Information System Design (U08182)

Tutorial Exercise 2 Answer

1. What does the term pattern mean in the context of software development?
   According to one influential definition (Gabriel, 1996), a pattern comprises three elements: a context in which a given problem occurs repeatedly, a set of forces that influence or constrain the possible solutions and a software configuration that allows these forces to be resolved.

2. How do patterns help the software developer?
   A pattern captures and documents proven good practice in some aspect of software development.
   When using a pattern the solution that it embodies is contextualized and applied to the problem in hand

3. Why is the class constructor private in a Singleton pattern?
   The class constructor in the Singleton pattern is private so that it can only be accessed the class-scope instance() method. This ensures that the Singleton class has total control over its own instantiation.

4. What are the advantages of using the Singleton pattern?
   The Singleton pattern ensures that only one instance of a class exists and provides system wide access to that instance.

5. What is the Façade pattern, and when should the pattern be used?
   The Façade is a single class that is used to access a collection of classes. Use the Façade pattern when you want to provide a simple interface to a complex subsystem. Using this pattern helps to simplify much of the interfacing that makes large amounts of coupling complex to use and difficult to understand.

6. What implementation problems may occur when using the State pattern?

   The State pattern may have the following implementation problems.

   - If State objects cannot be shared amongst Context objects (i.e. are not pure state objects) then each Context object will have to have its own State object thus increasing storage requirements.
   - State objects may have to be created and deleted as the Context object changes state increasing the processing requirement.
   - The use of the State pattern introduces one additional message, which also possibly increases the processing requirement.

7. Explain the three types of design patterns.

   Creational – Concerned with the creation of object instances, separating the way in which this is done from the application.

   Structural – Concerned with the structural relationships between the instances, particularly using generalization, aggregation and composition.

   Behavioral – Concerned with the assignment of the responsibility for providing functionality amongst the objects in the system.

8. List two general dangers and two general benefits of the use of patterns.

   Two general dangers of using patterns are the inappropriate application of a pattern and some limitation on creativity. Two advantages of using patterns are the introduction of a reuse culture at the design level and the rich source of development experience they offer.

9. What are the main differences between the MVC architecture and the layered and partitioned architecture?

   The main differences between the MVC and the layered architecture include the update propagation mechanism and the separation of the presentation layer into the View and Controller components in the MVC.

10. Explain how the update propagation mechanism works in the MVC architecture.

    In the MVC when a change to the model data is detected by the model it informs each registered view/controller that some change has been made but does not provide any detail of the change (in the standard use of the architecture at least). It is then up to each view/controller pair to request from the model the latest version of data they require so that they can update themselves.

11. What role does each element of the MVC architecture play?

    Model classes provide the main functionality of the system.

    - View classes provide a display of the attributes of objects to the user.
    - Controller classes respond to interaction from the user.

12. Explain how the Java observer and observable classes are used to implement an MVC architecture.

    The model class needs to extend (i.e. subclass) the Java Observable class. In doing so the model class has access to all the methods of the Observable class.

    Every View class needs to implement the Java Observer interface, by creating an update method that is applicable to the class. All View objects must register with the model class by invoking the addObserver() method of this subclass.

    When the controller class changes values in the model class, the model class must execute the setChanged() and notifyObservers() methods of the Observable class. Each view gets simultaneously updated with the current values represented by the model.