# Constructors, Overriding, and Overloading – Answers

Question 1

d) Declare all constructors using the private access modifier.

If no constructors are declared explicitly; then the compiler will create one implicitly, and it will have the same access modifier as the class. The explicit declaration of any constructor will prevent the creation of a default constructor. If all constructors are declared *private*, then code outside of the class will not have access to the constructors and will not have the ability to create an instance of the class. Constructors do not return a value and constructor declarations do not include a return type, so the keyword *void* is not applicable to a constructor declaration.

Question 2

1. **The compiler will create a default constructor if no other constructor is declared.**
2. **The default constructor takes no arguments.**
3. **If a class A has a direct superclass, then the default constructor of class A invokes the no-argument constructor of the superclass.**

If no constructor is declared explicitly, then the compiler will implicitly insert a default constructor. The default constructor takes no arguments. The primordial class Object has no superclass; so the default constructor of type Object does not invoke a superclass constructor. If a class A has a direct superclass, then the default constructor of class A will invoke the no-argument superclass constructor. It is unlikely that the real exam would try to trick you with a question that requires you to know that the constructor of type Object does not invoke a superclass constructor. For the purposes of the real exam, it might be safer to overlook that particular unique feature of type Object. If a subclass constructor attempts to invoke the no-argument superclass constructor when none exists, then a compile-time error is generated. The access modifier implicitly assigned to the default constructor is the same as that assigned to the class. The default constructor does not have a *throws* clause. Consequently, a compile-time error is generated if the no-argument constructor of the superclass has a *throws* clause.

Question 3

1. static
2. volatile

*Serializable*, *Runnable*, *Externalizable*, and *Cloneable* are all interfaces. *Thread.run* is a method. The keywords *static* and *volatile* are field modifiers.

Question 4

1. Compiler error at 3.
2. Compiler error at 4.

Class A and C are not declared in the same package; therefore, class C does not have access to package access method, m4. Since class C extends class A, class C does have access to the protected method, m2, of class A.

Question 5

1. 1

A compile-time error occurs at the line marked 1, because the array variable declaration cannot be used to specify the number of components contained in the array. Instead, the dimension expression should be contained in an array creation expression such as the following, new int[3].

Question 6

1. Compile-time error

The array initializer, ((1,2),(3,4,5),(6,7,8,9)), generates a compile-time error, because the curly braces have been replaced by parentheses. The array initializer should have been specified as follows: {{1,2},{3,4,5},{6,7,8,9}}.

Question 7

e) Prints: 2,3,5

Each of the three array variable declarations, a1, a2 and a3, is different in terms of the position of the square brackets, but each declares a variable of type int[][]. Each of the three declarations contains an array initializer. In each case, the initializer could be simplified as follows: {{1,2},{3,4,5}}. The initializer creates an array containing two components, and each is a reference to an array containing components of type int. The size of each of the subarrays is different: The size of the first is 2 and the second is 3. The array access expression, a1[0][1] means a[1st subarray][2nd component], which is the value 2.

Question 8

**b) The compiler attempts to create a default constructor for class B.**

**d) Compile-time error at 2.**

If no constructor is declared explicitly, then the compiler will implicitly create a default constructor that accepts no parameters, has no *throws* clause, and invokes its superclass constructor. Since class A has an explicitly declared constructor, the compiler will not create an implicit default constructor for class A. Class B does not have an explicit constructor declaration, so the compiler attempts to create a default constructor for class B. Since class A does not have a no-parameter constructor, the attempt by class B to invoke the no-parameter constructor of A would fail. As a result, a compiler error is generated at marker 2.

Question 9

a) private

Constructors are not inherited and cannot be overridden, so there is no need for the *final* modifier in a constructor declaration. Furthermore, an *abstract* constructor would be useless, since it could never be implemented. The *static* modifier can be applied to fields and methods, but not to constructors.

Question 10

b) Prints: DDDD

The instance method that is invoked depends on the run-time type of the object – not on the compile-time type of the reference. In each case, the method m1 is invoked on an object of type D; so the implementation of m1 in type D is selected each time.

Question 11

**b) The relationship between a class and its superclass is an example of an "is-a" relationship.**

**c) The relationship between a class and an object referenced by a field within the class is an example of a "has-a" relationship.**

Inheritance is an example of an "is-a" relationship, because the subclass "is-a" specialized type of the superclass. The relationship between a class and an object referenced by a field declared within the class is an example of a "has-a" relationship, because

Question 12

a d

If an alternate constructor invocation appears in the body of the constructor, then it must be the first statement. The same is true for a superclass constructor invocation. A compile-time error is generated if a constructor attempts to invoke itself either directly or indirectly.

Question 13

e) None of the above

Line 4 does not generate a compile-time error. The reference named base actually refers to an instance of type Sub, so the reference may be cast to type Sub.

Question 14

b) Prints: ABC

The method invocation expression d1.m1(a1) uses reference d1 of type D to invoke method m1. Since the reference d1 is of type D, the class D is searched for an applicable implementation of m1. The methods inherited from the superclasses, C, B and A, are included in the search. The argument, a1, is a variable declared with the type A; so method A.m1(A a) is invoked.