Homework 3

Start Assignment

Due Oct 14 by 5pm **Points** 75 **Submitting** a file upload
File Types java, mp4, and pdf **Available** until Oct 14 at 10pm

Pong :: Two Ways

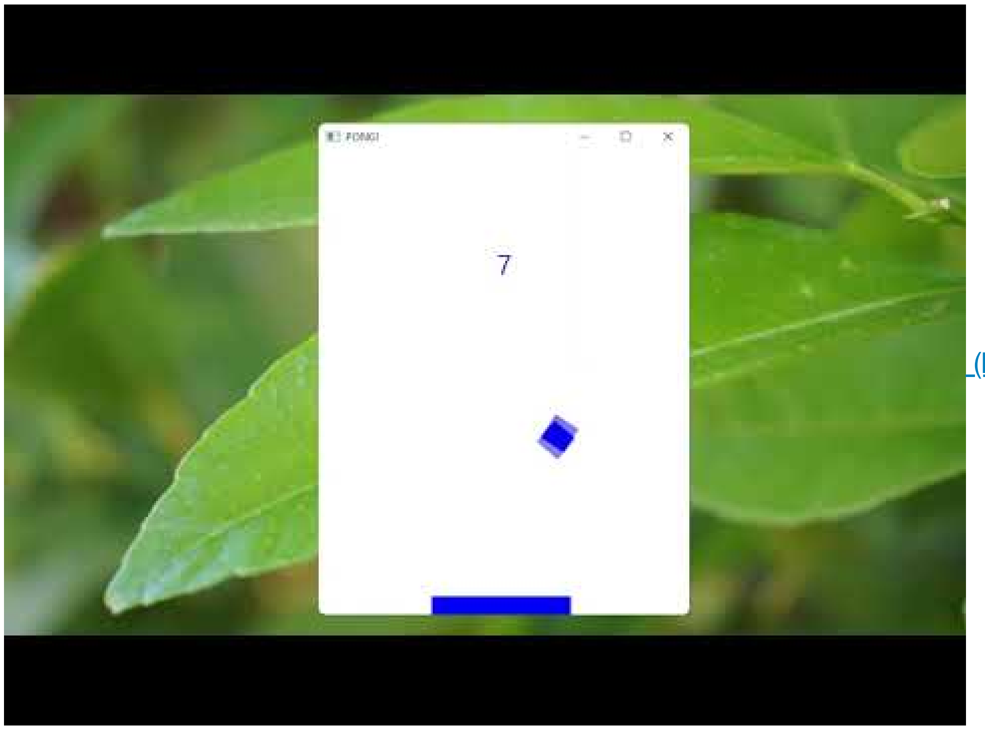
In this assignment you will be writing a simple one player Pong video game using JavaFX and you will be writing it two ways. To be more clear, you will be writing in a straightforward linear style and then refactoring your code into a more object oriented style leveraging basic inheritance.

There's not really a lot of code in this assignment and you won't really have to completely write the assignment twice. While refactoring you will be lifting code into methods and into objects and trying to make good decisions about what should go where and what your object hierarchy should look like.

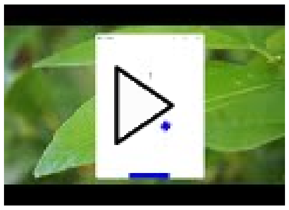
Once you've completed both programs, you will complete a short writeup where you explain your choices. My advice is to comment your refactored coded excessively in order to create the content for your writeup. Then, remove the excessive comments and just put them in your writeup. As we will see, comments are a code smell, however, they can be useful during development. It's often better to make notes as you go so that you have the content later to organize into your writeup.

