CSE 11 **START EARLY!**
Fall 2013
Program/Homework Assignment #2 (100 points) -(Corrected Version)

Due: 11 October 2013 at 11pm (2300)
Book Exercises Cover Chapters: 3 - 4
This is a combination of written responses (25 points) and a larger program (75 points)

Note that in all commands, the "$" indicates the shell prompt. You do not type in the $, only what comes afterwards. You hit "enter" or "return" to execute a command in the shell

Please note that: JAEA = "Java, An Eventful Approach". A problem as JAEA: Exercise 1.5.2 means do exercise 1.5.2 in your textbook.

Exercises are optional, not graded.

Exercise #1: Browse to http://blog.linuxacademy.com/linux/ssh-and-scp-howto-tips-tricks/ to learn about ssh (secure shell) and scp (secure copy). These are very useful for remote access to servers and copying files from one unix host to another.

Homework Problems:
The goal of this assignment is to create a standalone class and a program that uses that class. You should do problems 1-5 first, then do problem 6.

Answers for Problems 1 – 5, should be placed in a file call `PR2.txt.` The top of file you have your student ID number, Name, Login similar to the following (replaces with Id, Name, and Login) . Make sure that ID, NAME, LOGIN appear on separate lines.

```
ID: A1111111
NAME: Philip Papadopoulos
LOGIN: cs11e
```

Make sure that answers to each problem are clearly labeled in your PR2.txt file. Something like

Problem 1:
<answers to problem 1>

Problem 2:
<answers to problem 2>

would be sufficient. Make it is easy for the Reader to see your answers.

Problem #1: **(5 points)**

JAEA: Exercise 3.4.2

Problem #2 (5 points)

What is the value of `y` at the end of the following code segment?

```
int x, y;
x = 34;
y = 10;
if (x >= 20  && (y = 18) == 18) {
     y = 17;
}
```

Problem #3 (5 points)

What is the value of `y` at the end of the following code segment? And What is printed on the console? Why?

```
int x, y;
x = 34;
y = 10;
if (x >= 20 && (y++ > 10)) {
     System.out.println("Welcome to CSE11!");
}
```

Problem #4 (5 Points)

JAEA: Exercise 4.8.5

Problem #5 (5 points)

