**Chapter 19: Object-Oriented Programming: Inheritance**

**Section 19.1 Introduction**

19.1 Q1: Select the *false* statement regarding inheritance.

1. A derived class can contain *more* attributes and behaviors than its base class.
2. A derived class can be the base class for other derived classes.
3. Some derived classes can have multiple base classes.
4. Base classes are usually more specific than derived classes.

**ANS d. Base classes are usually more specific than derived classes.**

19.1 Q2: Which of the following is *not* a kind of inheritance in C++?

1. public.
2. private.
3. static.
4. protected.

**ANS c.** static**.**

19.1 Q3: The *is-a* relationship represents.

1. Composition.
2. Inheritance.
3. Information Hiding.
4. A friend.

**ANS b. Inheritance.**

**Section 19.2 Base Classes and Derived Classes**

19.2 Q1: Which of the following is most likely a base class of the other three?

1. automobile.
2. convertible.
3. miniVan.
4. sedan.

**ANS a.** automobile**.**

19.2 Q2: Which of the following is *not* a good example of a hierarchy likely to be modeled by inheritance?

1. Airplanes.
2. Geometric shapes.
3. Animals.
4. Prime numbers.

**ANS d. Prime numbers.**

19.2 Q3: To declare class subClass a privately derived class of superClass one would write:

1. class subclass : private superClass
2. class subclass :: private superClass
3. class subclass < private superClass >
4. class subclass inherits private superClass

**ANS a.** class subclass : private superClass

**Section 19.3 Relationship between Base Classes and Derived Classes**

19.3 Q1: Assuming the definition,

class BasePlusCommissionEmployee : public CommissionEmployee

which of the following is *false*?

1. The colon ( : ) in the header of the class definition indicates inheritance.
2. The keyword public indicates the type of inheritance.
3. All the public and protected members of class BasePlusCommissionEmployee are inherited as public and protected members, respectively, into class CommissionEmployee.
4. CommissionEmployee is the base class and BasePlusCommissionEmployee is the derived class.

**ANS: c. All the** public **and** protected **members of class** BasePlusCommissionEmployee **are inherited as** public **and** protected **members, respectively, into class** CommissionEmployee**.**

19.3 Q2: Assuming the following is the beginning of the constructor definition for class BasePlus-CommissionEmployee which inherits from class Point,

BasePlusCommissionEmployee::BasePlusCommissionEmployee( string first,   
 string last, string ssn, double sales, double rate, double salary )  
 : CommissionEmployee( first, last, ssn, sales, rate )

The second line:

1. Invokes the CommissionEmployee constructor with arguments.
2. Causes a compiler error.
3. Is unnecessary because the CommissionEmployee constructor is called automatically.
4. Indicates inheritance.

**ANS a. Invokes the** CommissionEmployee **constructor with arguments.**

19.3 Q3: Which of the following is *not* one of the *disadvantages* of using the “copy-and-paste” approach to duplicating code from one class into another class?

1. Errors are prone to be spread around.
2. It is time consuming.
3. It forces the system to store many physical copies of the code, creating a code-maintenance nightmare.
4. All of the above are disadvantages of the “copy-and-paste” approach.

**ANS d. All of the above are disadvantages of the “copy-and-paste” approach.**

19.3 Q4: protected base class members *cannot* be accessed by:

1. Functions that are neither friends of the base class, derived-class member functions nor friends of a derived class.
2. friends of the base class.
3. Functions that are not derived-class member functions.
4. friends of derived classes.

**ANS a. Functions that are neither** friend**s of the base class, derived-class member functions nor** friend**s of a derived class.**

19.3 Q5: When should base class members be declared protected?

1. When all clients should be able to access these members.
2. When these members are used only by member functions of this base class.
3. When these members should be available only to derived classes (and friends), but not to other clients.
4. The protected access specified should never be used.

**ANS c. When these members should be available only to derived classes (and** friend**s), not to other clients.**

19.3 Q6: From most restrictive to least restrictive, the access modifiers are:

1. protected, private, public
2. private, protected, public
3. private, public, protected
4. protected, public, private

**ANS b.** private**,** protected**,** public

**Section 19.4 Constructors and Destructors in Derived Classes**

19.4 Q1: When an object of a derived class is instantiated, the \_\_\_\_\_\_\_\_\_\_ constructor initializes the \_\_\_\_\_\_\_\_\_ members.

1. Base class, base class.
2. Derived class, base class.
3. Base class, derived class.
4. Derived class, public.

**ANS: a. Base class, base class.**

19.4 Q2: Base class constructors and assignment operators:

1. Are not inherited by derived classes.
2. Should not be called by derived class constructors and assignment operators.
3. Can be inherited by derived classes, but generally are not.
4. Can call derived-class constructors and assignment operators.

**ANS: a. Are not inherited by derived classes.**

19.4 Q3[C++11]: Which of the following statements about inheriting base class constructors is *false*?

1. To inherit a base class’s constructors, you write the following line of code in the derived class definition (*BaseClass* is the base class’s name):

using BaseClass::BaseClass;

1. If an inherited base-class constructor has default arguments, the line of code in Part (a) causes the compiler to generate a derived-class constructor with the same default arguments.
2. By default, each inherited constructor has the same access level (public, protected or private) as its corresponding base-class constructor.
3. If the derived class does not explicitly define constructors, the compiler generates a default constructor in the derived class—even if it inherits other constructors from its base class.

**ANS: b. If an inherited base-class constructor has default arguments, the line of code in Part (a) causes the compiler to generate a derived-class constructor with the same default arguments. (A base-class constructor’s default arguments are *not* inherited. Instead, the compiler generates overloaded constructors without default arguments in the derived class.)**

**Section 19.5** public**,** protected **and** private **Inheritance**

19.5 Q1: Which forms of inheritance are *is-a* relationships?

1. All forms of inheritance are *is-a* relationships.
2. Only public and private.
3. Only public and protected.
4. Only public.

**ANS: d. Only** public**.**

19.5 Q2: When deriving a class from a protected base class, the public members of the base class become \_\_\_\_\_\_\_\_\_ and the protected members of the base class become \_\_\_\_\_\_\_\_\_\_?

1. protected, private
2. public, private
3. protected, protected
4. public, protected

**ANS: c.** protected**,** protected

**Section 19.6 Software Engineering with Inheritance**

19.6 Q1: Theoretically, clients do not need to see the \_\_\_\_\_\_\_\_\_ of classes from which they derive other classes.

1. Header files.
2. Source code.
3. Object code.
4. Interface.

**ANS: b. Source code**.

19.6 Q2: Which of the following *is true* about using inheritance in software engineering?

1. Common attributes and behaviors should be factored out of closely related classes and placed into a base class from which the original classes can now inherit.
2. It is best to create a huge class library to make it easy for a client to find the most appropriate class for his or her needs.
3. A class produced through inheritance should be as large as possible to fully encompass all of the functionality it should offer.
4. The standard C++ libraries that are shipped with C++ compilers are usually enough to accomplish anything an application might need to do.

**ANS: a. Common attributes and behaviors should be factored out of closely related classes and placed into a base class from which those original classes can inherit**.