Demo Video

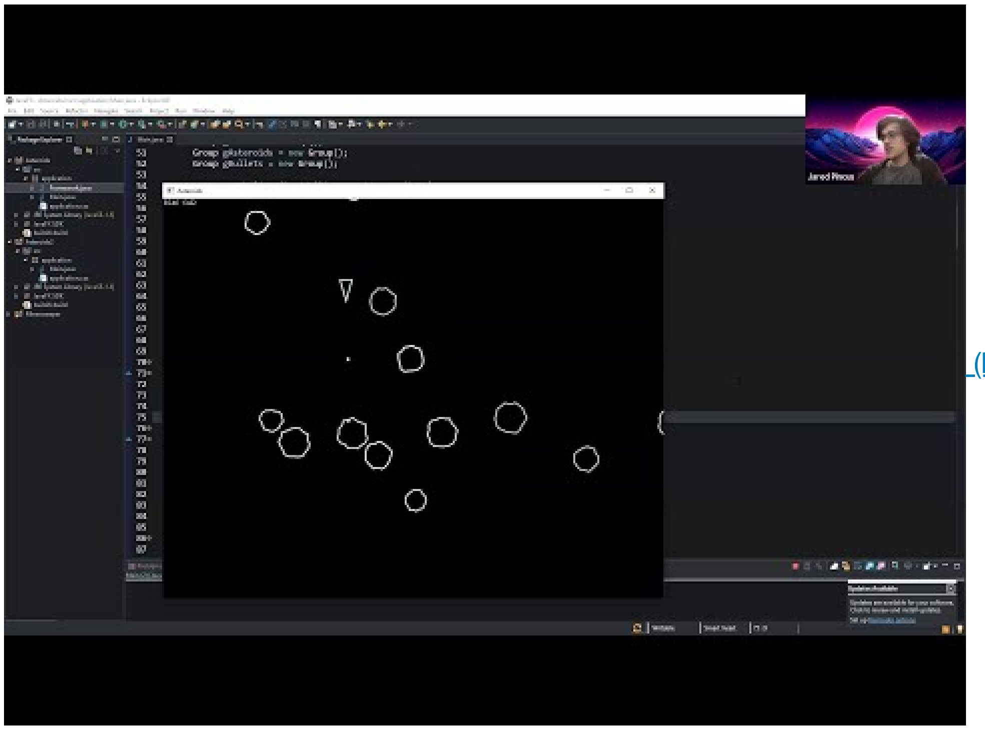
First, watch the demo video so that you know what you will be creating.



(https://www.youtube.com/watch?v=juahrnTmYeo)

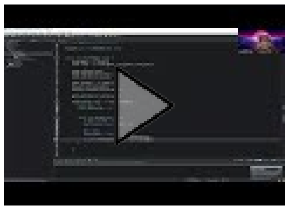

(https://www.youtube.com/watch?v=juahrnTmYeo)

Now that you know what it is, you can learn how to program something like this in a fairly linear style by watching the following video from the gaming club at Stevens Institute of Technology (https://www.stevens.edu/). This enthusiastic programmer creates a simple asteroids game in JavaFX. Almost everything that you need to create Pong is in this video. Watch this before you start digging around on the internet for tutorials.



```
Group gAsteroids = new Group();
Group gBullets = new Group();
```

[(https://www.youtube.com/watch?v=ceMz7JMkKuw)](https://www.youtube.com/watch?v=ceMz7JMkKuw)



[(https://www.youtube.com/watch?v=ceMz7JMkKuw)](https://www.youtube.com/watch?v=ceMz7JMkKuw)

You can even download the source from the linked Github site. The video is easy enough to follow. If you are really new to Java or are intimidated by JavaFX then I recommend that you just type the Asteroids project along with the video to get up to speed.

In any case, watch as much, or as little, of the video as you need to get started on your own Pong game.

Pong 1.0

Now that you're up to speed on JavaFX and the basic linear style, you will want to code your first version of the video game. In the Asteroids game the author uses a simple class hierarchy for his game objects where all game objects extend the abstract class *PhysicsObject*. For this first version of Pong, it's not necessary to create a class hierarchy as the objects are so simple. You may simply encode them as fields in the main class, e.g.:

```
Rectangle bat = new Rectangle(...);
Rectangle ball = new Rectangle(...);
```

Of course you can separate declaration from initialization if you wish.

The purpose of this first project is for you to code freely and solve all of the simple algorithmic problems such as bat/ball/wall intersection. You do not have to write the sound class at all, it is provided in the appendix. Your first version should be as close as possible to the given demo. Your application should respond to two keystroke events: The “i” key will display or hid the fps information and the “s” key will enable and disable sound.

