Portfolio Part 6: Finishing Touches

- **Name**: Heath Younkin
- **Dot Number**: younkin.35
- **Due Date**: 12/10 @ 11:59 PM

Assignment Overview

By now, you should have an entire component implemented at all levels. For example, you should have two interfaces, an abstract class, and a kernel implementation.

Now, we want you to start producing your actual portfolio. To do that, you need to do a few things first:

1. **You need to add testing**: tests can be written basically as soon as you have designed your interfaces. Of course, because of how many changes your design has probably had up to this point, writing tests that early may lead to a lot of rework. Therefore, there is no better time to start testing than now that you have something more concrete.
2. **You need to share some use cases**: while having an API is useful, it's not always clear how and why someone would use it. Therefore, you should probably replicate some of the proof-of-concept work you did previously using your complete component.
3. **You need to publish your work**: if you haven't yet already, now would be a good time to publish your component to a public repo like GitHub.

These are the three main tasks we want you to complete to round out your portfolio project. However, there are tons of ways to take your work to the next level, such as:

1. **Publish your documentation**: one of the perks of using JUnit is that you can directly convert all those comments you made in your code to HTML files that can be hosted as Java documentation. How do you think the OSU API works?
2. **Start a CI pipeline**: it's very likely that you won't touch your component again after this class. That said, maybe you want to actual publish your component to a dependency manager like Maven. If so, it's a good idea to setup some automated testing and deployment through "Continuous Integration." GitHub Actions is a common tool for this, so why not look into it?
3. **Track version history**: in the world of software development, it's very important that you version your API with a version number. There are many different techniques for versioning software, but a date-based system is pretty easy. If you release your component on a particular date, use that date as the version number. Alternatively, you can make use of other techniques like semantic versioning.

Many of these additional techniques are somewhat out of the scope of this course, but just knowing about them could set you up for long term success.

Assignment Checklist

To be sure you have completed everything on this assignment, we have littered this document with TODO comments. You can browse all of them in VSCode by opening the TODOs window from the sidebar. The icon looks like a tree and will likely have a large number next to it indicating the number of TODOS. You'll chip away at that number over the course of the semester. However, if you'd like to remove this number, you can disable it by removing the following line from the `settings.json` file:

```
"todo-tree.general.showActivityBarBadge": true,
```

Which is not to be confused with the following setting that adds the counts to the tree diagram (you may remove this one as well):

```
"todo-tree.tree.showCountsInTree": true,
```

Assignment Learning Objectives

Without learning objectives, there really is no clear reason why a particular assessment or activity exists. Therefore, to be completely transparent, here is what we're hoping you will learn through this particular aspect of the portfolio project. Specifically, students should be able to:

1. Design a test plan for a software sequence component
2. Provide example use cases of a software sequence component
3. Use version control software to share a repository of code
4. Reflect on the software development process and the individual's own growth

Assignment Rubric

Unlike previous parts of the portfolio project, in this part of the project, you'll receive a summative assessment, which usually involves a more critical lens. In other words, at this stage, we're expecting you to demonstrate some mastery of the material. See the rubric items for specifics.

Learning Objective	Subcategory	Weight	Missing	Beginning	Developing	Meeting

Learning Objective	Subcategory	Weight	Missing	Beginning	Developing	Meeting
Students should be able to design a JUnit test plan that thoroughly tests all component methods	Procedural Creation	15	(0) No attempt is made to test the component	(5) Minimal testing is provided that only covers a small portion of the codebase	(10) A test suite is designed but misses some key test cases	(15) A thorough test plan is designed that covers the entire codebase
Students should be able to generate two qualitatively different use cases of their component	Factual Creation	10	(0) No attempt is made to generate two use cases of the component	(3) At least one use case is provided	(7) Two use cases are provided but are not significantly different from each other	(10) Two qualitatively different uses cases are provided showcasing the range of possible uses for the component
Students should be able to reflect on the software development process	Metacognitive Evaluation	15	(0) No attempt is made to reflect on the software development process	(5) A low effort reflection is made to answer the questions and/or clearly falls from a tool like ChatGPT	(10) A genuine reflection is made but some answers are thin or missing	(15) A genuine reflection is provided that shows metacognition (i.e., a deep consideration for how you've changed or developed)

Learning Objective	Subcategory	Weight	Missing	Beginning	Developing	Meeting
Students should be able to judge the polish of their component	Procedural Evaluation	10	(0) No attempt is made to polish up the component	(3) The overall level of polish is consistent with expectations from previous assignments, but no further effort is made to show pride in the work	(7) Polish is demonstrated through adherence to the OSU discipline only; no personal discipline or style is demonstrated	(10) Component demonstrates a high level of pride in the work by not only adhering to the OSU discipline but also by adopting a personal style, which can be demonstrated by documenting the use cases and testing, updating the main README, curating the GitHub page layout for readers, etc.

