CS101 Assignment Two

Objective: Using conditionals and loops, we can construct interesting animated graphics objects. In this assignment, you will learn how to build some virtual toys that can move like real ones. The skeleton codes are attached with this document. They are also online in the course website. This assignment is due at mid-night on February 14.

1 Digital Clock

In this project, we write a program to simulate a clock as shown in Fig. 1. The clock has three hands and a panel with markers and digit labels. This clock looks like a clock and more importantly it should click exactly like a real clock.

As an example, we can draw the second hand of the clock using the following code:

```java
x = secondHandLength * Math.cos(secondHandAngle);
y = secondHandLength * Math.sin(secondHandAngle);
StdDraw.setPenColor(StdDraw.BLACK);
StdDraw.setPenRadius(secondHandThickness);
StdDraw.line(0, 0, x, y);
```

In this example, the second hand has length secondHandLength, thickness secondHandThickness and angle secondHandAngle with the x axis. Note that the angle increases when the hand rotates counter-clock-wise. The panel, markers and the digits can be drawn similarly with functions in the StdDraw class. Please check the APIs at http://www.cs.princeton.edu/introcs/15inout for more details.

Making an animation is not quite different from drawing static pictures. The only difference is that we need to “refresh” the drawing in fixed time intervals. Refreshing means that we need to
erase the old drawing and put a new one on the canvas. The canvas in StdDraw is like a real canvas: Each time you draw something on it, it will stay there; The new drawing will occlude the old ones (the ink is nontransparent). Therefore, you can erase something by placing another background color object on the top. The easiest way of refreshing a frame is to erase everything on the canvas and then draw the new one. But a more efficient way is to erase only the parts that need redrawn. In this project, the panel of the clock does not change with time. So, you can draw the panel once and update the circular region inside the panel (the area in which the hands move). Showing an animation can be done based on the following pseudo-code:

```java
initialize the locations of objects;
while (true) {
    draw objects at current locations;
    update object locations;
    wait for a given time;
    clear the drawing region;
}
```

We can use function StdDraw.show(n) to delay for n milliseconds. In each new pictures, we update the location and appearance of objects (in this project, we update the locations of clock hands).

We are now ready to make the clock. The clock should have all the components as shown in Fig. 1. The appearance of your clock does not have to be exactly the same as the clock shown. You are encouraged to try even more fancy clocks. Your program should ask for 3 arguments for the starting hour, minute and second and set the right clock hands locations based on these inputs.

## 2 Planet System

In this project, we will simulate a planet system that contains the Sun, the Earth and the Moon. We do not need to accurately model the size and distance of these objects. We just want to simulate their periodic movements. The Sun in fact just sits there. The earth rotates 360 degrees around the Sun in 365.26 days (In our program, our days go faster than the real ones.). The Moon finishes one cycle in 29.5 days. We use red ball, blue ball and yellow ball to represent the Sun, the Earth and the Moon respectively. Fig. 2 shows different phases of the movements.

![Figure 2: The Sun, the Earth and the Moon.](image)

This problem is similar to the previous one. We can use very similar technique to generate the animation. One difference is that in this project objects follow ellipse trajectories (from our point
of view) but not circles. Assume that we use the same coordinate system settings with the clock program: \( x \) is in \([-1,1]\) and \( y \) in \([-1,1]\). You can use the following Java code segment to draw the Earth following an ellipse orbit.

```java
double earthX = 0.75 * Math.cos(earthAngle);
double earthY = 0.75 * Math.sin(earthAngle) * 0.25;
StdDraw.setPenColor(StdDraw.BLUE);
StdDraw.filledCircle(earthX, earthY, 0.15);
```

In the above code, we place the earth on an ellipse orbit with the radius 0.75 and angle earthAngle. 0.25 is a scaling factor that crashes a circle into an ellipse. In this computation, we assume to use an orthogonal projection: objects do not become smaller when they move farther away. This greatly simplifies the problem but still gives reasonably good results.

The Moon is a bit tricky. It also has an ellipse orbit, but its center moves with the Earth. We use the following code segment to draw the Moon.

```java
double moonX = earthX + 0.3 * Math.cos(moonAngle);
double moonY = earthY + 0.3 * Math.sin(moonAngle) * 0.25;
StdDraw.setPenColor(StdDraw.YELLOW);
StdDraw.filledCircle(moonX, moonY, 0.05);
```

To make the Earth and the Moon move around, you just need to change the earthAngle and moonAngle from 0 to \(2\pi\) in a loop based on their periods. Usually 100 uniform partitions of \([0, 2\pi]\) can generate smooth enough movement.

