CS 273 Laboratory 10: Debugger II

This lab gives you additional experience using the debugger in BlueJ. Its focus is debugging when a program has multiple methods and classes.

Preliminaries

1. Flag Application

In this laboratory, you will be using the BlueJ debugger to control the execution of a program that draws an American flag on a window. This flag is built from objects:

- A `Flag` object contains:
  - 13 `ColorRect` objects, which represents the stripes on the flag. A `ColorRect` is a rectangle that knows its color.
  - A `StarRect` object, which represents the square with stars on it. A `StarRect` contains:
    - ⇒ A (blue) `ColorRect` object.
    - ⇒ 9 `StarRow` objects. Each `StarRow` object represents a horizontal row of stars. It contains a set of:
      - → `Star` objects. A `Star` is a star-shaped `Polygon` that has a color. Because it is a `Polygon`, it implicitly contains an x- and y-position.

In other words, each star and rectangle that you see is an individual object that knows how to paint itself.

Each of the above classes implements:

- A constructor, that allows an object of that class to be created. Most of the above object types simply create all of the sub-objects.
- A `paint` method, that paints the object on the screen. Most of the above object-types simply invoke paint on all their sub-objects.

Note: there are many instance variables in this lab that are presently declared as `public`, that would eventually be changed to `private`. Leaving them public allows us to have access to objects' internal structure while debugging. We have declared these instance variables as `public//private` , as seen in the ColorRect class:

```
// a polygon that defines our rectangle
    public//private
      Polygon p;
```

(Thus the `private` is commented out.) This would allow you to easily find and change them to `private` at some point in the future.

2. Running the Flag Application

<u>In order to run this flag application</u> you must perform the following steps:

- Right click on the Flag file.
- Select "new Flag()".
- Accept the default object name by pressing "OK". (Alternatively, you can give it different name, if you'd like.) A red icon at bottom of main BlueJ window which represents the flag object you've just created.
- Right click on the newly created object and select "void view()"

3. Restarting BlueJ (should it hang)

BlueJ often refuses to let you recompile because it thinks that something is still running. If you get into that situation, and "terminate" does not work, you might try the following in order to "cleanly" restart after having been at a breakpoint:

- Clear all your breakpoints
- Click "Continue" to allow the flag window to be completely drawn
- Close the flag window by pressing its "close" icon
- At this point, it still may not let you recompile. If this is the case, perform these steps:
    - Delete the flag object (red icon in main BlueJ window) by right clicking on it and selecting "Remove" (e.g., "flag_1")

At this point, it still may not let you recompile. If this is the case, follow these steps:

- Create a new flag object by right-clicking on "Flag" and selecting "new Flag()".
- At this point, it often lets you recompile. If it doesn't, then you will have to kill BlueJ and restart it.

Laboratory

Create a folder named `lab10` in the `cs273` area of your network drive. Go to the course website and download the software for Lab 10 (`lab10.zip`). Unzip the software into the `lab10` folder you just created.

Part 1: Examine the Java files, and become familiar with the program by making some simple changes

1. Study the constructors in `Flag`, `StarRect`, and `StarRow` to see how they create their sub-objects. Examine the `paint` methods to see how they paint their sub-objects.

2. To help you become familiar with the objects, open the project and run it according to the instructions in “3. Running the Flag Application” on page 2.
3. Then, make changes to the source code so that the following happens. *Take care to make your changes so that they are easy to undo*.
    - The second star in each row does not get painted. (Remember, array indexing starts from 0, so the index of the second star is 1.)
    - The color of the middle red stripe is black.

checkpoint 1 (15 points): Show your lab instructor or assistant the executing application.

Part 2: Arrange your windows so that they will not overlap the debugger

The purpose of this part is to get the debugging window "out of the way" of the window. During the debugging of a `paint` method, if a window covers and uncovers the window, the system may send additional `paint` commands to the window, causing it to hang. When you're debugging the application, it is often stopped, and therefore is unable to repaint itself. Follow these steps:

1. *Undo* the changes that you made in part 1. You should now have a real American flag again.
2. Run the application.
3. Open editor windows for all the .java files except for `FlagViewer.java`. Move all of the editor windows so that none of them cover any portion of the running application—the top-left quadrant of the screen--but are still as visible as possible. You should move them all to the same part of the screen. During the course of this lab, you should avoid moving these windows over the space where the flag will appear.
   In later steps, whenever we want to debug the application's `paint` method (or something it calls), we will
    a. Clear any breakpoints that might be set.
    b. Set one or more breakpoints (e.g., the first statement of the paint method in `Star.java`).
    c. Run the application in the debugger.
4. Perform the steps a, b and c above. When you get to step b, set a breakpoint at the first statement of the `paint` method in `Star.java`. After you do step c, it should stop at that breakpoint; the flag should be painted, but without stars (because they are not painted before the breakpoint stops the application). (If the debugger covers the flag-window when it comes up, move the debug-window, terminate the application using the debugger’s `Terminate` button; then rerun the application.)

checkpoint 2 (15 points): When you reach this point, show your lab instructor or assistant the application and your `Star.java` file with execution stopped at the breakpoint. Then do several `Continue` operations. For each one, a single star should appear on the blue rectangle.

