Portfolio Part 3: Component Interfaces

- **Name**: Jacksen Cooper
- **Dot Number**: cooper.3135
- **Due Date**: 10/24/25

Assignment Overview

By now, you have had the opportunity to create three high-level component designs and even prove that at least one of them is viable. Hopefully, at this point, you've also received feedback on your designs. Now, you will have the opportunity to put together the client-side interfaces, specifically the kernel and enhanced interfaces.

The kernel interface provides the minimal functionality you would expect of your data type. By convention, we would name our kernel interface `ComponentKernel`, where `Component` is the name of your component. As an example, the kernel interface for the `NaturalNumber` component would be written as `NaturalNumberKernel`, and it's skeleton would look as follows:

```
public interface NaturalNumberKernel extends Standard<NaturalNumber> {
    ...
}
```

Similarly, the enhanced interfaces provides all of the methods that we want to layer on top of the kernel. Again, by convention, we use the name of the component directly for the enhanced interface. For example, `NaturalNumber` would be the name of the enhanced interface, and its skeleton would look as follows:

```
public interface NaturalNumber extends NaturalNumberKernel {
    ...
}
```

Remember, the interfaces are what the client will see, so you should really be considering the types of functionality that your user would want. If it helps, ask yourself the following: what would I (as in you, the reader) want this component to be able to do? Now that you've had a chance to create a proof-of-concept, you should be able to answer this question to some extent.

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. Identify areas of improvement in previous designs and strategies to address them
2. Provide contracts for clients via method headers
3. Assemble a kernel interface in line with the software sequence discipline
4. Assemble an enhanced interface in line with the software sequence discipline

Assignment Rubric

Again, to be completely transparent, most of the portfolio project, except the final submission, is designed as a formative assessment. Formative assessments are meant to provide ongoing feedback in the learning process. Therefore, the rubric is designed to assess the learning objectives *directly* in a way that is low stakes—meaning you shouldn't have to worry about the grade. Just do good work.

Learning Objective	**Subcategory**	**Weight**	**Missing**	**Beginning**	**Developing**	**Meeting**
Students should be able to list changes to their design	Factual Memory	2	(0) No attempt made to list design changes	(.5) Changes were listed but not in the CHANGELOG	(1.5) Changes are listed but in chronological order	(2) Changes are correctly listed in CHANGELOG in reverse chronological order
Students should be able to provide a hierarchy diagram of their component	Conceptual Understanding	2	(0) No attempt was made to include a hierarchy diagram	(.5) A hierarchy diagram is included but does not follow the OSU discipline	(1.5) A hierarchy diagram is included with some minor mistakes or missing elements	(2) A proper hierarchy diagram is included which showcases where the two interfaces fit

Learning Objective	Subcategory	Weight	Missing	Beginning	Developing	Meeting
Students should be able to generate client-sided documentation for their component	Factual Creation	3	(0) No attempt is made to generate documentation	(1) Documentation is included but does not follow expectations from our discipline (e.g., no OSU-specific annotations)	(2) Documentation is included but is no exhaustive	(3) Documentation is included that follows all OSU best practices, such as including standard JavaDoc as well as parameter modes and design by contract as needed
Students should be able to design kernel and enhanced interfaces	Procedural Creation	3	(0) No attempt is made to design kernel and enhanced interfaces	(1) Interfaces are created by do not follow OSU discipline	(2) Interfaces are created with minor bugs like not getting the hierarchy right	(3) Interfaces are created that properly following the OSU discipline

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

1. Because these assignments build on each other, it's important to acknowledge the growth and development of your work. Therefore, you must show what has been changed following the proof-of-concept as it pertains to the interface design.
2. To ensure you have a conceptual understanding of the kernel and enhanced interfaces as well as how they fit in the larger picture, you must include a hierarchy diagram for your component.
3. Both interfaces must include full documentation for all method headers. Documentation must meet the bare minimum expectations for JUnit, which means including `@param` for every parameter and `@return` for every return type. Documentation must also meet the bare minimum expectations for design by contract, which includes an `@ensures` for every method. All other aspects of contracts should be included where applicable, such as `@requires`, `@replaces`, `@clears`, and `@updates`. There is no requirement to use mathematical notation in your contracts.
4. The kernel interface must, at the very least, inherit from `Standard` and compile. In addition, it must contain a handful of minimally viable method headers (i.e., the minimum number of methods needed to model the data type). Do not include fields or method implementations. However, you may include any nested interfaces needed for your data type. You may also include constants that may be helpful throughout the design. Similarly, the enhanced interface must, at the very least, inherit from the kernel interface and compile. In addition, it may contain as many method headers as you would like, but the methods must be able to be implemented by the kernel and `Standard` methods. Again, do not include any method implementations or fields at this point.

Pre-Assignment Tasks

Before you start crafting your interfaces, you must demonstrate that you know how everything fits in the discipline hierarchy. You have seen many of these diagrams already, so you should be able to draw one. Feel free to make a hierarchy diagram using whatever tools you would like. Then, include a picture of it in this folder. You may also embed it just below using markdown syntax (i.e., `![ALT TEXT](path/to/file)`).

To start making your interfaces, make a branch off of main in your new repo called something like `interfaces`. 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 draft two interfaces in line with the course discipline: a kernel interface and an enhanced interface. Because it is unlikely that you've done this before, you may consider browsing some examples directly from the API. For example, here is the NaturalNumberKernel source code. Your kernel interface should look something like this. Meanwhile, here is the NaturalNumber source code. Similarly, your enhanced interface should look something like this.

As with the previous assignment, you will share no code here. Instead, create your two interface files in `src`, and follow the submission instructions below.

Post-Assignment Tasks

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

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 kernel and enhanced interfaces 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 3 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 interfaces assignments 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 edit their contracts. Specifically, you should be helping them with the readability of their contracts. If something isn't clear to you, it's probably not clear to others, so help them communicate their contracts better. 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 | | ---------- | ------- | ---------- |v ---------- | ---------- | | 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.