1) Question 1 is about what a class is, what is an object, and about instance variables vs local variables. This question asked you to select all the options that were true. In this case all three given statements were true. A class lets you create instance variables and methods which act as data and actions. An object is another way to say an instance of a class. Finally, any instance variables made will always have a default value, but any local variables made in methods, or any code block do not have default values.

2) Whenever there is a local variable and an instance variable with the same name (which is allowed since they exist in different context) by default the java compiler will use the local variable over the instance variable. This means if we needed to use the information from an instance variable, we need to specify that it is from the object. In the class that an instance variable is created the keyword *this* would be used to refer to the object of that class. The answer for this question was **false**.

3) Instance fields which are instance variables and instance methods belong to instances of a class, in other words, they belong to objects of a class and thus an object needs to be created in order to use those fields. The answer for this question was **true**.

4) The given statement is **true**. The java file must have the same name as the public class in the file. It is possible to put other classes in the same file, but they cannot have the public access modifier.

5)

object	heap
primitive variables	stack
references	stack
method frames	stack
String pool	heap

6) The key for this question is to understand how objects are created. By counting the number of *new* keywords, we will be able to know how many objects we have because when *new* is used, an object is created. So, in the given code there is *new* used 3 times which means there is a total of **3 objects** in this snippet. The other lines such as the Three t3 or Three t4 = t3 are just references being made to no object or to an object that already exists. Again, the actual objects are related to how many times we use *new*.

7) This question is meant to test your understanding of an instance method signature in relation to local variables being use alongside instance variables. The key things to pay attention to is that this is an instance method because it is setting up the instance variables for the object – That means that it should not be static. The return type is void because the method call in the main method does not get stored into anything, so there is no return value. Finally, we know that the method needs to have some parameters because there is arguments given in the method call of a String and a number. All of these conclusions

can result in only one possible method signature: **public void setInto(String names, int numOfTests).**

8) The main focus in this question is about instance variables and local variables. An object of the Ball class is created. Then a String 'color' is also made inside of the main method, which means it is a local variable. In the following line we assign "red" the local variable 'color'. In the final line we are calling the String 'color' instance variable from the Ball object, but since we never touched this variable since it is an instance variable, not the local one we created. The value that is printed is **null**. If the local variable color was printed, then it would give "red".

9) The main idea in this question is about how objects are printed. If we take a look at the Box class, we have two instance variables, a method called print, and a method called setInfo. In the other class we have a main method which is created a Box object. Then the setInfo method with given arguments is run to set the initial values of 'width' and 'length'. In the last line the box object is being printed by giving the reference name. The toString() method is was defined so it is automatically invoked to give us the following result: **Width: 5.0, Length: 11.0**

10) This question combines most of the concepts covered in the previous questions. The main idea here is about printing objects, but we also need to pay attention so the instance and local variables. If we take a look at the makeSport() method that is used to set the initial values of the instance variables we can conclude that 'name', 'numOfPlayers' and 'teamBased' will be assigned with the given argument of the makeSport() method. Now analyzing the main method, we have a Sport object being created. Then we have the makeSport() method being called with some given arguments and in the last line we have the object bring printed. By looking at the toString method format we can see how the object is printed. The correct print for this snippet will be: **The sport is Soccer and has 22 players in teams: true**

11) A static member is something that belongs to the class. That means any object created from that class will share those static members. As a result, the given statement is **true**: a static variable of a class can be called by an object a class. Usually, static members are accessed by the class name directly, not by objects, but it is possible to use the objects as the reference.

12) A static block is executed whenever the class is loaded, meaning whenever it is used at all. It is also always the first thing executed. So, the given statement is **false** because the static block is not linked to the objects of a class. The block will only be executed once, the first time the class is used, and not again whenever an object is created.

13) The statement says a static method can be called by the class name. That statement is **true** since static members are meant to be called by the class name.

14) From the given options there is two that cannot have the static keyword applied to them: **package** and **constructor**. Besides those the variables, methods, and class are allowed to be static. Note that the static class is used for inner classes (class inside of another class).

