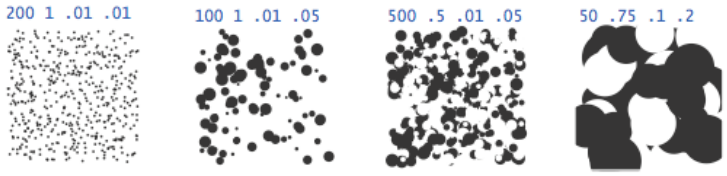
Lab01 - Princeton circles

Due: Fri Apr 3, 2015

Here's the assignment. It's from Princeton (la-dee-da): introcs.cs.princeton.edu/java/15inout/

26. Write a program `Circles.java` that draws filled circles of random size at random positions in the unit square, producing images like those below. Your program should take four command-line arguments: the number of circles, the probability that each circle is black, the minimum radius, and the maximum radius.



Looks like fun. Let's go! Step by step.
Like so many tasks, the key to programming is to partition the problem into manageable chunks. Divide and conquer! We'll try to that here in Lab01.

Step 1. Hello, Lab01

Create your `Lab01` class. Let's put `main()` in there.

```
public static void main( String[] args) {
   System.out.println( "Hello, Lab 01");
}
```

Step 2. JFrame

We don't have the code yet, but we can outline our steps in pseudo-code. Place these steps in your `main()` use comments.

```
main {
  create a frame
  create a panel for our circles, put it in the frame
  create some circles, adding them to the panel
  make the frame visible
}
```

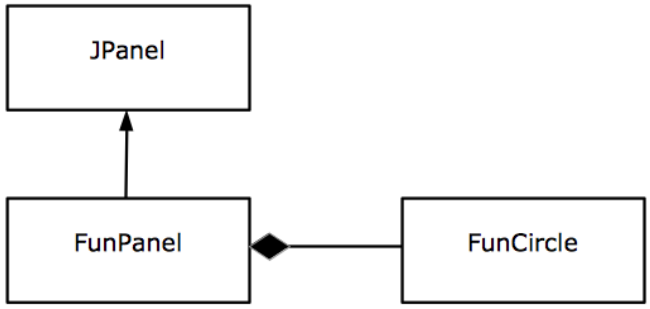
On page 367, the `JFrame` class is introduced in our text. Create your frame and make it visible. Set the background color to something exotic.

On page 880, there's a section on drawing shapes in `JApplet` or `JPanel` classes. Try to create a `JPanel`, and draw one circle in it. You pick the location, size and color.

Step 3. UML

Once we're able to create a `JFrame`, add a `JPanel` to the frame, and draw a circle in the `JPanel`, then we're ready to tackle the meatier Princeton circles problems.

This is a UML class diagram describing a design for our solution. We'll run with this after our class discussion on Ch 11 Inheritance.



From this diagram, we know that:

- `FunPanel` is-a `JPanel`, and
- `FunPanel` has-a `FunCircle`

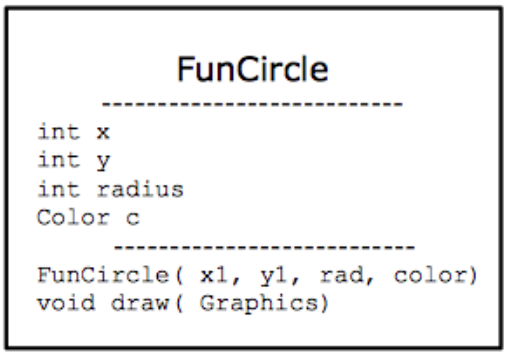
Step 4. FunPanel

Create your own panel that is-a `JPanel`. Override the `paintComponent()` method to draw just one circle. There's an example of this on page 898-899 in our text.
Don't forget to paint the superclass!

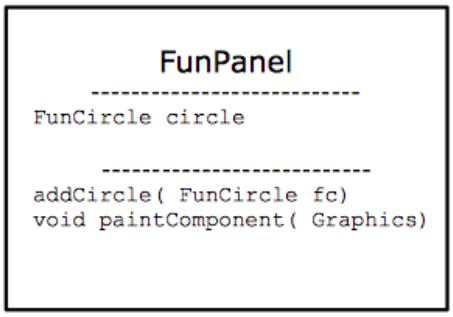
Step 5. FunCircle

Drawing one circle is nice. Drawing many is better. This step has 3 parts: a, b, c.

a) Create FunCircle class - To create a circle we need to store its location, radius and color. We need a way to draw it. Here's a more detailed UML diagram of `FunCircle`.



b) Change FunPanel - We also need to change `FunPanel` so that we can add circles to it and draw them. Here's the UML for these changes:



c) Change main() - In `Lab01`, change `main()` to create a `FunCircle` and then add it to the `FunPanel` using the new code you've written.

Step 6. ArrayList of circles

Wait a second… we're still only drawing one circle. To draw many circles, we need to change `FunPanel` to keep track of many circles rather than one. Use an `ArrayList` to do this. See page 515 for help doing this.

Here are the necessary changes to FunPanel:

- ➔ Add class variable: `ArrayList<FunCircle> circles;`
- ➔ Change `addCircle()` to use your list.
- ➔ Change `paintComponent()` to draw all the circles in the list.

To test this change, create and add a few circles in `main()`. And run it!

Step 7. Many, random circles

This lab is getting long, so I have provided you with code to randomly choose things like circle location, radius, and color. The class is `RandomHelper`, and it's in my k: drive.

In our final step, we'll create hundreds of circles in a loop and add them to our panel.
Remember the 4 Princeton experiments way back at the beginning? Let's create a method in `Lab01` to try each case: `princeton1()` through `princeton4()`.
My `princeton1()` method is on the last page as an example.

Once you get the feel of things, feel free to change the parameters to get results that you find fun and interesting.
thanks… yow, bill

```
public static void princeton1( FunPanel fp) {
    Dimension dim = fp.getPreferredSize();  // panel size

    for( int i = 0; i < 500; i++) {
        // get the location, radius and color of the circle
        Point p = RandomHelper.randomLocation( dim.height);
        int radius = 20;
        Color c = RandomHelper.randomColor();

        // create the circle and add it to the panel
        FunCircle fc = new FunCircle( p.x, p.y, radius, c);
        fp.addCircle(fc);
    }
}
```