Labwork 6 Adapter Façade Composite 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 **Labwork 3 Part 4** the SeaBattle and LandBattle Abstract Factory game. 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: Adapter (15 points)

Create a new project called **Lab6Part1**. You have been supplied with a Hierarchy of classes of Animals in a Jar file called ZooCode_Missing_Adapter.jar. All of the classes inside this Jar file are part of the same animal hierarchy and are all designed to use a polymorphic feed() method so that the feeding instructions for all animals can be output in the same way using **dynamic binding**. IMPORTANT: The code is missing two classes called LizardAdapter.java and AlligatorAdapter.java which should be **Adapter** classes to call a feed() method for Lizards and Alligators.
You have also been supplied with a second Jar file called LizardAndAlligator_Code_Supplied.jar. This jar file contains a previously written lizard.java and alligator.java class which has the feeding instructions for Lizards and Alligators BUT DOES NOT CONFORM TO THE INTERFACE OF THE ANIMAL HIERARCHY IN THE ZooCode_Missing_Adapter.jar file; instead the lizard and alligator classes use methods called feedLizard() and feedAlligator() [TO CONFORM TO THE ANIMAL INTEFACE THEY SHOULD CALLED **feed()**)

You have DELIBERATELY not been given the source code for lizard and alligator classes, instead only the .class files has been supplied: YOU MUST NOT WRITE A NEW LIZARD AND ALLIGATOR CLASS AND YOU WILL NOT BE SUPPLIED WITH THE LIZARD AND ALLIGATOR CLASS SOURCE CODE (you will need to import this Jar into your zoo project; do this using Project->Properties->Libraries (tab)->Add External Jar and include the LizardAndAlligator_Code_Supplied.jar into your project).

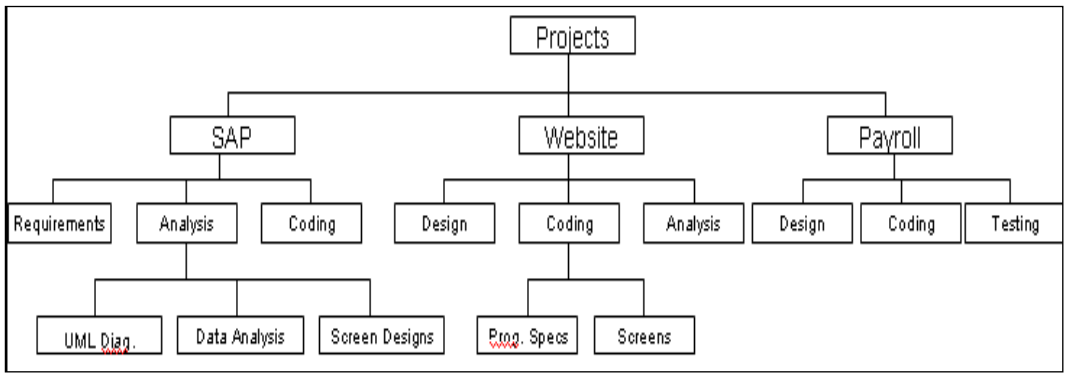
YOUR TASK: Make the lizard AND alligator class that was supplied work with the Animal hierarchy by using the **Adapter design pattern**. This will involve writing

the missing LizardAdapter and AlligatorAdapter classes within the Animal hierarchy and calling the Adaptee's non-complying feedlizard() and feedalligator() methods from a compliant feed() method in the new adapter classes. [Hint: You will know when this is working you will see what type of food the lizard and alligator eats without ever having needed to see the original source code!!!]

- Create the LizardAdapter class (5 points)
- Create the AlligatorAdapter class (5 points)
- Test the program and verify type of food for both animals (5 points)

Part 2: Composite (15 points)

Create a new project called **Lab6Part2**. The IT department in a large consultancy company needs to track the performance of each project and the cost of completing the project. Write a program that displays the project chart below in a JTree (use the example in the lecture as the basis for this: where AbstractEmployee was the abstract Composite now AbstractProject is the abstract base class; where Boss was a node now Project objects are the nodes and where Employee was the leaf the project task (Task: to represent a Root task) are now the leaf objects).
When the user clicks on a project, deliverable or task the cost required to complete that component is displayed.



Proposed marking scheme:

- AbstractProject class for composites (3 points)
- Concrete node class for Project (concrete Composite) (3 points)
- Concrete root (leaf) for Task (or Root\Leaf) (3 points)
- Create sample leaf and node objects (see above, use sample data) (3 points)
- GUI to test the composite structure (3 points)

Part 3: Façade (a basic Math interface for younger Java programmers)

(20 points)

Problem: You have been given the job to teach programming to young novice programmers and cover some basic Mathematics functions. Younger programmers in Java (or those less familiar with Maths) may find some of the functions in the Math module a bit hard to understand, e.g. pow, rint, abs, cbrt, sqrt. Also many of the functions may not be relevant to the younger programmers whose Math skills are not advanced

Possible solution: Use the façade pattern to provide a better interface to the Math class for younger programmers.

Create a new project called **Lab6Part3**. Apply the Facade pattern by placing a new façade called **JuniorMathFacade** in front of the existing Math class available with the Java programming environment (do not re-code the Mathematical functions themselves, simply place a façade in front of the existing Math class calls). The following class methods can be used in the façade:

- absoluteValue(int) – this will wrap abs(int)
- cubeRoot(double) – this will wrap cbrt(double)
- squareRoot(double) – this will wrap sqrt(double)
- xToThePowerOfy(double,double) – this will wrap pow(double, double)
- roundDoubleToInteger(double) – this will wrap rint(double)

Proposed marking scheme:

- Create façade class JuniorMathFacade (2 points)
- Place façade in front of existing Math methods (3 each) (15 points)
- Write a test program to uses <u>all</u> JuniorMathFacade methods (3 points)