**Assignment – 1**

1. JDK – The Java Development Kit. Contains all the tools needed for writing, debugging and executing Java code, as well providing an environment and pre-compiling support.  
   JRE – The Java Run-Time Environment. Comes bundled in with the JDK. It provides the environment for the user to work with, usually through an IDE like Eclipse. It is the implementation of the JVM.  
   JVM – The Java Virtual Machine. The secret behind Java’s platform independence. It loads, verifies and executes the user’s code and provides the run-time environment, or JRE.
2. There are three ways to create an object other than using the NEW keyword.  
   **-** Using the Class name and its public default constructor (if it has one).  
   **-** Using the clone() method to create a copy of an existing object.  
   **-** Using the object in a deserialized form.   
   EX. MyObject() obj = (MyObject)instream.readObject();
3. The **java.lang.OutOfMemoryError** is an exception which is thrown in one of the following circumstances.
   1. The object is out of memory, and no more space can be allocated via Java’s Garbage Collector.
   2. The **swap** space is too low, and allocation isn’t possible.
   3. There is no more **PermGen** space available.
   4. An oversized Array that exceeds the JVM Memory limit.
   5. An attempt was made to load a native method, but the requested native method was larger than the JVM Memory limit.
4. The JVM is primarily an interpreter, though it does compile the bytecode of the CLASS file produced by the user.
5. The **loadClass** command only loads the class but doesn’t initialize the object. The **Class.forName** command does both.
6. It can be done, but it’s not recommended. Only do it if the Garbage Collector’s services are urgently needed after a set of operations.
7. The storage of object member variables and function local variables is on the **Stack.**
8. All member variable must be loaded into the **Heap.** As such, they have to be initialized with default values when an instance of class is created. In case of local variables, they don't get loaded into the **Heap** – they are stored in the **Stack** until they are to be used.
9. Java has no need for pointers, largely as a design choice. The JVM moves the data in the memory quite a lot of times during the run-time – having pointers would make that quite difficult. Another reason is that Java deals mostly with objects, not primitive values.
10. Java has three kinds of Class Loaders.
    1. Bootstrap Class Loader – Loads JDK internal classes.
    2. Extensions Class Loader – Loads JDK extension classes, mostly from the library.
    3. System Class Loader – Loads from the current CLASSPATH. Needs to be set manually.
11. **See Answer 10.**
12. Instance and static variables use the **PermGen** section of the **Heap** memory.
13. **PermGen** is a section of the **Heap** memory used to store the data assorted with Objects, Instanced sections and static variables.
14. **Metaspace** is the replacement of **PermGen** from Java 8 onwards. It essentially does the same thing as **PermGen** – with a lot more flexibility in memory allocation and size.
15. The following things happen in sequence in object creation.
    1. Memory is allocated.
    2. Fields are initialised to their default values.
    3. The first line of the specified constructor is invoked.
    4. Instance initialiser is invoked, and all fields are set to the specified values, if any.
    5. The rest of the constructor code is executed.
16. The JVM comprises the following sections.
    1. **Heap** Memory.
    2. **Non-Heap** Section.
    3. One small section for the JVM Metadata.
17. The code block with the static modifier signifies a class initializer - without the static modifier the code block is an instance initializer. Class initializers are executed in the order they are defined when the class is loaded. Instance initializers are executed in the order defined when the class is instantiated, immediately before the constructor code is executed and immediately after the invocation of the super constructor.
18. We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class. If static method is redefined by a derived class, then it is overloaded, not overridden.
19. **See Answer 18.**
20. The **this** keyword works as a reference to the current Object whose Method or constructor is being invoked. It can be used to refer to any member of the current object from within an instance Method or a constructor.
21. The **super** keyword is used inside a sub-class method definition to call a method defined in the super-class. Private methods of the super-class cannot be called. Only public and protected methods can be called by the **super** keyword. It is also used by class constructors to invoke constructors of its parent class.
22. The **this** keyword is always used to refer to the current object. The **this()** method is used to access one constructor from another where both constructors belong to the same class.
23. A static method is a method that’s invoked through a class, rather than a specific object of that class. Static methods can only access static variables – they can’t use anything that’s specific to that particular object. Non-static methods (or instance methods) must be called on a specific object and can use the object’s instance data. A final method is just a method that cannot be overridden (explicitly final) – while static methods can be overridden (implicitly final).
24. The **main()** method in Java is static because it can then be invoked by the runtime engine without having to instantiate an instance of the parent class.
25. Memory for static variables is created only once in the program at the time of loading of class. These variables are preceded by the static keyword. Static variables can be accessed with their class reference. Memory for non-static variables is created at the time of create an object of class. These variables should not be preceded by any static keyword. These variables are accessed with their object reference.
26. The **super** keyword is used to access the members of the super class and to send the arguments from a sub class to a super class constructor. The **this** keyword refers to the current class object and is used to send the argument within the same class constructor.
27. Java uses only the pass by value system.
28. A constructor in Java is a block of code that is like a method that’s called when an instance of an object is created. It doesn’t have a return type, is not considered a member of a class and is called automatically when a new instance of an object is created.
29. Yes, we can use Copy Constructors. However, it is recommended to use the **clone()** method instead, unless it’s absolutely necessary.
30. Constructor Chaining is the process of calling one constructor from another constructor with respect to current object.
31. Yes, by using the **super** keyword.
32. No Argument Constructors just use no arguments when they are being called.
33. **See Answer 32.**
34. If a Constructor is made private, then it can only be accessed inside the class. There are two reasons to do this.
    1. Internal Constructor Chaining.
    2. Singleton class design paradigm.
35. Constructor Overloading is generally utilised when there is a need to initialise an object in a different way, such as in multithread code.
36. No, the functionality is built into the Garbage Collector and is unnecessary.
37. Native methods are methods written in other languages like C/C++. Native methods can be called from Java using the JNI. Native methods are used when the implementation of a particular method is present in languages other than Java.
38. **See Answer 23.**
39. **See Answer 23.**
40. No, a private method cannot be overridden since it is not visible from any other class.
41. It generally refers to the practice of labelling a method as **private**, so that only the members of that class can access that method.