CSC 360 Programming assignment #3
Due date: Wednesday, May 30 by noon

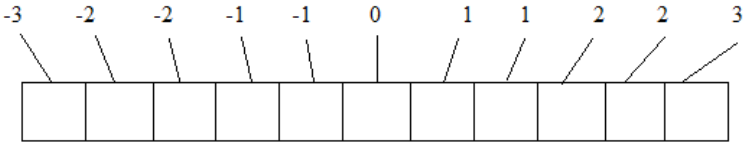
For this assignment, you will create the game Breakout. If you do not know Breakout, it is a game with a bouncing ball, paddle and a series of bricks. The object is to hit the ball with the paddle so that it bounces to the other end of the game field and hits one or more bricks. As each brick is hit, it disappears and the user scores points. If the user can clear the entire field of breaks, then the game resets with a new field of bricks. You may have already implemented a portion of this if you have done chapter 15 exercises 5 & 6. If you haven't look at my solution to see my code. You should start with your own code though as you will want to implement this in your own way.

How the game works:

1. Have bx, by, bdx and bdy variables to represent the ball's location and velocity. The ball is a Circle.
2. Have px and pdx to represent the paddle's upper-left corner and its motion in the x direction. The paddle does not move in the y direction. The paddle is a Rectangle.
3. Have any array of Bricks. Bricks are defined below as a separate class (either a nested inner class or a class in a separate file, your choice).
4. Declare the following as instance data as they will be shared among various methods and event handlers:
    a. Circle ball – the ball (needed to add to and remove from the Pane as it moves)
    b. Rectangle paddle – the paddle (add and remove from Pane as it moves)
    c. int score, lives – player's current score and lives remaining
    d. Text description – output of the current score and number of lives remaining
    e. bx, by, bdx, bdy, px, pdx as noted above (used in several locations)
    f. the array of Bricks (this is a 2-D array)
    g. the Pane and the Timeline object
    h. a Random generator
5. Your main method will call launch
6. Your start method will instantiate or initialize all of your instance data
    a. for the ball, set bx to the middle horizontally and by to a value beneath the Bricks but well above the paddle, give px a value in the middle horizontally, pdx should start at 0, bdx and bdy should be given random values as long as bdy is positive (so that initially, the ball is moving downward), limit bdx, bdy, pdx to a reasonable range (say -3 to +3)
    b. create the Text, Circle and Rectangle objects and add them to the Pane; also draw four Lines to have borders around the playing field
    c. instantiate all of the Brick objects and for each Brick object, pass it the Pane object; Brick will contact a draw method that will draw a Brick onto the Pane (described below)
    d. create a Timeline object with its own event handler (described in 7) and attach to the Scene a Key EventHandler (described in 8), use a reasonable duration (mine was 15, if you have a larger number, you will want to allow larger ranges for bdx/bdy and pdx)
    e. set the timer to repeat indefinitely and start animation, add the pane to the scene and the scene to the stage, show the stage

7. Your Timeline object needs an ActionEvent handler; this can be defined as a lambda expression, anonymous inner class, or nested inner class, your choice. This handler will work as follows:
    a. Move the ball.
        i. See if it has hit the left, right or top wall and if so, bounce it off (multiply bdx or bdy by -1 depending on which wall it hit)
        ii. See if it has hit the paddle and if so, bounce it back (multiply bdy by -1); also change bdx based on where the ball hit the paddle (see the figure below)
        iii. If the ball has gone passed the paddle in the y direction, the user loses that life, subtract 1 from lives, and if lives is 0 stop the timer and display a "game over" type message, otherwise remove the Text object from the Pane, create a new one with the new number of lives, and add the new Text object to the Pane and start the ball anew (I have a separate launchBall method to start the ball each time needed)
        iv. See if the ball has hit a Brick and if so, remove the Brick. This is described in the Brick class below. If there is a hit, update the score, remove the Text object from the Pane, create a new Text object with the new score, add the new Text object to the Pane.
    b. Move the paddle. If the paddle has hit the left or right border, stop it from moving. In my program, I rebounded the paddle the other direction setting the pdx value to 1 or -1 depending on which wall it hit (left wall, pdx = 1, right wall, pdx = -1)
8. The Key Handler needs to determine which key was pressed. If left arrow then decrement pdx and if right arrow then increment pdx. Keep pdx within a reasonable range, say -3 to +3. If down arrow (or space bar, your decision), set pdx to 0 (this stops the paddle).

