Portfolio Part 4: Abstract Class

- **Name**: Jacksen Cooper
- **Dot Number**: cooper.3135
- **Due Date**: 11/7/25 @ 1:50

Assignment Overview

Now that you've had a chance to refine your designs a bit, it's time to start writing some code. In this assignment, you will be making your first abstract class. The abstract class will take on the name `ComponentSecondary`, where `Component` is the name of your component. For example, if you're making a NaturalNumber component, the abstract class would be called `NaturalNumberSecondary` as follows:

```
public abstract class NaturalNumberSecondary implements NaturalNumber {
    ...
}
```

Inside the abstract class, you will implement every secondary method you specified in the enhanced interface. Because the secondary abstract class is layered over the kernel interface, you cannot implement the methods of your enhanced interface using the underlying representation. As a result, these methods must be implemented using the kernel methods only.

Surprisingly, you have done this several times before without probably realizing it. For example, when you first learned recursion in software 1, you were tasked with implementing the secondary methods of NaturalNumber. At the time, they were static methods, but you were tasked with only implementing the secondary methods using the kernel methods.

You did this a few other times as well. For example, we asked you to implement the secondary methods for set at one point. We did this by having you extend `Set1L`, so you could override the implementation of `remove()` and `add()`. At the time, we didn't force you to only use kernel methods, but the premise remains the same. Meanwhile, in a later lab, you were tasked with implementing the secondary method `sort()` of `Queue`.

Once you have implemented all of the secondary methods, you must also implement the key `Object` methods. It's up to you to decide which ones you want to implement, but `toString()` and `equals()` are a great start. You may also implement `hashCode()` if you so choose. Note that these methods do not have access to the representation, so you must also implement them using the kernel methods only.

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. Generate a list of changes since the previous iteration of a project
2. Use the kernel and Standard methods to implement a series of secondary methods that compile
3. Identify method preconditions and check them appropriately (i.e., follow design-by-contract)

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	3	(0) No attempt made to list design changes	(1) Changes were listed but not in the CHANGELOG	(2) Changes are listed but in chronological order	(3) Changes are correctly listed in CHANGELOG in reverse chronological order

Learning Objective	Subcategory	Weight	Missing	Beginning	Developing	Meeting
Students should be able to carry out implementations of all enhanced methods and common methods	Procedural Application	4	(0) No attempt is made to implement any enhanced methods	(1) An attempt is made to implement the enhanced methods, but they clearly do not work OR implementations make use of global variables and/or kernel implemnetation details	(3) All enhanced methods are implemented, but no common methods are implemented	(4) All enhanced methods are implemented, and common methods are implemented or justifiably excluded (preferably with a comment)
Students should be able to check that calls to kernel methods follow their contracts	Factual Evaluation	3	(0) No attempt is made to respect kernel contracts	(1) Some kernel method contracts are respected	(2) Most kernel method contracts are respected	(3) All kernel method contracts are respected

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

1. In keeping with the concept of iteration, the assignment must detail all of the changes from most recent submission (i.e., since the interfaces). Make sure to explain what was changed and why.
2. The abstract class implementation must implement all methods from the enhanced interface (i.e., all secondary methods), and these methods must be implemented using only the kernel and Standard methods. In addition, the abstract class must implement `toString()` and `equals()`, but you may implement `hashCode()` as well. In general, the methods do not have to be 100% correct, but their logic must make sense (i.e., no low effort implementations).
3. When implementing the secondary methods, you must respect the contracts of the kernel methods. In other words, if a kernel method has a precondition, the client of the kernel method must check the precondition. If there is no method available to check the precondition, it must be added to the kernel.

Pre-Assignment Tasks

> To start making your abstract class, make a branch off of main in your new repo called something like `abstract-class`. 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 create an abstract class that falls from the interfaces you previously designed. Because it is unlikely you have done this before, consider browsing some examples in the API. Unfortunately, there are not many. However, you might browse SimpleWriterSecondary or SimpleReaderSecondary.

As with the previous assignment, you will share no code here. Instead, create your abstract class file 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 abstract class 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 4 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

Criteria of Constructive Feedback	Missing	Developing	Meeting
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.