CSE 143, Winter 2010
Programming Assignment #1: Rectangle-Rama (20 points)

Due Thursday, January 14, 2010, 11:30 PM

thanks to Mike Clancy of UC Berkeley for the original idea of this nifty assignment!

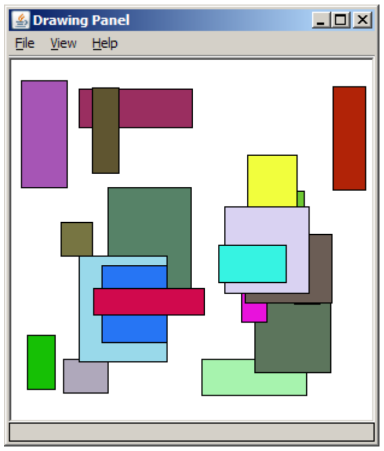
This program focuses on using the `ArrayList` collection class and basic object-oriented programming. Turn in a file named `RectangleManager.java` online using the turnin link on the Homework section of the course web site. You will also need the support files `DrawingPanel.java`, `Rectangle.java`, and `RectangleMain.java` from the Homework section of the course web site; place these in the same folder as your program or project. (If you use Eclipse, you may need to put these files in the `src/` subdirectory of your project if such a subdirectory exists.)

You should not modify the provided files. The code you submit must work properly with their unmodified versions.

Program Description:

In this assignment you will write the logic for a graphical program that allows the user to click on rectangles. The graphical part of the code has already been written for you; your code does not need to directly draw any graphics.

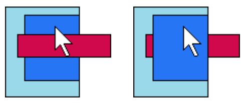
The main client program you should run is the provided `RectangleMain` class. When it runs, it will create a graphical window on the screen; this is an object of type `DrawingPanel`. The panel will display a list of rectangles. Each rectangle is represented as an object of the provided class `Rectangle`. Each rectangle's position, size, and color are randomly generated by `RectangleMain`. The overall list of rectangles should be stored and maintained by your code in the `RectangleManager` class you will write.



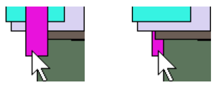
In the screenshot to the right, notice that some rectangles overlap and occupy some of the same (x, y) pixels in the window. In such a case, the rectangle created later is "on top" of prior rectangles and may partially cover them on the screen. You should think of the overall list of rectangles as having an ordering, where rectangles stored earlier in the list are closer to the "bottom" and ones stored later in the list are closer to the "top". This is sometimes called a 3-dimensional ordering or *z-ordering*.

The graphical user interface ("GUI") displays the rectangles and allows the user to click on them. Depending on the kind of click, one of four different actions occurs:

- If the user clicks the **left mouse button** while the mouse cursor points at a rectangle, that rectangle is moved to the very *top* of the z-ordering (the end of the list of rectangles).

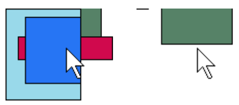


- If the user clicks the **left mouse button** while holding down the **Shift key**, while the mouse cursor is pointing at a rectangle, that rectangle is moved to the very *bottom* of the z-ordering (moved to the start of the list of rectangles).



- If the user clicks the **right mouse button** while the mouse cursor is pointing at a rectangle, that rectangle is removed from the list of rectangles and disappears from the screen.

- If the user clicks the **right mouse button** while holding down the **Shift key**, *all rectangles that occupy that pixel* are removed from the list of rectangles and disappear from the screen.



If you use a Mac with a 1-button mouse, you can simulate a right-click with a Ctrl-click (or a two-finger tap on the touch pad on a Mac laptop). If the user clicks the window at a pixel not occupied by any rectangles, nothing happens.

If the user clicks a pixel that is occupied by more than one rectangle, the top-most of these rectangles is used. (Except if the user did a Shift-right-click, in which case it deletes all rectangles touching that pixel, not just the top one.)

Note that **your code does not need to directly detect mouse clicks**. Our provided code detects the mouse clicks and responds to them by calling various methods on your `RectangleManager` object, as described on the next page.

Implementation Details:

Your `RectangleManager` class should store a list of rectangles as a **field of type `ArrayList`**. The various methods listed below will cause changes to the contents of that list. Remember to `import java.util.*;` to use `ArrayList`.

The following sections describe in detail each method you must implement in your `RectangleManager` class. For any methods that accept objects as parameters, you may assume that the value passed is not `null` or otherwise invalid.

`public RectangleManager()`
This constructor should initialize a new rectangle manager object. Initially the manager is not storing any rectangles.

`public void addRectangle(Rectangle rect)`
This method should add the given rectangle to the end of the rectangle manager's list of rectangles.

`public void drawAll(Graphics g)`
This method should cause all of the rectangles in the rectangle manager to draw themselves on the screen using the given graphical pen. You do not need to do this yourself directly by calling methods on the `Graphics` object; each `Rectangle` method already has a `draw` method that it can use to draw itself. You just need to pass on the `Graphics` object to each rectangle to do the rest of the work. Draw the rectangles in order from bottom (start) to top (end) of your manager's list.

Recall that in order to refer to type `Graphics`, you must `import java.awt.*;` in your code.

`public void raise(int x, int y)`
The graphical user interface ("GUI") calls this method on your rectangle manager when the user left-clicks. It passes you the x/y coordinates the user clicked. If these coordinates touch any rectangles, you should move the topmost of these rectangles to the very top (end) of the list. If the coordinates do not touch any rectangles, no action or error should occur.

