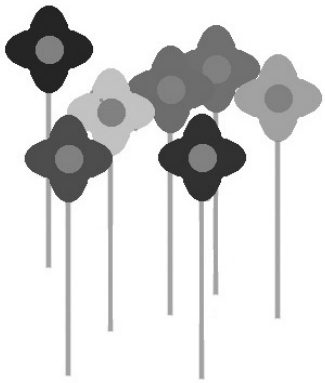
Lab 4: Part 1
Growing Flowers[1]
CS 1101B – Fall 2014
Due *(both Parts 1 and 2)*:
November 2, 10pm (Monday lab)
November 3, 10pm (Tuesday lab)

Objective: To gain experience defining a class and its methods.

Your program in Part 1 of this lab will draw and grow flowers like the following:



In Part 2 (October 27^{th} and 28^{th}), you will add additional classes to make the behavior of the program more interesting. **You will submit a single program on November 2/3 that fulfills the requirements of both parts of the lab.**

Program Requirements

The canvas should initially start empty. When you press the mouse in the canvas, a flower should start growing at that point. Initially it will be just a sprout, which you should represent as a small, red, solid circle centered on the point where the mouse was pressed. As you drag the mouse around, it should grow: i.e., a stem (a narrow, green, solid rectangle) should grow upwards with the sprout centered on its top end. When it reaches its full height, the stem should stop growing and petals should appear, with the sprout still visible in front of the petals. The color for the petals should be chosen randomly. The flower won't grown any more, but if you click on the petals after it has bloomed, the flower will change color. When you are happy with the flower's color, you can grow another one by clicking somewhere else in the window. When the mouse moves out of and then back into the window, the scene should reset itself to be empty.

A few notes:

1. In the `grow` method you need to write, you can produce the "growth" of the flower by increasing the height of the `FilledRect` representing the stem (using the `setHeight` mutator method) *and* moving the `FilledRect` up by the same amount. Note that when the flower reaches its full height, the `grow` method should make it sprout petals, appropriately colored.

[1] Adapted from a lab provided with *Java: An Eventful Approach*, K. Bruce, A. Danyluk, and T. Murtagh

2. After the flower has blossomed, nothing should happen if you keep dragging the mouse. Depending on how you write your code, this may require you to check whether the petals have already been created. You can do this as follows: Suppose `petal1` is a `FilledOval`. The following code:

```
if (petal1 == null) {

}
```

   will execute the code inside the `if` only if `petal1` has *not* been created. Similarly, the following code:

```
if (petal1 != null) {

}
```

   will execute the code inside the `if` only if `petal1` *has* been created.

3. If you stop dragging the mouse before the flower has reached its full height, you don't need to be able to continue growing it.

4. If you click on a bloomed flower **after** starting the next sprout, it should not change the color of the older flower (instead, it should simply start the next sprout as if you had clicked outside of any flower).

Your program will be divided into two classes: the usual `Events` class that extends the `FrameWindowController` class (what you're used to doing) and a separate `Flower` class. I have provided the complete code for the `Events` class and a bare skeleton of the `Flower` class on the class webpage.

You should not modify the `Events` class in any way except to provide a program summary at the top of the file, i.e. you need to understand what the `Events` class does and describe it in this comment. Your main job is to implement the `Flower` class so that it works with the `Events` class. In particular, the `Flower` constructor should expect three parameters: the `Location` where the `Flower`'s stem should be planted, a `double` specifying the maximum height of the `Flower`, and the canvas. The `Flower` class should define the following methods used by the controller to implement the functionality described above:

1. `changeColor`: sets the color of the petals to a random color,

2. `flowerContains`: returns true if the petals or center of the flower contain the `Location` object sent in; otherwise, it returns false, and

3. `grow`: makes the flower grow a bit if it has not already reached its full height and sprouted petals; if it has reached its full height, this method should make it sprout petals and color the petals.

Submitting Your Work

Do not submit your work until November 2/3, at which time you will submit a single program that fulfills the requirements of both parts of the lab.