15) The Main class has an instance variable 'n' defined with a value of 10. It also has a static variable 'z' defined at 20. An object of the Main class is made called obj. Using this object as a reference the static variable 'z' is changed to 40. Then another object of the Main class is created. Using the obj2 reference of the second object the instance variable of that object is changed to 5. When the print statement is run the values of obj2 object are printed. First the instance variable is printed which was 'n'. The value for that one was 5 and the value of 'z' is printed too giving 40. The reason it is 40 is because the 'z' was static. Even though we used the first object, obj reference to change the value it will also be changed to the second object, because it is static. All objects of the class share the same static variable.

16) This question asks about which method signature will allow the code to compile. Taking a look, the method body we can determine that we need some parameter named 'nums' from the method. We can also conclude that 'nums' should be an array because we are using it in the Arrays.sort() method. In the following line we can see that we are returning the first element of our array. Now taking a look at the main method, we see that we have an int array with some values. In the next line we have a int variable 'min' which is getting a value from a method called minNum and it is accept the 'arr' as an argument. At this point we can conclude that the method name will be 'minNum' and the parameter type will be an int array. We should also notice that we are using the method right away without any object, so the method has to be static. By the variable we are storing the result into we can always tell that the return type will be int. After all these conclusions we can say the method signature would be: **public static int minNum(int [] nums)**

17) The Book class has a static String declared with a value J. K. Rowling. It also has static block which changes the value of the author to Stephen King. The static block would be run whenever the class is used. In the class there is also a main method which create an object of the Book class. At this point the class is used, which means the static block is executed given our author String a value of Stephen King. On the next line the author variable is accessed by the object reference and reassigned to Christopher Paolini. Another Book object is created, and the two object references check if the author variable they have access to are equal. This statement will be **true** no matter what the values actually were because the variable is static, so it is shared with both objects.

18) This class has a static method run(), a main method, and a static block. In this case the first section executed is the static block which gives us 3. After the static block the only other thing that will run is the code in the main method, so the final output is **32**. The run() method is never called so it is never executed.

19) The given statement is **true**. The *new* keyword is used to call the constructor of a class to create an object of that given class.

20) The given statement is **true**. One of the rules to create a constructor is that it must have the same name as the class. If you try to make a constructor with a name that is not the same as the class name, there will be a compile error.

21) The given statement is **false**. The default constructor is a no argument constructor given by the compiler but if any constructor is created the default constructor is no longer there.

22) From the given statements these are the correct ones: **Constructors are used to initialize variables, Constructors can be overloaded,** and **Constructors are invoked to create objects of a class.** The other statements are not correct.

23) There are two classes in this snippet: Pond and Water. The Pond class has a reference of Water 'water' and has a constructor which we need to finish. The body in the constructor is to assign the reference of 'water' to an actual object of Water by providing some input. If we look at the water class, we can see a String instance variable 'color' and a constructor which accepts a String 'color' to set the initial value of the instance field of the object. Given all this information we know the constructor on line 5 should be named Pond to match the class it is in. We also know that there is some variable called 'color' which is being used in the Water constructor, but it does not exist in the Pond constructor, so it must be coming from the arguments of the constructor. The access modifier is not important in this question so we will keep in public. The only constructor that we could put on line 5 with the given information is: **public Pond (String color).**

24) There is two classes in this code snippet: Bed and Pillow. The Bed class has a String 'size' and an ArrayList of Pillows 'pillows'. There is also a constructor which accept two argument and initializes the instance fields of the Bed objects. The size is simply stored into the variable but for the number of pillows that are stored into the 'pillows' ArrayList depends on the for loop which will execute based on the second given argument. Each iteration of the for loop creates a Pillow object and adds it to the 'pillows' ArrayList. Now looking at the Pillow class we have an instance String 'material' which does not do anything in this snippet. There is also a main method which is making a Bed object and giving the arguments needed. The object is made with the 'size' set as "king" and the second argument of "4" determines how many times the for loop should run, and in other words how many Pillow objects are added to the ArrayList 'pillows'. In the print statement the size of the instance field of 'pillows' is printed, which is just how many Pillow object were made. The output will be **4**.

25) We have a class B which has a static int 'count'. We also have three constructors, one which has no parameters, one that has an int parameter, and last one with a String parameter. Looking at the Test class where the main method is using the B class:

Code	Count value
new B();	0 + 1 = 1

B a = new B();	1 + 1 = 2
B a2 = new B("word");	2 + 4 = 6
B a3 = new B(5);	6 + 5 = 11

The final value of count in the last line is **11.**