Chapter 5 – Inheritance

Answers

1. A

   On line 11, when the value of the variable “count” is first printed out, it still has its initial value of 0. Then when the sub class Puppy is created on line 12, the constructor of the Puppy class is called. Since there is no call to a “super” constructor, the compiler automatically calls the no-argument constructor of the super class Dog, which increases the value for the variable “count” by 1, to make it have the value of 1. Then the Puppy constructor is called, which increases the value of the “count” variable by 1 again, making its value to be 2.

2. A

   A regular class (as opposed to an inner class) can never have the access modifier “private”, otherwise a compilation error will occur. On line 12 an instance of a class is created with the modifier “final”, which is allowed. This means that the object that this object variable refers to cannot be changed.

3. B

   It does not make sense to allow a final class to have an abstract method, since all abstract methods need to be overridden when a class is sub classed, and final classes can never be sub classed. An abstract method cannot be static though.

4. B

   If a method in a super class is private then its sub classes do not inherit or see this method, so answer B is OK.

   If the method “zzz” in the super class had the “public” access modifier placed before it, then the sub class would inherit this method. Therefore if the sub class wants to override the method, the access modifier there would need to be “public” as well, since when overriding a method the access modifier of the new method cannot be more restrictive than the access modifier of the old method, and public is the least restrictive access modifier.

5. A

   In this example class “Base” is the parent class and class “Derive1” is a sub class of class Base. Class Base defines a method called “method”, and class Derive1 properly overrides this method.

   In the “main” method, an instance of each the Base class and the Derive1 class are created, and the method “method” is called for each of them. When “method” is called on the Base object, that method in class Base is called, which prints the word “Tel-Aviv”. And when “method” is called on the Derive1 object, that method in class Derive1 is called, which prints the word “Haifa”.

6. A,C

   In the constructor “public Derive(String str) “ there is no call to “this” or to “super”, so the compiler will automatically call the constructor in the super class that receives no arguments. This constructor would look like answer A “Base()”, and needs to be explicitly written in the base class.

   In the constructor “public Derive(int j, int k) “ there is a call to “super(j, k)”, so this requires that the super class contain a constructor like this.

7. A,B

   We can assume here that this example is talking about 2 classes, one called “Car”, and a sub class of this called “PrivateCar”. Since the class “PrivateCar” is a sub class of “Car”, so it automatically inherits the fields mentioned in C, D and E. The fields in A and B are items that only “PrivateCar” contains, and not “Car”, so these should be included as fields in the class “PrivateCar”.

8. A, D, E

   The answer A is acceptable because it properly overloads the method in the super class. It does not throw the IOException like the super class method does, but this is OK. The overriding method can just not throw a checked exception that the overriden method does not throw, but it does not need to throw the exception at all.

   B is incorrect since it is trying to override the method, but the access modifier (package friendly) is more restrictive than the access modifier "public" of the overridden method, so this is not valid.

   C is incorrect since it is throwing a checked exception that the overridden method does not throw.

   D and E are correct, since they are just defining new methods and are not trying to override at all.

9. D,E

   D and E are correct since they both define methods properly.

   A is incorrect since a method cannot be both final and abstract. An abstract method is meant to be overridden, and a final method can never be overridden.

   B is incorrect since the abstract modifier cannot be used on variables. And C is incorrect since "friendly" is not a modifier or keyword in the Java language.

10. B

    An abstract method cannot be final since the idea of an abstract method is that sub classes should override it, and final methods cannot be overridden.

Every class with at least one abstract method is itself final. And since final classes cannot be sub classed, it does not make sense for them to have abstract methods.

11. A,C

A regular class we want to extend can not have the access modifier "final", otherwise a compilation error will occur, so A is one correct answer.

Answer C is also a correct answer since in class "Derive", on line 3, an attempt is made to override the method "method". But the access modifier here (package friendly) is more restrictive than the access modifier "protected" on the original method, and this is not allowed when overriding.

For answer B, the "final" modifier here just means that this variable "p" cannot change the reference that it holds. No where in the code is this reference trying to be changed, so this change is not needed.

For answer D, the name of the variable is fine here, since it does not need to have the same name as the variable in the method it is overriding.

For answer E, changing the type of the variables from float to int has no effect on this code.

12. A,C

When overriding a method the access modifier for the new method must not be more restrictive than the access modifier for the original method. In this case the access modifier for the derived method is "protected", so the access modifier for the original method can either be the same, "protected" or more restrictive "private", and cannot be "public".

