Chapter 3 – Classes

Answers

1. A

   A static variable is part of a class itself, and not part of a specific object of the class as instance variables are. Therefore for a class that contains a static variable, there is one of these variable, regardless of the number of objects created from the class. And every time this variable is changed, no matter how it is changed, the others who access it see the updated value.

   In this example the static variable is called “sum”. Static variables can be accessed by creating an object of the class, and then by accessing the variable through the object, such as in lines 9, 11, 13 and 14. The more common way to access a static variable though is by the class name directly, without creating an instance of the class, as is done in line 7. And in line 12 the static variable “sum” is increased directly. This is allowed here since the static method “main” is part of the class that holds the static variable “sum”, so it has direct access to it. In this example the static variable “sum” is increased 5 times and decreased 1 time, so the result is 24.

2. A

   Classes can contain one or more static initializer blocks, which looks like a method but only has the word “static” and curly braces with statements in them. The statements in here are used to set initial values for static variables in the class. When the class is loaded, these blocks of code are each called once, in the order that they appear. Only after that is the “main” method called.

   In this example there are 2 static blocks, one increases the value of x to 60, and the second divides it by 15, making the value of x to be 4, the answer. The “main” is called and prints out the 4.

3. A

   The method “verMethod” is called passing a char variable as a parameter. Since there are no versions of this method that accept a char, the compiler sees that since a char can be implicitly converted to an int, it will use that version of the method. So the text “int version” is printed.

4. A

   Both object references and primitive types can be converted to other types when they are passed to methods or are assigned to a new variable of a different type. This conversion occurs automatically by the compiler if it is allowed. Or, if it is not allowed, the programmer can cast the value to be of the new type so that the compiler will accept it.

5. A

   Both object references and primitive types can be converted to other types or cast to other types as needed. Some conversions are done automatically by the compiler, and therefore a cast is not needed. With primitive values, if the type is converted to a larger type, then automatic conversion occurs. If the type is to be converted to a smaller type, then a cast is required. For objects the rules are more complex, but in general, subclasses can be converted to object types of their super class automatically, but the other direction would require a cast.

6. A,F

   For A, line 9 will not compile since a Float is a subclass of Object, so a cast to a float would be needed to fix this line “floty = (Float)obj;” And for F, line 5 will not compile since all of the constructor’s for the class float require that a value be passed to them.

   B is fine, since in line 8 an array is being put into a variable of type Object, and this is valid. C is fine since a Float object is being constructed properly. D is fine since an array of Strings is being created properly. And E is not true since the code will not compile, as mentioned above.

7. A,D

A is fine since when passing an int to a method that can take a float, the int will automatically be converted to a float and then passed to the method. D is fine since it is passing 2 int values, and there is a constructor that accepts 2 int values. The others are all not OK since there are no constructors for the method that accept the number and types of values passed.

8. B,C,D

When overloading a method, the new methods must have different arguments. This can be either a different number or type of arguments, or a different order of the types of arguments. The return value does not matter in overloading.

B, C and D are all proper overloading methods, since their argument list is different than the original method. A and E are really the same method as the original, and there cannot be 2 of the same methods in a class, so these are not legal.

9. A,D

Source files may have many different classes written in them, but they can only contain one class that is declared as public. The name of the file itself needs to be the same name as this public class.

10. B,C

When overloading a method, the new methods must have different arguments. This can be either a different number or type of arguments, or a different order of the types of arguments. The return value does not matter in overloading.

In this example both B and C have different argument lists, so they are valid for overloading. A, D and E are not valid since the argument list is the same as the original method. For A, the return value is different, but this is not a difference that matters in overloading.

11. D

There are many ways to try to encourage the garbage collector to clean up memory, but none of them can be 100% guaranteed.

12. A

In this example a new object called Zombit is created, and it has an instance variable called “val” that has its value set by passing it to the constructor. Here “val” it is set to 12. The method “jump” then increased the value of this instance variable “val “by 2, to make it 14, which is the answer printed.

