CSC 7200
Week 4 Programming Assignment

Design Patterns Assignment

Create a rocket-powered duck using the code found in the HFDP Strategy source code. Modify the test class (e.g., MiniDuckSimulator or MiniDuckSimulator1) to demonstrate the new rocket-powered behavior. This type of duck also makes a whoosh type quack. Ensure that the test class also shows this new kind of quacking.

HeadFirst Design Patterns (HFDP) code is here: http://www.headfirstlabs.com/books/hfdp/ (in particular, the code itself is at this link http://www.headfirstlabs.com/books/hfdp/HeadFirstDesignPatterns_code102507.zip)

JUnit Testing Assignment

In addition, create a JUnit project using the code in the MyClass and MyClassTest classes below. Execute the code using the Eclipse or NetBeans JUnit plug-in. Compare your results as shown in the JUnit Viewer with the sample shown in the "JUnit Tests" part of the Lecture and also the Vogella Tutorial linked from the Lecture. Correct the error in the MyClassTest code and re-run. Does a green bar now show in the Viewer?

MyClass

```
package junit.test;

public class MyClass {
	public int multiply(int x, int y){
		return x * y;
	}
}
```

MyClassTest

```
package junit.test;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class MyClassTest {
	/**
	 * Test method for {@link junit.test.MyClass#multiply(int, int)}
	 */

	@Test
	public void testMultiply() {
```

```
            MyClass tester = new MyClass();
            assertEquals("Result",50, tester.multiply(10, 4));
            //fail("Not yet implemented");
      }
}
```

Java Graphics in a Paddle-Ball Game Assignment

The purpose of this week's programming assignment is to learn how to deal with events from multiple sources. In addition, we will learn how to use some of Java's graphics capabilities for drawing shapes in a display area. The end result of this week's programming assignment will be a simple paddle-ball game. Along the way, we'll learn more about Timers, Mouse Events, and Graphics in Java.

Paddle Ball Game Overview

The paddle-ball game is a simplification of the Pong game. In the Pong game, a ball is moving around the display, bouncing off walls. The player moves a paddle in one dimension, trying to hit the ball whenever possible. If the ball hits the paddle, it bounces off and continues on its way. If the paddle misses the ball, the ball goes off the end of the display area and the player loses a point.

In our paddle ball game, the ball moves around the display, bouncing off walls just as in the Pong game. The player controls a paddle trying to hit the ball just like in the Pong game. If the ball hits the paddle, it bounces off and continues on its way. The main difference in our game is that if the player misses the ball, the ball simply bounces off the wall behind the paddle and continues to travel. There is no scoring in this game.

Object Oriented Design

If we analyze the game description we can see there are several different objects which are interacting. There is a ball object which is traveling around the display. There is a paddle object which is being moved by the player. There is a display which is presenting the graphical representation of the game to the user. These objects are readily apparent from the description of the game.

The ball object has a size and a position within the field of play. It also has a direction of travel in both the X and Y dimensions. The ball must be able to update its position when needed. Since the ball normally bounces off walls, it needs to know the length and width of the field of play when updating its position. The ball must also be able to draw itself.

The paddle has a size and a position within the field of play. The paddle must be able to update its position within the field of play. The paddle must also be able to draw itself.

The display has a display area which has a length and width. It has a means for drawing graphical shapes within the display area. The display area must be able to update the display area the position of the ball or paddle changes. The display itself has no concept of the game or its objects, so it will need to interact with some other object to draw all the game objects.

At this point, we have a ball, a paddle, and a display object. What is missing is some object which provides the game context. The ball has no awareness of the paddle. The paddle has no awareness of the ball. The display has no awareness of either the ball or paddle. We need an object which enforces the rules and concepts of the game and manages the behaviors of the other objects involved in the game.

Let's call this the controller object. The controller has a ball, a paddle, and a display object. The controller manages game events. It gets mouse events and tells the paddle to update its position based on the position of the mouse cursor. It gets timer events and tells the ball to update its position based on the size of the display area which it gets from the display. It tells the display when to redraw the display area. It also provides a method which the display object uses to request the controller to draw the current state of the game. The controller is responsible for telling the ball and paddle to draw themselves.

The controller is also responsible for detecting when game objects are interacting. Specifically, it must be able to determine when the ball has made contact with the paddle. This requires that the controller is able to determine if a specific surface of the ball is in contact with a specific surface of the paddle. When this is detected, the controller tells the ball to reverse the direction of travel in the X or Y direction. This places a requirement on the ball and paddle to provide methods allowing the controller to determine the X or Y position of any given surface of the ball or paddle.

