Lab 4: Moving Lines with the Command Design Pattern

Instructor: Mackale Joyner

Course Website: https://www.clear.rice.edu/comp504

Goals for this lab

- Become familiar with the Command Design Pattern

Important tips and links

NOTE: The instructions below are written for Mac OS and Linux computers, but should be easily adaptable to Windows with minor changes. For example, you may need to use \ instead of / in some commands.

Note that all commands below are CaSe-SeNsItIvE. For example, be sure to use "F18" instead of "f18".

1 Checkout Lab 4 SVN Repo

Check out your Lab 4 SVN folder from the remote SVN in IntelliJ from either the "Check out from Version Control" option on the welcome screen or the "Checkout from Version Control" option accessible under the top menu's VCS tab. Use the plus button in the pop-up window to add the following SVN folder:

```
https://svn.rice.edu/r/comp504/turnin/F18/NETID/lab4
```

with NETID replaced by your Net ID. Click the Checkout button to start downloading this repo.

You should see a lab3 directory created in your working directory with the source code for this lab.

2 Lab 4 Exercise

In this lab, you will use the Command design pattern to draw multiple lines that move across the canvas. Recall that the Command design pattern separates the client from the receiver. The command `execute` method will determine which receiver is affected by the command. In this exercise, the model will determine where to initially draw the moving lines and how fast each line should move across the canvas. For a moving line, the view will update the line position every .2 seconds. The controller will process REST method requests from the view and call the appropriate `DispatchAdapter` function in the model to update the line positions. The controller may also signal the adapter to switch strategies for all lines in the `LineWorld`. The Command design pattern should be used to implement this functionality. The next sections detail what you'll need to implement in the model, view, and controller.

2.1 Model

You will need to modify the `MovingLine` class in the model. The `MovingLine` constructor calls the `init` method to initialize the line with a location, random velocity, and random strategy. The line will begin moving once it is drawn on the canvas. The initial strategy is chosen randomly. If the strategy is the horizontal strategy, the horizontal strategy will move the line horizontally to the right. There is no collision detection with the right wall of the canvas. As a result, the line will keep moving right until the line moves off the canvas.

Each line should switch strategies when the user clicks on the `switch strategy` button. The horizontal strategy will switch to the vertical strategy. The vertical strategy moves the line vertically down. A line will keep this strategy until the `switch strategy` button is clicked again. The vertical strategy will switch to the composite strategy. The composite strategy has the horizontal and vertical strategies as children. Recall, the only job the composite strategy has is to call each child's update operation. All lines with the composite strategy WILL NOT MOVE. You cannot use any control logic in the `MovingLine` update method. The moving lines will keep using the composite strategy until the `switch strategy` button is clicked again. The composite strategy switches back to the horizontal strategy. The only `TODO` in `MovingLine` is to implement the `update` method. You should use a factory method to create `CompositeStrategy` objects since the lines don't care how it's created.

The `DispatchAdapter` sits in between the client (user) and the model. The `DispatchAdapter` will communicate with the controller to determine what type of command should be sent to all the lines in the `LineWorld`. Each `DispatchAdapter` method used to communicate with the controller and the model needs to be implemented. The singleton design pattern should be used where appropriate. Again, there should be no control logic in the `MovingLine` update method.

2.2 Controller

You'll need to implement the endpoints in the `LineDrawController` class. The controller will now interact with the model by using the `DispatchAdapter` to update the `LineWorld`.

2.3 View

You'll need to implement the `line`, `update`, and `reset` endpoints in `view.js`. The view ensures that all the lines start moving immediately after they are drawn on the canvas. The view updates the line position every .2 seconds. After implementing the view endpoints, you should be able to start up a local server, draw multiple moving lines, and switch strategies for each line every time the user clicks the `switch strategy` button. All lines with the `CompositeStrategy` should not move. The line should start moving again once it has switched strategies. Note: since each line's initial strategy is randomly chosen, not all lines will be stopped.

3 Demonstrating and submitting lab work

Show your work to an instructor or TA to get credit for this lab. They will want to see your updated files that you will commit to Subversion and your program running in the Chrome web browser. Labs must be checked off by an instructor or TA by the following Monday at 11:59pm.

Please don't forget to commit your work to your svn repository. To perform an svn commit, select VCS on the menu and click on "Commit...". Add a commit message and click on the `Commit` button.