13. A

In this example a new object called “Harmony” is created, and it has an instance variable called “held”, that is set to the value of 1 in line 6. On line 7 the method “jump” is called, passing a copy of the reference to the Harmony object, which happens to be the same Harmony object that we had previously created. The method “jump” increases the “held” value by 1. Since the same Harmony object reference was passed, that “held” instance variable was changed to 2, so that is the value that is printed.

14. E

In this example a new object called “Harmony” is created, and it has an instance variable called “held”, that is set to the value of 1 in line 6. On line 7 the method “jump” is called, passing the value of the variable “held” (an int) to the method “jump”. In this case the method “jump” accepts the int and treats it as a local variable, having no connection at all to the object’s instance variable “held”. Therefore when the value of “held” is printed, it is still 1, and has not been changed.

15. A

 compilation error will occur on line 5 since a float number 24.2 is trying to be put into an int variable. This could result in a loss of precision, so the compiler would require a cast to an int on this number to make it a legal statement. The same type of error would occur below in the class Car on line 3.

16. A

An object of type Object is a super class to an object of type Float. Therefore a Float object can implicitly be converted into an Object type of object, but converting the other way won't work. The object "o" would need an explicit cast to type "Float" in order for this to compile.

17. C,D

C and D each call one of the constructors available for class "Question". A, B and E do not call one of the available constructors for the class. A is not valid since when at least one constructor is written for a class, there is no default (no argument) constructor created for that class.

18. B

The java.lang.Math class cannot be subclassed since the class has a final modifier.

19. B

An object cannot be created out of the java.lang.Math class since its constructor is private.

20. None

It is true that initializing the variable will solve part of the compilation problem, since local variables do not get assigned default values like class level variables do. Therefore in this case if the "if" statement returned false, the value in the local variable "result" will have never been assigned a value, and the value in "result" is always returned at the end of this method. The compiler sees this as a potential problem and does not allow the code to be compiled due to this.

The other problem here is that on line 5 a float variable, “number” is being assigned to an int variable “result”, and the compiler requires an explicit cast here since a loss in precision could occur in the conversion process.

21. A

The native modifier can only be used with methods, and allows the program to use a method written in a language other than Java.

22. A

When using the keyword “this” or “super” inside a constructor, it must be the first statement in the constructor.

23. A,B,C,D,E

There are a number of different types of collections in the Java language, and each of them implements the “Collection” interface. A simple collection has few restrictions, among them that the order does not matter and that they can hold duplicates. A list is a type of collection that is ordered, a set is a type of collection that does not allow for duplicates and a map is a type of collection that supports searching on key fields.

24. A

Instance variables are variables on a class level, and these all automatically get default values placed into them if they are not initialized. Local variables are variables within a method, and these variables all need to be initialized some way, or else the code will not compile.

25. E

The variable x is a local variable and therefore needs to be initialized in order for the code to compile. If the “if” statement returns true then x is initialized, but if it returns false, x was never set to a value.

26. A,B,C

A collection does not have a special order and allows for duplicates.
A set has no special order, and does not allow for duplicates.
A list has a special order and allows for duplicates.

27. A

The class “Try” has a static variable called “sum”, which by default has the value of 0. On line 7 this value is increased by 1 by accessing this value through the class itself, and not with an instance of the class. On line 9 this value is increased by 1 again, raising it to 2. Here the variable is accessed through an instance of the class.

On lines 11 and 12 this value in the variable sum is decreased each time by 1, bringing it back down to 0. Here again the variable is accessed through an instance of the class.

Since the variable “sum” is a static variable of the class “Try” there is only 1 copy of it, so all of these lines accessed the same copy of “sum”.

28. A

When there are static blocks in a class they are all run in order when the first object of the class is created. Their purpose is to initialize the static variables of the class. In this class the only static variable is “x”.

So first on line 6 x is increased by 20, so since x started out at 0 it now holds the value of 20. Then on line 14 the value of x is divided by 16, and the result is 1, since it is cut to an int.

Then the “main” method runs and prints out this x value, which is 1.

29. B

There is no way to totally force the garbage collection (memory release) in Java.

30. B

When using the keyword “this” or “super” inside a constructor, it must be the first statement in the constructor.

