CSDS 293
Software Craftsmanship
2024 Fall Semester

Programming Assignment 10

Due at the beginning of your discussion session on

Nov 7 – 15, 2024

Reading

In addition to the following topics, the quiz syllabus includes any material covered in the lectures:

- Sections 8.2 (introduction only), 8.3, 8.4 ("Avoid throwing exceptions in constructors ...", "Know the exceptions ...", and "Consider alternatives ..." only), and 8.5 in Code Complete.
- Items 9, 69, 73, 74, 75, 76, and 77 in Effective Java.
- The Error Propagation Guide in canvas in the Defensive Programming module.
- Section 19.6 in Code Complete, the Quick Reference Guide on Routine Names, and Repeated Code in canvas, and Items 28, 59, 60, and 62 in Effective Java.

Grading Guidelines

An automatic C (or less) is triggered by

- Any routine with complexity greater than 4, or by
- Any piece of code that is essentially repeated, or by
- Improperly named routines.

However, a submission that avoids these problems does not necessarily qualify for high quality craftsmanship. Starting with programming assignment 10, points will be deducted if code or branch coverage is incomplete.

Programming

In this programming assignment, you will apply the principles of defensive programming to the Social Network Analyzer that you developed in programming assignments 6 through 8.

The program should read the social network data from standard input. The data is encoded as a series of lines, where each line represents either a person or a connection:

- Person line format: `P [id] [name]`
- Connection line format: `C [id1] [id2]`

For example, a small social network might be encoded as:

```
P 1 Alice
P 2 Bob
P 3 Charlie
C 1 2
C 2 3
C 1 3
```

Your submission should be defensive about both non-compliant input formats and logical input errors. A non-exhaustive list of input problems involves non-numerical characters in IDs, improper separator characters, missing fields, and so on. A non-exhaustive list of logical input errors includes duplicate person entries, self-connections, connections to non-existent people, and so on. You are required to design, document, and implement a degree of robustness rather than pure correctness.

Design

Submit a separate file to document the error-handling architecture that you will follow in your implementation. This assignment can have multiple correct solutions and your job is to formulate one. The design document should cover the general concepts of the error-handling architecture, preferably with explanatory diagrams. The document should cover:

- A strategy for implementing robustness and handling errors (Section 8.3)
- Decisions on local or global error handling (Section 8.3)
- Presence of a barricade. You do not have to use a barricade, but if you do, you must present a diagram like Figure 8-2 explaining the identity of untrusted, validation (barricade), and internal (trusted) classes.

- The other factors in the defensive programming checklist at the end of chapter 8.

The implementation should carry out the decisions that you made in the design document. Conversely, ad-hoc implementations completely miss the point of the assignment.

Implementation

Implement your error handling architecture. Your implementation should be placed in socialnetwork.git. You will be asked to demonstrate that your code follows your error-handling architecture.

Submission

Make small regular commits and push your revised code and test cases on the git repository. Your submission should contain:

- A separate file to document your error handling architecture.
- Your implementation of the error handling architecture.
- A test suite that at a minimum achieves complete code and branch coverage.

General Considerations

Your code should have a reasonable number of comments, but documentation is going to be the topic of a future assignment. As a general guideline at this stage of the course, comments should be like those accepted in CSDS 132.

Discussion Guidelines

The project discussion will focus on defensive programming. You will be asked to demonstrate that your code follows your error-handling architecture.

Submissions

Make small regular commits and push your revised code and test cases on the git repository. Tag your release and push that tag to your git repository.