Lab 4: Part 2
Growing Flowers[1]
CS 1101B – Fall 2014
Due *(both Parts 1 and 2)*:
November 2, 10pm (Monday lab)
November 3, 10pm (Tuesday lab)

Objective: To gain more experience defining and using classes that interact.

Your program in Part 1 of this lab just grew flowers on a blank canvas. Nice flowers, but still.... In Part 2, you will add two additional classes:

1. a `Ground` class that will affect where the flowers can be planted, and
2. a `Sun` class that will affect whether or not they bloom.

Program Requirements

Differences from and additions to Part 1:

1. In Part 1, you didn't need a `begin` method. Now, in the `begin` method, you will create the ground, dig 1-3 furrows in it, and create a sun (more on these below).
2. Flowers will start growing only if the user presses on a furrow.
3. The sun will randomly appear or disappear whenever the user presses on a furrow to start a flower growing and, although the flower will grow regardless of whether the sun is out, it will not bloom if the sun is not out.
4. The flower will disappear in either of these two cases when the user releases the mouse:
   (a) if the flower has not reached its maximum height (regardless of whether it would have bloomed), or
   (b) if the flower has reached its full height, but has not bloomed because the sun is not out.
5. The height of the flowers should be about $\frac{1}{3}$ the height of the canvas, instead of $\frac{1}{2}$. (They were just *too* tall.)

First, a description of what the `Ground` and `Sun` classes need to do. The `Ground` class needs to have:

1. A constructor that can be sent a `height` and a `canvas` and creates a brown `FilledRect` that should be as wide as the canvas, and extend from the bottom of the canvas to `height` pixels above the bottom.

[1] An extension of a lab provided with *Java: An Eventful Approach*, K. Bruce, A. Danyluk, and T. Murtagh

2. A `digFurrows` method that can be sent an integer and, if the value of that integer is between 1 and 3 (including 1 and 3), it will create that many furrows (long, narrow, black `FilledOval`s). Note that the furrows should be spaced approximately evenly across the height of the ground. The ends of the furrows should be inset from the edges of the canvas about 40 pixels. If there is to be only one furrow, it should be centered (height-wise). See Figures 1 and 2 for pictures of what the ground should look like after two and three furrows have been dug, respectively.

3. A `furrowContains` method that can be sent a `Location` object and returns `true` if that location is in any one of the furrows. Again, this is a little tricky because you could have anywhere from 1 to 3 furrows. Remember that if you have a variable `objectVariable` that is referring to an object, you can check if the object actually exists by checking whether `(objectVariable != null)`. If it is not equal to `null` the object exists.

The `Sun` class needs to have:

1. A constructor that can be sent `x` and `y` coordinates that are the coordinates of the upper left corner of the bounding box around the sun, the diameter of the sun, and the canvas. It should create a yellow `FilledOval` with the specified diameter in the specified location.

2. A `randomSunComesOut` method that makes the sun come out (shows it) with a probability of 0.5 and, otherwise, hides it. In other words, regardless of whether the sun is currently out, it will be out after this method is called with a probability of 0.5. You can take care of the probability part as follows. The `nextDouble` method in the `Random` class generates a pseudo-random double value between 0.0 and 1.0, so if you have an `if` statement like this:

```
if (rand.nextDouble() < 0.5) {
     < statements >
}
```

   the statements inside the `if` will be executed with probability 0.5. Of course, the 0.5 should be a named constant...

3. A `sunIsOut` method that returns `true` if the sun is out; otherwise, it returns `false`.

Figures 1 and 2 show a canvas with the sun out, although it is a little hard to see since the color of the sun is so light.

Your `Flower` class, in addition to its functionality from Part 1, needs to have the following:

1. In the `grow` method, the flower should blossom only if the sun is out. **Note that this means the `Sun` object that was created in the `Events` class must be sent to the `grow` method so that the code inside that method can check if the sun is out, in order to do the apropriate thing regarding blooming**.

2. A `wither` method that hides any existing pieces of the flower.

3. A `hasBlossomed` method that returns `true` if the flower has bloomed; otherwise, `false`.

Note that you may need to add additional variables and/or constants to enable this additional functionality.