Pong 2.0

The second version of Pong must be “object oriented.” Oh no, that sounds fancy! I assume that you’ve been exposed to these ideas already and I further assume that you realize that, as far as OOP is concerned, that this isn’t a complex project. Hence, the goal here is to think and discuss these ideas while you design your project.

Inheritance and Composition

If you followed (and slightly extended) the model for Asteroids given in the demo, your basic class hierarchy might look something like the following:

```
// Your game object base class
//
abstract class PhysicsObject{}

// your bat and ball objects
//
class Bat extends PhysicsObject
class Ball extends PhysicsObject

// your game model, note, this is combined with the main application
// in Asteroids, but we're presenting it separately here
//
class Pong

// the main application class
//
class PongApp extends Application
```

In this hierarchy the commonality between Bat and Ball objects would be placed in PhysicsObject and accessed via inheritance but the relationship between Bat/Ball and Rectangle would be one of composition, i.e., a hasA relationship as opposed to an isA relationship. The Pong class implements the *business rules* of the application, e.g., how the elements interact, and when the score has increased. The PongApp class sets up the scene and stage as well as the event handling for all scene based mouse interaction. Note, for example, that in the demo, when the mouse enters the scene, the mouse cursor disappears.

Alternatively, your class hierarchy might look something like the following:

```
// your bat and ball objects
//
class Bat extends Rectangle
class Ball extends Rectangle

// your game model, note, this is combined with the main application
// in Asteroids, but we're presenting it separately here
//
```

```
class Pong extends Group

// the main application class
//
class PongApp extends Application
```

With this approach your Bat and Ball can no longer share anything that is note shared by Rectangle, however, they can also be directly inserted into the *scene graph*. This allows them to be treated as first class scene graph objects within a JavaFX application, but, limits what you can do in terms of *implementation inheritance*. Since the Pong model extends Group, it can hold scene graph objects and, consequently, but the root of the scene graph and inserted directly into the scene. One might find this approach convenient, however, one might also think that it violates *separation of concerns* as now our Pong model is, in some sense, a direct element of the user interface.

You are not limited to either of these approaches, but you must choose an approach and you must defend your approach in your writeup. Think about separation of concerns and this application, think about how this application might be extended? Do your choices materially impact the evolution of Pong?

In any case, your OOP version must have the following classes, it may have additional classes, you are free to define the class hierarchy as you wish.

```
class BinkBonkSound                          // sound class, provided in the appendix
class GameTimer                              // class that provides the information display
class ScoreDisplay                           // class that provides the score display
class Ball                                   // the ball
class Bat                                    // the bat
class Pong                                   // the game model
public class PongApp extends Application    // the main application
```

Notice that only PongApp is public. This so that you can point all of the classes in the same file. This is not standard practice in general, but it’s fine for a simple program like this and it makes grading easier.

Don’t get bogged down into the specifics of how the game works, e.g., in terms of timing. Copy the video presentation as best as you can and don’t hesitate to ask questions. For example, you must be able to impart english on the ball when you slide into it, this also starts the ball rotating, but the exact amount of both are adjusted to taste and we won’t worry too much about your program running exactly the same on my machine. There are some things that we can to do to improve this experience and I mention that in the video, but it’s not that critical for this assignment.

Writeup

Formatting

Let’s get this out of the way. I expect a consistent and appropriate format for all writeup submissions. You must include the following:

1. Title Page :: Contains nothing but the assignment name/number, your name, your last four of your student ID (not your SSN, it’s never going to be your SSN, ever), the course, e.g., CSC133, you may

include the section if you want to, and nothing else. Note: it does not include my name. **Do not put my name anywhere on your writeup.**

2. Your discussion :: For this assignment it does not need a header or sub-headers. If you insist on using a header you may use **Discussion**. If you would like to include a brief introduction, that's fine, give it a header of **Introduction**. For more complex assignments this is required, for this assignment it's fine to leave it off.
3. Your code in what amounts to appendices. For these each program should start on a new page with the name of the file at the top of the page as the header. The code should be formatted to not wrap. I find this to be much easier if you limit your column width while coding. Later we will discuss this a bit more in the context of *clean code*. If necessary you may shrink the font somewhat. However, if you are below 9 points with 1/2" margins and your code still wraps, then break the lines in your text editor.
    1. **Code in your writeup is black on a white background and is not screenshots**. I don't want to see syntax highlighting and I definitely do not want to see grainy screenshot fonts and weird artifacts along image edges. Copy and paste the *text* into whatever you are using to format your writeup and format your text with a monospace font in that environment.

