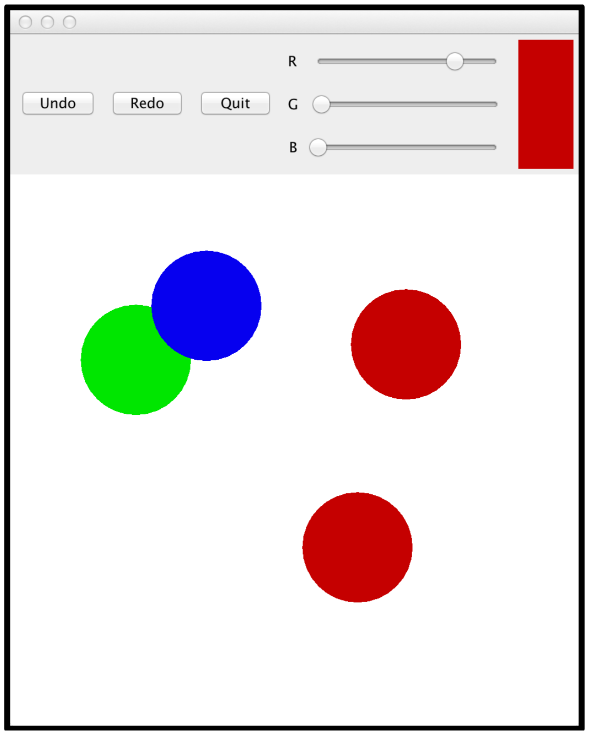
CS 46B
Lab: Stacks



In this lab you will experiment with applications of stacks. First you will use stacks to add UNDO and REDO features to a painting program. Then you will use a stack to detect and report errors in mathematical expressions.

As always: Work in pairs. One of you will be the "driver", who types into Eclipse. The other is the "scribe". Alternate jobs, week by week.

You will both submit lab reports.

Driver: At the top of your lab report write

CS 46B Lab Report
Your name
I was the driver, Xxxx Yyyy was the scribe.

Scribe: At the top of your lab report write

CS 46B Lab Report
Your name
I was the scribe, Xxxx Yyyy was the driver.

As you work through the lab assignment, instructions/questions highlighted in yellow should be discussed by both of you and addressed by the driver in the driver's report; instructions/questions highlighted in blue should be discussed by both of you and addressed by the scribe in the scribe's report.

PART 1: Undo

Create a Java project in an Eclipse workspace. Create a package called stacklab. Download StackLabStarters.zip from Canvas->Files. Unzip the zipfile and load the 4 files (Painter.java, UndoRedoPainter.java, and Circle.java) into your package.

The Painter app is a very simple painting program that lets you draw colored circles on the screen. The app has "Undo" and "Redo" buttons but for now they just print out messages. Your job is to implement undo and redo, without changing Painter.java.

Run the Painter app and play with it for a few minutes. When you click in the main panel, a circle is drawn. The circle's color is controlled by the 3 sliders, which set the color's red, green, and blue levels. Add some circles in various colors. Try clicking the Undo and Redo buttons and notice that they don't undo or redo.

Look at Painter.java. Unless you have studied Java Graphical User Interface ("GUI") programming, the source probably won't make much sense. That's ok. Look for the following:

- Instance variables `history` and `undoHistory`, which are both of type `Stack<Circle>`.
- Inner class `Canvas`, which has a method `paintComponent()`. This method loops through the `Circle` objects in the `history` stack, calling `paint()` on each one.
- Methods `undo()` and `redo()`, which just print messages.
- Methods `getHistory()` and `getUndoHistory()`, which return the 2 stacks.

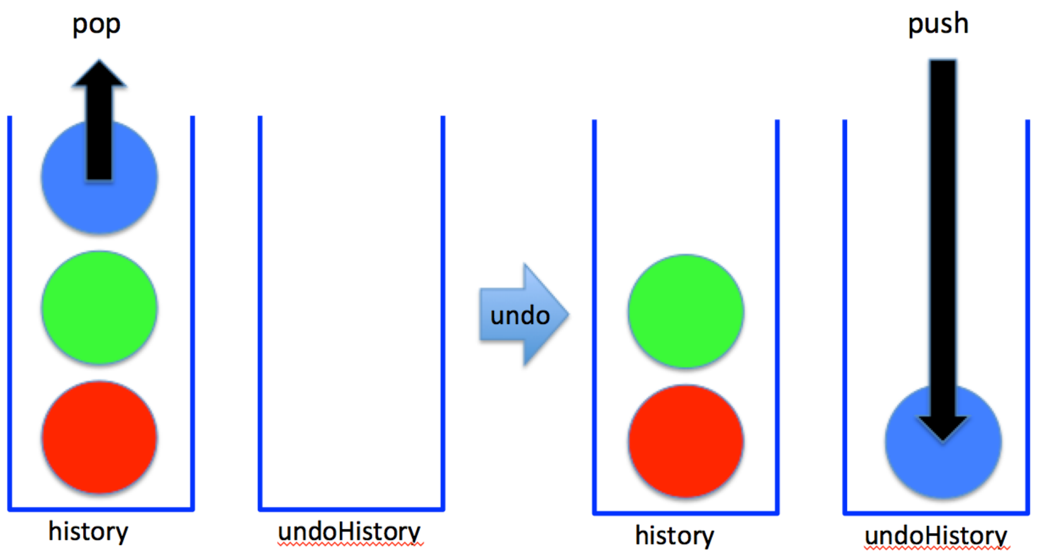
This app uses the `Circle` class to represent one circle. Look at the source. Notice that it models a center, a radius, and a color. Whenever the user clicks in the painting panel, one instance of `Circle` is constructed and pushed onto the history stack. Then the painting panel is cleared and the `Circle` instances are drawn on the screen, from the bottom to the top of the stack. (That's the loop you looked at in `paintComponent()`). Painting the screen happens too fast to see – it seems like just 1 new circle gets painted. To undo creation of the most recent circle, pop 1 element off the history stack and repaint the app. To undo creation of the 2nd most recent circle, pop a 2nd element of the history stack and repaint. And so on.