Below is further rationale/explanation for the rubric items above:

1. All component methods must be tested including the Standard, kernel, and secondary methods. The format of testing will be different from the style used in the course because it is unlikely that you will have an existing component to test against (more on this below).
2. There must be at least two different sample codes provided that show how the component might be used. For example, consider how `XMLTree` was used to create the `RSSReader` and the `RSSAggregator`. There is no expectation that you provide samples to this much depth, but two files with at least a main method would be excellent.
3. As you work through these finishing touches, take a moment to reflect on the software development process. There are reflection prompts below. You should also reflect on your growth as a developer and share details about what you've learned.
4. The overall component must exhibit a high level of polish. Because all previous parts of this assignment are meant to be formative, assessments were generally charitable. In this phase, you should submit a component as if you were a professional software engineer. Follow good practices like using good variable names, documenting your methods, and overall adhering to the discipline.

Pre-Assignment Tasks

To finish your component, make a branch off of main in your new repo called something like `finishing-touches`. There are many ways to do this, but my preference is to use GitHub Desktop. From there, you can click the `Branch` tab, select `New branch`, and name your new branch. Alternatively, VSCode has its own GUI for git. You can also make use of the command line directly in VSCode to run git commands. It's entirely up to you. Regardless of your choice, we'll want a branch that you can later make a pull request from with all your changes.

Note: because you may have changes still sitting in a pull request, you'll want to make this new branch directly from main. This may seem weird because you won't be able to see the other parts (e.g., your proof of concept) in VSCode. This is okay as parts 1-5 can be executed in isolation and merged together later. However, this does mean that you may be waiting for a pull request to see if your different features fit together. Once the pull request merges, you will need to pull the changes from main into your current branch to see them. If you don't like this workflow, you may try following the rebase strategies described here and here.

Assignment Tasks

Your primary task for this assignment is to polish up your code and get it ready to publish. To do that, you are being tasked with testing all of your existing code. You are also being tasked with coming up with at least two samples of your component being used for something. When all is said and done, you will publish your component to an open-source repository of your choosing.

Testing

As a part of testing your code, you will need to create a couple test files in the `test` directory. One of the files will be the JUnit test file for the kernel. Meanwhile, the other file will be the JUnit test file for the abstract class.

Unlike throughout the semester, testing will not involve testing against a reference implementation. As a result, testing is a bit messier and will involve calling kernel methods to check state. For example, here is a test case for one of the getters in NaturalNumber:

```
@Test
public void testIsZero() {
    NaturalNumber n = new NaturalNumber1L();
    assertEquals(true, n.isZero());
}
```

Note that this kind of testing is not as effective as the reference testing because it doesn't verify the entirety of the number's state. For example, it's possible that `n.isZero()` somehow changes the state of `n`. By only verifying the boolean value, there's no way of knowing if the remaining number is still intact. Therefore, it's probably better to write a test as follows:

```
@Test
public void testGetXOne() {
```

```
    NaturalNumber n = new NaturalNumber1L();
    NaturalNumber nCopy = new NaturalNumber1L();
    assertEquals(true, n.isZero());
    assertEquals(nCopy, n);
}
```

At least that way, you know the number is unchanged (i.e., `isZero()` properly restored `n`).

Use Cases

Another requirement to finish the project is to provide a couple of use cases for your component. To do that, the only expectation is that you generate two Java files in `src` with example code using your component either as part of the representation of some other proof-of-concept component or directly in `main`. For instance, the following class would be a extremely minimal example of how NaturalNumber might be used as part of a representation:

```
public class WholeNumber {

    private NaturalNumber n;

    public WholeNumber(int n) {
       assert n >= 0 : "Violation of: n >= 0";
       this.n = new NaturalNumber1L(n);
    }

    public void countUp() {
       this.n.increment();
    }
}
```