Finally, you need to change the `Events` class so that:

1. The ground, furrows, and sun are created in the `begin` method.
2. A flower will not start growing unless one of the furrows contains the point where the user pressed.
3. Before the flower is created, `randomSunComesOut` is called to make the sun either appear or disappear. Note that `randomSunComesOut` should not be called unless a flower is about to be created and start growing.
4. When the user releases the mouse, if the last flower created has not blossomed, it should wither (disappear).

See Figure 3 for a picture of the canvas with 3 furrows in the ground and 4 flowers that have bloomed.

Submitting Your Work

Please submit your program in the usual way via Blackboard. Remember that we need your whole project folder (the one with the name composed of your login id and lab number) and it must be compressed.

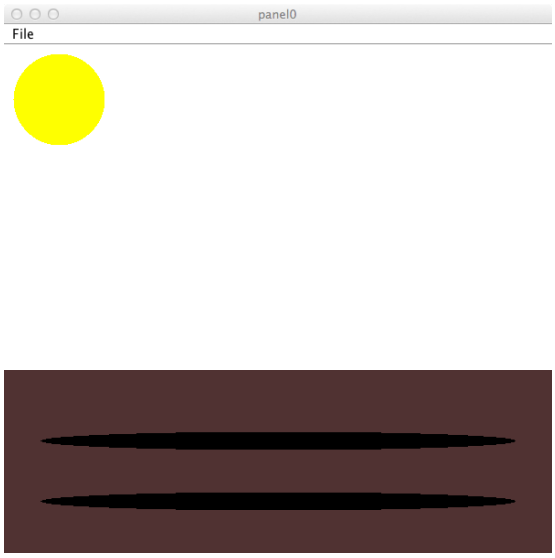


Figure 1: Ground with 2 furrows and sun out.

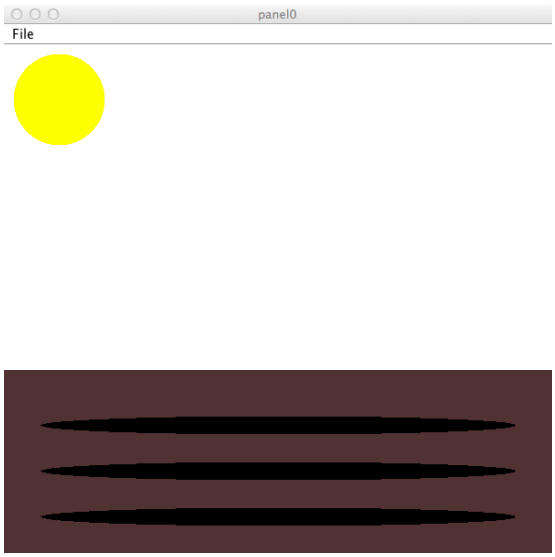


Figure 2: Ground with 3 furrows and sun out.

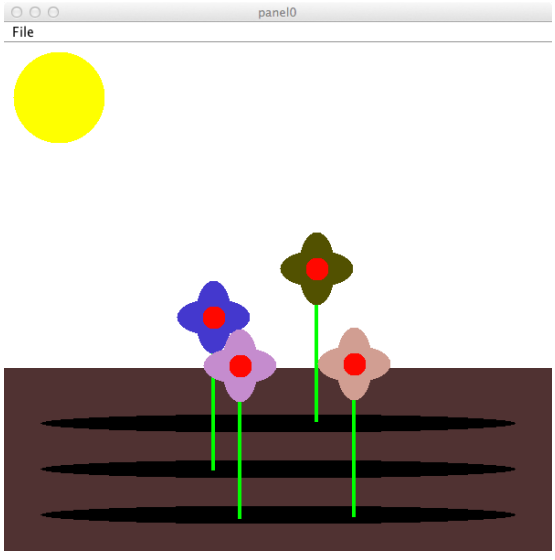


Figure 3: Four flowers that have blossomed.