Information Systems Analysis & Design (M8748)

Tutorial 8 Answer

1. What are the advantages of components?

   Use of a component saves time and work. A friend of one of the authors once said ‘have you ever wondered how much it would cost to make your own light bulb?’

2. Explain why the NIH syndrome occurs.

   It is reasonable not to trust components that we do not know sufficiently well. Some programmers feel that the only way that you can know a program sufficiently well is by writing it yourself. This is not always so reasonable. Another reason is that many people who enjoy technical challenge will gladly spend time solving a problem rather than use a ready-made solution. This may be satisfying for the developer, but can also be very expensive for the clients who must pay for the work.

3. What does object-orientation offer that helps to create reusable components?

   Objects are well encapsulated, and object structures can be designed this way too. The hierarchic nature of generalization abstracts out the more general features of a class. Hierarchic organization of models helps the developer to find components easily when they are needed. Composition encapsulates whole structures within a composite object.

4. Distinguish composition from aggregation.

   A component of a composition cannot be shared with another composition. The component has a coincident lifetime with the composition (although a component can be explicitly detached before the composition is destroyed). Neither of these is true for aggregation.

5. Why are operations sometimes redefined in a subclass?

   This is a basis for polymorphism. The superclass operation defines the operation signature, but each subclass has a different method that implements the behavior (see Chapter 10).

6. What is an abstract class?

   An abstract class has no instances and can exist only as a superclass in a hierarchy. It provides a generalized basis for concrete subclasses that do have instances.

7. Why is encapsulation important to creating reusable components?

In building a software system (or anything else) from ready-made components, it is important that the interfaces between components should be completely described, reasonably simple and preferably standardized. If this applies to a particular component, there is no need to know any of the details of its internal construction in order to understand how to use it. On the other hand, if it is necessary to know about its internal construction, then the interface is probably neither simple nor standardized (it will be unique to that component) and the component will be tightly coupled to any other components with which it interacts. This greatly increases the difficulty of making any change to the system as a whole, whether this is intended to bring an improvement or to repair a fault.

8. Why is generalization important to creating reusable components?
   Generalized components have the capacity to be reused in many different situations. As a component becomes more specialized, it can be used in fewer and fewer different situations.

9. When should you not use generalization in a model?
   Generalization should not be used when there is any characteristic of the proposed superclass that is not fully consistent with all its subclasses. Generalization should not be used when there is not a reasonable degree of overlap between the definitions of the superclass and its subclasses.

10. 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.

11. 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.

12. What is an antipattern?
    An antipattern captures practice that is demonstrably bad (by documenting an attempted solution that failed), and can also provide a reworked solution that can be applied within a specific context.