Polish

Overall polish involves a variety of efforts to demonstrate code quality. One of which would be directory structure. As an example, I would recommend that your directories look as follows before submission:

```
│   .gitattributes
│   .gitignore
│   LICENSE
│   README.md
│
├───.vscode
│       extensions.json
│       osu-cse-checkstyle-config.xml
│       osu-cse-formatter.xml
│       settings.json
```

```
│
├──doc
│   │   README.md
│   │
│   ├──01-component-brainstorming
│   │       01-component-brainstorming.md
│   │
│   ├──02-component-proof-of-concept
│   │       02-component-proof-of-concept.md
│   │
│   ├──03-component-interfaces
│   │       03-component-interfaces.md
│   │
│   ├──04-component-abstract-class
│   │       04-component-abstract-classes.md
│   │
│   ├──05-component-kernel-implementation
│   │       05-component-kernel-implementation.md
│   │
│   └──06-component-finishing-touches
│           06-component-finishing-touches.md
│
├──lib
│       components.jar
│       hamcrest-core-1.3.jar
│       junit-4.13.2.jar
│       README.md
│
├──src
│   │   NaturalNumberDemo.java
│   │   README.md
│   │   WholeNumber.java
│   │
│   └──components
│       └──naturalnumber
│                       NaturalNumber.java
│                       NaturalNumber1L.java
│                       NaturalNumberKernel.java
│                       NaturalNumberSecondary.java
│
└──test
    │   README.md
    │
    └──components
        └──naturalnumber
                        NaturalNumber1LTest.java
                        NaturalNumberTest.java
```

In addition, you'll want to take into account everything you've learned over the last two semesters to ensure your code is of good quality. That includes but is not limited to documenting your methods with parameter modes, giving your variables good names, making good use of whitespace, respecting CheckStyle, and just generally not taking shortcuts. Ultimately, you should be incorporating the feedback you've received throughout this process into your final product. **Have some pride in your work!**

Post-Assignment Tasks

The following sections detail everything that you should do once you've completed the assignment.

Reflection

Now that you have created a software component from scratch, it's time to reflect on that process as well as take some time to think about your growth as a developer. Below, you will find a series of reflection prompts. Take some time to fill them out honestly.

> A common gripe that students express is how much they feel the work they do in class fails to map to the real world. Now that you've had a chance to complete the portfolio project, how much better (or worse) do you think you understand software development and why?

I understand software development much better now because I have had the chance to apply the concepts learned in class to a real project. This project has allowed me to use the skills I have learned in class in a practical way involving my ineterests. I think I have a greater appreciation for software development now in that I can see the creativity involved in it more. That being said, the classroom can only teach so much, and completing this project has highlighted gaps in my knowledge that I will need to address on my own.

> Also, did the portfolio project surface any gaps in your own knowledge of software development. If so, what are those gaps and how did you address them?

Yes, it allowed me to understand how classes, interfaces, and abstract classes work together to create a complete component. I addressed this gap by looking at the OSU API and understanding how they structure their components and what is overridden, etc. As I mentioned before, this project has covered some gaps and revealed others. I still working on my general skills regarding VSCode and GitHub, as well as how languages work within these environments.

> Finally, as a part of completing the portfolio project, to what extent has your perspective of software development changed, if at all? In other words, is software development something you still enjoy? If not, why not?

I have learned to appreciate it more because I have made a practical use for it instead of just doing assignments for class. It has made me more interested in my capability for what I can do with software engineering. This project has also shown me how important it is to actually know how to code and not just vibe code. If people only use AI for real world projects, holes and errors will sprout more than if they coded it themselves with a lack of experience.

> One of the challenges of completing the portfolio project is picking up a lot of skills on your own. Some of these skills are, of course, software skills. However, there are plenty of other skills you may have picked up

> through this process. Therefore, the first question is what skills did you pick up through this process?

I have learned how to read APIs and also how to structure my own components in a way that follow the API structure. This also means getting better at documenting my code so that even after I have worked on it I explicity know what is going on. I have also learned how to use GitHub and GitHub Desktop to manage my code and version control. Much like any engineering project I have also honed my skill to work through tough problems and create an end product that I truly understand the inner workings of.

> The follow-up question is: could you rephrase these skills you picked up as bullet points that you could put on a resume? Try it below.

- Proficient in designing and implementing software components using Java
- Experienced in reading and understanding APIs to inform component design
- Skilled in documenting code effectively for long-term maintainability
- Familiar Git and GitHub
- Strong problem-solving abilities in software development projects

> Next, how has working on this project affected your career trajectory? In other words, do you now hate the topic you picked? Or, are you even more interested in it? Both outcomes are valuable to your personal development.

I think it has made me more interested in software development because I can see how it can be applied to real world problems and also to my own interests. This upcoming summer I will be interning as a software engineer and after the completion of this project I feel much more comfortable stepping into the role. I originally picked ECE because I didn't want to go all in on software development. Picking the degree and enjoying both software and hardware has made me realize the kind of work I want to do in the future, which would be a job straddling the fence dividing the two.