The `Ball` Class

Given that the ball has a size, a position, and moves a specific amount on each update, the Ball class will need member variables for all of these attributes. The size of the ball (diameter) and the number of pixels to move per update should be established when a ball object is created (explicit value constructor). To simplify things, the distance to move (increment) will initially be the same in the X and Y directions. The X and Y position members need to be set to some starting position by the constructor. The X, Y position of the ball is the upper left corner of the minimal box which contains the ball.

When the X increment is positive, the ball is traveling to the right. When it is negative it is traveling to the left. When the Y increment is positive the ball is traveling down. When it is negative it is traveling up.

According to the description in the previous section, the ball must provide the position of its edges to the controller. To do that, the Ball class provides the following methods:

getTop, getBottom, getLeft, getRight

The first two methods return the Y coordinate of the top/bottom edge of the ball. The other two methods return the X coordinate of the left/right edge of the ball. These values can easily be calculated from the X, Y position and the diameter of the ball. The Ball class must provide a method which takes a Graphics object parameter and uses it to draw the ball at its current position.

When the controller detects that the ball has hit the paddle, the controller must tell the ball to reverse its direction of travel in either the X or Y direction. To be flexible, the Ball class should provide two methods, one to reverse the X direction and one to reverse the Y direction. To the ball, reversing direction simply means negating the value of the X or Y increment.

The controller also must know whether the center of the ball is within the boundaries of the paddle in order to detect contact. This means the Ball class must provide methods to report its horizontal or vertical center. This can be easily computed from the current position and the diameter.

Finally, the Ball class provides a method which the controller calls any time the ball needs to change position. This method adds the X increment to the current X position, and the Y increment to the current Y position. The Ball class is responsible for detecting when it has encountered a wall. So, this method needs to know where the edges of the game area are. This method must be given the length and height of the game area as parameters. The following conditions must be checked for detecting contact with a wall:

1. Top edge of ball <= 0 then reverse Y travel direction
2. Bottom edge of ball >= height then reverse Y travel direction
3. Left edge of ball <= 0 then reverse X travel direction
4. Right edge of ball >= length then reverse X travel direction

The `Paddle` Class

The Paddle class has a length and a position. Both of these should be initialized by an explicit value constructor. That allows the controller to determine the paddle length and the position of the paddle relative to the wall. We will simplify things by stating that the paddle will have a horizontal orientation and will move left to right only. The paddle cannot move in the vertical direction once it has been created. The X and Y position of the paddle should represent the center of the top surface of the paddle. This will support the correlation of the mouse cursor position to the center of the paddle.

The draw method of the Paddle class takes a Graphics object parameter and draws a filled rectangle based on the current position of the paddle. Drawing a rectangle is based on the upper left corner of the rectangle as its reference position. Calculation of the reference position is easily

done using the X and Y position of the paddle and the length of the paddle. The Paddle class controls what the width of the paddle will be.

To support the controller, the Paddle class must have methods which return the position of the top, bottom, left, and right edges of the paddle. The controller needs this information for detecting when the ball and paddle come into contact. The getTop and getBottom methods return a Y position. The getLeft and getRight methods return an X position. These values are easily calculated from the X and Y postion, and the length and width of the paddle.

Finally, a method must be provided to allow the controller to change the position of the paddle based on the X and Y coordinates of the mouse cursor. This method must restrict the movement of the paddle to changes to the X coordinate only.

The `Display` Class

The Display class provides a window to present the visual rendering of the state of the game. It also must interact with the controller. The easiest way to create a Display class which can draw various graphics is to have the class extend the JPanel class and override the paintComponent method. This method should simply fill a rectangle with some background color. The size of the rectangle is simply the size of the panel which is acquired from the getWidth and getHeight methods of the JPanel class. After drawing the background, the display object must pass its Graphics object to the controller, enabling the controller to arrange for the other game components to draw themselves.

The only other method needed in this class is an explicit value constructor which takes a controller object and stores it into a member variable. The constructor also creates a JFrame object storing it into a member variable. The display object adds itself to the JFrame. The frame initialization should be completed, including setting the initial size of the window which will display the game.

The `Controller` Class

The controller manages all aspects of the game including various game parameters. It determines the diameter of the ball and the distance the ball travels with each move. The distance travelled should be less than the diameter. The speed of ball movement is controlled by a combination of distance travelled and frequency of timer events. Timer events should be set up to occur around every 25 milliseconds. The length of the paddle should be at least twice the diameter of the ball. The game is to be set up so the paddle moves horizontally near the top of the display area. There should be at least 5% of the display height between the paddle and the top of the display area.