31. D

In the “main” method in this example a new Zombit object is created, and the reference to this object is put into the variable “z”. The Zombit constructor accepts an int value, which is 12 passed here, and assigns it to the instance variable called “result”.

Then the “jump” method of Zombit is called, passing the value in the instance variable called “value”, which holds the value 0 since it was never assigned a value. This method causes the instance variable “value” of this instance to increase its value by the number passed. Since the original number was 0 and the number passed was 0, this value remains as 0. Then the this value in the “value” variable of the instance is printed to the screen.

32. C, D, E

Answers C and D are correct since they pass 1 or 2 ints, and there are constructors that accept 1 or 2 ints as parameters. Answer E is correct since a char value can be implicitly converted to an int value, so the constructor that accepts 1 int value is used in this case.

Answers A and B are incorrect since there are no constructors available to accept either no arguments or a floating point argument.

33. A

Line 3 will not compile since it is trying to convert a reference of type object into a reference of type Float, and this is not valid since the Float class is a sub class of the Object class.

34. B

More than one class can be defined in a single source file, but a source file can only contain one class that has the access modifier “public”. If a source file has a “public” class then the name of that file needs to be the same name as the name of this.

35. B

In Java all variables are defined within classes.

36. D

This code does not compile since line 8 calls an empty constructor, and the class does not provide one.

37. D,E

In the “main” method of this example 5 new “Pompa” objects are created. In the constructor of the class “Pompa” the static variable “count” is increased by 1, therefore after 5 “Pompa” objects are created this “count” value is 5.

After this the variable “pompa5” that holds a reference to one of the “Pompa” objects is set to null, which clears the reference in the “pompa5” variable so now it does not hold a reference to any object. When an object variable is set to null it is then available for the Garbage Collector to clear the memory space that it is taking up. Before the Garbage Collector cleans up the memory space it calls the method “finalize()”. In this example the method “finalize()” is overridden, and the new method reduces the value of the “count” variable by 1.

It can never be known if the Garbage Collector will clean up the memory right away or not. Therefore the next line of code that prints this “count” value to the screen can be either 4 or 5. It would be 4 if the Garbage Collector cleaned up the memory for this object right away since the “finalize()” method would have been called in this case. And it would be 5 if the Garbage Collector did not do its job yet.

38. C

In the “main” method here an object of type Chompy is created and its reference is put into the variable “first”. The constructor used is the one that accepts an int value and another Chompy object. The reference to this Chompy object passed is placed into the “next” variable of the instance, and the int value passed is assigned to the “id” value of the instance.

The new Chompy object that is created and has its reference placed in the “next” variable, also has its own copy of the instance variable called “id”. The number 102 that was passed is entered into here.

Then in “main” this “id” value of the object that “next” refers to is printed, which is the 102.

39. F

In the “main” method of this example 4 Rectangle objects are created, each having their own x and y int variable values, which are passed as parameters to the Rectangle constructor.

Then a variable of type double called “sum” is created. Then the method “area” is called on each of the 4 Rectangle objects, returning the area of the rectangle as a double value. Each of these values are added to the “sum” variable value, to get to the value of 17.0. The answer is 17.0 and not 17 since this value is held in a variable of type double.

40. E

This code does not compile since both of the constructors for the class “Plomba” are declared as “private”, and the “main” method tries to create instances of this class.

41. D

This code does not compile since the “main” method, which is a static method, is trying to access the non-static instance variable “num”, and this is not legal. A static method can only have direct access to static variables.

42. C

In the main method 2 Rublika objects are created, and their references are placed into the variables “first” and “second”. Then the “connect” method is called on “first” passing “second” as a parameter. This method sets a reference of the object in “second” to be placed into the instance variable “sun” for the object held by “first”.

Then in main a variable called “sum” is created and it is set to hold the sum of the return values of the method “getArea” on “first” and “second”. When “getArea” is called on “first”, it sees that “sun” is not null, so it first calls “getArea” on the object referred to in “sun”, which is the same object referred to by “second”. This call to “getArea” returns 20 (5 x 4). This is then added to the width x height of “first” which is 200 (20 x 10). The resulting sum of these 2 is 220. This sum is then added to a “getArea” call on “second”, which returns 20. So the total sum to print is 240.

