Chapter 6 – Inner Classes

Answers

1. A,E

   Answer A is correct since if an inner class is not a static class then one way to create an instance of that class is to first create an instance of the outer class, and then create an instance of the inner class, based on the outer class. If an inner class is a static class, then its outer class object does not need to be created in order to access it.

   Answer E is correct, since an inner class can be declared static.

   Answers B and C are incorrect since an inner class can be defined as anonymous or can be declared private, but they don't have to be either.

   Answer D is incorrect since an inner class created within a method cannot access instance variables that are in its enclosing method. It can though access final variables within its enclosing method.

2. A

   An instance of an inner class cannot exist without its outer class. In this line of code "new Outer().new Inner()" one reference is created, that holds an object of the inner class called "Inner".

3. A,B,C,D

   Answers A and B are correct since an inner class (unless static), even within a method, can always access instance variables of its outer class.

   Answer C is correct since this variable var3 is a final variable passed into the enclosing method, and inner classes created within methods can access the final variables of their enclosing methods.

   Answer D is correct since the variable var5 is directly passed into this method of the inner class, so for sure it is available here.

   Answer E is incorrect since the variable var4 is an instance variable of the method that the inner class is created in, and as a rule an inner classes created within a method cannot access instance variables of its enclosing method unless they are final.

4. B

   In this example on line 5 an anonymous inner class is created that implements the interface "BambaInterface". But a compilation error occurs on this line since when implementing an interface arguments cannot be passed, and here the parameter "3.4" is being passed.

5. B
   An anonymous inner class does not have a name, so no constructor can be defined for it. An anonymous inner class is created within a method by using the keyword “new”, and then referring to a class that the anonymous class is subclasses (or to an interface that the anonymous class is implements.) During this line of creation it can pass parameters to the constructor of its super class, but it cannot redefine a constructor.

6. A
   An inner class can be declared as abstract like any regular class can be.

7. B
   By definition an inner class either extends another class or implements an interface.

8. A
   The JVM does not distinguish between an inner class and a regular class. When compilation occurs, a file is created for each inner class the same way a file is created for each regular class. The naming convention for this inner class file is different though. The name contains the name of the outer class, then a $ sign, then the name of the inner class. For example: “Outer$Inner.class”

9. A
   An instance of an inner class can always access the instance variables of its outer class, even the private ones. It is like the inner class is part of its outer class.

10. B
    If an inner class is declared inside a method, and instance of the inner class can only access final variables of the method, and not other instance variables of the method. This is because when a method has finished running its instance variables disappear, but the object that it created may still be around, so that object cannot hold instance variable values that may not exist anymore.

11. C
    In the “main” method here a new inner class object is created and its reference is placed into the variable called “popy”. The constructor of this class sets the variable “luckyNumber” to hold the value of 65. Then the “getLuckyNumber()” method is overridden just for this object, by placing it between “{“ and “};”.

    After this the method “getLuckyNumber()” is called on the “popy” object, and the newly defined method is used. This overridden method multiplies the value in the “luckyNumber” variable by 100 and returns this value, which is 650.

12. A
    In this example an instance of the outer class “Oliv” is created and its reference is put into a variable called “oliv”. The constructor of this class initializes the instance variable “luckyNumber” of this instance to hold the

value of 14. Then the method “setInner()” is called on “oliv”, which creates an instance of an inner class of type “Oliv” and puts its reference into a variable called “inner”.

After this, the “setLucky()” method is called using the “inner” variable, so the version of this method used is the one defined for the inner class, which is located in the “setInner()” method. This sets the variable “luckyNumber” of the “inner” class to have the value of 12.

Then the value of the variable “luckyNumber” of the outer class “oliv” is printed to the screen, which is 14. The creation of the inner class has nothing in the end to do with this answer.

13. F
In the “main” method a reference to a new “Oliv” object is created and placed into the variable “oliv”. It uses the constructor that accepts a reference to a new “Oliv” object as a parameter, and places this reference into the variable “inner” of the original object, who’s reference is held in“oliv”.

This second “Oliv” object also uses the constructor that accepts a reference to a new “Oliv” object, and again places this reference into the variable “inner” of this second object.

The third “Oliv” object calls the constructor that accepts and int value, and places this value (here 34) into the variable “luckyNumber” for this third object. Since no “Oliv” object reference was passed for this third object to be created, the “inner” variable holds a null value for this object.

The line: ”oliv.inner.iner.setInner()” calls the “setInner()” method on the third object created. This third object is accessed by referring to the first object “oliv” and then referring to its “Oliv” object reference placed in “inner”, and then again referring to the “Oliv” object reference placed in the “inner” variable of the second object. This “setInner()” method creates a new “Oliv” object and places its reference into the “inner” variable of this third innermost object. And then it overrides the method “setLucky” for this third object.

This new “setLucky” method is called on this third “Oliv” object, and the new “setLucky” version is used. The int value of 12 is passed to this method, but it is irrelevant here, since here the “luckyNumber” is always set to be 6, which is what is set for this third object.

Then on the last line of “main” this “luckyNumber” variable value for the third object is printed to the screen.

14. B
In the “main” method a reference to a new “Oliv” object is created and placed into the variable “oliv”. It uses the constructor that accepts a reference to a new “Oliv” object as a parameter, and places this reference into the variable

“inner” of the original object, who’s reference is held in“oliv”.

This second “Oliv” object also uses the constructor that accepts a reference to a new “Oliv” object, and again places this reference into the variable “inner” of this second object. Then the third “Oliv” object is created the same way as the second.

The fourth “Oliv” object calls the constructor that accepts and int value, and places this value (here 35) into the variable “luckyNumber” for this fourth object. Since no “Oliv” object reference was passed for this fourth object to be created, the “inner” variable holds a null value for this object.

Then the method “setInner” is called on the third “Oliv” object. This method creates a new “Oliv” object and places its reference into the ”inner” variable for this third object, which replaces (overwrites) the previous fourth “Oliv” object. This method also overrides the “setLucky” method for this fourth object so that is takes the int argument passed and places it into the “luckyNumber” variable. The next line calls the “setLucky” method on this fourth object passing the number 12, so this new version of the method is used.

Then the last line in this “main” method prints this “luckyNumber” value for the fourth “Oliv” object, which is 12.

15. A

In this example a String array containing 5 names is created, and then an array of type “InnerParty” is created that can hold references to 5 objects. Then there is a “for” loop to fill this “InnerParty” array.

The “for” loop creates 5 “InnerParty” objects and puts their references into the 5 spots in the “InnerParty” array. As each InnerParty object is created one of the names in the String array is passed along with the current array index. The constructor then sets the “name” variable for each of these objects to contain the name value in the String that was passed to it.

Back in the for loop, for each array item the “doSomething()” method is called, referring to the inner class (“inner”) of this “InnerParty” class. Both the outer class “InnerParty” and the inner class “inner” contain a “doSomething()” method, but here this method of the inner class is called.

This “doSomething()” method in the inner class takes the value in the “name” variable of the outer class, adds the text “98” to the end of this name to create a new String, and puts this new String value into the “name” variable of the inner class.

Then the program calls the “getInner()” method on the reference to an “InnerClass” object that resides in the number 1 spot in the “InnerParty” array. This method returns the value in the “name” variable of the inner class, which is “Haim98”.