> Finally, consider the skills you've picked up and your current career trajectory. What are some things you could do to continue on your career trajectory? Also, who are some mentors you could contact to help you stay on your path?

I could contiue making projects like this outside of class to continue to develop my skills. I can now research how these software systems and languages that I have come to understand connect with the hardware I wish to work with. I am very fortunate to have a lot of great people to look up to, one of which first piqued my interest in software development and the ECE major. I will continue to stay in touch with him and seek his advice as I grow throughout my career journey.

Changelog

At the end of every assignment, you should update the CHANGELOG.md file found in the root of the project folder. Here's what I would expect to see at the minimum:

```
# Changelog

All notable changes to this project will be documented in this file.
```

```
The format is based on [Keep a Changelog](https://keepachangelog.com/en/1.1.0/),
and this project adheres to [Calendar Versioning](https://calver.org/) of
the following form: YYYY.0M.0D.

## YYYY.MM.DD

### Added

- Designed test suite for <!-- insert name of component here --> component
- Designed two different use cases for <!-- insert name of component here -->
component

### Updated

- Changed design to include ...
```

Here `YYYY.MM.DD` would be the date of your submission, such as 2024.04.21.

You may notice that things are nicely linked in the root CHANGELOG. If you'd like to accomplish that, you will need to make GitHub releases after each pull request merge (or at least tag your commits). This is not required.

Submission

Assuming that your project is in a GitHub repo somewhere and your changes are on a proof-of-concept branch, then what we'll want you to do is create a pull request of all your changes. Pull requests are pretty easy to make if you're using GitHub Desktop. Just click the `Branch` tab and select `Create pull request`. This should pull up your browser with the pull request form ready to complete. Give your pull request a good title like "Completed Part 6 of the Portfolio Project" and briefly describe what you've done. Then, click "Create pull request".

If all goes well, you should have a pull request that you can submit to Carmen via its URL. The URL should be in the form: `https://github.com/username/repo-name/pull/#`

Note: you are the owner of the repo, so you are not required to wait for feedback before merging. After all, the main purpose of the pull request is to put all your changes in once place for a code review. However, I highly recommend keeping the pull request open until at least a peer has had a chance to look over your changes. Otherwise, you defer needed changes to later pull requests, which could sacrifice the overall quality of your work or result in major rework.

Peer Review

Following the completion of this assignment, you will be assigned three students' component abstract classes for review. Please do not spend a ton of time on your reviews, **perhaps 10-15 minutes each**. Your job during the peer review process is to help your peers work through the logic of their implementations and identify gaps in their use of design-by-contract (i.e., forgetting checks for preconditions). If something seems wrong to you, it's probably a good hunch, so make sure to point it out.

When reviewing your peers' assignments, please treat them with respect. We recommend using the following feedback rubric to ensure that your feedback is both helpful and respectful (you may want to render the markdown as HTML or a PDF to read this rubric as a table).

Criteria of Constructive Feedback	**Missing**	**Developing**	**Meeting**
Specific	All feedback is general (not specific)	Some (but not all) feedback is specific and some examples may be provided.	All feedback is specific, with examples provided where possible
Actionable	None of the feedback provides actionable items or suggestions for improvement	Some feedback provides suggestions for improvement, while some do not	All (or nearly all) feedback is actionable; most criticisms are followed by suggestions for improvement
Prioritized	Feedback provides only major or minor concerns, but not both. Major and minar concerns are not labeled or feedback is unorganized	Feedback provides both major and minor concerns, but it is not clear which is which and/or the feedback is not as well organized as it could be	Feedback clearly labels major and minor concerns. Feedback is organized in a way that allows the reader to easily understand which points to prioritize in a revision
Balanced	Feedback describes either strengths or areas of improvement, but not both	Feedback describes both strengths and areas for improvement, but it is more heavily weighted towards one or the other, and/or descusses both but does not clearly identify which part of the feedback is a strength/area for improvement	Feedback provides balanced discussion of the document's strengths and areas for improvement. It is clear which piece of feedback is which
Tactful	Overall tone and language are not appropriate (e.g., not considerate, could be interpreted as personal criticism or attack)	Overall feedback tone and language are general positive, tactul, and non-threatening, but one or more feedback comments could be interpretted as not tactful and/or feedback leans toward personal criticism, not focused on the document	Feedback tone and language are positive, tactful, and non-threatening. Feedback addesses the document, not the writer

Assignment Feedback

If you'd like to give feedback for this assignment (or any assignment, really), make use of this survey. Your feedback helps make assignments better for future students.