name of the field (qualified by the class name if it is a static field) and the **value** is the description of the field. For full credit, you should list **all** fields of `Object o`, including inherited, private, and static fields.

Method **`updateObject`** takes a map from field names to their desired values, and an object to perform those writes to. It should write the provided value to the provided field in the object. For instance, calling `updateObject(o, { "a": 0 , "b": 1 })` should write the value `0` to field `o.a` and the value `1` to field `o.b`.

Static Fields

When the argument of **`describeObject`** is of a class with static fields, they should be indicated by prepending the name of the class to the name of the field as the key to that field's entry. For instance, if class `A` has a static field `foo`, the key for that field should be `"A.foo"`.

When the argument of **`describeObject`** is an instance of `java.lang.Class`, then the method should describe **only** the static fields belonging to the class represented by the argument. For instance, using the example in the paragraph above, `describeObject(Class.forName(A))` should return a map with a key `"A.foo"` .

Value Formatting

The main challenge in implementing **`describeObject`** is to format the value of each field in an appropriate way given its type. Values should be formatted as follows:

- Primitive types
  - `int` fields are described as the result of `Integer.toString(int)`
  - `char` and `boolean` fields are similar to `int` but using methods `Character.toString(char)` and `Boolean.toString(boolean)`, respectively
  - `long` fields are described as the result of `Long.toString(long)` followed by the string `"#L"`. E.g., `872349234#L`
  - `float` and `double` fields are similar to `long`, but described with `Float.toString(float)` and `Double.toString(double)` and followed by the string `"#F"` and `"#D"`, respectively
  - `byte` fields are described as a string starting with `"0x"` and followed by the hex value of the byte, obtained through `Integer.toHexString(int)`. E.g., `0xA1`
  - `short` fields are described as a string starting with zero `"0"` and followed by the octal representation of the number through `Integer.toOctalString(int)`
- Boxed types (e.g. `Integer`, `Character`)
  - All boxed types are described with a string starting with `"Boxed "` and followed by the primitive description of the boxed value (see above)
- Null references are described as the string `"null"`
- All other objects are described by a method named `"debug"` if there is one (see below), and by their `"toString"` method otherwise

Grad Students Only: Debug Method

Some types have a `debug` method that should be called instead of `toString` when formatting values of that type. A class has a `debug` method if:

- It has a non-static method called `debug` that returns a `String` and does not take any arguments.
  - Your solution should call this method with the value as the receiver
- It has a static method called `debug` that returns a `String` and takes an argument of the right type.
  - You should pass the value being described as the single argument
  - If there are many such static methods, your solution can call any method that takes an argument compatible with the type of the object being described

Grad Students Only: Exceptions

You may encounter exceptions when trying to display the value of a field, e.g. exceptions thrown by the `toString` method. In that case, the description of that field should be as follows:

- For errors, "`Raised error: `" followed by the fully qualified name of the class of the error. For instance "`Raised error: java.lang.Error`"
- For checked exceptions, "`Thrown checked exception: `" followed by the fully qualified name of the class of the exception. For instance, "`Thrown checked exception: java.io.IOException`"
- For unchecked exceptions, "`Thrown exception: `" followed by the fully qualified name of the class of the exception. For instance, "`Thrown exception: java.lang.NullPointerException`"
- If the exception arises from a reflection-based method invocation, it will be wrapped in an `InvocationTargetException`. In this case, you should display the underlying exception (formatted as above), not the `InvocationTargetException`.

Entry Point

You should create a new class, in a new file, where you will implement your solution. You should change method Main.getInspector so that it creates an instance of the class you added. You should not change any other part of the code that is provided to you.

```
public abstract class Main {
    static ObjectInspector getInspector() {
        throw new Error("Not implemented");
    }
}
```

Due Date and Resubmission Policy

This assignment is due on **February 27 2023** (Monday) at **11:59pm CST**.

The code and date used for your submission is defined by the last commit to your Git repository.

Submission and Grading

This assignment is submitted through GitHub, and has an automatic grade component of 100%. You can check your current grade at any point by submitting your code and checking the autograder. The automatic grade is determined by 10 tests that will check if your project outputs the expected result. Each test is worth 10%.

Errors and Omissions

If you find an error or an omission, please post it on Piazza as soon as you find it.

Academic Integrity

The academic integrity policy described in the syllabus applies to this assignment. You are responsible for writing all the code that you submit. We will use an automatic tool that detects plagiarism on all submitted code, and we will investigate all instances where plagiarism is more than likely.

Please refer to the syllabus for the full academic integrity policy.