Sample Written Answer Exam Questions Part 2

EBU5304 Software Engineering 2018/19

Question on general aspects of design patterns

What is the general idea of a **design pattern** in software development?

- It is a general pattern for putting code together, given a particular name.
- It must be applicable in a variety of situations, but used to solve a particular sort of problem.
- Design patterns are a way of passing on expertise in programming, they represent solutions experts have found they are often using.
- They provide a vocabulary which helps developers communicate ideas about program structure.

Question on factory objects

Explain the **Factory Object** design pattern, and how its use fits in with the concept of separation

- A factory object is an object on which factory methods can be called.
- A factory method is a method that performs like a constructor, so a call to it creates and returns a new object, but it is a standard method call in syntax .
- A constructor call can only return an object of its actual type, but a factory method could return an object whose actual type is a subtype of its return type.
- The actual type of the object returned by a factory method could be determined by its arguments and other run-time factors.
- This helps establish separation since it means the object returned by the factory method is known to meet the specification of its return type but there is not the closer connection of the calling code being dependent on its exact class.
- A factory method may be used to hide the actual type of the object it returns from from the code which uses it, so it is viewed only in terms of a supertype.

Explain the **Factory Object** design pattern, and how its use fits in with the concept of separation

- With the use of a factory method, there is the additional separation of aspects of the new object being defined by the factory object and not by the calling code.
- So generalised code may be written, with aspects of object construction parameterised, meaning a factory object is passed to the code through a parameter.
- New objects used in the code are provided by calls of factory methods on this factory object.
- The behaviour of the generalised code can be varied by passing different objects to it, so enabling that code to be re-used in different ways
- Some aspects of the objects generated by calls of a factory method on a factory object may come from data stored in the factory object rather than from the object which made the factory method call.
- In some cases a factory method may return a reference to an existing object rather than construct a new object.
- For example in the Singleton design pattern or the Object Pool design pattern.

Question on the Decorator and Adapter design patterns

Give a full explanation of the **Decorator** design pattern, and explain how it differs from the **Adapter** design pattern.

- The Decorator design pattern is used in situations where we want to “decorate” code by giving it some extra behaviour.
- The idea is that the code we wish to decorate is contained in a class which implements a particular interface.
- The Decorator design pattern means having another class which implements the same interface (describing the “outer” or “wrapper” object).
- This class has a field of that interface type (used to refer to the “inner” or “wrapped” object).
- Each method in the interface is implemented in the wrapper class by a method which calls the same method with the same arguments on the inner object
- But it will also perform some additional task, the “decoration”.
- So, the wrapper object may be used in any place where the wrapped object was used so long as that place refers to it through a variable of the interface type.
- The code will then perform as if the wrapped object was used directly, as exactly the same method calls are made with the principle of dynamic binding, and also do the extra work.

Give a full explanation of the **Decorator** design pattern, and explain how it differs from the **Adapter** design pattern.

- The Adapter design pattern also has an inner and outer object and an associated interface.
- But only the wrapper class implements the interface.
- The inner object is of a class that does not implement the interface.
- The methods of the interface are implemented by code which calls what method or combination of methods in the inner object is judged to be the equivalent of the method of the interface in terms of its general specification.
- The Adapter design pattern is used when we have a class of objects which has already been developed and we want to be able to adapt objects of that class so they can be passed to code and used as if they were of the interface type.

Question on the State design pattern

Explain what is meant by the **State** design pattern. State under what circumstances it is used. Give the benefit that using it provides to a software system.

- The State design pattern involves an object that has within it a reference to another object that implements the same interface.
- A method call on the outer object will be implemented by the same method call being made on the inner object.
- With the State design pattern, under some circumstances when a call to one of the methods in the interface is made, the inner object will be replaced by another object of a different class that implements the same interface.
- This is used in cases where an object starts off best implemented in one form, but changes made to it by methods called on it may make it better implemented in another form.
- So the basic idea of the State design pattern is to enable there to be a single object whose internal representation can change, with the change managed inside the class of the object.

Explain what is meant by the **State** design pattern. State under what circumstances it is used. Give the benefit that using it provides to a software system.

- An example is an object representing a set of integers, which could be implemented by an array of booleans or an array of integers.
- The representation by an array of booleans is suitable when the range of integers in the set is small, but unsuitable if the range of integers is large.
- So the State design pattern could be used where an object representing a set of integers has inside it a reference to an object representing a set of integers
- Initially the inside object would be one using an array of booleans as its implementation, but if numbers outside a particular range were added to the set, it would replace it by an object using an array of integers to represent the set.
- The new inside object would be given the elements of the old inside object.
- Code making use of the outside object could just use it as a set of integers, without having any extra code to change it from one representation to the other when that is a good thing to do.
- In this way it meets the Dependency Inversion Principle, as it means the code using the set of integers does not have any dependency on the different versions of code that could implement it.