Portfolio Part 1: Component Brainstorming

- **Name**: Heath Younkin
- **Dot Number**: younkin.35
- **Due Date**: 9/19

Assignment Overview

The overall goal of the portfolio project is to have you design and implement your own OSU component. There are no limits to what you choose to design and implement, but your component must fit within the constraints of our software sequence discipline. In other words, the component must extend from Standard and must include both a kernel and a secondary interface.

Because this is a daunting project, we will be providing you with a series of activities to aid in your design decisions. For example, the point of this assignment is to help you brainstorm a few possible components and get some feedback. For each of these components, you will need to specify the high-level design in terms of the software sequence discipline. In other words, you will describe a component, select a few kernel methods for your component, and select a few secondary methods to layer on top of your kernel methods.

You are not required to specify contracts at this time. However, you are welcome to be as detailed as you'd like. More detail means you will be able to get more detailed feedback, which may help you decide which component to ultimately implement.

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. Integrate their areas of interest in their personal lives and/or careers with their knowledge of software design
2. Determine the achievablility of a software design given time constraints
3. Design high-level software components following the software sequence discipline

Assignment Rubric: 10 Points

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 identify their values, interests, and/or goals as they relate to their designs	Metacognitive Memory	3	(0) No attempt to summarize values, interests, and/or goals	(1) A brief description of values, interests, and/or goals is provided but lacks depth	(2) A description of values, interests, and/or goals is provided by are not related to designs	(3) A description of values, interests, and/or goals is provided and relates to designs
Students should be able to predict the feasibility of their designs	Metacognitive Understanding	3	(0) No attempt to design components that are feasible	(1) At least one component is feasible	(2) At least two components are feasible	(3) All three components are feasible
Students should be able to use the OSU discipline in all three designs	Metacognitive Application	4	(0) No attempt to follow the OSU discipline in designs	(1) At least one design follows the OSU discipline	(3) At least two designs follow the OSU discipline	(4) All three designs follow the OSU discipline

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

1. Each design must align with your personal values and long-term goals. Because the goal of this project is to help your build out a portfolio, you really ought to care about what you're designing. We'll give you a chance to share your personal values, interests, and long-term goals below.
2. Each design must be achievable over the course of a single semester. Don't be afraid to design something very small. There is no shame in keeping it simple.
3. Each design must fit within the software sequence discipline. In other words, your design should expect to inherit from Standard, and it should contain both kernel and secondary methods. Also, null and aliasing must be avoided, when possible. The methods themselves must also be in justifiable locations, such as kernel or secondary.

Pre-Assignment

> Before you jump in, we want you to take a moment to share your interests below. Use this space to talk about your career goals as well as your personal hobbies. These will help you clarify your values before you start brainstorming. Plus it helps us get to know you better! Feel free to share images in this section.

Career goals are still something I am working on. I know that I want to work in a field that does something related to technology and electronics. Whether that be hardware, software, etc. I do not know. In terms of interests I am interested in many different things. Movies, music and djing, art (ceramics), skiing, enjoying the outdoors and more.

Assignment

As previously stated, you are tasked with brainstorming 3 possible components. To aid you in this process, we have provided some example components that may help you in your brainstorming. All of these components were made at some point by one of your peers, so you should feel confident that you can accomplish any of them.

There is no requirement that you use any of the components listed above. If you want to model something else, go for it! Very common early object projects usually attempt to model real-world systems like banks, cars, etc. Make of this whatever seems interesting to you, and keep in mind that you're just brainstorming right now. You do not have to commit to anything.

Example Component

To help you brainstorm a few components, we've provided an example below of a component you already know well: NaturalNumber. We highly recommend that you mirror the formatting as close as possible in your designs. By following this format, we can be more confident that your designs will be possible.

