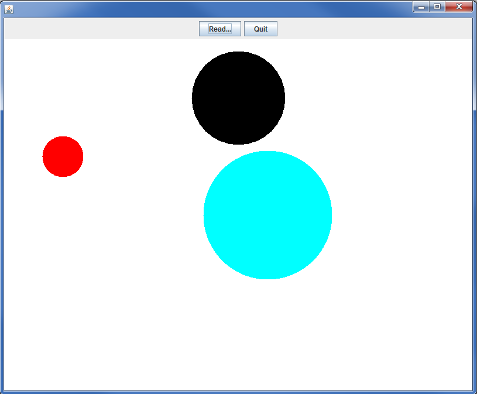
CS 46B

Lab 4[[1]](#footnote-0)



In this lab, you’ll use your knowledge of Java I/O and exceptions to complete a painting program that draws dots on the display.

**Timing Note:**

Please adhere to the timing schedule provided to complete each part. If you find that you cannot finish a section before the scheduled time or you are running behind, please seek extra assistance from your lab instructor or learning assistant. They will provide hints to help you complete the section on time. Additionally, during the last 45 minutes of the lab, you will complete the exit interview questions provided as the check-out quiz on Canvas. Therefore, ensure that your group completes the activity on time

[Section 1 - Fixing the Dot class](#_heading=h.1fob9te) (needed time 20 min)

[Section 2: Fixing the DotReader class and creating the DotException class](#_heading=h.26in1rg) (needed time 20 min)

[Section 3 - Fixing the DotDisplay class](#_heading=h.1fob9te)(needed time 20 min)

[Section 4 - Debug Your Program(](#_heading=h.3j2qqm3)needed time 30 min)

Estimated time: 1.5 hrs

**Lab Grade**

Labs are a core component of the course, and a lot of your learning happens when you have to take what you learned in the lecture and apply it in practice. Labs are a required part of the course, and missing more than two labs will result in failing the course.

# Your lab grade will be based on three components:

# **Check-in Quiz 2 points:** This is an easy quiz about the material covered during the week and takes 15 minutes at most.

# **Collaboration 3 points + Project Compile and Execution 3 points:** Your lab instructor examines and grades your project as a team on your laptop to make sure it compiles and executes. Also, make sure that you collaborate with your teammate(s). Working in groups to solve problems is an important skill that computer scientists embrace. It is important not to leave group members behind or to let each just do the work independently. Also, all team members need to submit their Jar files on Canvas. Your files will be checked randomly. If your jar file is missing or does not compile and run as expected, 50% of your grade will be deducted. So please make sure that your Jar file is complete and sound before you upload it on Canvas.

# **Exit Interview 2 points:** To receive credit for this lab, your group will complete an exit interview. To get an idea of the kinds of questions that will be asked, look at the questions highlighted in blue that you encounter as you complete the lab instructions. ***To help you prepare for the exit interview, I suggest tackling the questions when you encounter them in the lab instruction, discussing them as a group, and then writing down what you think the answer is.***

# **Important note:**

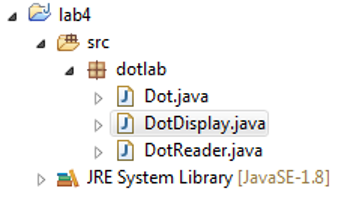
For each lab, your group will complete an exit interview. Completing this interview will give you credit for the lab. If you are absent from a lab, you can make up the interview during your lab instructor's office hours (only for those 2 allowed missing labs). Note that you can miss at most two interviews. If you miss more than two, you will fail the course.

The exit interview will be approximately 10 minutes, and you will rotate through who is the group leader (in charge of answering the questions). These interviews are not so much about getting the right answer but serve as a way for you to demonstrate how you are thinking about the problems and how your understanding of the material is evolving. They also provide an opportunity for the lab instructor or learning assistant to help me, the instructor, understand where you are struggling. You will get credit for completing the interview provided that you have made a good-faith effort to complete the lab.

One goal of the labs and homework assignments is to support you in learning how to write code. When working in small groups, it is important that all of you understand the code that is being written. If you are relying on your group mates to do all the programming, the exams and grading interviews will be challenging. **Also, note that your lab exams will be completed and graded individually.** With that said, make sure you understand each lab activity well and feel confident that, if needed, you can complete that individually, or if you find yourself understanding the concept quickly, please slow down and try to get your group mates up to speed. The best way to really understand the material is to explain it to someone else. And “explain it” doesn’t mean just showing them your code or telling them exactly what to do; it means helping them figure out how to do it on their own.