JAEA: Exercise 4.8.8

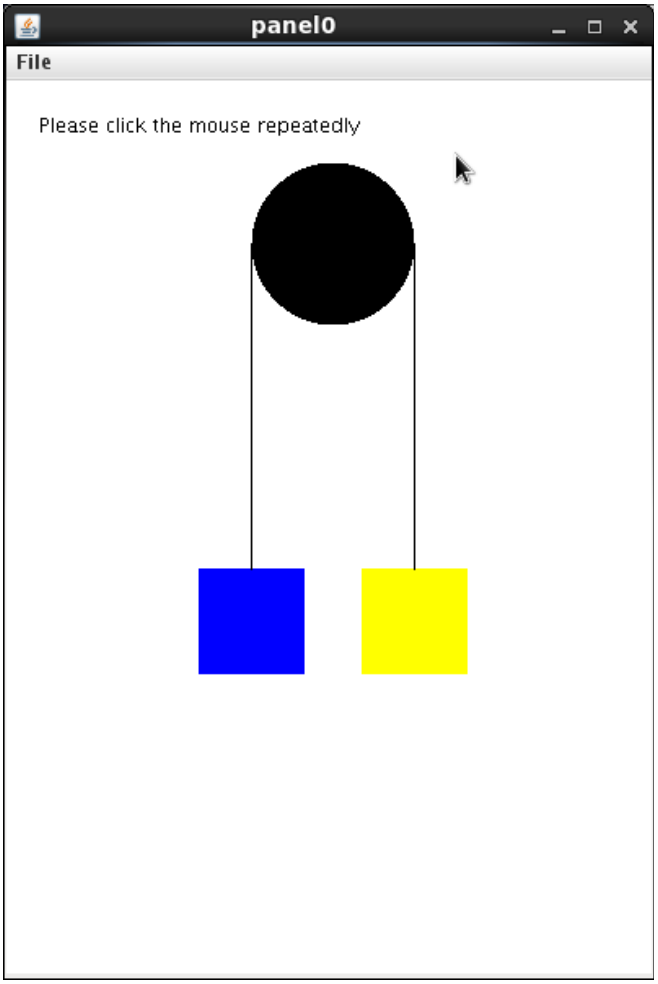
-- Continued on next Page --

problem #6 (75 points)

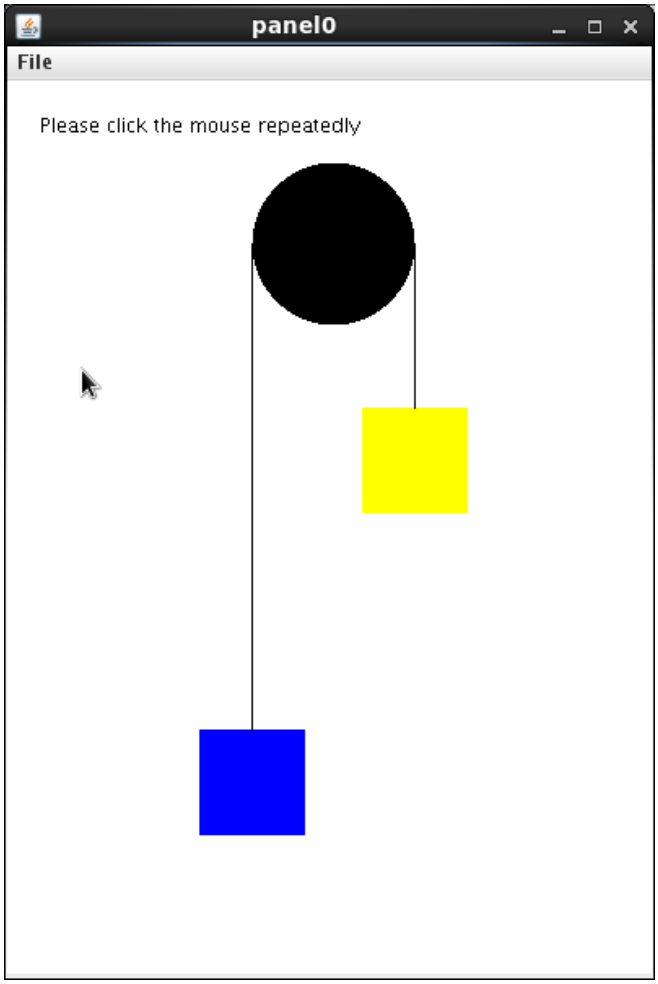
The Balanced Rope and Pulley

READ THE ENTIRE ASSIGNMENT BEFORE STARTING

You are to write a java program called BlockAndTackle with a supporting class called WeightBox. Together, these two classes will "simulate" a balanced rope and pulley system using objectdraw. When your program is complete, your program will look like the following two drawings.



Drawing 1: Initial Configuration of Balanced Pulley



Drawing 2: Positions after mouse has been clicked a few times on the left half of the screen

The Weight Box Class

You begin by creating the WeightBox class. You should download WeightBox.java from the class website. It is an *outline* of the WeightBox class in which all methods and constructors are stubs. It will compile, but it has no useful functionality. One of your tasks will be to fill out the class definition. The outline enables you to more quickly write and test some working code.