- Example Component: `NaturalNumber`
  - **Description**:
    - The purpose of this component is to model a non-negative integer. Our intent with this design was to keep a simple kernel that provides the minimum functionality needed to represent a natural number. Then, we provide more complex mathematical operations in the secondary interface.
  - **Kernel Methods**:

        - `void multiplyBy10(int k)`: multiplies `this` by 10 and adds `k`
        - `int divideBy10()`: divides `this` by 10 and reports the remainder
        - `boolean isZero()`: reports whether `this` is zero
    - **Secondary Methods**:
        - `void add(NaturalNumber n)`: adds `n` to `this`
        - `void subtract(NaturalNumber n)`: subtracts `n` from `this`
        - `void multiply(NaturalNumber n)`: multiplies `this` by `n`
        - `NaturalNumber divide(NaturalNumber n)`: divides `this` by `n`, returning the remainder
        - ...
    - **Additional Considerations** (*note*: "I don't know" is an acceptable answer for each of the following questions):
        - Would this component be mutable? Answer and explain:
            - Yes, basically all OSU components have to be mutable as long as they inherit from Standard. `clear`, `newInstance`, and `transferFrom` all mutate `this`.
        - Would this component rely on any internal classes (e.g., `Map.Pair`)? Answer and explain:
            - No. All methods work with integers or other NaturalNumbers.
        - Would this component need any enums or constants (e.g., `Program.Instruction`)? Answer and explain:
            - Yes. NaturalNumber is base 10, and we track that in a constant called `RADIX`.
        - Can you implement your secondary methods using your kernel methods? Answer, explain, and give at least one example:
            - Yes. The kernel methods `multiplyBy10` and `divideBy10` can be used to manipulate our natural number as needed. For example, to implement `increment`, we can trim the last digit off with `divideBy10`, add 1 to it, verify that the digit hasn't overflown, and multiply the digit back. If the digit overflows, we reset it to zero and recursively call `increment`.

Keep in mind that the general idea when putting together these layered designs is to put the minimal implementation in the kernel. In this case, the kernel is only responsible for manipulating a digit at a time in the number. The secondary methods use these manipulations to perform more complex operations like adding two numbers together.

Also, keep in mind that we don't know the underlying implementation. It would be completely reasonable to create a `NaturalNumber1L` class which layers the kernel on top of the existing `BigInteger` class in Java. It would also be reasonable to implement `NaturalNumber2` on top of `String` as seen in Project 2. Do not worry about your implementations at this time.

On top of everything above, there is no expectation that you have a perfect design. Part of the goal of this project is to have you actually use your component once it's implemented to do something interesting. At which point, you will likely refine your design to make your implementation easier to use.

Component Designs

> Please use this section to share your designs.

- Component Design #1: SongSort
  - **Description**:
    - A component that functions in tandem with a typical djing software but allows for more customization in terms of how you can sort your music and analyze it. This software could provide a better calculation of the time/measurement (bars) in between parts of songs (phrases).
  - **Kernel Methods**:
    - `void song()` - classify a song from which one could pull info or add song to groups
    - `string title()` - store a songs title, return string
    - `string artist()` - store a songs artist, return string
    - `string genre()` - store a songs genre, return string
    - `string key()` - store a songs key, return string
    - `double bpm()` - store a songs bpm, return double
    - `double length()` - store a songs length, return double
    - `void group(String str)` - store the songs in a created group
  - **Secondary Methods**:
    - `void songInfo()` - would store songs info: title, artist, key, bpm, genre, etc.
    - `string getGroup()` - would return a songs group
    - `double getBar(double d)` - get what bar a song is at at a given time
    - `string phrase()` - provide what kind of phrase a song is at at a given bar
    - `void setPhrase(String str)` - set a name for a phrase
  - **Additional Considerations** (*note*: "I don't know" is an acceptable answer for each of the following questions):
    - Would this component be mutable? Answer and explain:
      - Yes, one would want to be able to change the groups a song is in, its phrases, and possibly other info.
    - Would this component rely on any internal classes (e.g., `Map.Pair`)? Answer and explain:
      - Likely not, everything will be based off of 'song' objects
    - Would this component need any enums or constants (e.g., `Program.Instruction`)? Answer and explain:
      - Possibly for keys: '12A, 1B, etc.' and the beginning of the song ex: "0.00"
    - Can you implement your secondary methods using your kernel methods? Answer, explain, and give at least one example:
      - Yes, `songInfo()` for example would call the info kernel methods that hold the songs title, bpm, etc.

