**Due Date:** December 10, 2018 @ 11:59 PM

**Assignment Objectives:**

* Implement a project using abstraction, interfaces, and polymorphism
* Practice proper object oriented programming techniques and design
* Develop algorithms to solve problems.

**Book Material Covered:**

* Chapter 5 - 11

**To complete this assignment:**

* Check out the proj04 directory from the SVN repository:  
  https://csprojects.cs.ndsu.nodak.edu/csci160/2018/fall/svn/username/proj04
* Look over the assignment and make a quick mental estimate of how much time you expect proj04 to take you to complete. You should record this initial estimate in the proj04 Questions document.
* Create all of the various interfaces, abstract classes, and classes based on the API documents included with the assignment. You can make additional classes as needed for various features you want to add.
* Complete the GamePanel class by instantiating and using instances of the classes you have created and other provided classes.
* Complete the Blackboard portion of the assignment by answering the questions listed in the proj04 Questions Word document. Also, indicate the enhancements added to your game as a part of the Questions document.
* Compare the actual amount of time it took you to complete the assignment with your original estimate. This may be helpful for developing your estimation abilities going forward.

**Grading:**

Blackboard Questions 5 Answer and complete the Blackboard questions

Comments 25 Add comments to your code to improve readability

Style / Formatting 25 Format code properly to improve readability

Proper Inheritance Use 15 Use Abstraction and Inheritance appropriately

Functionality 75 Ensure program works as described

Design 20 Practice good Object Oriented techniques

Visual Look / Style 10 Visual appeal of program.

Enhancements 25 Additional functionality added (see relevant section)

Total: 200

**Instructions:**

You will be creating a simple game modeled after the brick-breaking game Breakout where a ball bounces off the sides and top of the screen and destroys bricks that it comes into contact with and is kept in play by a user-controlled paddle at the bottom of the screen.

Your program should exhibit the following behavior:

* The ball will bounce off the side or top walls.
* If a ball comes into contact with a brick, the brick will be removed, the player’s score will be increased, and the ball will bounce off of it.
* If the ball comes into contact with the paddle, the ball will bounce off of it.
* The player can control the paddle using the keyboard.
* The paddle should not be able to move off of the screen.
* If the ball touches the bottom of the screen the player will lose a life.
* If the player runs out of lives, they will get a game over message.
* If the player destroys all of the Bricks, they will get a message indicating they won.
* Regardless of whether or not a player wins or loses they will be asked if they want to play again.
* Bricks should have different colors based on their row or some other design.
* There is at least one uncontrolled, moving obstacle between the paddle and the bricks.
* The ball will bounce off of the obstacle if it comes into contact with it.
* When a brick is destroyed, there is a chance that the game will spawn an extra life power-up.
* The extra life power-up will travel down the screen and will disappear after moving far enough.
* The player can collect the extra-life power-up by touching it with the paddle.

**Provided Classes:**

The following classes are provided either partially or in entirety for your use.

GameFrame:

This class contains the main method of the application and is used to create the Window application and start up the game. It is not necessary to modify this class at all. If you are adding a main menu as an enhancement, modifying this class may be necessary.

KeyController:

This class implements the KeyboardListener interface and provides the ability to control the game with keyboard commands. It is not necessary to modify this class at all. However, you will need to call various methods of this class to determine if a key is being pressed. By default there are methods included that will tell you if any of the arrow keys or the space key are being pressed.

If you want to determine if other keys are being pressed, call the getKeyStatus method which takes a single argument of type int corresponding to a keyboard key and returns a boolean value indicating whether or not that key is being pressed down at the time.

You can look up the different keyboard codes online to find the values for different keys or you can use static contants of the KeyEvent class (e.g. KeyEvent.VK\_P which is the key code for the 'p' key on a keyboard) instead.

GamePanel:

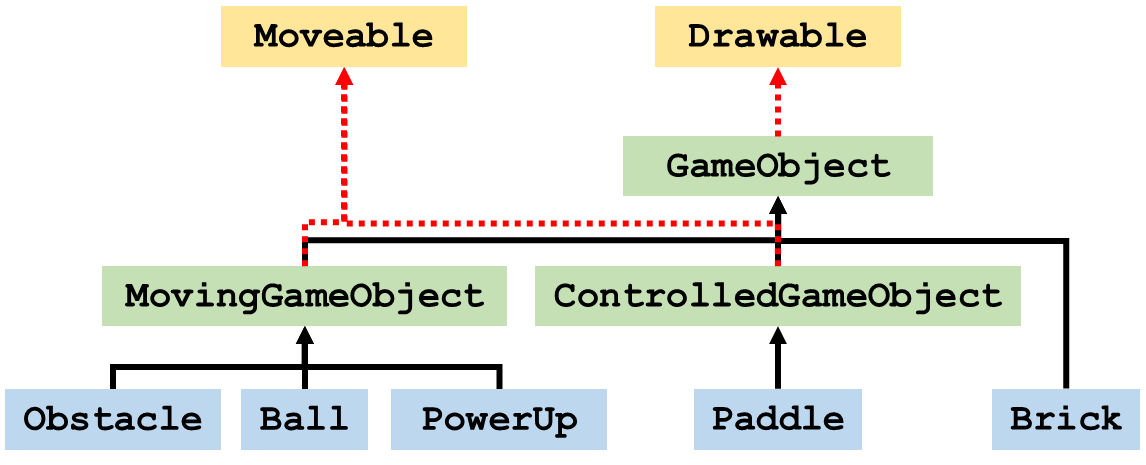
This class contains the majority of the code that will drive your game. The code necessary to animate the game has already been provided, so it is not necessary for you to implement any of this functionality yourself. The class also contains some empty methods that you should use as the code that is already written will call these methods:

