**Chapter 9 Class and Method Design  
Test Bank**

**Multiple choices**

1. Detailed design is important for two reasons. First, preexisting classes and components need to be understood, organized, and pieced together, and second, \_\_\_\_\_.

a. it is common for the project team to write some code and produce original classes that support the application logic of the system

b. it is tempting to jump in without planning

c. modular design is outdated

d. the SDLC continues to be used instead of a more reliable method of analysis for systems

e. top-down modular approaches are an acceptable method of analysis that must be performed during the planning phase

Ans: a

1. In an object-oriented system, changes can take place at the following level of abstraction.

a. variable

b. method

c. class/object

d. cluster (examples: partition, package)

e. all of these

Ans: e

1. \_\_\_\_\_ has emerged as the standard for the design of object-oriented systems.

a. Java

b. C++

c. VisualBasic

d. UML

e. Microsoft .NET

Ans: d

1. In terms of levels of abstraction, which of the following is at the lowest level relative to the rest?

a. system

b. partition/package

c. library

d. class/object

e. method

Ans: e

1. \_\_\_\_\_ means having the ability to send the same message to different objects, which can be interpreted differently by different objects.

a. encapsulation

b. polymorphism

c. inheritance

d. coupling

e. cohesion

Ans: b

1. \_\_\_\_\_ suggests that only the information required to use an object should be available outside the object.

a. Encapsulation

b. Information hiding

c. Polymorphism

d. Inheritance

e. Cohesion

Ans: b

1. \_\_\_\_\_ refers to the level of interdependency or interrelationship among the modules in a system.

a. coupling

b. cohesion

c. connascence

d. multiple inheritance

e. single inheritance

Ans: a

1. A class/object should only represent one thing, and a method should only solve a single task. This principle is often referred to as \_\_\_\_\_.

a. coupling

b. cohesion

c. connascence

d. multiple inheritance

e. none of these

Ans: b

1. What are the two types of coupling in object-oriented systems?

a. interaction, data

b. data, inheritance

c. data, stamp

d. interaction, inheritance

e. data, common

Ans: d

1. Which of the following types of interaction coupling is “best?”

a. stamp

b. data

c. content or pathological

d. common or global

e. control

Ans: b

1. Object-oriented systems have three general types of cohesion: \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_.

a. method, class, inheritance

b. method, generalization/specialization, inheritance

c. generalization/specialization, class, object

d. method, class, generalization/specialization

e. functional, sequential, procedural

Ans: d

1. *Fan-out* refers to \_\_\_\_\_.

a. the number of attributes a class has

b. the number of messages sent by a method

c. the number of times a message is sent by a method

d. the number of times an attribute is passed from a class

e. none of these

Ans: b

1. Which of the following Structured English statements is an advanced form of an IF statement?

a. Action statement

b. For statement

c. While statement

d. Case statement

e. Do statement

Ans: d

1. Constraints can be written in either a natural language, a semiformal language, or a formal language such as \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

a. UML

b. Structured English

c. Use case diagrams

d. English

e. UML’s Object Constraints Language

Ans: e

1. There are three different types of constraints typically captured in object-oriented design: preconditions, postconditions, and \_\_\_\_\_\_\_\_\_\_\_\_\_.

a. regular conditions

b. guard conditions

c. exception handling

d. invariants

e. contracts

Ans: d

1. Essentially, all OCL expressions are simply a declarative statement that evaluates to either being true or false. If the expression evaluates to true, then the constraint has been \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

a. satisfied

b. executed

c. always true

d. served as an invariant

e. skipped

Ans: a

1. Like other problem domain models, the \_\_\_\_\_\_\_\_\_\_\_\_\_, contracts, and method specifications in class and method design need to be verified and validated.

a. constraints

b. classes

c. behaviors

d. associations

e. polymorphisms

Ans: a

1. The constraints and contracts in class and method design models were derived from the \_\_\_\_\_\_\_\_\_\_\_\_ requirements and the problem domain representations.

a. business

b. regularly

c. non-functional

d. functional

e. reliability

Ans: d

**True/False**

1. Many project teams are too quick to jump into writing code for classes without first designing them.

Ans: True

1. Polymorphism is the mechanism that combines the processes and data into a single object.

Ans: False

1. Subclasses inherit the appropriate attributes and methods from the superclass above them.

Ans: True

1. Because of potential problems, developers must be aware of the effects of modifications in a superclass and in each of the subclasses that will inherit the modification.

Ans: True

1. It is impossible for a subclass to inherit from more than one superclass.

Ans: False

1. C++ forces programmers to write code with good levels of interaction coupling.

Ans: False

1. High levels of inheritance coupling in a system are always good and desirable.

Ans: False

1. You should maximize connascence within an encapsulation boundary and minimize connascence between the encapsulation boundaries.