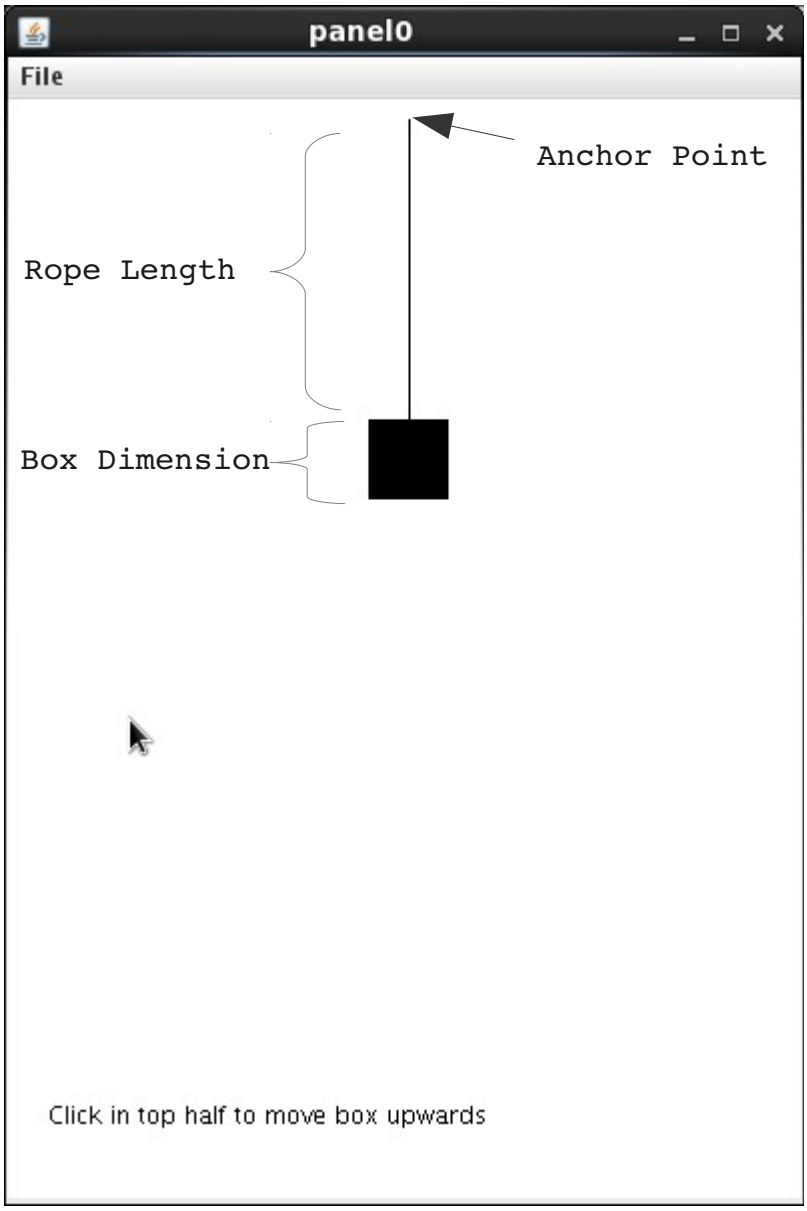
It is important to understand that WeightBox is not a program, and it can be used with other programs

(not just BlockAndTackle, the main program for the full assignment). We'll describe the class and an initial *testing* program called WeightTest.java. The testing program will help you develop the WeightBox class before your write BlockAndTackle.

The following picture shows what is displayed by the WeightTest program when WeightBox has been properly implemented. If you just have the outline of WeightBox, you will see an empty screen with just the instructions.

WeightTest.java is also provided to you and helps you follow a good development methodology. That methodology is based upon testing small functional components and gradually adding more complexity. Instead of building the complete program all at once, we build in incremental steps and use small tests to verify the correctness at each stage of development. In this case, WeightTest is a test driver program that can assist us in developing the full WeightBox class. The following picture is annotated to help you understand the description of WeightBox:

-- Continued on next Page --



Drawing 3: WeightTest using the WeightBox class. Click in the top half of the window to raise the WeightBox. Clicking in the bottom half lowers the box.