13. A

    When an object reference is converted in either an assignment or when passed to a method, the same rules apply for both. If the object being passed to or assigned is the same class or a subclass of the object that is accepting it, the conversion is done implicitly. Otherwise an explicit cast would be needed to convert the object type.

14. A

    Since no call is made to “super” in any of the constructors of the sub class “Derive”, they will all implicitly call the no argument constructor of the super class “Base”, as is in answer A. The other constructors in B, C, D and E may exist in the Base class, but they are not mandatory in this case.

15. C,D,E

    Answers C, D and E are all acceptable since they are not trying to override the method in the base class. We can see this since although the method name is the same, the parameter lists are different.

    Answer A is not acceptable since it is trying to override the method in the super class, but is throwing an exception that is a super class of the exception in the original method, and this is not legal.

    Answer B is not acceptable since it is trying to override the original method with an access modifier that is more restrictive.

16. A

    When overriding a method, the access modifier of the new method must not be more restrictive than the access modifier of the old method. It can be the same or less restrictive.

17. A

    When the word “super” is used in the constructor of a derived class, it must always be the first line in the body of the constructor of the derived class.

18. A

The access modifier “protected” is less restrictive that having no access modifier (package friendly).

If “protected” were to be put before this base class method, then when the sub class tried to override it a compilation error would occur since the sub class has the package friendly modifier, and when overriding a method the access modifier cannot be more restrictive than the original access modifier.

19. A,B

The first constructor in the sub class “Derived” explicitly calls the constructor of the super class passing an int value with “super(num)”. Therefore the super class must have an explicit constructor written that accepts one int.

The second constructor in the sub class “Derived” makes no call to super, so it implicitly calls the no argument constructor of the super class. Since we know that the super class already has one constructor that accepts an int, we know that the default constructor that accepts no arguments is not implicitly created by the compiler. Therefore the super class must contain an explicit version of a no argument constructor.

20. B

The access modifier of an overriding method may not be more restrictive than the access modifier of the original method. In this case the access modifier of the new method is “private”, so the only access modifier that would be allowed for the original method would be “private” as well.

21. B

When overriding a method the access modifier of the new method can be the same or more public than the access modifier of the old method, but not more private.

22. A

    In the “main” method of the program an instance of the object “Try1D” is created, and its method “tutu” is called passing the value of 10. In this method the value of the instance variable “luckyNum” is then set to be the number passed, 10, and that is the value printed to the screen.

    The method “tutu” in the sub class is not overriding the method “tutu” in the super class since the argument list that the 2 methods receive is not the same. Therefore the fact that the method in the sub class has an access modifier that is more restrictive that the access modifier in the super class is irrelevant.

23. B

    A class in Java can implement multiple interfaces. This is one of the main advantages of an interface.

24. A

    An interface can extend multiple interfaces, using the keyword “extend”.

25. B

    At line 20 a variable of the interface type “Computer” is created, and a reference to an object of type “TV” is placed into it. This is valid since the TV class implements the interface “Computer”.

    Then at line 21 the code does not compile, since the method “openTV” is being called on a variable of type “Computer”, and that interface does not define this method, so it does not recognize it. This method “openTV” exists in the class TV, so in order to fix this code an explicit cast is needed to turn the variable “myComputer” into a TV before the method is called on it. For example:

    ```
    ((TV)myComputer).openTV();
    ```

26. A

In Java a class can extend only one class.

27. B

Although an interface and an abstract class are both abstract, they are not the same.

All of the methods in an interface are implicitly abstract, whereas an abstract class can contain abstract methods as well as methods that are not abstract.

A class can extend only one other class, whether that extended class be abstract or not, whereas a class can implement as many interfaces as it wants to.

28. A

In this example the sub class and the super class both define a static int variable called “nony”, each with different values, and the sub class overrides the method “getLuck()”, which returns the value in “nony”. The “main” method makes an instance of the sub class and puts its reference into a variable of the super class type. Then it calls the “getLuck()” method on this instance. The “getLuck()” method of the sub class is used here. Although the variable “t” is the type of the base class, the value that it holds is a reference to an object of the subclass, and the actual object type that the reference refers to determines the actual version of the method to be used.

29. A

In the “main” method of this example a “Child” object is created, and then the method “getNum()” is called on it. This method, with no arguments passed, is defined in the Child class, so that is the method that is called.

