**Workout – 2018-03-13 – Adapter & Façade Patterns Name: Hannah Laws**

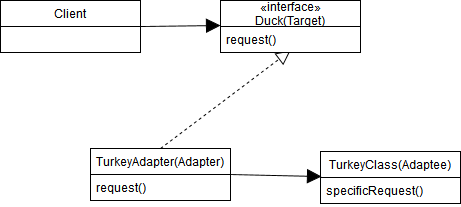
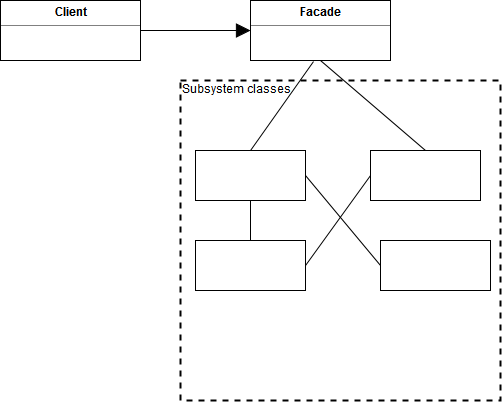
Instructions: Create a directory named workout20180313. When you are finished with this workout, place the following in that directory:

1. A copy of this document with the answers to questions 1-6 filled in.
2. The java files after you are finished with question 7: Client, MyCanvas, MyNewCanvas, MyNewCanvasAdaptor.

When completed, zip up your workout directory and submit it to ASULearn.

1. Below is code from the Turkey Adapter problem from the Book.

|  |  |
| --- | --- |
|  | public class DuckTestDrive {  public static void main(String[] args) {  MallardDuck duck = new MallardDuck();  Duck turkeyAdapter = new TurkeyAdapter(new WildTurkey());  System.out.println("\nThe Duck says...");  testDuck(duck);  System.out.println("\nThe TurkeyAdapter says...");  testDuck(turkeyAdapter);  }  static void testDuck(Duck duck) {  duck.quack();  duck.fly();  }  }  public interface Duck {  public void quack();  public void fly();  }  public class MallardDuck implements Duck {  public void quack() {  System.out.println("Quack");  }  public void fly() {  System.out.println("I'm flying");  }  }  public interface Turkey {  public void gobble();  public void fly();  }  public class WildTurkey implements Turkey {  public void gobble() {  System.out.println("Gobble gobble");  }  public void fly() {  System.out.println("I'm flying a short distance");  }  }  public class TurkeyAdapter implements Duck {  Turkey turkey;  public TurkeyAdapter(Turkey turkey) {  this.turkey = turkey;  }  public void quack() {  turkey.gobble();  }  public void fly() {  for(int i=0; i < 5; i++) {  turkey.fly();  }  }  } |

1. Draw the UML diagram from the code for the Turkey Adaptor. Label the client, the adaptor, the adaptee, and the target.
   1. 
2. Why do you think the textbook authors make the turkey fly 5 times when a duck's fly method is called? What is the important issue here relative to the adaptor pattern?
   1. A turkey flies longer than a duck, so the duck’s flying method has to be called multiple times to simulate that. This brings up the point that an adapter may not be perfect.
3. Here are some questions about the adaptor pattern in general.
4. Write the definition.
   1. The adapter pattern converts the interface of a class into another interface the client expects. Adapter lets clases work together that couldn’t otherwise because of incompatible interfaces.
5. Is there a cost in runtime efficiency for using the Adaptor Pattern? Explain.
   1. The Adapter Pattern make your system more complicated than needed, and since it relies on compilation, this would mean less efficiency at runtime.
6. Can you do implement class adaptors in Java? Why or why not?
   1. No, because Java doesn’t allow for multi-inheritance.
7. Let's compare and contrast Adapters and Decorators.
   1. How are they the same?
      1. They both wrap objects.
   2. How are they different?
      1. Decorator allows for new behavior
      2. Adapter uses libraries and subsets with changing code.
      3. Adapter always converts the interface being wrapped.
8. Façade Pattern
   1. Write the definition.
      1. The Facade Pattern provides a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
   2. Draw the generic Façade Pattern UML.
      1. 
   3. What is being "wrapped" in the Façade Pattern?
      1. method calls to other components.
   4. Explain 1-Click as a classic example of the Façade Pattern…