In English: A square box is suspended from a rope, that has an initial length. The rope is anchored at the anchor point. The rope can be shortened or lengthened and the box moves with the rope. The anchor point never moves.

You should implement the following constructor and methods. You will need to choose class variables to keep track of specific instances.

WeightBox Class description (methods and functionality)

Constructor

The constructor for WeightBox is as follows

```
public WeightBox(Location anchorPoint, double ropeLength,
```

```
        double boxSize, DrawingCanvas canvas)
    anchorPoint is a fixed end of a rope. It does not move
    ropeLength is the initial length of the rope
    boxSize is the width and height, in pixels, of a box
```

it is suggested that the rope be implemented using the Line class of objectdraw. The box can be implemented as a FilledRect. The box of boxSize x boxSize is placed at ropeLength pixels below the anchorPoint. A line is drawn between the anchorPoint and the top of the box. Notice that the center of the box is directly below anchorPoint.

Methods

```
public void hoist(double deltaY)
     This raises or lowers the box by deltaY pixels.
     If deltaY < 0 it raises the box.
     The rope should be shortened by deltaY amount.
     The hoist method should never raise the box above the
     anchor point


public double getRopeLength()
     returns the current length of the rope.  It should
     never be negative.

public void setColor(Color newColor)
     sets the color the box to newColor
```

Developing the WeightBox class

The supplied WeightTest.java does not use getRopeLength() or setColor(). In other words it is not a complete test driver. This means that you can develop your WeightBox class in stages.

1. First, create the constructor Compile this first version of WeightBox.java and then use it in WeightTest. This will enable you to verify that the constructor is working properly. WeightTest will use your partially-implemented class to create the initial drawing (centered, box of the correct size, rope drawn correctly. Make certain that the drawing appears as it should.
   Since you have an empty hoist() method, clicking on the mouse will do nothing, and that is expected.
2. Now modify your WeightTest class, to implement a working `hoist` method. You can use WeightTest to verify/validate that your hoist method works correctly. Check, for example, that the box is never hoisted above the anchor point. Check that the rope is lengthened/shortened properly.
3. Do NOT go beyond steps 1 and 2 UNTIL they are working correctly You must learn to debug your programs in stages and it is tempting to write all methods first and then debug. That is a bad approach. Dividing up a complicated program task into smaller and testable sub-tasks is a much better way to go. This staged development approach is essential when