43. A

Tracking the code gives that 'duba1.isOk() && !duba1.isAllOk()' boolean expression is true.

44. A

This code does not compile since the class “Tropy” has the access modifier “private” and this is not allowed for a regular class.

45. A

In this example a new Gorky object is created, and its instance variable “theNumber” is set to a value of –5. Then the method “moveTheNumber” is called on the Gorky object, passing the value 32. The code in the method “moveTheNumber” does not actually change the value of the “theNumber” variable, so when this variable is printed, its value is still –5.

46. A

In this example 2 objects of type BitMove are created and their references are placed into the variables “bitMove1” and “bitMove2”. When each of these objects is created the constructor is passed the value of 1, and this number is placed into the “num” instance variable for each of the 2 objects.

Then the “moveTheBits” method is called on “bitMove1”, passing one value, a 64. This uses the version of the method that accepts one int value, and performs a << operation on the value in “num”. The following is performed: 1<<64, which results in 1. For efficiency reasons, when using a bitwise operator (e.g. <<, >> or >>>) it first calculates the residual received when dividing the required number of steps (the right operand) by the number of bits required to represent the value on which the bitwise shift should work (the left operand). In our case it is 32 (each int value is represented using 32 bits). For that reason, shifting left/right an int value 64 steps is the same as shifting it 32 or 96 or even 0 steps.

Then the “moveTheBits” method is called on “bitMove2”, passing two values, 1 and 32. This uses the version of the method that accepts two int values, and also performs a << operation on the value in “num”. But here this operation occurs within a loop, and the number of times it occurs is the second value passed to the method, which is 32 here. The value after the << is the value of the first argument passed to the method. So the operation “num<<1” occurs 32 times, which results in 0.

47. A

In the Java language, all variables that are passed to methods are passed by value, both primitives and objects.

For primitives, for example, if an int value of 7 is passed to a method, the method contains a totally separate variable than the original variable, but both variables contain the value of 7. If the local variable within the method changes the value of the variable to a different number, this has no effect on the original variable's value that was outside the method.

With objects, the reference to the same object is passed to a method. Therefore the object variable outside the method and the one inside the method both point to the same object. Due to this, if any properties of the object are changed within the method, this affects the object that the variable outside the method points to as well, since they both point to the same object.

48. C

In this example there is a static variable called “sum”, which is of type int. The initial value of this variable is 20. On line 7 the value of “sum” is increased by 1, making it 21. This is done by accessing the variable through its class name, and not through an instance of the class, which can be done for static variables and not for instance variables. On line 8 the value of “sum” is increased by 1 again to make it 22. This time the variable “sum” is accessed through an instance of the class.

On line 11 the value in “sum” is decreased by 1, making it 21. Again this is done by accessing “sum” from another instance of a class. And on line 12 the value of sum is decreased by 1 again, making it 20. This time “sum” is accessed through the class name again.

On line 12 the value of “sum” is printed, and it is now 20.

There is only 1 copy of a static variable that is connected to its class. All objects created from the class access the same static variable. And as seen above, static variables can be accessed through the class itself or through instances of the class. They all affect the same variable.

49. E

This code will not compile since it is trying to create instances of the Plomba class on lines 14 and 15, and the two constructors of this class are marked as private.

50. A

In this example a new instance of a Zomba object is created, a local variable called “number” is given the value of 99. This variable value is then passed to the “jump” method, but the value passed is used in the method only, and has no effect on anything outside of the method. After this the value in the “number” variable is printed, and it still contains the number 99.

51. D

Answer D is correct since it is passing an int value, and there is a constructor to accept an int.

Answer A is not correct since there is no no-argument constructor available.

Answer B will not work since although it is passing a floating point value, by default floating point values are of type double, and there is no constructor that accepts a double. The value passed would need to be cast to a float in order for this to work.

Answers C and E are not correct since there are no constructors available that accept these values.

