Homework 3 – Inheritance & Monsters

**Due: Friday, October 18th by 11:59PM (Friday night)**

It’s common to create a hierarchy of classes when representing multiple types of objects with shared attributes. In Geometry Wars, for instance, each enemy shape has different behaviors, but each one has a color, direction and position on the screen. These attributes can be represented as fields in a parent class. Unique behaviors can be defined in classes that inherit from the base enemy class.

This homework is different than the ones you have completed before. A framework has been created for you, and you will be using that framework. This framework allows the console window to function as if it were a more game-like application. Rather than all of the code appearing in Main, the Game class has a loop (called the “game loop”) through which the game is simulated and drawn to the screen.

Part of your job in this assignment is to read through the starter code to understand what it does. *Note: you will create several new classes for this assignment, but will only modify one of the four included code files.*

# Task Overview

This is a brief overview of the tasks you must complete for this assignment. Specifics are given in the corresponding sections later in this document. You should ***read the entire document*** before starting.

* Begin with the **provided starter code**
* **Read** through the documentation below about the starter code architecture
* Create a **base Monster class** which represents a keyboard-controlled ASCII monster
* Create **4 subclasses of Monster**, each of which has a different behavior as outlined later
* Add code to the **Homework3Game class** to instantiate, update and draw your monsters
* **Don’t edit** the KeyboardHelper class, the Game class or the Main method!

# Starter Code

There is a starter project you’ll need to use for this homework. The starter project has four classes that help you treat the console window like a game window. Before you add any code to your assignment, run it and take a look at the code to see how it works. (The Console window may flicker depending on your computer, and that’s ok! The Console window was never designed to be used like this, so there are limitations.) Here’s a breakdown of the classes:

**KeyboardHelper** – ***Do not edit!***

This is a helper class that uses some Windows-level functions to get raw keyboard input (rather than relying on Console.Read() or other Console methods). It contains the following:

* **KeyCode** – An enumeration of several commonly used keys
* **IsKeyDown** – A static method that returns true if the key with the specified KeyCode is down, and false if the key is up
* **IsKeyUp** – A static method that returns true if the key with the specified KeyCode is up, and false if the key is down (this is the inverse of IsKeyDown).

**Game** – ***Do not edit!***  
This class drives the “game”. It contains a game loop, which will continually call its Update and Draw methods at specific intervals until the game is over. It contains the following:

* Fields
  + **gameOver** – A Boolean that controls the game loop
  + **targetFPS** – The approximate frames per second for the game loop
  + **timeBetweenFrames** – How long to wait to achieve the target FPS
* Properties
  + **GameOver** – Gets or sets whether the game is over
  + **TargetFPS** – Gets or sets the target FPS & calculates the time between frames
* Constructor
  + **Game** – Takes an optional targetFPS value and sets gameOver to false
* Methods
  + **GameLoop** – Calls Update and Draw until the game is over
  + **Update** – A virtual method you can override to update your game. By default, it Sleeps a specific amount of time to achieve the target FPS
  + **Draw** – A virtual method you can override to draw your game. By default, it clears the console window

**Program.cs & the Main method** – ***Do not edit!***

The **Main** method simply creates a Homework3Game object (see below) and launches its game loop

**Homework3Game** – ***Edit this!***

This class inherits from the Game class and overrides its Update and Draw method. You will need to edit this class for the assignment.

* Fields: These are just examples – remove them and add some of your own (see the next section)
* Constructor: You’ll need a constructor, but what it does is up to you
* Methods
  + **Update** – This method overrides the base class’s Update, but also calls that version to ensure the program runs at the correct frames per second. Other than the call to base.Update(), you can remove the example code in this method.
  + **Draw** – This method overrides the base class’s Draw, but also calls that version to clear the console window. Other than the call to base.Draw(), you can remove the example code in this method.

# Where to Begin

You’ll be creating several “monsters” that move around in the console window. Each monster should be shown as a different colored ASCII character, and will be represented in your code by a different class. Each class will need to inherit from a base class called Monster.

Each monster sub-class will need its own behavior as it moves around the screen. A list of required behaviors can be found in the next section.

# Monster Class

You’ll be creating a base “Monster” class, which holds common information that all Monsters share and has common methods for actions that all Monsters can perform.

**Fields**

The Monster class will need fields for the ASCII character that represents the Monster in the console window, the X & Y positions of the Monster inside the window and the color of the Monster (which will be a ConsoleColor value).

**Properties**

Create properties for any fields that you may need to access from outside the class.

**Constructor**

Make a parameterized constructor that accepts any required data and sets all the fields.

**Methods**

The Monster class will need a **Draw** method, which moves the Console’s cursor position to match the Monster’s current position, sets the Console’s foreground color to the Monster’s color and then “writes” the Monster to the Console.

You’ll also need to create a virtual **Update** method that checks the keyboard for input and lets you move the monster around the screen. Be sure to clamp the monster’s position to the Console window’s width and height; if you attempt to set the cursor position outside the window, it throws an exception. This particular method will be overridden in child classes.

# Sub Classes

Make 4 sub classes of Monster. Each one should have a different visual representation (a different single ASCII character is fine) and a different color, which can be hardcoded as part of each class’s base constructor call.

For example, if you had a subclass called Bouncer, all Bouncers might be represented by a green letter “B”. The Bouncer class could pass these options directly into the base class’s constructor.

Each subclass should override the Update method and provide different behavior (you don’t need to call the base class’s Update here). Each time Update is called, something about the monster should change on the screen. You are required to implement the following behaviors:

* A monster that bounces around the screen – when it hits a wall, it will “bounce” off
* A monster that wraps around the screen – when it hits a wall, it “wraps” to the opposite side of the screen (left to right, top to bottom, etc.)
* A monster that moves in a regular pattern – Perhaps it moves in a set rectangular pattern, or a circular pattern (look into the Math.Sin() and Math.Cos() functions for this).
* A monster that moves around randomly – Uses a random number generator to move

# The Homework3Game Class

Inside this class, create variables to hold the 5 different types of monsters: a base Monster object and the 4 sub-class objects. You can take advantage of Polymorphism and create a single List<> (the built-in C# data structure – no need to re-code it from scratch here) that can hold all of the monsters.

The Update method should call update on each monster, and the Draw method should call Draw on each monster. When the program runs the 4 monster subclass objects should move automatically and the base Monster object should be controllable by the user.