If this child object were to try to call the “getNum” method passing an int value, a compiler error would occur, since that method is defined in the super class with the modifier “private”, so sub classes cannot access it.

30. D
A compilation error would occur on line 12 since a static method is trying to access a non-static variable, and this is not legal. Also the static method “main” cannot access the non-static variable “count” directly, and it tries to do so.

31. B,C
In this case both of the constructors in the sub class “Derived” use the “super” keyword to call a constructor of the super class. One of the constructors passes 1int, and the other passes 2 ints, so the super class needs to define constructors of these types, as is seen in answers B and C.

In this case answer A, which contains a no argument constructor would not be mandatory in the super class, since the keyword “super” is used in all of the constructors of the cub class. Although it would still be recommended to put it in just in case other classes try to call it.

Answers D and E could be in the super class, but are not mandatory.

32. B,C
The method in the super class has the package friendly access modifier, so when overriding this method in the sub class the access modifier can either be the same or less restrictive. The access modifiers “public” and “protected” are both less restrictive than package friendly, so answers B and C are correct.

33. A,C
The method in the super class has the package friendly access modifier, so when overriding this method in the sub class the access

modifier can either be the same or less restrictive. The access modifiers “public” and “protected” are both less restrictive than package friendly, so answers A and C are correct.

34. A

In the “main” method a variable of type “Father” is created, called “child”, and a reference to an object of type “Child” is placed into it.

Then the “getNum()” method is called on the variable

“child”. Since the actual object that the reference refers to is of type Child, the “getNum()” method in the Child class is the one that is called, so the number 20 is printed.

35. None

None of the answers are correct here. Since none of the methods in the sub class use the “super” keyword to call a constructor of the super class, they implicitly call the no argument constructor of the super class. So, the super class either needs to have no constructors defined for it, that way the compiler will implicitly insert the default no argument constructor, or the super class can have a no argument constructor explicitly written in it.

36. A,B,C

In this example the Derive class is not overriding the “shemeshMethod” method from the super class since the argument list is different from the original. So there are no restrictions as to what the access modifier can be for this new method.

37. A,B,C

Since the method “zzz” in the super class has the access modifier “private”, any method that overrides it can use any of the access modifiers available. This is because “private” is the most restrictive of the access modifiers, and when overriding a method the new method cannot have an access modifier that is more restrictive than the original method, and that is not possible here.

38. D

The code will not compile here since the sub class “Child” is attempting to override the method “getNum()” with an access modifier that is more restrictive than the original method has. The package friendly access modifier is more restrictive than the protected access modifier.

39. C,F

The second constructor in class “Derived” uses the “super” keyword, passing 2 ints, so the constructor in answer C would be mandatory.

The first constructor in class “Derived” does not use the “super” keyword to call the super class constructor, so implicitly the no argument constructor of the super class is called here, which is in answer E. Since we know that there is another constructor defined in the super class that accepts 2 ints, this no argument constructor must be explicitly written in the super class as well.

All of the other answers could be used as versions of the constructor in the super class, but they are not mandatory in this example. The only exception here would be that answers A and C are defining the same constructor, since the name of the variable is not relevant here, just the type, so only one of these could be present in a class.

40. A

Given these answers, the only constructor that would be mandatory to define in the super class would be the no argument constructor in answer A. This one is mandatory since we know that other constructors need to be defined for this super class as well, so the default no argument constructor is not provided in the super class.

In addition to the no argument constructor, the super class must define at least 2 other constructors, one that accepts 2 int values, and one that accepts 3 int values. None of the other answers fit these requirements.

41. A,E

    A is mandatory since we know that other constructors need to be defined for this super class as well, so the default no argument constructor is not provided in the super class.

    The constructor in E accepts 3 int values, and that matches one of the constructors in the sub class.

    In addition to these, the super class would need to provide a constructor that accepts 2 int values. None of the other answers fit this requirement.

42. C

    Line 3 is fine since all of the 3 variables defined will be “public static final int”. To make the code clearer though, it would be better to define each of these variables on a separate line, but it will compile this way in any case.

    A compilation error would occur on line 6 since the variable “X” is final, and therefore its value cannot be changed.

43. B

    In this example the class “Puppy” is not a sub class of the class “Dog”. In the “main” method of class “Puppy” first the value of the static int “count” is printed. It is still at its initialized value of 2. Then a Puppy object is created, which increases the count value by 1 in the constructor. Then the value of count is printed again, so it is now 3.

