**Design “Class” Diagram**

Color picker:

The color picker changes state based on what color the user selects. This color is displayed back to the user.

The color is also passed to the palette allowing it to use the color.

Images:

Images have a source image and that image is displayed to the user.

The palette sets the images.

Circler maker:

This class is designed to generate circles and ellipses.

It includes attributes like radius, height, and width to define the dimensions of shapes.

generateBracketCircle(radius) - creates a 2D array of strings with proper characters representing a circle

generateBracketCircle(width, height) - creates a 2D array of strings with proper characters representing a ellipse

generateShape() - displays the shape on the screen.

User:

This class is designed to handle user accounts.

Stores information about their account

Also stores all the palettes they saved.

*<For each part:>*  


The diagram above the class design of Builder’s Block. Essentially, the website is split into multiple pages. Each page interacts with itself and only itself. The circle generating page has nothing to do with the block palette generating page. For the palette page, this is represented in the diagram with the palette, color picker, and image classes being connected, as the palettes are generated by parsing image data with information given by the color picker. It is not connected to the circle maker class at all. The user class exists to keep track of and store user data, mainly favorited palettes, username and password. It uses the palette class because it stores palettes.

**Design Patterns**

**State**



We chose the State design pattern to manage the different states of the Color Picker and Filter classes effectively. This pattern is ideal for scenarios where an object’s behavior changes based on its internal state. For example, the Filter class updates the palette generated based on the color selected in the Color Picker. This approach simplifies the management of complex state transitions and enhances maintainability. The Palette class can serve as the context in this design. It stores multiple attributes such as paletteSize, baseRange, randomBlocks, and now includes images for associated visual data. The state of the palette may change depending on user interactions, such as selecting a new color in the color picker or generating shapes with the circle maker.

An abstract interface defines operations that handle changes to the palette's contents. This ensures consistent behavior across all state implementations, such as updating the image or recalculating the random blocks based on the selected color picker value.

The implementation is illustrated in the excerpt of the class diagram, where the Color Picker and Filter classes interact. Each state (e.g., different color selections) triggers specific actions when generating palettes ensuring a seamless user experience.

Selecting a new color in the color picker triggers a transition to colorstate, updating the random blocks and filtering images. Generating a shape via the circle maker transitions the state to ShapeState which appends new shape-related data to the palette. Saving a palette triggers a transition to UserSavedState, ensuring proper storage via the user class.

Selecting a new color in the color picker triggers a transition to ColorState updating the randomBlocks and filtering images. Generating a shape via the circle maker transitions the state to shapestate which appends new shape-related data to the palette. Saving a palette triggers a transition to UserSavedState ensuring proper storage via the user class.