# Part 0: Setup

Create an eclipse workspace for lab 4. Create a new Java Project called 'lab4'. In lab4->src, create the dotlab package and drag the Dot.java, DotDisplay.java, and DotReader.java starter files into the package. Your Package Explorer should look like this (possibly with red error marks):



# Section 1: Fixing the Dot class

You are going to update the Dot class to properly read the essential data to draw a dot on the screen.

## Step 1: Adding Instance Variables

In Dot.java, below the LEGAL\_COLOR\_NAMES code, add the instance variables necessary to store the

* color name (a String),
* x and y coordinates (ints),
* radius (another int) of a dot.

## Step 2: Adding Getter Methods

Write a public getter method for each of the instance variables. The getters should be called

* getColorName()
* getX()
* getY()
* getRadius()

## Step 3: Adding a Constructor

Immediately after the instance variables, make a constructor for the Dot class that takes a color name, an x value, a y value, and a radius and sets the corresponding instance variables to these values.

## Step 4: Checking for valid colors

The LEGAL\_COLOR\_NAMES array contains the only strings that are allowed as values of colorName. In your constructor, before setting your color instance variable, ensure that it is a color that can be found in LEGAL\_COLOR\_NAMES. If the color cannot be found in LEGAL\_COLOR\_NAMES, throw an IllegalArgumentException whose message reports the name and explains that it is a bad color name. Do I need to declare that the Constructor throws an IllegalArgumentException? Why or Why not?

### Testing the Constructor

Test that you're catching bad color names correctly. Create a main() method in Dot.java that creates 2 instances of the Dot class. The first Dot should use a legal color name with some reasonable values for its x, y, and radius, and should be created without a problem. The second Dot should use an illegal color name with some legal values for its x, y, and radius, so constructing it should cause an exception. What exception message is printed to the console?

## Step 5: Updating the toString method

Printing out an object can be extremely useful during debugging because it can provide clues about the cause of a bug. In main(), delete the code that creates your illegal Dot, and print out the contents of your legal Dot using .toString(). (Remember, .toString() simply returns a string, it doesn't actually print it out). Does the println method automatically call the toString method? What does the toString method print out?

### Initial toString method

That's weird. That doesn’t look like a Dot. Find the API page for the Object class and look up the toString() method, which Dot inherits. (Note: there are lots of ways to learn what Object’s toString() method does, including asking someone, googling, or checking StackOverflow. The only official, guaranteed correct, source of the information is the API page. This is why I’ve been having you all read the APIs during the in class exercises.) Summarize what the documentation explained in your own words. Why do you see what you see when you call .toString() on our Dot object?

### Overriding toString method

Override the toString() method so that it'll print out something useful for your Dot object. In Dot.java, write a toString() method that will provide the dot color, x and y coordinates, and radius in a readable format. How did you decide what format the String should take?

### Testing toString method

Make sure that the toString() method is working correctly. In your main() method of Dot.java, print out a valid Dot object. What happens when you print a valid Dot object now?

# Section 2: Fixing the DotReader class and creating the DotException class

The next goal is to be able to read descriptions of dots from a file (the file 3dots.txt is in the starter folder), and convert the descriptions to instances.

### Finishing the DotReader class

Each line of a file will have 4 properties separated by commas: the color name, an X position, a Y position, and a radius.

1. In DotReader.java, write the method readDot() that returns type Dot, takes no arguments, and throws IOException.
2. Within the readDot() method, use the BufferedReader instance variable provided to read a **single line** and break it at the commas into an array of Strings. For example, the input line “BLUE,100,200,300” should result in the array { “BLUE”, “100”, “200”, “300” }. Note there are no blank spaces in the input.
   1. If you have trouble with parsing the line, try looking up **split()** in the String API page; **use** “,” as the regex arg.
   2. If the line read from the BufferedReader is null, indicating end-of-file, your readDot() method should also return null to indicate end-of-file.
3. Remember that the Dot ctor needs 3 int arguments, but if you use **String’s split()** method, you get an array of Strings. You’ll have to convert (for example) the String “100” to the int 100. If you have trouble with converting a String to an int, look up the **parseInt()** method on the API page for Integer.

### Creating a DotException

But wait, what if you have an irresponsible user who tries to add more properties, like flavor, onto one line? (Maybe that user likes Dots candies.) You have to check that your array has only 4 properties in it, no more, no less. If the array is a different size than what we expected, we'll throw our own exception: a DotException.

1. Create a new class called DotException that's a subclass of Exception
2. It should have a constructor that accepts a single String argument, and calls the superclass constructor using that String argument.

Is the DotException a RuntimeException or a checked exception? Why did we choose to make it that type of exception?

