CIS 201 Computer Science I
Fall 2009 Lab 03

September 14, 2009

- Practice building FANG programs.
- Work on writing classes with less guidance.
- Practice setting up Java expressions.
- Introduction to the `Math` class.

Checkpoint 1 Let there be light!

Set up the class. Create a folder for `Lab03` in your CIS 201 directory. In that folder, define a `class`, `Orbits`. This class is the main FANG class for our program; it extends `Game` so you should `import` the appropriate library.

Compile and test your program (this is done often to check for typos).

Begin creating the fields. Add a *field* to the class called `sun`. The sun is a circle. Make sure you `import` the right library(ies). (You can compile again if you want, as a "spell-check" to make sure you included the right library and that you spelled the type of the field correctly. This is the last time this lab mentions compiling.)

Add a `setup` method. Make sure you have the proper *method header* for the method. Make sure you can explain the four parts of a method header as well.

In the body of `setup` instantiate the `sun`: 0.1 screens in diameter, centered on the screen, and a bright color you have never used before (`file:///home/student/Classes/201/SCG/SCG.pdf` is the book; one of the appendices lists all of the available color names). Make sure the `sun` is added to the game.

Show your work on Checkpoint 1 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 2 The Firmament

Construct the earth. The `earth` is a field that holds another circle. The `earth` is a small, 0.03 screen diameter, circle. It should be some shade of green. We The following discussion of simplified orbital mechanics determines where the `earth` begins the game.

Simplified Orbital Mechanics The `earth` orbits the `sun`. First, we assume that the orbit is circular and along the *unit circle*. That orbit is centered at the top of the screen and is too big for our needs; we adjust that after figuring out how the big orbit works.

The *unit circle* is the set of points (x, y) such that $x^2 + y^2 = 1$. From trigonometry (let us know if you have not yet had it) you should know that all such points (x, y) are of the form

$$(x, y) = (\cos(\phi), \sin(\phi))$$

where ϕ is the angle between the points $(1, 0)$ and (x, y) along the unit circle. (The angle ϕ is measured in *radians* – remember that there are 2π radians along the circumference of the circle.)

We want to simulate the movement of a point (x, y) along the unit circle orbit, by moving it a small amount around the circle in each frame. (Hope you're both thinking `advance` at this point.)

The point (x, y) on the unit circle has the form $(\cos(\phi), \sin(\phi))$ and we want to rotate the point (x, y) by θ radians (where θ is a small number) around the circle, we would want the new point (xx, yy) to be at location $(\cos(\phi + \theta), \sin(\phi + \theta))$.

Of course, you remember your trigonometry sum formulas:

$$xx = \cos(\phi + \theta) = \cos(\phi)\cos(\theta) - \sin(\phi)\sin(\theta)$$

$$yy = \sin(\phi + \theta) = \sin(\phi)\cos(\theta) + \cos(\phi)\sin(\theta)$$

Since $x = \cos(\phi)$ and $y = \sin(\phi)$, these formulas can be written as

$$xx = x\ \cos(\theta) - y\ \sin(\theta)$$

$$yy = y\ \cos(\theta) + x\ \sin(\theta)$$

Starting the Earth. Initially the earth is at $\phi = 0$. Track the location of the earth with two `double` fields, `ex` and `ey` (earthX and earthY). At $\phi = 0$, `(ex, ey)` $= (1, 0)$. Initialize the fields and set the location of `earth` to `(ex, ey)`.

Show your work on Checkpoint 2 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 3 Let there be orbit.

Add `advance`. What is the method header of `advance`? What is the type of the `parameter`? What does the value of the parameter mean? How have we used it in the past?

Move `earth`. In `advance` calculate the new values for `ex` and `ey` and then set the location of `earth` to the new coordinates. The earth should go around its orbit once every six seconds. Newton's apple fell at a rate of three seconds per screen. How, in `advance` did we scale the rate of fall to keep it smooth, even if `advance` were not called right on time?