The 1-Click allows the client to automatically placing the order and buying the product. The client doesn’t need to enter in additional information such as the mailing address, maybe the credit card number, and where to ship the item. Instead, it is all contained into a button so that the client doesn't need to do all the other steps.

1. Does either of the following classes violate the Principle of Least Knowledge? Explain
2. public House {  
    WeatherStation station;  
    // other methods and constructor  
    public float getTemp() {  
    return station.getThermometer().getTemperature();  
    }  
   }
   1. This code does not violate the design principle because getTemp() does not need an object to be passed as a parameter.
3. public House {  
    WeatherStation station;  
    // other methods and constructor  
    public float getTemp() {  
    Thermometer thermometer = station.getThermometer();  
    return getTempHelper(thermometer);  
    }  
    public float getTempHelper(Thermometer thermometer) {  
    return thermometer.getTemperature();  
    }  
   }
   1. This code violates the Principle of Least Knowledge because the getTemp() method needs to get the Thermometer object from station and then call the getTemperature() method.
4. Why might we favor the second class over the first for software maintenance?
   1. We’d be able to switch out the thermometer for testing, rather than messing with the original thermometer class.
5. An Adapter Coding Problem

You have a drawing program with millions of lines of code that for years has been using a vendor-provided MyCanvas class with the following methods to draw shapes:

* clear();
* setLineColor(int rgb);
* setFillColor(int rgb);
* drawSquare(int xPosition, int yPosition, int length);  
   // x and y give the top-left corner
* drawRectangle(int xPosition, int yPosition, int topLength, int sideLength);   
  // x and y give the top-left corner
* drawRightTriangle(int xPosition, int yPosition, int verticalLeg, int horizontalLeg);   
  // verticalLeg and horizontalLeg can be positive or negative to give any orientation to the Right Triangle
* drawTriangle (int[]xPosition, int[] yPosition)  
  // each array has 3 coordinate points for the 3 corners of the Triangle
* drawLine (int xStart, int yStart, int xEnd, int yEnd);

Your company was in the process of developing a new version of the software that used a new vendor-provided class. However, your project has been delayed by one year and the vendor is discontinuing support for the old class. You must quickly make the old code fit MyNewCanvas class.

MyNewCanvas has the following methods:

* clear();
* drawShape(int[] xPosition, int[] yPosition, int sides, int lineColor, int fillColor);   
  /\* The xPosition and yPosition arrays have the x and y coordinates for each  
   “corner” of the shape. They are “parallel arrays”.  
   sides gives the number of sides to the shape (1=a point, 2=a line, 3=triangle, 4=square, etc.  
  \*/

Your job is to develop an adaptor that can be used by the drawing program to produce drawings using the new Canvas class. Sample code is provided on ASULearn that includes your Client, MyCanvas, MyNewCanvas, and a sample NewReleaseClient that will not be ready in time to meet the vendor's deadline.

Here are your specific tasks:

1. Change the name of the "MyCanvas" class to "MyOldCanvas".
2. Create an interface named "MyCanvas" with all the methods from the MyOldCanvas class.
3. Write a MyNewCanvasAdaptor that implements the MyCanvas interface that runs the methods using the MyNewCanvas class.

Hints

* Because of differences in how the Old and New Canvas classes do the work, your adaptor will have to store "state" information about the current line color and current fill color. (The Old Canvas class set the line color and fill color with method calls BEFORE any shape was drawn. The New Canvas class, specifics the colors in the draw method directly.)
* Write the drawRightTriangle method last. It's the hardest.
* Test your MyNewCanvasAdaptor class incrementally.