# CIS 201 – Computer Science I
# Laboratory Assignment 3

## Introduction

In this lab, you will produce a simulation of the earth orbiting the sun.

## First, the sun

### Creating the file `Orbits.java`

Make a directory `Lab03` in which you will do your work. It should be a subdirectory of your `CS1` directory, as you have done with previous labs. Change directory to `Lab03`.

Create a Java program named `Orbits.java` in your `Lab3` directory. Your program should define a class `Orbits` that extends the FANG `Game` class. Your program should look like this:

```
import fang.core.Game;

public class Orbits extends Game {

}
```

Compile your program to be sure you have no typos. You can run your program, too, but don't expect it to do much.

### Create appropriate variables

Modify your program so that it includes the following private instance variable declaration:

```
private OvalSprite sun;
```

Put this as the first line inside your `Orbits` class. Be sure to indent it properly – use the `Tab` key to do so. You'll also have to import the FANG `OvalSprite` class, so add the line

```
import fang.sprites.OvalSprite;
```

to the top of your program.

A FANG ’game’ automatically calls a `setup` method whose purpose is to do any initialization in preparation to start the ’game’. (We will use the term ’game’ here, even though what we are creating is a simulation, not a real game.)

Put your `setup` method below the declaration of your `sun` variable. It should look like this:

```
public void setup() {
  // code for setup goes here
}
```

This method has the following signature:

|  |  |
|---|---|
| Visibility | : `public` |
| Return | : `void` |
| Name | : `setup` |
| Parameters: | none |

Now start to fill in the `setup` method body by defining (also called *instantiating*) the `sun` variable to be a circle with diameter 0.1. A circle is an `OvalSprite` with the same height and width. Set its color to `red` and its location to `(0.5, 0.5)` – the center of the game canvas. Finally, we will add the `sun` sprite to the game canvas. Here is the code you should insert into the `setup` method body:

```
sun = new OvalSprite(0.1, 0.1);
sun.setLocation(0.5, 0.5);
sun.setColor(getColor("red"));

addSprite(sun);
```

Compile your `Orbits.java` program and fix any problems until it compiles without error.

### Checkpoint 1

**Run your program, and show us your results. Be prepared to tell us about the method signatures of `setup`.**

## Next, the earth

### A simplified orbit

Since the earth will be orbiting the sun, we will need to understand how orbits work. To simplify things for now, we will assume that the orbit consists of points on the *unit circle*: those points $(x,y)$ satisfying the condition that $x^2 + y^2 = 1$. Since you’ve had trigonometry (let us know if this is not the case), you should know that all such points $(x, y)$ are of the form

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

where $\phi$ is the angle between the points $(1, 0)$ and $(x, y)$ along the unit circle. (Normally, the angle $\phi$ will be measured in *radians* – remember that there are $2\pi$ radians along the circumference of the circle.)

## Orbital movement

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 at each time interval. The `advance` method will be used to animate the earth orbit by changing the location of the earh a small amount each time interval. *The FANG game engine calls the* `advance` *method about 20 times per second, so the animation should be reasonably smooth.*

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

Do you remember your trigonometry sum formulas? Here they are:

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

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

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

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

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

## Programming an earth orbit

Now you will create an `earth` sprite to follow an orbit around the unit circle. Just leave the sun on your game canvas for the time being.

First, create an `earth` instance variable, simlar to your `sun` variable:

```
    private OvalSprite earth;
```

We also need instance variables to store the $(x, y)$ coordinates of the earth. We'll call them `ex` and `ey`:

```
    private double ex;
    private double ey;
```

Put these instance variables together with the `sun` instance variable at the top of your `Orbits` class.

In your `setup` code, define (instantiate) an `earth` object to be an `OvalSprite` with diameter 0.03 (it's smaller than the sun!), make it green (what else?), and define the `ex` and `ey` variables to be 1.0 and 0.0, respectively – this means that the earth will start off at coordinates (1,0), which is the upper right corner of the screen. (We call the variables `ex` and `ey` because they represent the $(x,y)$ coordinates of the earth.)

Set the location of your `earth` object to `(ex, ey)`, right after your assignments to `ex` and `ey`. Be sure to add the `earth` sprite to your canvas, along with what you did for the `sun`.

### Checkpoint 2

**Compile and run your program at this point, and show us your results. (And be prepared to tell us you knew the sum formulas for** $\cos$ **and** $\sin$**.)**

## Moving the earth

### Advancing on `advance`

Now write an empty `advance` method with the proper signature:

```
public void advance(double dT)
```

By *empty*, we mean that there should still be a set of curley braces after the method signature to hold the body of the method, but there should be no code (yet) inside the method body. Put this method following the end of the `setup` method but before the final curley brace at the end of the `Orbits` class. Remember that the `advance method` has a `double` parameter, which we have called `dT`. (You could give it any variable name you like, for that matter, but it's best to choose a name that is descriptive.) Compile and run your code to be sure that you haven't typed anything incorrectly.

### Moving the earth

At each call to `advance`, we want to move the earth's `(ex,ey)` coordinates a small amount around the unit circle, by θ radians. The value we choose for θ will depend on how fast we want the earth to move.

Declare the following new variable in your instance variable section:

```
private double eThetaSpeed; // how fast to advance earth, in radians/second
```

In your `setup` code, give the value of 0.1 to `eThetaSpeed`. This will move the earth along the unit circle orbit a small amount each time the `advance` method is called. Since there are $2\pi$ radians in a circle, a speed of 0.1 radians/second would mean that the time to go around a complete circle would be $20\pi$ seconds, or a little more than half a minute.