Content

I want you to discuss the choices that you made in terms of your object decomposition and why you made them. Justify your approach as best as you can. Do not use filler words or hyperbole. Do use technical language. You may lean on ideas in the Martin Robillard textbook if this helps. You are limited to two pages max.

Submission

You will submit the following:

1. PongApp.java - This is version 1
2. OPongApp.java - This is version 2, with your chosen hierarchy
3. hw1.pdf - This is your writeup
4. hw2.mp4 - This is a video (1 minute max), see below

To be clear, yes, you will submit both java files separately and you will also submit them as a part of the writeup. The video should show you compiling and running the game from your development environment. Demonstrate as much as you can in a minute and narrate your video. I want to hear you talk about your app.

The video must be an mp4 file, and links to external sites are not acceptable. If you have never done this before then I recommend OBS Studio. It's free and easy to capture your screen to make a video. Once captured, choose file/remux, find your captured video file, and tell it to "remux" and it will create the mp4 for you in the same place, easy peasy and totally freesy.

Appendix

```
class BinkBonkSound {

    // magic numbers that are not common knowledge unless one
    // has studied the GM2 standard and the midi sound system
    //
    // The initials GM mean General Midi. This GM standard
    // provides for a set of common sounds that respond
    // to midi messages in a common way.
    //
    // MIDI is a standard for the encoding and transmission
    // of musical sound meta-information, e.g., play this
    // note on this instrument at this level and this pitch
    // for this long.
    //
    private static final int MAX_PITCH_BEND = 16383;
    private static final int MIN_PITCH_BEND = 0;
    private static final int REVERB_LEVEL_CONTROLLER = 91;
    private static final int MIN_REVERB_LEVEL = 0;
    private static final int MAX_REVERB_LEVEL = 127;
    private static final int DRUM_MIDI_CHANNEL = 9;
    private static final int CLAVES_NOTE = 76;
    private static final int NORMAL_VELOCITY = 100;
    private static final int MAX_VELOCITY = 127;

    Instrument[] instrument;
    MidiChannel[] midiChannels;
    boolean playSound;

    public BinkBonkSound(){
        playSound=true;
        try{
            Synthesizer gmSynthesizer = MidiSystem.getSynthesizer();
            gmSynthesizer.open();
            instrument = gmSynthesizer.getDefaultSoundbank().getInstruments();
            midiChannels = gmSynthesizer.getChannels();

        } catch (MidiUnavailableException e) {
            e.printStackTrace();
        }
    }

    // This method has more comments than would typically be needed for
    // programmers using the Java sound system libraries. This is because
    // most students will not have exposure to the specifics of midi and
    // the general midi sound system. For example, drums are on channel
    // 10 and this cannot be changed. The GM2 standard defines much of
    // the detail that I have chosen to use static constants to encode.
    //
    // The use of midi to play sounds allows us to avoid using external
    // media, e.g., wav files, to play sounds in the game.
    //
    void play(boolean hiPitch){
        if(playSound) {

            // Midi pitch bend is required to play a single drum note
            // at different pitches. The high and low pongs are two
            // octaves apart. As you recall from high school physics,
            // each additional octave doubles the frequency.
            //
            midiChannels[DRUM_MIDI_CHANNEL]
                    .setPitchBend(hiPitch ? MAX_PITCH_BEND : MIN_PITCH_BEND);

            // Turn the reverb send fully off. Drum sounds play until they
            // decay completely. Reverb extends the audible decay and,
            // from a gameplay point of view, is distracting.
            //
            midiChannels[DRUM_MIDI_CHANNEL]
                    .controlChange(REVERB_LEVEL_CONTROLLER, MIN_REVERB_LEVEL);

            // Play the claves on the drum channel at a "normal" volume
            //
            midiChannels[DRUM_MIDI_CHANNEL]
                    .noteOn(CLAVES_NOTE, NORMAL_VELOCITY);
        }
    }

    public void toggleSound() {
        playSound = !playSound;
    }
}
```