Ans: True

1. Fan-out refers to the number of attributes passed by one object to another.

Ans: False

1. A derived attribute results from storing the value of a computation.

Ans: True

1. Contracts document the message passing that takes place between objects.

Ans: True

1. Method specifications can be somewhat vague, as the programmers will have a good idea of what the analyst needs done in the system.

Ans: False

1. Clicking a mouse can be an event.

Ans: True

1. A Case statement is an advanced form of an If statement.

Ans: True

1. A For statement simply performs some action.

Ans: False

1. When a calling method passes a variable to the called method, the two exhibit content or pathological coupling.

Ans: False

1. The *signature* of a method is completely specified by the name of the method and the parameters that must be passed to the method.

Ans: False

1. A self-contained, encapsulated piece of software that can be plugged into a system is often referred to as a design pattern.

Ans: False

1. Method signatures, in conjunction with frameworks, class libraries and design patterns, offer excellent opportunities for reuse.

Ans: False

1. Using an attribute to store the value of a computation is the use of a derived attribute.

Ans: True

1. Moving the attributes to a calling class when it is the only class that needs those attributes is not a method of optimizing the design of a system?

Ans: False

1. A post-condition is a constraint that must be met for a method to execute.

Ans: False

1. A pre-condition is a constraint that must be met after the method executes.

Ans: False

1. Invariants model constraints that must always be true for all instances of a class.

Ans: True

1. Contracts contain a detailed algorithmic description of how the method is to work.

Ans: False

1. A listing of the items that trigger the functionality in the program is known as events.

Ans: True

1. Written documents that include explicit instructions on how to write the code to implement the method are called method specifications.

Ans: True

1. A Case statement in Structured English specifies some action to be performed.

Ans: False

1. An *If* statement in Structured English controls actions that are performed under different conditions.

Ans: True

1. A Case statement in Structured English has several mutually exclusive branches.

Ans: True

1. Constraints can only be written in either a semiformal language or a formal language.

Ans: False

1. There are only two different types of constraints typically captured in object-oriented design: preconditions and postconditions.

Ans: False

1. Invariants are a type of constraints captured in object-oriented design

Ans: True

1. Essentially, all OCL expressions are simply a declarative statement that evaluates to either being true or false.

Ans: True

1. Since a manager has a higher position than an employee, it is a good idea to design class Manager as a superclass, and the class Employee inherits class Manager.

Ans: False

1. One of inheritance abuses in Object-oriented design is the inheritance was used to implement an association relationship.

Ans: True

1. In order to verifying and validating class and method design, all constraints, contracts, and method specifications must be tested.

Ans: True

1. UML’s Object Constraint Language is a formal language.

Ans: True

**Short Answer**

1. What is *coupling*? Is this desirable or undesirable in a system? Why?

Ans: Coupling refers to the level of interdependency or interrelationship among the modules (classes, objects, and methods) in a system. The higher the interdependency is, the more likely changes in part of a design can cause changes to be required in other parts of the design. Thus, you want to minimize coupling in a system, if possible. There are two types of coupling in an object-oriented system: inheritance coupling and interaction coupling.

1. What is interaction coupling? How can interaction coupling be minimized?

Ans: The two types of coupling in OO systems are interaction coupling and inheritance coupling. Interaction coupling deals with the coupling among methods and objects through message passing. To minimize this form of coupling, Lieberherr and Holland proposed the Law of Demeter. This law minimizes the number of objects that can receive messages from a given object. Interaction coupling is difficult to totally eliminate, but the best coupling is “no direct coupling” where the methods do not relate to one another at all. The worst form of this coupling is “content or pathological” coupling where a method of one object refers to the inside (hidden) parts of another object; this type of coupling violates the principle of encapsulation and information hiding.

1. What is inheritance coupling? How can inheritance coupling be minimized?

Ans: Inheritance coupling deals with how tightly coupled the classes are in an inheritance hierarchy. The issues of inheritance conflicts, redefinition capabilities, and dynamic binding all make a high level of inheritance coupling undesirable. To avoid these problems, the guidelines presented in the text state that you should ensure that inheritance is used only to support generalization/specialization (a-kind-of) semantics, and avoid all other uses.

1. Describe the concept of inheritance. How does this impact the development of object-oriented systems?

Ans: Inheritance is a form of reuse. Classes can reuse attributes and methods that were defined in other classes. Thus, a class can be defined, and the attributes and methods can automatically flow to a subclass (e.g., person may be the class, and employee and student could both be subclasses). The subclass would then have the same attributes and methods as the superclass, and may have additional attributes and/or methods that are unique to the subclass. There are two types of inheritance: single inheritance and multiple inheritance. Single inheritance allows a subclass to have only one parent class. With multiple inheritance, a subclass may inherit from more than one superclass. Single inheritance causes inheritance conflicts, as per which a subclass may have the same name as an attribute/method of a superclass. Multiple inheritance may cause even more inheritance conflicts, which arise from conflicts between two or more superclasses that a subclass inherits.

