Labwork 2 – Object Orientation with Design Patterns 2015

This lab is worth 5% or 50 points from the total 500 points for labwork this semester

Part 1: MVC Pattern (10 points)

Create a class called Hobby, which has the attributes, hobbyName and a constructor for a Hobby object, which accepts a String name as a parameter. Create a class called HobbyList, which holds a Vector or Array of 10 hobbies (these two classes will represent the data (or model) in a System). Add an addHobby (adds a Hobby to vector) and getHobby (returns Vector of hobbies) methods to the HobbyList class. Create two view classes, call one view class the ListView (use a JList GUI to display the hobbies held in the HobbyList) and call the other class the ComboView class (use a JComboBox). In each GUI provide a way to add to the HobbyList and another button to refresh the list display. Keep the listener class(es) separate to the GUI classes (the listener class\classes will represent the controller\controllers).

- Hobby class (Model) (2 points)
- HobbyList with sample data (Model) (2 points)
- JList view class (View) (2 points)
- Combo view class (View) (2 points)
- Event listener class\classes (Controller) (2 points)

Part 2: A Simple Factory Pattern (10 points)

Extend the NumberList example used in the lecture notes to handle alphabetic character lists (A,B,C..) in addition to the integer and double lists it already handles. The sum() method can have an overridden blank sum() method implementation since addition of alphabetic lists is not relevant. Draw a UML diagram of the new NumberList hierarchy, which includes the AlphaCharacterList subtype.

For example:
1 2 3 4 5 6 7 8 …… (as in lecture example with IntegerList class)
1.1 1.2 1.4 0.1 0.99 …. (as in lecture example with DoubleList class)
A, B, C, D…. (new AlphaCharaterList class)

- Created alpha character subclass (2 points)
- Implement new NumberFactory with Alpha lists (3 points)
- Test the new alpha list factory works using test class (3 points)
- UML of new hierarchy (2 points)

Part 3: Another Simple Factory **(10 points)**

Create a set of application programs to store a list of bank account numbers. The account numbers can either be current accounts or investment accounts. An example of the lists is as follows:

C10001, C10002, C10003…
I20001, I20002, I20003…
where 'C' is for current account and 'I' is for investment account.

Use the simple factory pattern to handle both lists and write a test program to display the contents of both lists (hint: use the 'C' and 'I' in factory to return correct subtype).

- Implement abstract Account class (2 points)
- Implement CurrentAccount subclass (2 points)
- Implement InvestmentAccount subclass (2 points)
- Implement the account simple factory (3 points)
- Test new hierarchy and factory (1 points)

Part 4: Simple Factory with GUI **(20 points)**

Problem: Displaying fields to Irish users', which are irrelevant (like ZipCode).
Possible solution: use the simple factory design pattern to return the correct field based on the user's location.

Create a factory called AddressFactory and a GUI called AddressGUI to demonstrate how a simple factory might solve this annoying address problem for Irish users (i.e. being asked to enter irrelevant data in forms). The GUI should provide the user with a dropdown menu for two different types of Address forms to enter data into a database\program. This will involve creating a base class called Address and two subtypes of address called USAddress and IrelandAddress. The USAddress subclass will return a list of fields to display in the GUI which includes specialized input fields for ZipCode and State. The IrelandAddress subclass will not have a ZipCode and will instead return a specialized list of fields that includes the County. Once the user enters data each address should be stored in a Vector of Address types held in the GUI. Finally, provide a view in the GUI to output all of the addresses stored in a Vector sequentially.

- Address superclass with generic attributes\behaviours (3 points)
- USAddress subclass to build address panel\fields (3 points)
- IrelandAddress subclass to build address panel\fields (3 points)
- AddressFactory to return correct subtype (3 points)
- GUI to provide form input based on list returned (3 points)
- GUI view to display all addresses from Vector (2 points)

- Test factory program with at least 5 addresses (3 points)