Joseph Aulner
CSCI 2510-001
Assignment 1

Paint Application

Paint

The `Paint` class is the entry point for the application. It is responsible for creating the `JFrame` in which the game interface will be displayed, creating and managing the rendering thread, and defining the primary game loop. An effort was made to reduce this class to a simple window/thread manager by extracting all game state/logic and rendering methods to the relevant packages.

Packages

The paint application is divided up into three primary packages: `logic`, `render`, and `util`. These packages segregate the primary concerns of the paint application, keeping track of the game state, rendering visual output for the user, and providing common utilities primarily for handling IO, respectively.

`logic`

The `logic` package contains the classes responsible for managing the state of the application. The primary class, `GameState`, stores the shapes that have been drawn, the available drawing tools, and the available colors. It provides a set of simple getters and setters to update the game state, as well as a method to update the game state in response to mouse and keyboard input. I chose to write `GameState` as a singleton to maintain a single point of truth for the state of the application and remove the need to pass the state instance around for input processing, etc.

`logic.shape`

The shape package contains the classes that represent render-able shapes. The abstract `DrawShape` class provides a common interface for rendering each shape and allows us to store all drawn shapes in a single collection in the central game state.

`DrawShape` also provides a `color` property and requires the implementation of a `render` method. The `Line` and `Rectangle` classes contain the implementation details of storing the location information and rendering each shape.

`logic.tool`

The `tool` package contains the classes responsible for handling the state and logic specific to each drawing tool. The abstract `DrawTool` class allows all tools to be stored in a single collection in the central game state and requires a common interface for processing input and rendering menu items and drawing previews.

The `LineTool`, `RectangleTool`, `PolyLineTool`, and `FreeDrawTool` provide the implementation details for each mode. Additionally, `PolyLineTool` inherits from `LineTool` so it can reuse the drawing preview and implement its own input processing. Each tool's constructor calls upon the base `DrawTool` constructor with a reference to the current state and the name of the file where the tool's menu icon is located.

Note that because `GameState` instantiates the draw tools within its own constructor, calling `getInstance` within the common tool constructor return a null reference. This is addressed by having the state constructor pass a reference to itself to the constructor.

`render`

The render package contains the classes responsible for displaying graphical output to the user. The primary `PaintCanvas` class is based on the `ActiveRenderingExample` class from the book and is responsible for creating a buffered canvas, overriding the default mouse cursor, and rendering each frame when called on in the main game loop. The `render` method handles most of the responsibility for this class by drawing all the objects on screen in the following order: drawn shapes, the active tool's drawing preview, the menu, and the custom cursor. Rendering of shapes, the draw tool preview, and individual menu items is delegated to the respective object.

The `MenuItem` class provides a uniform way to reference all colors and tools from the canvas. It also provides a single method used to set the active color/tool when the item is clicked.

`render.images`

The images package is simply a storage location for the images used in rendering the draw tool menu items.