We used the time parameter passed in to `advance`. We need to do the same. How many *radians* per second does the earth move? (If we multiply that value by seconds, we get the radians, the value for θ.) Set up a field, `eAngularVelocity`, to hold the angular velocity of `earth`. The field should be `final` (what does that mean) and have its value set when it is declared. Use the value `Math.PI` for π.

In `advance`, declare a local variable (how is a local variable different than a field?), call it `eTheta`. Set it to the amount of angle `earth` advances this frame.

Using the formulae from above,

$$xx = x\ \cos(\theta) - y\ \sin(\theta)$$
$$yy = y\ \cos(\theta) + x\ \sin(\theta)$$

which values here are the new location for `earth`? Remember that `ex` and `ey` are the coordinates of the planet. The formualae become:

```
double exx = ex * Math.cos(eTheta) - ey * Math.sin(eTheta);
double eyy = ey * Math.cos(eTheta) + ex * Math.sin(eTheta);
```

The above declares two more local variables, `exx` and `eyy`. Local variables do not carry their values over to the next time the method in which they are declared is called.

The final step of moving `earth` is to update the fields tracking it (to the new coordinates) and then reset the location of the sprite:

```
ex = exx;
ey = eyy;
earth.setLocation(ex, ey);
```

Show your work on Checkpoint 3 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 4 Earth should orbit the sun.

Determine the orbit. The unit circle makes everything easy to calculate; it also is the wrong orbit. Create a `final` field, `eRadius`. Set it to the radius for the `earth` orbit. Make it smaller than 0.5 so that the planet is always visible (0.45 works).

Relocate `earth`. Whenever you set the location of `earth`, instead of using `ex` and `ey` directly, scale each by `eRadius` and move the center of the orbit to the center of the `sun`. That is easier than it sounds.

Scaling by the radius means multiply each by the radius:

```
                                ex * eRadius,                ey * eRadius
```

Moving the center just means adding the difference between the center of the unit circle, $(0, 0)$ and the center of the `sun`. We know the center of the `sun`, numerically, but let's have the object give it to us.

```
                    sun.getX() + ex * eRadius, sun.getY() + ey * eRadius
```

This is used, each time you locate the `earth`:

```
earth.setLocation(sun.getX() + ex * eRadius, sun.getY() + ey * eRadius);
```

Have `earth` orbit the `sun`.

Show your work on Checkpoint 4 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 5 Define a method.

Factor out the calls to `setLocation`. The two calls to `earth.setLocation` are just the same. One software engineering mantra is "Do not repeat yourself." If you have to type it twice, you have to remember to fix it twice if it is broken, you have to check for typos twice, and you will never, ever, stop at just two copies.

Write a method to do the work for you and then call it from each of the spots where `earth` is located:

```
private void locateEarth() {
  earth.setLocation...
}
```

Put the method definition in the `class Orbits` but not within any other method. It is called with no parameters. Replace the direct `setLocation` calls with calls to `locateEarth`.

When you do the same thing over and over, write a *method* to do it for you, one you can call from all the places the task must be performed. If the task is almost the same, think about parameters to make each call unique. Over all, remember `Do not Repeat Yourself.`

Show your work on Checkpoint 5 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 6 Let there be Moon

The earth looks lonely. Create a circular sprite 0.01 screens across. Color it some light shade of gray. The sprite should be assigned to a field called `moon`.

Duplicate the fields used to keep track of `earth`'s location and orbit and instead of starting them with an `e`, start them with an `m`:

```
private OvalSprite moon;
private double mx;
private double my;
private final double mRadius = 0.045;
private final double mAngularVelocity = eAngularVelocity * 12;
```

The moon orbits faster and closer around the earth than the earth does around the sun. Now copy the `advance` code for `earth` and apply it to `moon`. Two things to keep in mind while doing that:

- The `moon`'s location is relative to `earth`, not `sun`.
- *Do not Repeat Yourself.* (Make a method to position the `moon`).

Show your work on Checkpoint 6 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Log off of the lab computer you are using before leaving they lab. Anyone entering the lab has unlimited access to your files if you remain logged on. **DO NOT** turn off lab computers! They are a shared resource and there might be someone else logged in to "your" machine.