Part 3: Use the debugger to watch the flag get drawn object-by-object

Two of the most important debugger-commands are:

- `Step`[1] This executes one line in the current method. When the method finishes, it returns to the method who called it.
- `Step Into` If the current line contains a method-call, it drops into the code for that method, so you can watch what is happening inside the method. If the current line has no method-call, it behaves like "step".

By using these commands, you can watch the program executed on a "step-by-step" basis.

You can also terminate execution using the debugger's `Terminate` button.
Set a breakpoint at the first statement of the `paint` method in `FlagCanvas.java`. Run the application in the debugger. After the breakpoint is hit, use the "Step" and "Step into" buttons to get yourself in a position to do each of the checkpoints below. The only breakpoint that should be set during any of these is the one at the first line of the `paint` method in `FlagCanvas.java`. You should use the "Step" and "Step into" buttons to navigate to appropriate portions of the program as it executes.

checkpoint 3 (10 points): Show your lab instructor or assistant the stripes being drawn one at a time each time you hit "Step" buttons twice.

checkpoint 4 (10 points): Show your lab instructor or assistant rows of stars being drawn one row at a time each time you hit the "Step" button twice.

checkpoint 5 (10 points): Show your lab instructor or assistant stars being drawn one star at a time each time you hit the "Step" button twice.

If you have *fully* completed all the above checkpoints, you now have a grade of D- (60) for this lab.

Part 4: Examine the call stack

The *call stack* (which is labeled the "Call Sequence" in the BlueJ debugger) is the list of methods that are currently in the process of running: the first element of the call stack is the method that is currently running; then next is the method that called it; below that is the method that called it; and so on. You can examine the call-stack and find out where each call "came from" using the call-stack window. The *call stack* is located on the left had side of the debugging window brought up by BlueJ.

1. Terminate the application using the `Terminate` button in the debugger.
2. Run the debugger on the application as before, with a breakpoint set at the first statement in the `paint` method of `Star.java`. When you hit the breakpoint, click on the various methods in the call-stack window; watch each method be

[1] In other debuggers, this is sometimes called "Step Over"

displayed, with the cursor at the point of call. (Ignore "system" methods that you do not recognize, such as `dispatchEventImpl` or anything that starts with `sun.awt` or `java.awt`.

<u>checkpoint 6 (10 points):</u> Show your lab instructor or assistant the that you can examine the call-stack.

If you have *fully* completed all the above checkpoints, you now have a grade of C- (70) for this lab.

Part 5: Use the inspector to examine and modify objects

BlueJ has three variable windows that are shown when you are debugging. They are the *static*, *instance* and *local* variables and are located on the right hand side of BlueJ's debugging window.

1. Terminate the application using the `Terminate` button on the debugger.
2. Restart debugging as above, ensuring that the application window overlaps no others. Set a breakpoint at the first statement of the `paint` method in `FlagCanvas.java`. Use "Step into" and "Step" to get to the point in `StarRect.java` where it is just about to paint star row 4 (that is, the fifth row, because Java starts counting from zero).

3. Before it paints row 4, use the variable window to inspect row 4:
   - double-click on the row instance variable.
   - Click on one array position and click Inspect.
   - Click on star and then inspect.
   - Choose one star and click on inspect.
   - You can see the data now in the star object.

<u>checkpoint 7 (10 points):</u> Show your lab instructor or assistant that you can examine the objects in this way. Show the window indicating the value of npoints of a star object.

If you have *fully* completed all the above checkpoints, you now have a grade of B- (80) for this lab.

Part 6: More Object Modification

1. Close all Object Inspector windows and terminate the application using the debugger's `Terminate` button.
2. Create a new Flag object, but do not right-click and invoke the `view` method on the Flag.
3. Change the color of the third star in the fourth row by doing the following:
   - Find the object you want by navigating (via inspectors) down from the Flag object (e.g., flag_1 red icon) to find the object you want. That is:
     - o Right-click on the Flag object and select the `Inspect` button.
     - o Inspect the `starRect` object

    - Inspect the array of `StarRow` objects
    - Select and inspect the fourth `StarRow` object
    - Inspect the array of `Star` objects
    - Select the third `Star` and click the `Get` button
    - Enter a name of the object (or just use the default) and select `Ok`.
    - This `Star` sub-object will now appear along the bottom row of BlueJ's main window (next to the `Flag` object). You may need to close the inspector windows to see it.
  - You can right-click on the `Star` object, and apply a method (in this case, <u>setColor</u>) to it.
    - Java will bring up a dialog box that will prompt you for the argument(s) to the method; type it in. (BlueJ seems to need the `java.awt.Color.green` rather than just `Color.green`.)
  - You can now perform a `view` on the Flag object in order to see the effect.

  This is permanent change to that `Star` object. The star will remain green until that star (or the entire flag) is deleted. If, on the other hand, you created a new flag object, it would be colored normally.

4. Use a similar process to change the `Flag`'s top white stripe to be yellow and its bottom red stripe to be gray. **<u>Again, do not modify the code; rather modify the object using the debugger.</u>** In order to see the effect, you may need to force a repaint on the flag (e.g., by minimizing and unminimizing the flag window).

<u>checkpoint 8 (20 points):</u> Show your lab instructor or assistant your running application with one star that is green, one stripe that is yellow, and another stripe that is gray.

If you have *fully* completed all the above checkpoints, you now have a grade of A (100) for this lab.

Part 7: finish up

Close all windows. Log off. You're done!