52. D

This example defines a class called Zombit, which has a static

variable called bomba that has an initial value of 0.
In the “main” method 3 new Zombit objects are created, each one passing an int value to the Zombit constructor. The constructor increases the value in bomba by the number passed in the constructor. On this case the numbers 2, 3 and 2 were passed to create the 3 objects, so the bomba value is increased to 7 (0 + 2 + 3 + 2 = 7). This value is then printed.

53. B
This example defines a class called Plomba, which has an instance variable called “id” of type int. In the “main” method 2 Plomba objects are created. One uses the no argument constructor, which sets the “id” value to be 0 for that object. The other uses the constructor that accepts an int value, passing a 72, and that value is placed into the “id” variable for that object. Then the “id” values for the 2 objects are added together, and their sum is printed. The sum od 0 + 72 = 72.

54. C
This example defines a class called Zombit, which has an instance variable called “value”. In the “main” method a Zomba object is created, and the value 1 is passed to its constructor. The constructor then assigns this value to the variable “value” for the object.

Next the “jump” method is called on the Zombit object created, passing in the number 2. In this method, the value of the “value” variable for this object is increased by 2, making it 3.

Back in the “main” method, the value of the “value” variable for the object is then printed, printing the number 3.

55. D
Both of the constructors available here accept one number, one accepts a float and one accepts an int. Answer D passes one int parameter to the constructor, so this is valid.

Answer A passes no parameters to the constructor, and since there are no constructors written that accept 0 parameters this answer is not valid. Answers B and C are not valid since they pass more than 1 number in the parameter list, and there are no constructors that accept this. And answer E is not valid since it passes a String as a parameter, and there are no constructors that accept a String.

56. D

In this example the class defines a static int called “bomba’ that starts out with a default value of 0. Then 3 “Zombit” objects are created, each one passing an int value to the constructor. The constructor adds each of the values passed to the variable “bomba” resulting in a total sum of 37. This value is then printed.

57. B

In this example two “Plomba” objects are created. Each Plomba object has its own instance variable called “id”, which is an int, and which is initialized by default to 0. The first object created calls the no argument constructor of the class, which sets the “id” value for this object to be 8. The second object created passes an int value, so it uses the constructor that accepts an int value and places this value into the “id” variable of the object. The value passed here is 72. The sum of these 2 values, 80, is then printed.

58. E

In this example a new Zombit object is created, and it contains the instance variable “value” with an initial value of 8. When the “jump” method is called, the local variable in the method is increased by 1, but this has no effect on the value in the variable called “value”. Then the value of the “value” variable is printed, and it still has its original value of 8.

59. A, D

Answer A is valid since it calls a no argument constructor, and there is one available. Answer D is valid since it is passing a char value, which can implicitly be converted to any numeric number that is at least the size of an int, so the constructor that accepts a double value is called.

Answers B, C and E are not valid since there are no constructors available to accept the type and number of the parameters passed.

60. E

In this example the constructor of the class “Zombit” is private, but since it is only accessed from the same class there is no compilation error. The class contains a static int variable called “bomba”, which has an intial value of 1. Each time a new Zombit object is created the constructor receives an int value as a parameter. The constructor takes the value passed and multiplies it with the current value in the “bomba” variable, and the result is the new value in “bomba”. So here 3 objects are created, passing a 2, a 3 and then a 2, which results in the number 12. This number is then printed.

61. D,E

Answer D passes 2 int values to the constructor and Answer E passes 1 int value to the constructor, and since there are constructors defined to accept 1 and 2 int values these answers are valid. Answers A, B and C are not valid since there are no constructors available to accept the number and types of parameters passed.

62. A

This example has a class called Try, with a static variable of type in called “sum” that has an initial value of 20. In the main method, this value in “sum” is increased and decreased a number of times, accessing this variable in a few different ways. The variable is accessed directly though the class name, such as in “Try.sum++”, and though an object that is created, for example in the line “t2.sum++”. Each time the value of sum is changed, whichever way it is changed, the same variable “sum” is accessed. In the end of main the value of “sum” is decreased by 1 and then printed, and it then has the value of 19. In this last case, the variable “sum” is accessed directly, since the “main” that is accessing it is in the same class that the “sum” variable is defined in.

