**B3. Java is Platform independent.?**

When you compile Java programs using javac compiler it generates bytecode. We need to execute this bytecode using JVM (Java Virtual machine) Then, JVM translates the Java bytecode to machine understandable code.

You can download JVM's (comes along with JDK or JRE) suitable to your operating system and, once you write a Java program you can run it on any system using JVM.

**B4. Three flavors of Java.**

ME(Micro Edition) for mobile,  
SE(Standard Edition) for desktops,   
EE(Enterprise edition) for enterprise.

**B5. How many types of memory areas are allocated by JVM?**

Java Virtual Machine is one of the most important components which does particular types of work.

1. Loading of code
2. Verification of code
3. Executing the code
4. Providing a run-time environment for the users

All these functions take different forms of the memory structure. These data structures are:

* Heap
* Stack
* Program Counter Register
* Native Method Stack

**Heap memory** structure is usually implemented for allocating memory dynamically. Variables assigned with this type of memory structure can be allocated at runtime, but they have slow accessing of memory.

**Stack memory** structure is mostly implemented for providing static memory allocation. Programmers could make use of stack if they knew in advance how much memory needs to be allocated for the storage of data.

**Program Counter Register:** Programs are a set of instructions or orders feed to a computer for performing. These instructions are delivered to the processor by the program written by a human. Program counter register holds the address of the upcoming instructions to be executed.

**Native** methods form a stack that is primarily implemented to line up with your system calls as well as libraries scripted in different computer language.

**B6. What is the latest version of Java?**

The latest version of Java till date is Java SE 11.0.2 released on 15th January 2019. The latest version as of now is Java SE 12.0.2.Feb 12, 2019

**B7. What is Write Once, Run Anywhere (WORA)?**

**WORA** which is abbreviated as **Write Once Run Anywhere** is the feature applicable to those programs which hold the capability to execute itself on any operating systems or on any machine. This terminology was given by Sun Microsystem for their programming language - Java. According to this concept, the same code must run on any machine and hence the source code needs to be portable. So Java allows run Java bytecode on any machine irrespective of the machine or the hardware, using [JVM (Java Virtual Machine)](https://www.w3schools.in/java-tutorial/java-virtual-machine/). The bytecode generated by the compiler is not platform-specific and hence takes help of JVM to run on a wide range of machines. So we can call Java programs as a write once and run on any machine residing anywhere.

**B8. Is Java a pure/fully object oriented language?**

Java is not Pure Object Oriented Programming language. Because use primitive data type. No. Java is not fully object-oriented: the basic types(int, float,boolean..etc) are not objects.

**B9. What is bytecode?**

Bytecode. Bytecode is program code that has been compiled from source code into low-level code designed for a software interpreter. It may be executed by a virtual machine (such as a JVM) or further compiled into machine code, which is recognized by the processor. ... Below are examples of Java bytecode instructions.

**B10. What is Heap space in Java?**

Java applications are only allowed to use a limited amount of memory. This limit is specified during application start-up. To make things more complex, Java memory is separated into two different regions. These regions are called Heap space and Permgen (for Permanent Generation):

The size of those regions is set during the Java Virtual Machine (JVM) launch and can be customized by specifying JVM parameters -Xmx and -XX:MaxPermSize. If you do not explicitly set the sizes, platform-specific defaults will be used.

The java.lang.OutOfMemoryError: Java heap space error will be triggered when the application **attempts to add more data into the heap space area, but there is not enough room for it.**

Note that there might be plenty of physical memory available, but the java.lang.OutOfMemoryError: Java heap space error is thrown whenever the JVM reaches the heap size limit.

**B11. Difference between EAR, JAR and WAR file in J2EE.**

**JAR File (Java Archive)**

**JAR** file is like **winzip** file compressed with **JDK** software. The problem of popular winzip file is it should be unzipped on Windows OS only. **winzip is platform-dependent**. A winzip file cannot be opened or unzipped from Linux. For that matter, **all zipping algorithms are platform-dependent**. The interesting point with JAR file is it can be zipped and unzipped by JVM irrespective of OS. That is, wherever Java is working, the JAR file can be created or unzipped. It can be said, **JAR is platform-independent way of creating a zip file**.

A JAR file extension is **.jar** and is created with **jar command** from command prompt (like **javac** command is executed). Generally, a JAR file contains Java related resources like libraries, classes etc.; but need not be. It can contain any non-sense (non-related) files like .txt, .doc, .gif etc., just like a winzip can contain.

All the options of JAR command, security aspects, creating JAR file with unzipping are discussed with examples at [JAR (Java ARchive) Files](http://way2java.com/java-lang/jar-java-archive-files/).

**WAR File (Web Application ARchive)**

A **WAR** file is simply a JAR file but contains only Web related Java files (but not Web unrelated files) like Servlets, JSP, HTML, Database Java Beans, web.xml file, Property bundles, JavaScript, shopping carts etc. necessary to develop Web applications. The advantage of WAR file is it can be deployed easily on client machine in a Web server environment. The extension of WAR file is **.war** but ofcourse created with JAR command only.

Two programs are given on the creation of WAR file.

**1.** [Servlet WAR File in Tomcat with Creation, Deployment, Execution](http://way2java.com/servlets/servlet-war-file-tomcat-creation-deployment-execution/)

**2.** [Servlet WAR File in Weblogic Creation and Deployment](http://way2java.com/servlets/servlet-war-file-weblogic-creation-deployment/)

**EAR File (Enterprise Application ARchive)**

**EAR** file contains **Enterprise application** related files (J2EE) like XML, EJB modules etc. Ir is also created with JAR command only but with extension **.ear**. EAR file is deployed in an **application server**.

**B12. Explain memory leak in Java.**

A Memory Leak is a situation **when there are objects present in the heap that are no longer used, but the garbage collector is unable to remove them from memory** and, thus they are unnecessarily maintained.

A memory leak is bad because it **blocks memory resources and degrades system performance over time**. And if not dealt with, the application will eventually exhaust its resources, finally terminating with a fatal java.lang.OutOfMemoryError.

There are two different types of objects that reside in Heap memory — referenced and unreferenced. Referenced objects are those who have still active references within the application whereas unreferenced objects don’t have any active references.

**The garbage collector removes unreferenced objects periodically, but it never collects the objects that are still being referenced.** This is where memory leaks can occur:

**B13. How Garbage collection works in Java?**

Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program.

**B14. Does Java garbage collector clean both heap and stack memory?**

There is no garbage that lives on the stack. Garbage collection only happens on heap for Objects that are no longer reference from any GC roots. Stack is where your local variables (like primitives and object references) live. ... Its objects which don't have a strong reference from a thread stack which can be cleaned up.

**B15. Why garbage collection is required in Java?**

It makes **java** memory efficient because **garbage collector** removes the unreferenced objects from heap memory. It is automatically done by the **garbage collector**(a part of JVM) so **we** don't **need** to make extra efforts.