Inspector for Java

António Menezes Leitão

March, 2014

1 Introduction

In order to debug an application, it is useful to *inspect* its objects. An *inspector* describes the state of an object and can also support making changes in that state.

2 Goals

Implementation of an *inspector* of Java objects. The inspector can be started from any point of a Java program, accepting as argument an object that should be inspected.

The inspector must be started using the following form:

```
new ist.meic.pa.Inspector().inspect(object)
```

This means that you need to implement the class `ist.meic.pa.Inspector` with a construtor that does not accept arguments. The class must also provide a method with the following signature:

```
public void inspect(Object object);
```

As a result of calling the previous method, the inspector must then present all the relevant features of the object, namely:

- The class of the object.
- The name, type, and current value of each of the fields of the object.
- Other features that you feel are important.

The inspector should then enter a read-eval-print loop where it accepts commands from the user, executes the corresponding actions and prints the results. All interactions (both printing of the object features and reading of inspection commands) should be done on `System.err`.

3 Interface

The interface for interacting with the inspector is based on a textual representation that is printed on the console. As an example, consider the following Java classes:

```
package a;

public class B {
  private String c;
  protected int d;
}

class E extends B {
  boolean f;

  public int g(int h) {
    return d+h;
  }

  public static long i = 10L;
}
```

Upon inspection of an instance of `E`, the following (example) text is printed:

```
a.E@1a2961b is an instance of class a.E
----------
private java.lang.String c = "Hello World"
protected int d = 42
boolean f = true
```

After the presentation of the information, the inspector prints the prompt `>`, signaling that it is waiting for commands. You must implement, at the very minimum, the following commands:

- `q` Terminates inspection, allowing the calling program to proceed its execution.
- `i` ***name*** Inspects the value of the field named *name* of the object currently presented and makes that value the current inspected object.
- `m` ***name value*** Modifies the value of the field named *name* of the object currently presented so that it becomes *value*. This command must support, at minimum, fields of type `int`.
- `c` ***name*** ***value***$_0$ ***value***$_1$ `...`***value***$_n$ Calls the method named *name* using the currently presented object as receiver and the provided values as arguments and inspects the returned value, if there is one. This command must support, at minimum, calling methods that do not have parameters and methods that require arguments of type `int`.

As an example, after the presentation of the instance of class `E` described previously, the following interaction would place the value `99` in the field `d` and would call method `g` with argument `1`:

```
> m d 99
a.E@1a2961b is an instance of class a.E
----------
private java.lang.String c = "Hello World"
protected int d = 99
boolean f = true
> c g 1
100
>
```

3.1 Extensions

You can extend your project to further increase your grade above 20. Note that this increase will not exceed **two** points that will be added to the project grade for the implementation of what was required in the other sections of this specification.

Be careful when implementing extensions, so that extra functionality does not compromise the functionality asked in the previous sections.

Some of the potentially interesting extensions include:

- Allowing the user to navigate back and forth in the graph of inspected objects.
- Allowing the user to save inspected objects for further recall (e.g., to use as arguments in method calls).
- Allowing the user to inspect shadowed superclass fields.

4 Code

Your implementation must work in Java 6.

The written code should have the best possible style, should allow easy reading and should not require excessive comments. It is always preferable to have clearer code with few comments than obscure code with lots of comments.

The code should be modular, divided in functionalities with specific and reduced responsibilities. Each module should have a short comment describing its purpose.

5 Presentation

For this project, a full report is not required. Instead, a public presentation is required. This presentation should be prepared for a 15-minute slot (approximately, 8 slides), should be centered in the architectural decisions taken and might include all the details that you consider relevant. You should be able to "sell" your solution to your colleagues and teachers.

6 Format

Each project must be submitted by electronic means using the Fénix Portal. Each group must submit a single compressed file in ZIP format, named as `inspector.zip`. Decompressing this ZIP file must generate a folder named `g##`, where `##` is the group's number, containing:

- the source code, within subdirectory `/src`
- the slides of the presentation,
- an Ant file `build.xml` file that, by default, compiles the code and generates `inspector.jar` in the same location where the file `build.xml` is located.

Note that it should be enough to execute

```
$ ant
```

to generate (`inspector.jar`).

The only accepted format for the presentation slides is PDF. This file must be located at the root of the ZIP file and must have the name `p1.pdf`.

7 Evaluation

The evaluation criteria include:

- The quality of the developed solutions.
- The clarity of the developed programs.
- The quality of the public presentation.

In case of doubt, the teacher might request explanations about the inner working of the developed project, including demonstrations.

The public presentation of the project is a compulsory evaluation moment. Absent students during project presentation will be graded zero in the entire project.

8 Plagiarism

It is considered plagiarism the use of any fragments of programs that were not provided by the teachers. It is not considered plagiarism the use of ideas given by colleagues as long as the proper attribution is provided.

This course has very strict rules regarding what is plagiarism. Any two projects where plagiarism is detected will receive a grade of zero.

These rules should not prevent the normal exchange of ideas between colleagues.

9 Final Notes

Don't forget Murphy's Law.

10 Deadlines

The code and the slides must be submitted via Fénix, no later than 19:00 of **March**, **28**.

The presentations will be done during the classes after the deadline. Only one element of the group will present the work and the presentation must not exceed 15 minutes. The element will be chosen by the teacher just before the presentation. Note that the grade assigned to the presentation affects the entire group and not only the person the will be presenting. Note also that content is more important than form. Finally, note that the teacher may question any member of the group before, during, and after the presentation.