### Updating DotReader to throw a DotException

Now that you have your own specific Exception, use it in your DotReader class. **In DotReader.java**, make sure that the array from the parsed line is of exactly size 4. If it's not, throw a new DotException that gives a brief description and the line from the file that's causing a problem.

You’ll have to change the readDot() declaration to declare that the method throws DotException. It’s ok to have multiple exception types after “throws” – just separate the types with commas. To throw the exception, do something like this:

DotException de = new DotException(*a good message*);

throw de;

Why is it useful to have created our own DotException? What happens if the readDot() declaration doesn’t say that the method throws DotException?

# Section 3: Fixing the DotDisplay class

Now that you have a functional Dot and DotReader class, put them to work to make the application complete.

Notice that DotDisplay doesn’t compile because there’s a call to a read() method that doesn’t exist yet. You probably won't understand much about this source. That’s ok, all you need to do is give it a read() method.

## Implementing the read method

In DotDisplay.java create a read() method like this:

private void read(File f) throws IOException, DotException

{

FileReader fr = new FileReader(f);

BufferedReader br = new BufferedReader(fr);

// (you’ll write code here)

br.close();

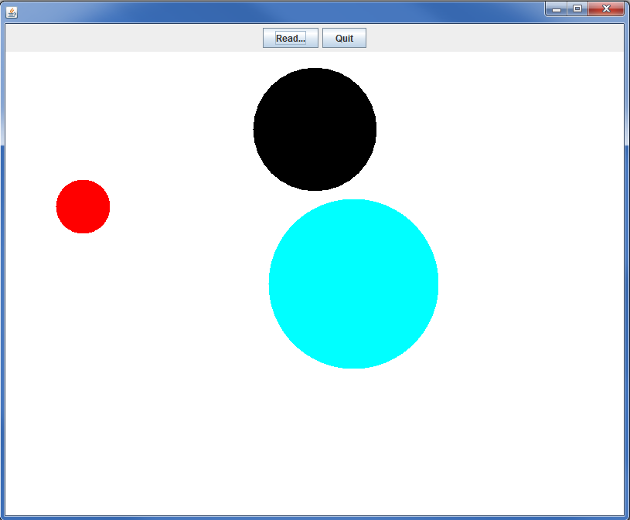
fr.close();

}

Change the read method so that it uses a DotReader to read every dot in the file. For every dot that you read, call addDot(*yourDot*). This call passes the dot to the display code

### Test you program

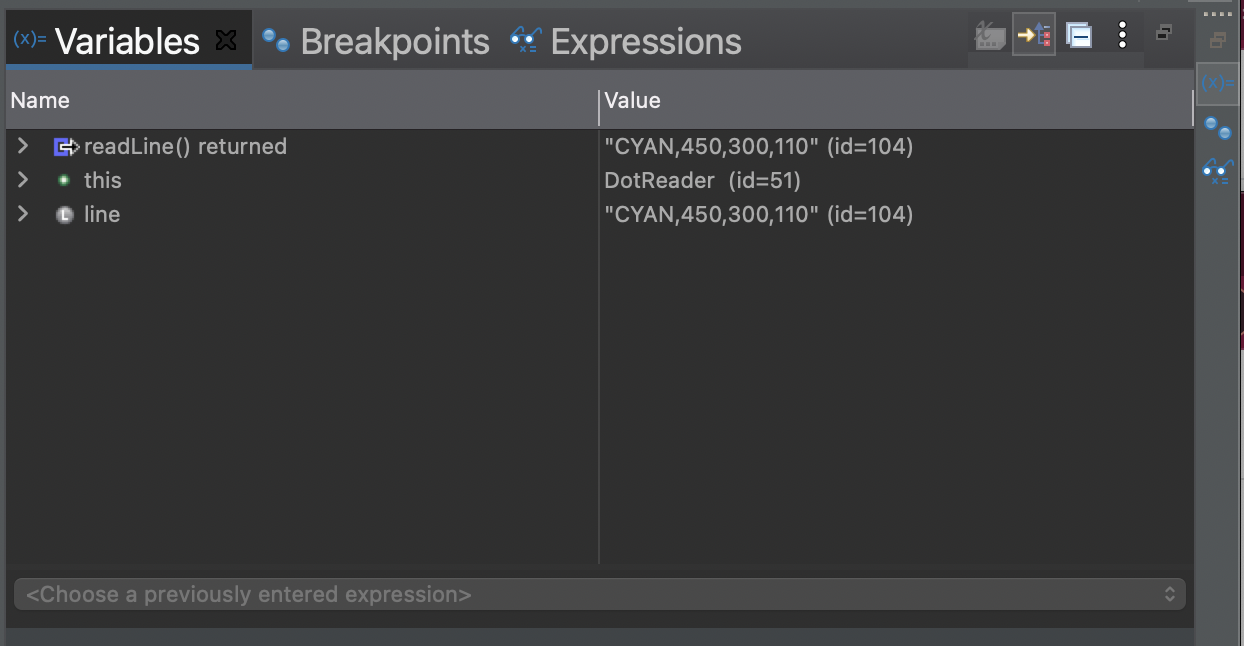
Run DotDisplay.java and click 'Read...' Use the navigator to find and open the '3dots.txt' that you downloaded with the rest of this assignment. You should have a window that looks like this:

