Objectives

- Learn to recognize Code Smells in existing code and be able to name them.
- Learn how to resolve Code Smells with the aid of refactoring.
- Learn how to apply manual refactoring, as well as automatic refactoring available in Eclipse.

Recommended Readings

- Chapter 24 of Code Complete 2 (available at LMS): It is an introduction to refactoring — what it is, why we need it, how to do it. This chapter contains a list of **code smells** and advice on resolving each of the code smells. Remember to use these in answering the questions.

Pairs

- You will work with a pair partner for this assignment.
    - Ideally you would be working on one computer, as in the actual pair programming.
    - We recommend contacting your partner and scheduling a common time slot early.
- In addition, each of you needs to submit the peer self-evaluation form **independently**.
    - Complete an honest evaluation of work effort for the assignment.
    - The teaching staff can adjust the individual score based on the evaluation.

Getting Started

- Download the code and import it using Eclipse.
- Read the README.pdf. It tells you what you need to do for the assignment.
- One of your pair should create code base repository, and invite the other. Your repo should be named homework4, and contain trunk, ANSWER.md, gitlab-ci.yml, etc. in top directory.
- You might notice some peculiarities with the code because it was initially written in C++. As such, the code doesn't always follow Java conventions.

Refactoring

- There are 4 refactorings and 4 questions that you need to answer. For each **refactoring**, you will be graded on how thorough your refactoring is. For instance, were you able to eliminate all the duplication that was mentioned in README.pdf?
- Also, the refactored version of the code should exhibit more communicative code. Communicative code has the following properties:
    1. Descriptive variable names
    2. Judicious use of comments
    3. Intention revealing method names
    4. Intention revealing class names
    5. Follows Java conventions
- For each **question**, you will be graded on
    1. The explanations and the examples that you give in your answers.
    2. Our review of your code and the refactorings that you performed.
    3. The functionality should not be changed, e.g., logging messages, output formats, etc.
    4. The tests in the project are all pass.
    5. The small refactoring steps must be **recorded by git commits.**
- For each part, feel free to justify your decision on why you chose to refactor or not to refactor. You are free to cite from the readings, where appropriate.
- As required in README.pdf, **please commit your code after you finish each of the four parts** with the given commit message.
- Do not modify the directory structure, and test files (unless it’s necessary due to the change of function types by refactoring).

Continuous Integration (CI)

- You need to set up CI for this assignment. That is, whenever you push your changes, all tests are automatically executed and the results are reported.
- Basically all you need to do is writing ".gitlab-ci.yml" file. A sample project is provided to demonstrate the use of CI in GitLab. See https://gitlab.com/help/ci/yaml/README.md.
- Go to the menu "CI / CD" to the left of your Gitlab repo webpage for CI execution result. The default docker image used for the shared runner is "maven:latest".
- We recommend to do 'Do a Mock Installation' part in README.pdf, and then build your CI, and push to your repo before you start refactoring.

Handing In

- Please remember to commit often. When you are done, please do the following.
- Fill out **ANSWER.md**.
    - Modify the provided template to contain your names and answers to the questions.
    - **Please use code smell names from course slides or the book chapter** (Chapter 24 of Code Complete 2). This helps eliminate confusion on terms.
    - You can have the same code smell for different questions. But please don't put the same code smell for all four questions. (It's not good for learning.)
- Submit your code: push the finished code into your gitlab repo.
- Submit your ANSWER.md: Place your completed ANSWER.md in **top directory** of your repo.
- Submit your individually peer self-evaluation form (peer-self.md) **to LMS** (not GitLab).

Some Tips

- The first time you run LANTests.java as a JUnit test, one of the tests will fail. We have included instructions on how to fix this in README.pdf.
- Keep in mind the goal of the refactorings that you are performing: to make it easier to add new functionality in the future. Use this goal to motivate your refactoring decisions**.**
- **Remember to commit** after finishing each (small) step. This will severely affect your grade.