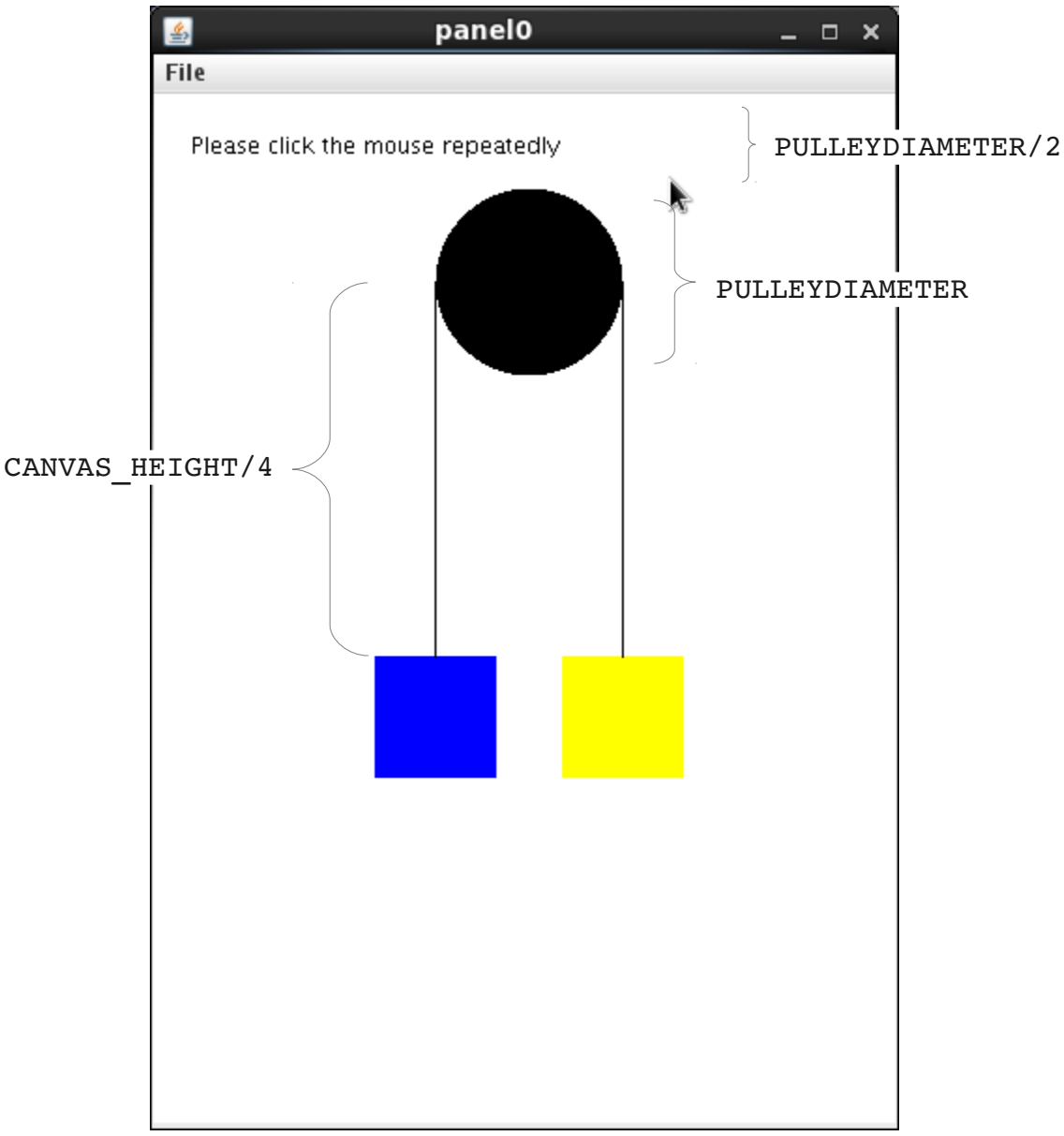
you get to larger problems.

4. Now that the constructor and hoist method are built and initially tested, you need to add functionality to both the WeightBox class (getRopeLength and setColor must still be implemented) and the WeightTest program. I would suggest implementing and testing setColor first. To finish this step you have to modify both WeightBox and WeightTest. At this point, all of the methods above should be implemented.
5. Make sure that your versions of WeightTest and WeightBox work properly. As an additional step of testing, you might modify WeightTest again to create two different instances of WeightBox, of different colors. Have them both go up/down on a mouse click

BlockAndTackle Program (A Balanced Pulley)

BlockAndTackle uses WeightBox, too. If you look at Drawing 1, above you can see that there are two WeightBox instances, one on each side of the pulley. In between them is a circle (the pulley) that appears to be a simple circle (actually that is all it is). You can (and should) use WeightTest.java as a basis for for your BlockAndTackle. Java program. Below is an annotated drawing the defines some constants. They are referenced below in the specifications of BlockAndTackle.java. The specifications may seem overwhelming at first, but all of them make sense.

-- Continued on next Page --



The following CONSTANTS must be defined in BlockAndTackle.java. Please put each constant declaration on a separate line (see the example in WeightTest.java). If you change hte numbers in the constants, your program should behave differently. In grading your program, we will use different values for these constants and recompile to verify that you are using these constants. We will only check with reasonable values of these constants.