- Component Design #2: LogicGate
  - **Description**:
    - A component that models digital logic. It would take multiple boolean inputs and produce a boolean output based on the type of gate. This could be used to build more complex gates and circuits.
  - **Kernel Methods**:

        - `void setInputs(boolean arr[])` - sets inputs for the gate
        - `void getInput()` - outputs inputs
        - `boolean getOutput()` - returns boolean output given the gates current input
        - `void setType(GateType g)` - sets the gates type AND, OR, NOT
        - `GateType getType()` - returns the gates type AND, OR, NOT
    - **Secondary Methods**:
        - `toggleInput(int inputIndex)` - toggle the value of a certain gates indexed input
        - `string toString()` - returns a string representation of the inputs and outputs: `"AND(1,1) = 1"`
        - `LogicGate clone()` - create a dupicate gate with the same inputs
        - `connectTo(LogicGate L, int inputIndex)` - link different gates together
    - **Additional Considerations** (*note*: "I don't know" is an acceptable answer for each of the following questions):
        - Would this component be mutable? Answer and explain:
            - Yes inputs and gatetypes can change.
        - Would this component rely on any internal classes (e.g., `Map.Pair`)? Answer and explain:
            - Yes possibly a `Connection` class for making circuits and more complex gates. It may also need a `GateType` class that defines the kind of gates that can be made, allowing the user to make new ones.
        - Would this component need any enums or constants (e.g., `Program.Instruction`)? Answer and explain:
            - Yes the basic gate types: AND, OR, NOT. Possibly even others like: NAND, NOR, XOR, XNOR, etc.
        - Can you implement your secondary methods using your kernel methods? Answer, explain, and give at least one example:
            - `toggleInput()` would require the use of `setInputs()` as it would have to flip an input value. `toString()` would call `getInput()`, `getOutput()`, and `getType()`.

- Component Design #3: Sprite

    - **Description**:
        - A component that represents a 2D object in a game. It has a position, size, image, and can be moved up or down on a screen.
    - **Kernel Methods**:
        - `void setPosition(double x, double y)` - set sprites position
        - `double getPosition()` - returns current coordinates
        - `void setImage(Image i)` - sets the sprites look
    - **Secondary Methods**:
        - `void move(double x, double y)` - shifts the sprite relative to its current position
        - `boolean collides(Sprite s)` - checks if sprite collides with another sprite
        - `boolean isVisible()` - checks if sprite should be rendered
    - **Additional Considerations** (*note*: "I don't know" is an acceptable answer for each of the following questions):
        - Would this component be mutable? Answer and explain:

        - Yes multiple attributes about a sprite change, specifically location.
    - Would this component rely on any internal classes (e.g., `Map.Pair`)? Answer and explain:
        - Possibly an `image` class or a `bounds` class to check for collisions.
    - Would this component need any enums or constants (e.g., `Program.Instruction`)? Answer and explain:
        - Possibly starting positions, direction and state.
    - Can you implement your secondary methods using your kernel methods? Answer, explain, and give at least one example:
        - Yes, `move()` requires the current location using `getPosition()` and the future location with `setPosition()`

Post-Assignment

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. Since this is likely the first time you've done this, we would recommend browsing the existing file. It includes all of the changes made to the portfolio project template. When you're ready, you should delete this file and start your own. Here's what I would expect to see at the minimum:

```markdown
# 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 a <!-- insert name of component 1 here --> component
- Designed a <!-- insert name of component 2 here --> component
- Designed a <!-- insert name of component 3 here --> component
```

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.

In the future, the CHANGELOG will be used to document changes in your designs, so we can gauge your progress. Please keep it updated at each stage of development.

Submission

If you have completed the assignment using this template, we recommend that you convert it to a PDF before submission. If you're not sure how, check out this Markdown to PDF guide. However, PDFs should be created for you automatically every time you save, so just double check that all your work is there before submitting. For future assignments, you will just be submitting a link to a pull request. This will be the only time you have to submit any PDFs.

Peer Review

Following the completion of this assignment, you will be assigned three students' component brainstorming assignments for review. Your job during the peer review process is to help your peers flesh out their designs. Specifically, you should be helping them determine which of their designs would be most practical to complete this semester. When reviewing your peers' assignments, please treat them with respect. Note also that we can see your comments, which could help your case if you're looking to become a grader. Ultimately, 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

Criteria of Constructive Feedback	**Missing**	**Developing**	**Meeting**
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.