Labwork 8 Iterator, Observer, Chain of Responsibility Pattern – Object Orientation with Design Patterns 2015

This lab is worth 6% or 60 points from the total 500 points for labwork this semester (includes 10 marks UML exam preps)

UML DIAGRAM DRAWING EXAM PREPS (10 Points)

Draw a UML diagram for **Part 3 of this lab (Lab 8 Part 3 Chain of Responsibility)**. This is an example of the Chain of Responsibility pattern. Include ALL class level details and relationships between participants in the diagram (Note: YOU DO NOT NEED TO HAVE IMPLEMENTED THE SOLUTION FULLY TO DRAW A UML DIAGRAM OF THE SYSTEM)

All pattern participants included\shown	(4 marks)
All relationships shown in correct UML syntax	(4 marks)
Diagram class level details (attributes\behaviours)	(2 marks)

Part 1: Iterator (15 points)

Create a new project called **Lab8Part1**. Create a test program called **TestJavaIterator** that will use the **Iterator** interface in Java. Also create your own polymorphic hierarchy, e.g., Animal, Pig, Sheep with one polymorphic method to output details of each member of the hierarchy using dynamic binding. Set the top class in the hierarchy to **abstract** (e.g. public abstract Animal). Create ten sample objects in the **TestJavaIterator** class and add them to a vector of the hierarchy super-type you have chosen (e.g. Vector<Animal>). Use the Iterator interface methods **hasNext()** and **next()** on the vector to apply dynamic binding to print out details on the objects in the vector (call the polymorphic details display method on each object). NOTE: YOU MUST USE THE ITERATOR INTERFACE IN THIS TASK.

Proposed marking scheme:

- Create your hierarchy of classes (5 points)
- Create vector of objects using the hierarchy base class (super) (2 points)
- Use the Iterator interface methods to print all object details (8 points)

Part 2: Observer **(15 points)**

Create a new project called **Lab8Part2**. Create a subject class called **OilPriceIndicator** that creates a class that extends JFrame and implements the Runnable interface. The **OilPriceIndicator** should simply display an oil price to the screen (using a JLabel) and update the display every second (change the price by a certain amount each second). Create **two interfaces** called **Subject** and **Observer**. The **Subject** interface will contain a **registerInterest(Observer observer)** method for observers to register and interest in the oil price method and the Observer interface will contain a **sendNotify(double oilPrice)** to update observers. Create two observer class called BritishOilPrice and JapanOilPrice both of which will implement the Observer interface and the Runnable interface and extends JFrame [these classes will also display the oil price from a label but must get the oil price sent to them by the Subject using **sendNotify(double oilPrice)**]. Add an observer list to the **OilPriceIndicator** in the form of a vector of observers and send an update to ALL observers on the list every time to the oil price changes to the observers. Finally create a main class called InternationalOilMarket that will do the following:

```
//Create the separate subject and observer GUI's
OilPriceIndicator internationalOilPrice=new OilPriceIndicator();
JapaneseInvestorGUI japaneseOilPrice=new JapaneseInvestorGUI();
BritishInvestorGUI britishOilPrice=new BritishInvestorGUI();

public InternationalOilMarket() {
        //Observers register interest
        internationalOilPrice.registerInterest(japaneseOilPrice);
        internationalOilPrice.registerInterest(britishOilPrice);
        japaneseOilPrice.setLocation(0, 350); //to avoid overlap GUIs
        britishOilPrice.setLocation(0, 700); // to avoid overlap GUIs

        //Start subject and observers, all will update when price changes
        new Thread(japaneseOilPrice).start();
        new Thread(britishOilPrice).start();
        new Thread(internationalOilPrice).start();
}
```

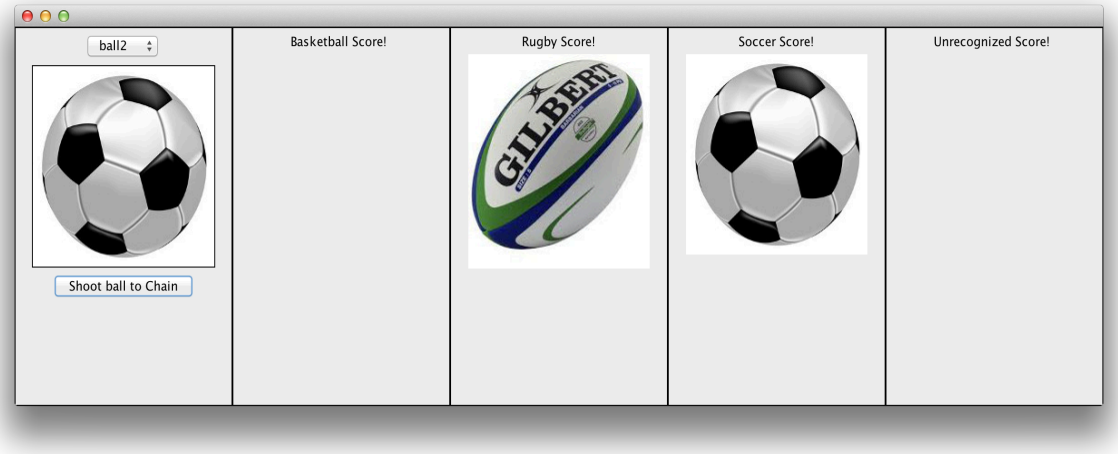
- Create the Subject interface (2 points)
- Create the Observer interface (2 points)
- Code the Subject interface to change the oil price and display (3 points)
- Code the Observers to update based on Subject class (6 points)
- Test main InternationalOilMarket class to run observer pattern (2 points)

Part 3: Chain of Responsibility **(20 points)**

Problem: You want to display a scoreboard that can handle the display of multiple sports within the same GUI

Possible solution: Use the chain of responsibility pattern to pass the image (or image object) along the chain until the proper sport class responds and displays the image in the correct scoreboard. You will add a chain for dealing with unknown sports also.

Create a new project called **Lab8Part3**. Create a **Chain of Responsibility** that will display the sport score in a customized window based on input from the user, e.g., if the user selects a ball that is a rugby ball an image of a rugby ball will be displayed, hitting the a "Send ball to Chain" button will send the image (or Icon object) along the chain to be caught and displayed by the correct scoreboard panel (name the images well and this should be manageable, rugby.jpg). The final interface should look like the interface below (when ball1 was selected and the button was hit the rugby chain member caught and displayed the image, when ball2 was selected and the button was hit the soccer chain member caught the image and displayed it):



Proposed marking scheme:

- Create Chain interface (3 points)
- Create main display GUI (3 points)
- Create Soccer, Basketball, Rugby and Unknown chain members (9 points)
- Build the chain using the addChain and sendChain (3 points)
- Run the chain and test implementation (2 points)