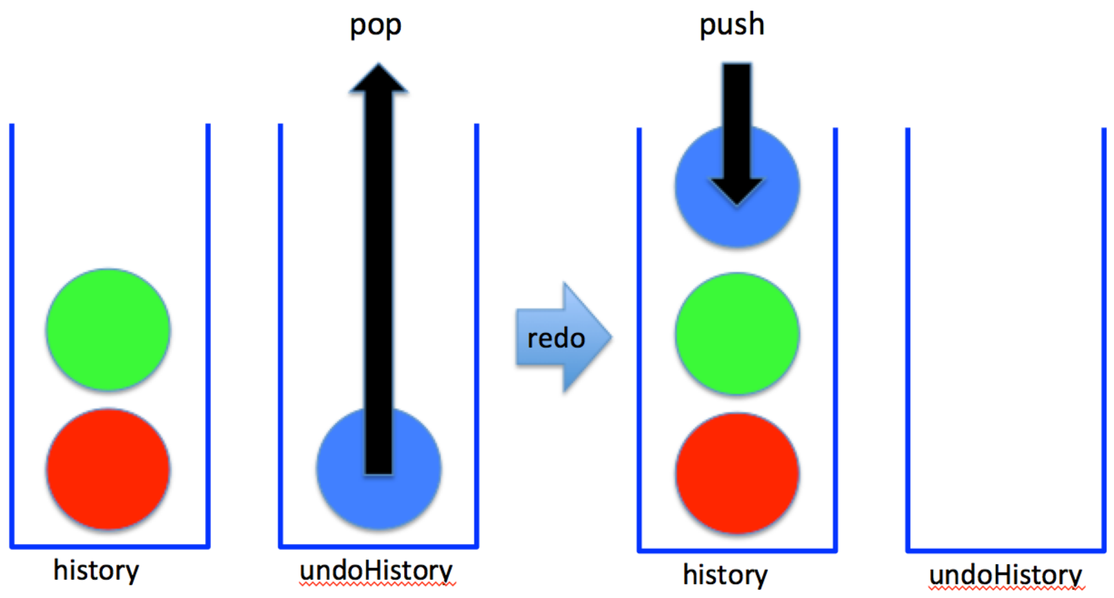
Painter.java is almost 200 lines of complicated code. Editing this code would be risky – you might break something and have trouble fixing it. Instead, subclass Painter.java (use starter file UndoRedoPainter.java). When you're done the subclass will only be around 35 lines, and you will be familiar with those lines because you're going to write them, so your job should be a lot simpler than modifying Painter.java.

Edit UndoRedoPainter.java, filling in method undo(). Your code should get the history stack from the superclass, pop off the top `Circle` object, and then call repaint(). To test, run UndoRedoPainter.java as an app. Add a red circle, then a green circle, then a blue circle. Driver: paste a screenshot into your report, and label it "1A". Scribe: If you click the Undo button, what should happen? Now click the Undo button. Driver: paste a screenshot into your report, and label it "1B". Scribe: is this the result you expect? (If not: change your expectations or change your code!) Driver: paste a screenshot into your report, and label it "1B".

PART 2: Redo

Some apps, like Word and Eclipse, support Undo but not Redo. So if you accidentally type ^Z (= undo), you're going to lose some work. "Redo" means "Undo the most recent Undo". Apps support "Redo" with a second stack, which we're calling the undo history. In undo(), `Circle` objects that are popped off the history stack during an undo operation should be pushed onto this undo history. In redo(), one `Circle` object should be popped off the undo history stack and pushed onto the history stack. See the figures below.





Think of a sequence of circle additions, undos, and redos that will verify if your code is working correctly. Scribe: describe your test sequence. Driver: start the app and do the test sequence. After each step, paste a screenshot into your report.

PART 3: Safety

When your app is working correctly, start it, add 2 circles, and then click Undo 3 times. Scribe: why does the app crash? Start your app again and then click Redo. Scribe: why does the app crash?

Fix your undo() and redo() methods so that they do nothing in these situations, rather than crashing. You will need to call the isEmpty() method of stack, which returns a boolean. The fix is just a few lines. Scribe: how did you fix the problems?

PART 4: Better Safety

The crash problems above happened because you requested impossible actions (undoing a circle that didn't exist, or redoing an undo that never happened). Obviously the worst way to deal with these problems is to do nothing, and hope users never request impossible actions. A better way is to detect the impossible requests and do nothing, as you did above. But the best way is to make it impossible for users to request impossible actions. This is easy with graphical apps, because controls like buttons can be enabled and disabled. The Painter superclass has methods setUndoButtonEnabled() and setRedoButtonEnabled(), which both take a boolean arg that determines whether the corresponding button will be enabled or disabled. These methods are inherited by your app subclass.

The Undo button should be:
1) Disabled when the app starts.
2) Enabled when a circle is added.
3) Disabled when an Undo causes the history stack to become empty.

(2) is already done by the Painter superclass. Implement (1) by adding a constructor to UndoRedoPainter that calls setUndoButtonEnabled(false). Implement (3) by adding code to your undo() method.

The Redo button should be:
1) Disabled when the app starts.
2) Enabled when an Undo is performed.
3) Disabled when a Redo causes the undo history stack to become empty.

Figure out where and how to do this.