44. E

    In this example the only problem is that the sub class “SecondClass” is trying to override the method “y” that exists in its super class “FirstClass”, but this method is marked as final in the super class, which means that it cannot be overridden. So by removing the word “final” the method “y” can be overridden and the code will run.

Answers A and B are not true since there is no “final” modifier in front of the variable “x”. Answer D is not correct since it is fine to declare this instance variable as final. This just means that the reference that the variable “FC” holds cannot be changed, but the values inside the object that FC points to can be changed.

45. A

A final method that is also abstract is not legal, and is contradictory, since a final method cannot be overridden, and an abstract method is meant to be overridden.

Method cannot be both static and abstract, if they are defined as such a compilation error would occur.

46. A,C,D

When overriding a method, the new method must not have an access modifier that is more restrictive than the old method. In this example both methods have the default package friendly access modifier.

Answers A and C are legal since these are placing the access modifiers “public” or “protected” before the sub class method, and these are less restrictive than the default package friendly. Answer B is incorrect since “private” is more restrictive.

Working in the opposite way, if the original method had the access modifier “private”, then the sub class could use any of the other access modifiers available, since all of the others are less restrictive. Answers E and F would not work since they would make the method in the super class have an access modifier that is less restrictive than the package friendly access modifier in the sub class.

47. A

In the “main” method of this example 2 objects are being created, one of type “Base” called b1 and the other of type “Derive1” called b2.

Then the “method” method is called on each of these instances. Each instance uses the “method” that is defined in its object class.

When b1 calls “method” and passes a 5, the Base’s method is used, and the value returned is 105.

When b2 calls “method” and passes a 6, the Derive1’s method is used, and the value returned is 600. Since the Derive1 class is a sub class of the class Base, when it is created it receives its own instance of the variable num, which is initialized at 100.

48. A,F

    Since the Derive class has a constructor that calls “super” passing an int value, we know that the super class must have an constructor to accept this, so answer F is correct.

    Since we know that the super class has at least one constructor explicitly defined for it, an explicit no argument constructor must also be defined for it. This is because the constructor from the sub class that does not call “super” will implicitly call this no argument constructor in the super class, and it will not be provided by default by the compiler in this case.

49. A

    The class “PrivateCar” automatically inherits a copy of all of the instance variables from its super class “Car”. So the only value mentioned that is only specific to “PrivateCar” is a Boolean value as to whether or not it has been in an accident, which is answer A.

50. A,D,E

    Answer A legally overrides the method. Answers D and E are creating new methods and are not overriding since their parameter list is different than the original method.

    Answers B anc C are not acceptable since they are attempting to override the method, but are providing access modifiers that are more restrictive than the one in the original method.

51. D,E

    D and E are both legal method declarations.

    Answer A is not legal since the modifiers abstract and final cannot be used together on a method. Answer B is not legal since an instance variable cannot have the keyword “abstract” before it. And answer C is not legal since the word “friendly” is not a keyword in Java.

52. B

    An abstract method cannot be final since a final method cannot be overridden and the idea of an abstract method is that it should be overridden. A variable cannot be marked as abstract, so answer B is incorrect.

53. A

    Final methods cannot be overridden, that is the definition of a final method. If a static method is overridden an error will not occur, just the first one will always be used any way.

54. A

    In the “main” method, since the variable “calc” is of type “Calculator”, it uses the static method “getnum()” from the Calculator class, which returns the number 7.

    Then since the reference in the variable “calc” points to an object of type “GreatCalculator”, when the non static method “getValue()” is called, it uses this method from the GreatCalculator class, returning a 4.

    So 7 + 4 = 11.

55. A

In the “main” method a variable of type “Car” is created, which holds a reference to an object of type “SportCar”. Then the values of the instance variables “brand” and “volume” are printed. Since both the classes Car and SportCar have instance variables with the same name, the compiler will use the first ones found, which are in the Car class, since that is the variable type that holds the reference, so the values in answer A are printed.

56. B

In the “main” method a variable of type “Car” is created, which holds a reference to an object of type “SportCar”. Then the values of the instance variables “brand” and “volume” are printed. Since both the classes Car and SportCar have these instance variables with the same name, this time there is an explicit cast on each of these variables to make them into type SportCar. So here the values from the SportCar class are used, which are in answer B.

57. C

This code will not compile when it reaches the line that says:
“numA = 8 + numB;”

This is because the variable numB was never declared, so the compiler does not recognize it. The line below this line declares it, but that is too late.