Assignment 4: Applying Design by Contract to an Existing Project

Software Design & Modeling

Due date: 2022-12-09 at 23:00

1 The assignment

Your assignment in a nutshell:

1. Pick a software **project** written in Java that you have access to. This can be:
   - an open-source project available on GitHub or other public repositories,
   - a package or component of the standard Java API,
   - a project you developed in the past (for example an assignment or course project), or
   - any other project whose source code is available and that you can share.
2. Identify a **project component** with clearly defined functionality. This typically corresponds to a small group of classes and interfaces related by inheritance. For example, a data structure consisting of an abstract interface and two concrete implementations.
3. Annotate the project component with **contracts** (pre- and postconditions, as well as class invariants) that characterize the expected behavior of the *public interface*. You should annotate at least 20 public methods with *some*[1] pre- and postconditions; and at least 2 classes (or interfaces) with *some* class invariant.
4. With **assertion checking** enabled, run the component with different inputs that exercise the annotated methods as diversely as possible. If the project includes tests, you can rerun those as well; in any case, you can write your own tests.
5. Write down a description of your work, from writing annotations to the outcome of your testing with assertion checking, and your findings in a **report**.

   Maximum length of the report: 8 pages (A4 with readable formatting) including any pictures and code snippets.

[1]That is, at least 20 methods should each have at least 1 precondition clause, at least 1 postcondition clause, or both.

The assignment must be done in *pairs* of students; student pairs must be different in each assignment; in other words, you should not work on this assignment with the same person you worked with on Assignment 1–3. Every group member should contribute equally to the work; the individual grading will also reflect this.

Once you have identified your partner for the assignment, write it in this spreadsheet **within 3 days** after this assignment is published. If you cannot find another student to work with within 3 days, the instructors will pair you up with some other available students.

This assignment contributes to **20%** of your overall grade in the course.

2 Tools and other resources

2.1 jSicko

Use the Java Simple Contract Checker `jSicko`:

`https://gitlab.com/usi-si-oss/codelounge/jSicko`

to process contract annotations in Java projects and add runtime checks when compiling. The repository includes basic examples of usage and common pitfalls. For more extensive examples, check out the other repository `https://gitlab.com/usi-si-oss/codelounge/jsicko-tutorials/` and the exercise of annotating `java.util.List` done in class (and available in iCorsi).

Your project should be compilable using Java 11 for `jSicko` to work properly. You should also be able to use `jSicko` with more recent version of Java, provided the code you annotate and analyze doesn't use any newer Java feature.

`jSicko` is a prototype developed by CodeLounge. If you find bugs or other issues, please report them to Andrea Mocci (the project maintainer) so that he can improve it.

Other languages? If you feel adventurous, or simply prefer to work with languages other than Java, you can look for a contract checking tool for languages other than Java and use it to do the assignment targeting another language.[2] If you choose to do so, please check with the instructors that your choice of tool and language is reasonable before working on the assignment.

2.2 Choosing the project

Look for components with a clearly defined interface whose behavior is amenable to being specified using simple assertions. *Libraries* are likely to be easier to specify than applications. *Data structure* libraries are natural candidates to be specified with contracts – but this is just a suggestion, not a requirement.

Concretely, here are some examples of projects that are suitable for analysis with jSicko:

- Data structures in `java.util`, in particular a combination of one interface (or abstract class) and a couple of concrete implementations;[3] for example `Map`, `HashMap`, and `LinkedHashMap`.

[2]For example, Deal is such a contract checking library for Python.

[3]With the exception of interface `List` and its implementations, as we analyzed these in class already.

- Other implementations of data structures, such as Google's Guava or Crater Dog's Java Collection Framework.
- Some classes and interfaces in package `java.util.stream`; for example, interface `Stream` (which includes several fully-implemented `default` or `static` methods).
- Implementations of date/time functionality, such as some classes in package `java.time`.
- Implementations of classic algorithms in Java, such as the following projects:
  `https://github.com/TheAlgorithms/Java`
  `https://github.com/phishman3579/java-algorithms-implementation`
- A (small and simple) Java project that you developed in the past.

Modifications. As much as possible, you should try not to alter significantly the structure or behavior of the source code you are annotating. However, it's quite possible that some contracts are impossible, or extremely cumbersome, to write without changing some aspects of the source code. If this turns out to be necessary, you are allowed to introduce any modifications. Remember to discuss and justify the main changes in the report.

2.3 Using jSicko

jSicko works generally well with "basic" Java features, but has some limitations when applied to code that uses recent (> version 11) Java features, or extensively relies on reflection or serialization. Therefore, make sure you have enough time to try out different projects, or to trim down a project's code to remove features that don't play well with jSicko, before you start writing contracts.

Another suggestion is to "start small": do not start annotating a big project with many tests. Instead, isolate a single class (or even parts of a class) and a bunch of tests of that class's methods, check that you can compile this small part of the project with jSicko (without including everything else), add some simple contract annotations to that part of the code, and verify that jSicko's checks work as intended (for example, try to trigger a contract violation by writing a test that violates a precondition). When you see that everything works as intended, you can proceed to add code to your build and annotate that as well.

Examples of contract annotations. The *tutorial* given in class where we annotated Java's `List` interface with contracts is a good example of how to proceed. Some additional examples of (mostly library) code annotated with contracts in JML (a notation different from jSicko's):

Java API examples: `http://www.eecs.ucf.edu/~leavens/JML/examples.shtml`

Algorithms in Java API: `https://bitbucket.org/caf/java.util.verified/`

Some of those contracts are significantly more complex and detailed than what you are expected to do for this assignment, but they might still give you ideas for the kinds of properties you can express using contracts.

3 What to write in the report

3.1 Report content

The structure of the report is free; it should include all the significant choices made and findings emerged during your work.

Things to include in the report's discussion:

- the project selection process, including mentioning projects you discarded;
- a short description of the selected project and project component with some stats (size, number of contributors, whether it's a library or application, and so on);
- how you approached writing contracts and what challenges you found in doing so;
- whether you had to modify the project to add some contracts, why, and how you modified it;
- a quantitative and qualitative summary of the contracts you wrote: number of pre- and postconditions, how detailed they are, and so on;
- a discussion of the outcome of testing with contract checking, and any interesting insight that came up;
- a brief (possibly speculative) discussion about whether your findings are likely to be applicable to other projects or, conversely, they are probably unique to the project you selected – and why you think this to be the case.

4 How and what to turn in

Turn in:

- Your **report** as a single PDF file in *iCorsi* under *Assignment 4*
- The **source code** of your project *before* and *after* annotating it with `jSicko` in the Git repository at `https://gitlab.com/usi-si-teaching/msde/2022-2023/software-design-and-modeling/assignment4-DbC/groupN`, where `N` is your group number in the group assignment spreadsheet.

  Use *branches* to store the *before* and *after* versions:

  1. add the project code *before* annotating to the `main` branch
  2. create a new branch called `annotated`
  3. add annotations, modifications, and new tests to the content of the `annotated` branch

  We will grade the most recent commit in the `annotated` branch before the project deadline.