1. What is *connascence*? Is this desirable or undesirable in a system? Why?

Ans: Connascence generalizes the ideas of cohesion and coupling, and combines them with the arguments for encapsulation. This means that two modules (classes or methods) are so intertwined that if you make a change in one, it is likely that a change will be required in the other. This is very similar to coupling and as such should be minimized. However, when you combine coupling with the encapsulation levels, it’s not quite that simple. In this case, you want to 1) minimize overall connascence by eliminating any unnecessary connascence throughout the system, 2) minimize connascence across any encapsulation boundaries, such as method boundaries and class boundaries, and 3) maximize connascence within any encapsulation boundary.

1. What are the four opportunities for reuse identified in the text?

Ans: The four opportunities for reuse are patterns, frameworks, class libraries, and components. A pattern was defined in Chapter 7 as “a useful group of collaborating classes that provide a solution to a commonly occurring problem.” A framework is a set of implemented classes that can be used as a basis for implementing an application. For example, frameworks are available for CORBA and DCOM that enable you to base the implementation of part of the system architecture layer. A class library gives you a set of implemented classes that were designed for reuse. A component is a self-contained, encapsulated piece of software that can be plugged into a system to provide a specific set of required functionalities.

1. What are the five optimizations that should be done to create a more efficient design?

Ans: First, review the access paths between objects. If the path is long and the message is sent frequently, a redundant path should be considered.

Second, each attribute for each class should be reviewed. Which method uses the attributes and which objects use the methods should be determined. If the only methods that use an attribute are read and update methods and only instances of a single class send messages to read and update the attribute, the attribute may belong with the calling class instead of the called class. Moving the attribute will speed up the system.

Third, review the direct and indirect fan-out of each method. If the fan-out of a method is high relative to that of other methods in the system, the method should be optimized.

Fourth, look at the execution order of the statements in often-used methods. Sometimes rearranging some of the statements can make it more efficient.

Finally, avoid re-computation by creating derived attributes.

1. What are the three types of constraints typically captured in object-oriented design?

Ans: The three are pre-conditions, post-conditions, and invariants.

Pre-conditions are constraints that must be met for a method to execute. A post-condition is a constraint that must be met after the method executes or the effect of the method execution must be undone. Invariants model constraints that must always be true for all instances of a class.

1. Use Structured English to write the logic for the process “register for a class” at your college or university. Focus on a single class, but consider the possible exceptions that may occur when you try to register for that class.

Ans: The answers will vary for this problem depending on the particular registration process (and possible exceptions) that exists at your institution. However, they certainly should consider what would happen if the class is closed, if they don’t have the necessary pre-requisite for the course, if there is a hold on their registration, etc.

1. Why is it necessary for an analyst to detail the specifications for the individual classes and methods? What techniques are useful for doing this documentation?

Ans: The analyst must provide explicit instructions on how to write the code to implement each method so that the programmers who write the code will know what to do. The specifications need to be very clear and easy to understand or the programmers will have to try to decipher vague or incomplete instructions. A method specification form can be used to document these specifications. The analyst will have to determine the triggers or events that start the particular method, the arguments received and returned, messages sent, and the specifications for the algorithm(s) used in the method.

1. What is the Object Constraint Language? Give an OCL example.

Ans: The Object Constraint Language (OCL) is a complete language designed to specify constraints. Essentially, all OCL expressions are simply a declarative statement that evaluates to either being true or false. If the expression evaluates to true, then the constraint has been satisfied. For example, if a customer had to have a less than a one hundred dollar balance owed to be allowed to place another credit order, the OCL expression would be: *balance owed <=100.00.*

1. What are the different ways to optimize an object system?

Ans:

1. Review the access paths between objects.
2. Review the attributes of each class.
3. Review the direct and indirect fan-out of each method.
4. Look at execution order of the statements in often-used methods.
5. Avoid re-computation by creating derived attributes.
6. Briefly discuss the procedure for verifying and validating class and method design.

Ans: First, perform a walkthrough of all of the evolved problem domain representations. That is, all functional models must be consistent, all structural models must be consistent, all behavioral models must be consistent, and the functional, structural, and behavioral models must be balanced. Second, all constraints, contracts, and method specifications must be tested. The best way to do this is to role-play the system using the different scenarios of the use cases. In this case, we must enforce the invariants on the evolved CRC cards, the pre- and post-conditions on the contract forms, and the design of each method specified with the method specification forms and algorithm specifications.