Upon impacting the paddle, the ball should bounce backward. The bdy value should be negated (that is, bdy *= -1;). The bdx value will change based on which portion of the paddle was hit. Below is an example of how you might set this up. If you prefer to just keep bdx the same, that's fine, but it will make for a very boring game. The idea is that the further away from the center of the paddle that the ball hits, the greater the x velocity will be (negative if it hits on the left side of the paddle, positive if it hits on the right side).

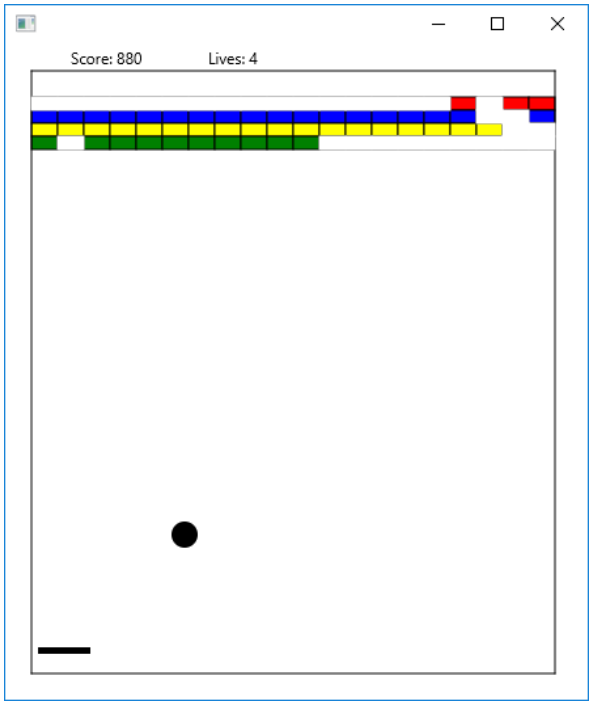


The Brick class represents a single Brick in the game. Each Brick will have an x, y coordinate (the upper-left hand corner of the Rectangle), a Color, a boolean of whether it is visible or not, and a value. For each row of Bricks, their value increases. So for instance, in my game of 4 rows, the top row is worth 40, the bottom row is worth 10. I also have given each row a different Color. The Brick class needs the following methods:

1. 4-arg constructor to set x,, y, value, Color. Assume visibility is true. You can also provide a 5-arg constructor which also receives the initial visibility. This allows you to generate a field of Bricks in which not all Bricks are visible to start. You could for instance create a pattern like a checkerboard to start a level, or randomly generate which Bricks are initially visible. This is an enhancement that you should only work on if you get the rest of the program working.

2. A collides method that receives the ball's location (bx, by) and returns true if the ball has hit this Brick, false otherwise. Remember that bx, by are the center of the ball while the Brick's x,y are its upper left hand corner. You will have to determine if any part of the ball hit any part of this Brick. Only test this if the Brick is currently visible. If it is not visible, it doesn't really exist.
3. A draw method which receives the Pane object from the start method. If the Brick is visible, then do these steps. Create a Rectangle with the x,y coordinate of this Brick, set its fill Color to the Color of this Brick. Draw the Rectangle on the Pane. Obviously if the Brick is not visible, you would skip this step. NOTE: I added a border around each Brick by using setFill (to the Brick's Color) and setStroke (to Black). This caused an oddity in that some Brick outlines disappeared once their neighbors were removed.
4. A remove method that, if invoked from your Timeline's action handler, causes this Brick to "disappear". This method receives the Pane. This method sets visible to false, creates a new Rectangle that is white and draws this Rectangle on the Pane. What this does is "replace" the Brick with a white one and thus renders it invisible. In my implementation, this method returned the value of the Brick that was hit to be added to the user's score. You can do this in other ways although removing a Brick by using pane.getChildren().remove(…); is harder because a Brick is not the same as the Rectangle object added to the pane.
5. If needed, accessor methods for getValue and isVisible.

Below is my version of the game after it has been played for a few minutes. The player was fortunate enough to get the ball up to the top where it wiped out most of the red Bricks. The paddle is near the left border (that isn't some stray line).



NOTE: your game does not have to be perfect. In mine, when the ball hits a wall, it actually goes slightly into the wall before bouncing. I also have a problem in that hiding the very middle of the

bricks has the ball go right through them (I haven't figured out why). The importance is being able to implement a game that uses the keyboard, a Timeline object, and have appropriate logic to deal with collisions and motions.

Do not make an overly elaborate version of this game unless you have time and only then if you are interested in seeing what else you can do with it. Get the game working first and foremost.

Submit all of your code (preferably a single program with one or more nested inner classes) and three screen captures – the start of the game, somewhere in the middle after some Bricks have been removed, and the end of the game. All of this can be placed into a single Word document. Submit your file or files to foxr@nku.edu.