The last problem we need to solve is occlusion. As shown in Fig. 2, the Earth and the Moon are occluded by the Sun if they move behind it. Similarly, the Moon is occluded by the Earth if it moves behind the Earth. To handle occlusions, we use the following observations:

- The Earth and the Moon can be occluded by the Sun only if the Earth moves on the inner half of the track (earthAngle > 0 && earthAngle < Math.PI); otherwise the Earth and the Moon will occlude the Sun.

- The Moon is occluded by the Earth only when it is on the inner half of the track (moonAngle > 0 && moonAngle < Math.PI); otherwise the Moon occludes the earth.

The above reasoning in facts extracts the relative depth of each object: An object can be occluded by another one only when it is farther away. To generate the occlusion effects, we draw objects in an order that is from the farthest to the nearest; we draw the occluded objects first and then the objects in the front last. We need the `if` statements we learned in our class to distinguish different occlusion situations.

### 3 What to Submit

You should submit the two java programs named as Clock.java and Solaris.java. Pay attention to good Java programming style. Zip or tar your two java files into a single zip file or tar file. Upload your zip (or tar) file to WebCT before the submission deadline. There will be 4 days grace period. But late submission would involve 10% point deduction for each day. Submissions later than 4 days are not accepted.
public class Second {
    public static void main(String[] args) {
        // set the scale of the coordinate system
        StdDraw.setXscale(-1.0, 1.0);
        StdDraw.setYscale(-1.0, 1.0);

        // initial second of the clock from the input
        int second = Integer.parseInt(args[0]);

        // initial angle for the second hand
        double secondAngle = -second * Math.PI/30 + Math.PI / 2.0;

        // draw the panel
        StdDraw.circle(0, 0, 1);

        // PLACE your code here for drawing the markers and digits on the panel

        // main animation loop
        while (true) {
            // clear the background
            StdDraw.setPenColor(StdDraw.WHITE);
            StdDraw.filledCircle(0, 0, 0.8);
            double x, y;

            // PLACE your code here for the hour hand and minute hand
            //

            // compute the location of the second hand
            x = 0.75 * Math.cos(secondAngle);
            y = 0.75 * Math.sin(secondAngle);

            // draw the second hand on the canvas
            StdDraw.setPenColor(StdDraw.BLACK);
        }
    }
}
StdDraw.setPenRadius(0.005);
StdDraw.line(0, 0, x, y);

// update the second hand angle
secondAngle -= Math.PI/30;
secondAngle = secondAngle % (2 * Math.PI);

// display and pause for 1 s
StdDraw.show(1000);
public class EarthSimple {
    public static void main(String[] args) {

        // set the scale of the coordinate system
        StdDraw.setXscale(-1.0, 1.0);
        StdDraw.setYscale(-1.0, 1.0);

        // initial values
        double earthAngle = 0;

        // main animation loop
        while (true) {
            // clear the background
            StdDraw.setPenColor(StdDraw.BLACK);
            StdDraw.filledSquare(0, 0, 2);

            // compute the location of the Earth
            double earthX = 0.75 * Math.cos(earthAngle);
            double earthY = 0.75 * Math.sin(earthAngle) * 0.25;
            earthAngle += 2*Math.PI/365;
            earthAngle = earthAngle % (2 * Math.PI);

            // You need to add codes to the following code
            // to include the Moon
            //

            if (earthAngle < Math.PI) {
                // the Earth is behind the Sun
                StdDraw.setPenColor(StdDraw.BLUE);
                // draw the Earth
                StdDraw.filledCircle(earthX, earthY, 0.15);
                StdDraw.setPenColor(StdDraw.RED);
                // draw the Sun
                StdDraw.filledCircle(0, 0, 0.3);
            }
        }
    }
}
else
{
    // the Earth is in front of the Sun
    StdDraw.setPenColor(StdDraw.RED);
    // draw the Sun
    StdDraw.filledCircle(0, 0, 0.3);
    StdDraw.setPenColor(StdDraw.BLUE);
    // draw the Earth
    StdDraw.filledCircle(earthX, earthY, 0.15);
}

    // delay for one day (for us 10ms)
    StdDraw.show(10);
}