**Sw Devt IV – Advanced .NET 420-411-DW  
Project 1 – Tetris  
Phase 2 – Business classes**

**Project Phase 1:** Your first attempt at design - *completed*

**Project Phase 2:** Complete implementation of all business classes, and perform complete unit testing

**On deck - Project Phase 3:** Add a Monogame UI which will take care of the game loop, user input and drawing on the screen.

**Phase 2**

The overall design of the business classes is provided in the last page of this document. Please see your instructor to discuss any design changes that you wish to make. You will notice that the UML provided is **not** complete: for example, the constructors are not indicated, nor the methods which must be implemented due to an interface, nor all the private instance variables and methods that you may wish to add.

**Rotation** is the trickiest part of Tetris. Here are two options for you to consider:

* 1. Recommended: set up a “template” of the possible rotation offsets when you instantiate each shape subclass; e.g. an array of Point[][] that contains the offsets that you will apply for the possible configurations of the Shape. For example, a shape with 4 possible rotations will have an array of 4 Point arrays; each Point array contains the offset Point to add to the Block’s position. You will have to adjust actual (x,y) positions of the Block based on the offset. The origin block will always have an offset of (0,0). This option is what is assumed in the design provided.
  2. alternatively, each Block can rotate on its own around another Block (i.e., the origin block). The pseudo-code to accomplish this:

Point newPosition;

newPosition.X = origin.Position.X – origin.Position.Y + currentPosition.Y;

newPosition.Y = origin.Position.X + origin.Position.Y - currentPosition.X;

currentPosition = newPosition;

Notice that not all shapes (i.e., the shapes with less than 4 rotations) follow this algorithm. So these shapes will need to override this general algorithm. This approach of calculating based on the origin block is a trickier; consider it only if you thrive on challenges!

If a rotation is not possible (if the new configuration is not a valid move), you again have options

1. do nothing – i.e., nothing changes. This is what my specs require.
2. keep rotating until you reach a valid configuration (i.e., in worst case, you will stop back at the current configuration)

**Unit testing:**

For every concrete class, code a matching unit test class (in a unit test project) to unit test its functionality. One of the advantages of coding to interfaces, as done in the design below, is that you can create test stubs (also called mocks or fakes) of interacting objects to facilitate your unit testing. For example, in the design below, the Blocks need a handle to an IBoard in order to validate if moves are permitted. Instead of instantiatiating a Board object and somehow filling it, you define and instantiate a FakeBoard which implements IBoard and that allows you to set up your test case situations more easily and with less dependencies on code that has not yet been tested.

**Stay tuned: Upcoming activities that will help you!**

We will add unit testing and events to the Pong lab!

**Tetris Business classes**