* + The **paint** method should contain method calls used to draw game objects or text to the screen.
  + The **updateGameState** method should contain code to move game objects and to check if game objects are colliding or if the game has ended.
  + The **setupGame** method should contain code to instantiate any instance variables and to setup the game. This method is called by the constructor method of the GamePanel class and can be used to restart the game.

**Recommended Classes:**

It is recommended that you create the classes for which you have been provided an API. These will provide you with most of what you need in order to complete this assignment. However, you may wish to add additional functionality to some of these classes beyond what is described in the API. GameObject and its subclasses and the Interface classes you create should be placed in separate packages.

The following diagram shows the suggest class hierarchy to be used:



The beige boxes represent interfaces, the green boxes are abstract classes, and the blue boxes are classes. Red dashed lines indicate that a class (or abstract class) implements an interface, and solid black lines indicate inheritance.

It is also highly suggested that you create a GameLogic class that contains several static methods such as checking for collision between the ball and walls, the ball and paddle, the ball and bricks, etc. This allows your GamePanel class to call these methods and reduce the overall clutter inside of your GamePanel class. It may also help you to focus on smaller problems without having to worry about the larger behavior of the program.

**Enhancements:**

Your program should contain at least three of the following in addition to the functionality described above:

* Bricks can take more than one hit to destroy and change color based on the hits they’ve taken. Alternatively as they take more hits they can show some visual sign of damage such as having cracks.
* When a Brick is destroyed, it doesn't just vanish, but is broken into smaller chunks that crumble. For add challenge, you can try to make the chunks fly off in certain directions based on where the Brick was struck when it was destroyed.
* The player can control the movement of the ball to some degree based on how they hit it with the paddle. Hitting it on the left or right side will increase the speed of ball in that direction while hitting it in center will decrease the horizontal speed.
* The paddle can wrap around the screen (i.e., if it moves off the left, it will start appearing on the right) instead of being stopped by it. For an extra challenge make the paddle able to be partially on both sides of the screen at once, rather than just warping from one side to the other.
* Modify the game so that the ball doesn't bounce off of the walls, but will appear on the other side. You cannot choose to implement this enhancement if you have also added the ability for the paddle to wrap around the screen as well.
* Add a high-score list to the game that can keep track of multiple plays and will persist even if the game is closed.
* Include multiple levels in the game so that if you destroy all of the bricks and have not run out of lives, you will get a new level, possibly with a different arrangement of bricks.
* Add additional simple power-ups to the game beyond an extra life. These can do things making the ball move more slowly, making the paddle wider, splitting the ball into multiple balls, or allowing the paddle to shoot lasers that can destroy bricks.
* Add a difficulty setting option to the game that changes how difficult the game is. You can do this by modifying the speed that the paddle can move, how fast the ball travels, and how wide the paddle is.
* Implement more realistic collision detection that is able to determine where the ball collided with something and how it should bounce off based on that. For example, if the ball hits the side of a Brick, it should continue its existing vertical movement.
* Add the ability for a player to pause the game by pressing some key on the keyboard. When this occurs, the game should display some indication that it is paused, such as letters in the Background of the game or having a pop-up.
* Add the ability for a player to control the paddle with a mouse in addition to a keyboard.
* Add a menu at the start of the game where the player can choose between options like starting a new game, viewing high scores, etc.
* Add a motion blur trail to the ball to give some additional appearance of movement and speed. When the ball is moving faster, it should have a longer motion trail. The trail should decrease in size and have the color fade out towards the end of the trail.
* Add sound effects and music to you game. You may want to use some of the classes included in lab10 to help you out with this. Alternatively you could record your own sound effects and load and play those from file.

**Where to Start:**

It is suggested that you complete all of your classes first by implementing them based on the API. Creating the interfaces first, followed by the abstract classes, and finally the concrete classes is the best approach.

Once those have been completed you can start to add small bits of functionality at a time. For example, you may just want to start be having a ball and getting it to bounce off of the walls before worrying about adding the paddle. After that you can add the paddle and get it to move before worrying about getting the ball to bounce off of it. Similarly, you may want to try just getting several rows and columns of bricks added before you worry about the ball being able destroy them.

It is also beneficial to create multiple helper methods that have very limited functionality. For example, you can create a method that will check to see if the ball is colliding with an obstacle and a separate methods to see if the ball has collided with any of the bricks.