`public void lower(int x, int y)`
The GUI calls this method on your rectangle manager when the user Shift-left-clicks. If these coordinates touch any rectangles, your method should move the topmost of these rectangles to the very bottom (beginning) of the list. If the coordinates do not touch any rectangles, no action or error should occur.

`public void delete(int x, int y)`
The GUI calls this method on your rectangle manager when the user right-clicks. If these coordinates touch any rectangles, your method should delete the topmost of these rectangles from the list. If the coordinates do not touch any rectangles, no action or error should occur.

`public void deleteAll(int x, int y)`
The GUI calls this method on your rectangle manager when the user Shift-right-clicks. If these coordinates touch any rectangles, your method should delete *all* of these rectangles from the list. If the coordinates do not touch any rectangles, no action or error should occur.

After any mouse click, the GUI also clears the screen and calls `drawAll` to re-draw all of the rectangles in your list.

Your manager's list stores `Rectangle` objects. Each `Rectangle` object has the following public methods:

`public int getX(),  public int getY(),  public int getWidth(),  public int getHeight()`
These methods return the rectangle's top-left x/y position, width, and height in pixels respectively.

`public Color getColor()`
These methods return the rectangle's color. You should not need to call this method.

`public void draw(Graphics g)`
Draws the rectangle on the screen using the given graphical pen.

`public String toString()`
Returns a text representation of the rectangle, such as `"(x=57,y=148,w=26,h=53)"`.
It can be useful to print a rectangle (or a collection of rectangles) with `println` to examine their state.

Development Strategy and Hints:

We have noticed that many CSE 143 students do not develop their code in stages and do not have a good idea of how to test their solutions. One of the most important techniques for professional developers is to write code in stages ("**iterative enhancement**" or "**stepwise refinement**") rather than trying to do it all at once. It is also important to test the correctness of your solution at each stage. We suggest that you write your constructor and `addRectangle` first, then `drawAll`. Then run the program to make sure that you can see the rectangles appear on the screen. Now write each click-related method one at a time and test each one individually to be sure it works before moving on to the next.

The program will not compile until your `RectangleManager` class contains all of the specified methods. But you can just write empty "**stub**" versions of the methods at first, then come back later and fill them in when you are ready. This will allow you to write each method and test it one at a time.

One part of this program involves figuring out which rectangle(s), if any, touch a given x/y pixel. You can figure this out by comparing the x/y position of the click to the x/y area covered by the rectangle. For example, if a rectangle has a top-left corner of (x=20, y=10), a width of 50, and a height of 15, it touches all of the pixels from (20, 10) through (69, 24) inclusive. Such a rectangle contains the point (32, 17) because 32 is between 20 and 69 and 17 is between 10 and 24.

If you're seeing compiler errors related to the `Rectangle` class, you may not have downloaded `Rectangle.java` or added it to your project properly. Java includes its own `Rectangle` class that is incompatible with our provided one.

If you have bugs or exceptions in your code, there are several things you can try. You can print out the state of your list and of each `Rectangle` object with temporary **`println` statements**. (Though this is a graphical program, `println` output does still appear on the console.) You should remove or comment out any such `println` statements later before you turn in the assignment. You can also use a **debugger** as found in jGRASP or Eclipse to pause the code at a breakpoint in the middle of an operation. Stepping through line-by-line can help you to see what has gone wrong.

A **sample solution** to this program is available on the course web site. You can use this to test your program's behavior.

Style Guidelines and Grading:

A major focus of our style grading is **redundancy**. As much as possible you should avoid redundancy and repeated logic within your code. One powerful way to avoid redundancy is to create "helper" method(s) to capture repeated code. It is legal to have additional methods in your `RectangleManager` class beyond those specified here. (We recommend that you declare any such methods to be `private` rather than `public`, so that code outside your class cannot call them.)

Your rectangle manager should maintain its list of rectangles internally in a field of type `ArrayList` as stated previously. You should not use any other data structures. Your code should use "generic" type parameters appropriately when creating any collection objects. You must declare such collections with a suitable element type within `<>` brackets. You should use the array list appropriately and take advantage of its ability to "shift" elements as they are added or removed. Your list should not store any invalid or `null` elements as a result of any mouse click activity.

Properly **encapsulate** your objects by making any data fields in your class `private`. Avoid unnecessary fields; use fields to store important data of your objects but not to store temporary values only used in one place. Fields should always be initialized inside a constructor or method, never at declaration.

You should follow good general Java style guidelines such as: appropriately using control structures like loops and `if/else` statements; avoiding redundancy using techniques such as methods, loops, and factoring common code out of `if/else` statements; properly using indentation, good variable names, and types; and not having any lines of code longer than 100 characters in length. (If you have any such lines, split them into two or more lines using a line break.)

You should **comment** your code with a heading at the top of your class with your name, section, and a description of the overall program. Also place a comment heading on top of each method, and a comment on any complex sections of your code. Comment headings should use descriptive complete sentences and should be written in your own words, explaining each method's behavior, parameters, return values, and assumptions made by your code, as appropriate.

Unless otherwise specified, your solution should use only material taught in class and in the book chapters covered so far.

For reference, our solution to this program is around **80 lines long** including comments (and has **36 "substantive" lines** according to our Indenter tool on the course web site). But you do not have to match this; it's just listed as a sanity check.