CANVAS_HEIGHT = 600;
CANVAS_WIDTH = 400;
BOXWIDTH = 65;
PULLEYDIAMETER = 100;
STEP = 33;

CANVAS_HEIGHT, CANVAS_WIDTH, BOXWIDTH, and STEP are already defined in the WeightTest.java program supplied, Their meanings in BlockAndTackle are identical. STEP is

how many pixels the WeightBoxes will move each time the mouse is clicked.

Functionality of BlockAndTackle

1. The canvas should be CANVAS_WIDTH x CANVAS_HEIGHT
2. Create a pulley that has Diameter PULLEYDIAMETER
3. The Pulley should be centered on the canvas.
4. The top of the pulley should be PULLEYDIAMETER/2 from the top of the canvas
5. Create two WeightBox instances.
6. The anchorpoints of the two WeightBoxes should be at either side of the pulley as illustrated in the drawing
7. Color the left WeightBox blue, the right WeightBox yellow
8. When you click on the left side of the canvas, the left WeightBox should be lowered STEP pixels, the right WeightBox raised STEP pixels (See 10 below for limits)
9. When you click on the right side of the canvase the exact opposite motion should occur (i.e, rightWeight box is lowered, left WeightBox is raised)
10. Neither box should be raised above the bottom of the pulley. Use the getRopeLength() method to figure out if raising STEP pixels is too large a step. For example, if a box is 7 pixels from the bottom of the pulley and STEP is 33, you would only raise 7 pixels. You use the hoist() method to raise (or lower) the box.
11. The boxes are considered "balanced" because if you raise one box by K units, you lower the other box by exactly K units. Please note that mathematically, K <= STEP. K == 0 when one box has reached its maximum height.
12. Please put in comments at the top of your program, the same, ID, NAME, LOGIN that is in your PR2.txt file. This will make grading MUCH easier. You will lose points if you do not do this.

Some hints on building BlockAndTackle

1. Copy your working WeightTest.java to BlockAndTackle.java. Edit appropriately so that it runs correctly before you make any changes.
2. Create the additional constants for BlockAndTackle, and use them to properly create the initial configuration, including coloring, initial rope length, etc.
3. modify the onMouseClick() method to raise one box, lower the other box appropriately. At this stage, test without insuring that boxes only go to the maximum height
4. modify the onMouseClick() method to insure that the neither box goes above the bottom of the pulley. Use the getRopeLength() methods in your WeightBox instances to compute how far you you could hoist the boxes.
5. Try some different constants in BlockAndTackle, recompile, and then test that your program still works as expected.

Make Copies of your Program Files as you go along, If you make a big mistake you can go back to the previously working code

Turning in your Program

YOU MUST BE ON THE LAB MACHINES FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the "bundleP2" program that will turn in the following three files

```
PR2.txt
WeightBox.java
BlockAndTackle.java
```

No other files will be turned in and and then **must be named exactly as above.** BundleP2uses the departments standard turnin program underneath.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

```
$ /home/linux/ieng6/cs11e/public/bin/bundleP2
```

The output of the turnin should be similar to what you saw in your first programming assignment.

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Suggestion: if you have classes that compile and do some or most of what is specified, turn them in early. When you complete all the other aspects of the assignment, you can turn in newer (better) versions.

Don't forget to turn in your best version of the assignment.

Frequently asked questions

Can I do this as an applet? No. BlockAndTackle should be a java program.

Do I really have to use the command line? I used Dr. Java in my high school class. Yes! Although we will have graphics output on many programs, we are not using any particular development environment.

What if my programs don't compile? Can I get partial credit? Depends on how close your program is to being something that should be working. You are highly encouraged to turn in code that compiles, even if you cannot implement all of the functionality.

Does WeightBox extend WindowController? No! It uses objectdraw, but the class does NOT extend WindowController.

It's possible to create BlockAndTackle and have it meet all the program requirements without using the WeightBox class. Can I do that? No! Part of this assignment is learning about creating and using new classes.

START EARLY! ASK QUESTIONS!