The Controller class contains a ball object, a paddle object, and a display object. In addition, the controller is responsible for managing the events that drive the game. Therefore, this class must implement the ActionListener interface to deal with Timer events which drive the ball. It must

also implement the MouseMotionListener interface to cause the paddle to move based on mouse movement.

The constructor of this class creates the three objects mentioned above. It also must create a Timer object which will fire every 25 milliseconds and send ActionEvents to the controller object's actionPerformed method. The last thing the constructor needs to do is to register the controller object as a mouseMotionListener on the display object. So, when the mouse cursor moves across the display area, MouseEvents will be delivered to the mouseDragged or mouseMoved methods of the controller object.

The controller provides a draw method which takes a Graphics object parameter and is called by the display object. This method should check to see if the ball and paddle objects exist, and if they do, ask each object to draw itself, passing the Graphics object into the draw methods of those objects.

The mouseDragged and mouseMoved methods are responsible for managing the padddle position. When a MouseEvent occurs and one of these methods executes, the method uses its MouseEvent parameter to get the current X and Y coordinates of the mouse cursor. These coordinates are given to the update method of the paddle which adjusts the paddle position based on these new coordinates. Once the paddle has been updated, the controller calls the repaint method on the display object to cause the display to reflect the new position of the paddle.

The controller's actionPerformed method is responsible for causing the ball to move and detecting whether the ball is in contact with the paddle. It calls the update method on the ball, passing in the width and height of the display which it gets from the display object.

To detect whether the ball has just contacted the paddle, the following steps should be executed:

1. Get the Y coordinates of the top of the ball and the bottom of the paddle.
2. For contact to have just occurred, the top of the ball must be less than or equal to the bottom of the paddle, and it must also be greater than or equal to the bottom of the paddle minus the amount the ball just moved.
3. If step 2 is true, the horizontal center of the ball must be between the left and right ends of the paddle.
4. If step 3 is true, tell the ball to reverse its vertical direction.

Finally, the actionPerformed method invokes the repaint method to draw the display with the new position of the ball.

Play the Game using a `PaddleBallGame` class

Create a test application that creates a controller object and play the game! Experiment with changing the game parameters and observe how the game reacts. When the game works, take a

screen shot of the game. Turn in ALL your code along with the screen shot. Make sure that your code is fully commented!

Deliverables

- Modified code for the rocket-powered duck and test class
- Answer to the question of whether the green bar now shows in the Viewer
- Screenshots of the running PaddleBall program
- Source Code for all 5 classes
- Paste Screenshots + Source Code into a single DOC/DOCX file

Lab Grading Criteria

```
Design Patterns/JUnit     20 pts.
PaddleBall Program       100 pts.
Screenshots + Source Code 30 pts.
-----------------------------------
Total   150 pts.

*** Program1:
- No Ball class: -30
- Ball class does not have an explicit-value constructor: -6
- Ball class does not have size or position fields: -6/each
- Ball class does not have getTop(), getBottom(), getLeft(), or
getRight() methods: -9/each
- Ball class does not have methods to reverse X or Y directions:
-9/each
- Ball class does not have method to report its center: -9
- Ball class does not have method to change its position: -9

- No Paddle class: -30
- Paddle class does not have an explicit-value constructor: -6
- Paddle class does not have length or position fields: -6/each
- Paddle class does not have getTop(), getBottom(), getLeft(),
or getRight() methods: -9/each
- Paddle class does not have a draw method: -15
- Paddle class draw method does not take a Graphics object: -6
- Paddle class does not have method to change its position: -9
```

- No Display class: -30
- Display class does not extend JPanel: -15
- Display class does not override the paintComponent() method: -6
- Display class does not have an explicit-value constructor that takes Controller object: -6
- Display class's constructor does not create and use a JFrame object: -9

- No Controller class: -30
- Controller class doesn't contain Ball, Paddle, or Display objects: -15pts/each
- Controller class does not implement the ActionListener interface to deal with Timer events: -9
- Controller class does not implement the MouseMotionListener interface to move paddle: -9
- Controller class' Constructor does not create Ball, Paddle, Display, or Timer objects: -6/each
- Controller class does not have a Draw method which takes a Graphics object and passes it to the Ball and Paddle objects: -6
- Controller class does not have mouseDragged and mouseMoved methods: -9/each

- Things don't work in general: -6/instance
- No screenshots: -30