In your `advance` method, you can now write the code to move the earth by `eThetaSpeed` radians per second along the unit circle. Because of the orientation of the *x*- and *y*-axes in our game canvas, moving in the positive direction around the unit circle (since `eThetaSpeed` is positive) will appear to be moving clockwise on our game canvas. If you want to look at it like most mathematicians do, you'll have to go behind your computer screen and stand on your head, in which case the earh will appear to be going counter-clockwise.

Remember the formulas from above:

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

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

Also remember that `eThetaSpeed` is in radians per second, so we multiply this by `dT` to get the number of radians to move. We can compute this value and store it in a local variable `eTheta`, as follows:

```
    double eTheta = eThetaSpeed * dT;
```

Put this at the beginning of your `advance` method. (The use of `double` here creates a new variable `eTheta` "on the fly". Such a variable is called a *local variable*. This variable will exist throughout the execution of the `advance` method and will be discarded when `advance` finishes.)

Since we will also need cos and sin values of `eTheta`, add these lines as well:

```
    double ecosTheta = Math.cos(eTheta);
    double esinTheta = Math.sin(eTheta);
```

The `Math.cos` and `Math.sin` functions are built in to Java, so you don't need to import anything.

Since we are using `ex` and `ey` as the *x*- and *y*-coordinates of the earth on the unit circle, formulas described above for *xx* and *yy* above now become:

```
    double exx = ex*ecosTheta - ey*esinTheta;
    double eyy = ey*ecosTheta + ex*esinTheta;
```

Here we have substituted the variable `ecosTheta` for $\cos(\theta)$ and so forth. Also, we have defined two new local variables in the process: `exx` and `eyy`.

Once you have identifed the new earth coordinates, you will need to re-position the earth:

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

Put this code as part of `advance`, immediately after your evaluation of `exx` and `eyy`;

Compile your program and run the game. You should see the earth slowly advance clockwise in a circle with origin at the top-left corner with coordinates (0,0). If you let it run for a while, the earth will disappear from the screen on its way to the second quadrant. If you wait long enough, it will reappear at (1,0) and be visible again.

## Checkpoint 3

**Show us your work at this point.**

## Earth orbits the sun

Of course, the earth should be orbiting the sun, not the origin of our game canvas. You need to fix this.

### Earth radius around the sun

You first need to determine how far the earth should be from the sun as it orbits the sun. Create a variable `eRadius` that will hold this value. Since the sun starts out at (0.5,0.5) and we want the earth to be visible on the canvas as it orbits the sun, a reasonable radius of rotation around the sun would be 0.45.

As you have done before, create a private double instance variable named `eRadius` (along with the other instance variables) and assign its value to be 0.45 (in `setup`). Put this assignment immediately after your assignments to `ex` and `ey`.

Now, whenever you set the location of the earth (there are *two places* you need to do this: once in `setup` and a second time in `advance`) do the following:

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

This will put the earth's location *relative to that of the sun*. Since `(ex,ey)` are always on the unit circle, `(ex*eRadius, ey*eRadius)` will always be on a circle of radius `eRadius`.

Make these changes and run your game. You should see the earth stadily progress around the sun.

### Factoring out the `setLocation` calls

In the code above, you will have two places where you call

```
earth.setLocation(...)
```

where the code is *exactly the same*. Instead of having entered this twice – which increases the likelihood that you will make a mistake – you will create a separate method that does this.

Create a method as follows:

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

where the `setLocation` parameters are the same as in your current program. Put this method somewhere in your program, probably between the end of `setup` and the beginning of `advance`. You can put it pretty much anywhere in your `Orbits` class definition, but it can't be *within* any of the other methods.

Then, in the two places where you had `earth.setLocation(...)` – once in `setup` and a second time in `advance` – replace these with

```
locateEarth();
```

In general, if you find yourself doing a complicated line or lines of code more than once in your program, you can write a method that does the same stuff, and you can replace the lines where it originally occurred with method calls.

Compile and run your program to verify that your change hasn't affected the way your game runs.

## Checkpoint 4

**Show us your work at this point.**

## The moon and earth

Well, now that you have the earth orbiting the sun, you can have the moon orbit the earth. You have already done all the work for the earth orbiting the sun: all you need to do is write almost identical code for the moon orbiting the earth.

You'll need the following variables:

```
private OvalSprite moon;
private double mx;
private double my;
private double mRadius;
private double mThetaSpeed;
```

Make the moon be a white circle with diameter 0.01 and orbit radius 0.045. As with the earth, the `(mx, my)` coordinates will be on the unit circle, starting at (1.0, 0.0); Choose `mThetaSpeed` to be 1.0, so that the moon rotates around the earth faster than the earth rotates around the sun. The rest is just copying the code for the earth and doing the same for the moon. You'll need to have moon-related code in `setup` and `advance`. You can practically duplicate the earth-related code, only with appropriate modifications to be moon-related. Inside `advance`, you'll need local variables named

```
double mTheta
double mcosTheta
double msinTheta
```

defined appropriately, just as we did for the earth.

Oh, two things to remember:

- the location of the moon is relative to that of the earth, so you'll want to say something like

  ```
  moon.setLocation(earth.getX() + mx*mRadius, earth.getY() + my*mRadius);
  ```

- You'll want to make a `locateMoon` method that does this.

### Checkpoint 5

Show us your running game at this point.

## Documentation and formatting

Be sure that your code is properly indented and that you have used spaces appropriately. You will lose points if you do not do so.

Put documentation on top of each of your method definitions. Briefly describe what the method does.

Also, be sure your code has the ID block at the top (following your `import` lines) with your names (as author) and other required items as described in Lab 2.

### Checkpoint 6

**Print you code and include it with your checksheet document.**