****

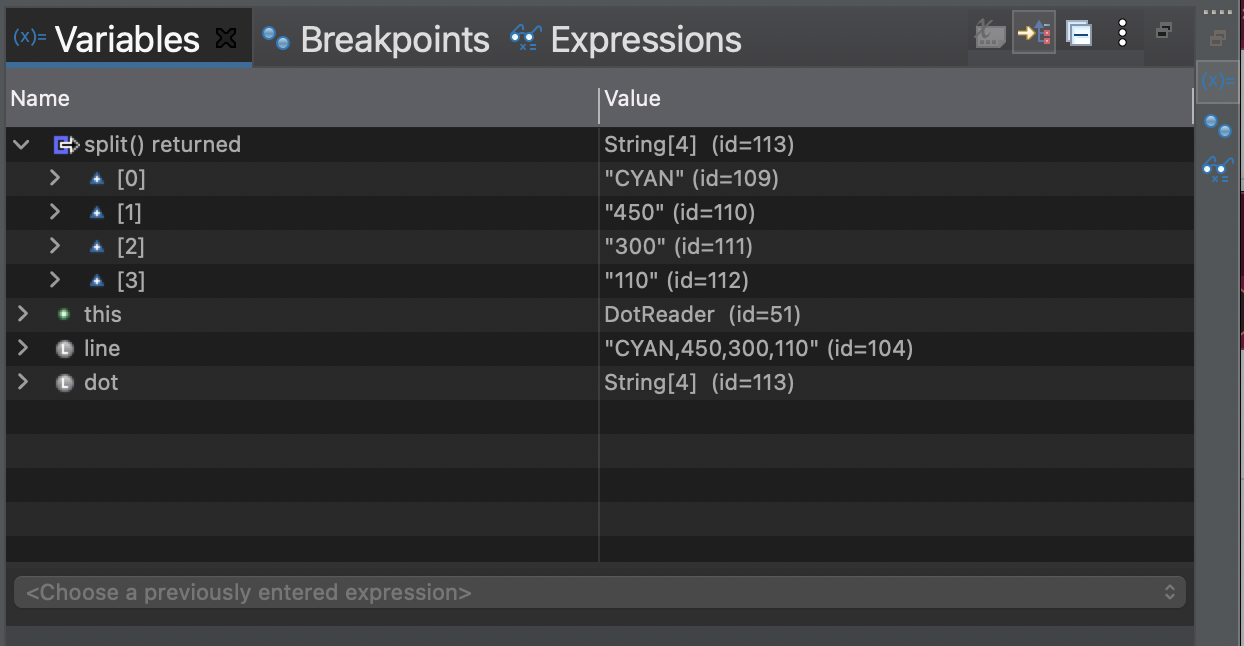
### Section 4: Debug Your Program

You might not get your program to work right away. Even if you do, let’s add a breakpoint at the first line in the readDot method in your DotReader class (the first line probably looks something like this… String line = br.readLine();

Run the program in debug mode. Click the step over button when you encounter the break point. What happened to the Variables in the debug window? They should look something like the image below



Now press the step over button until you get past the call to the split method. What does the Variables window look like now? It should look something like the image below.



How can using the debugger help you figure out potential problems with the code you just wrote?

### Testing your Exceptions

Great! Now make sure your IllegalArgumentException and DotException are working properly.

1. Edit the 3dots.txt file so that one of the lines causes an IllegalArgumentException.How did you edit the 3dot.txt file? What stack trace did Eclipse display when the exception was thrown?
2. Now undo the changes to 3dots.txt, then edit it once more so that one of the lines causes a DotException.How did you edit the 3dot.txt file? What stack trace did Eclipse display when the exception was thrown?

# Saving your work

It can be a good idea to back up your work in case you ever accidentally delete your eclipse-workspace. You can export your work as a jar file similar to lab 1.You can import jar files directly into Eclipse. Later this semester we will learn better ways to backup your work using version control and GitHub.

**Please submit your lab4.jar file on Canvas.**

1. Modified from material provided by Dr. Philip Heller and Dr. Cay Horstmann and Dr. Chakarov [↑](#footnote-ref-0)