63. E

This code will not compile since both of the constructors of the class “Plumba” are “private”, so they cannot be accessed out of the class, so no objects can be created from this class.

64. D,E

Answers A, B and C are not valid since there are no constructors available to accept the number and types of parameters sent.

Answer D is acceptable and uses the constructor that accepts 3 int values. The first value passed here is a double that is cast to a short. Passing a short value is valid here since a short can implicitly be converted to an int.

Answer E is valid since it passes 1 int value, and there is a constructor that accepts one int value as a parameter.

65. A,C,D

A constructor always has the same name as the class that it is in, and it never has a return value. Every class must have at least one constructor. If there was no constructor written for a class, then the compiler provides a default no argument constructor for the class. If at least one constructor is written for a class then no default constructor is created for that class.

A constructor can be private, which means that it cannot be accessed, so that class cannot be instantiated. Usually this is done when a class’ main use is to have a bunch of static variables and methods to be accessed.

66. B

On line 6 a new instance of the “Test” object is created and its reference is placed in a variable called “test”. On line 7 the “num” instance variable of the object is set to have the value of 100. On line 8 the “jump” method of the object that “test” refers to is called, and it is passing a copy of the reference to this same object.

In the “jump” method the “num” value of the object that the reference passed refers to is added to the “num” value of the current object. Since they both refer to the same object, the value is increased to 200. Then back in the “main” method the “num” value of the object reference in the variable “test” is printed, which is 200.

67. B,D

A normal class can have the access modifiers public or friendly (not really an access modifier), and cannot have the access modifiers private or protected.

68. A,B,C,D

Constructors and methods can have any one of these access modifiers.

69. A

In this class called “Demo” there is no explicit constructor, so the default constructor is called, which does nothing. Note that there is the method “public void Demo()”, which looks like a constructor at first glance, but it has the return value of void, and constructors have no return value. In this example, the method Demo() never gets called, so the value of the variable “magicNum” retains its default value of 0.

70. A

In the “main” method a new instance of the class “Demo” is created. Then the instance variable “number” of the class “Demo” is printed to the screen. The value of the variable “number” within “main” is a different variable than the one in Demo, and is not the one that is printed.

71. D

The variable “number” is declared and defined a few times in this program. This is confusing and not a great way to program, but it is legal and works. In the “main” method an instance of the Demo class is created, and then its “getNumber()” method is called. The “getNumber()” method has its own local variable “number” and that is the value that is returned (15) and then printed to the screen.

72. C

This code does not compile since the variable “m” was never initialized. The line “Math m” is a reference of class Math, but if this line tried to make an instance of the class Math as well, that line would not compile, since the class Math has a private constructor so an instance of that class cannot be made.

73. A

If a method accepts an Object as a parameter, then what it is actually receiving is a copy of the reference that points to the original object. So then there are 2 references pointing to the same object. If the method changes a value in the object itself, then since there is only one object, the change remains. If within the method the reference to the object is reassigned to point to a different object, then the reference outside of the method still points to the old object.

74. A

There may be one or more static blocks within a class, and each of them is called in order the first time the class is loaded. These blocks are used to initialize any static variables in the class.

75. A

A final variable is a constant in Java. This means that once a value is set for it, that value can never be changed. This value can be set when the variable is declared, or later on, as long as it is assigned a value before it is used.

76. A

All of these are correct statements.

77. A

A class must implement the Cloneable interface in order for objects of this class to have to the ability to be cloned. If the class does not implement this interface, and an attempt is made to clone an object of this class, a CloneNotSupportedException will be thrown.

78. A

When a shallow clone is performed, the primitive variables in the new object receive the same values as those variables in the old object contained. The object reference variables in the new object contain a copy of the reference to the same objects referred to by these variables in the old object. Therefore the object reference variables in the 2 objects contain references to the same objects.

79. A

Car implementation of the Cloneable interface doesn't create new objects for each one of the objects a Car object holds. For that reason, changing the owner name using car2 variable effects both Car objects.