1. What is JVM? Why is Java called the ‘Platform Independent Programming Language’?

JVM, or the Java Virtual Machine, is an interpreter which accepts ‘Bytecode’ and executes it.

Java has been termed as a ‘Platform Independent Language’ as it primarily works on the notion of ‘compile once, run everywhere’. Here’s a sequential step establishing the Platform independence feature in Java:

- The Java Compiler outputs Non-Executable Codes called ‘Bytecode’.
- Bytecode is a highly optimized set of computer instruction which could be executed by the Java Virtual Machine (JVM).
- The translation into Bytecode makes a program easier to be executed across a wide range of platforms, since all we need is a JVM designed for that particular platform.
- JVMs for various platforms might vary in configuration, those they would all understand the same set of Bytecode, thereby making the Java Program ‘Platform Independent’.

2. What is the Difference between JDK and JRE?

When asked typical **Java Interview Questions** most startup Java developers get confused with JDK and JRE. And eventually, they settle for 'anything would do man, as long as my program runs!!' Not quite right if you aspire to make a living and career out of Programming.

The "JDK" is the Java Development Kit. I.e., the JDK is bundle of software that you can use to develop Java based software.

The "JRE" is the Java Runtime Environment. I.e., the JRE is an implementation of the Java Virtual Machine which actually executes Java programs.

Typically, each JDK contains one (or more) JRE's along with the various development tools like the Java source compilers, bundling and deployment tools, debuggers, development libraries, etc.

3. What does the 'static' keyword mean?

We are sure you must be well-acquainted with the **Java Basics**. Now that we are settled with the initial concepts, let's look into the Language specific offerings.

Static variable is associated with a class and not objects of that class. For example:

```
public class ExplainStatic {       public static String name =
"Look I am a static variable"; }
```

We have another class where-in we intend to access this static variable just defined.

```
public class Application {         public static void
main(String[] args) {
System.out.println(ExplainStatic.name)         } }
```

We don't create object of the class ExplainStatic to access the static variable. We directly use the class name itself: ExplainStatic.name

4. What are the Data Types supported by Java? What is Autoboxing and Unboxing?

This is one of the most common and fundamental Java interview questions**.** This is something you should have right at your finger-tips when asked. The eight Primitive Data types supported by Java are:

- **Byte :** 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive)
- **Short :** 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).
- **Int :** 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive)
- **Long :** 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive)
- **Float**
- **Double**

Autoboxing: The Java compiler brings about an automatic transformation of primitive type (int, float, double etc.) into their object equivalents or wrapper type (Integer, Float, Double,etc) for the ease of compilation.

Unboxing: The automatic transformation of wrapper types into their primitive equivalent is known as Unboxing.

5. What is the difference between STRINGBUFFER and STRING?

String object is immutable. i.e , the value stored in the String object cannot be changed. Consider the following code snippet:

```
String myString = “Hello”; myString = myString + ” Guest”;
```

When you print the contents of myString the output will be “Hello Guest”. Although we made use of the same object (myString), internally a new object was created in the process. That’s a performance issue.

StringBuffer/StringBuilder objects are mutable: StringBuffer/StringBuilder objects are mutable; we can make changes to the value stored in the object. What this effectively means is that string operations such as *append* would be more efficient if performed using StringBuffer/StringBuilder objects than String objects.

```
String str = “Be Happy With Your Salary.'' str += “Because Increments are a myth"; StringBuffer strbuf = new StringBuffer(); strbuf.append(str); System.out.println(strbuf);
```

The Output of the code snippet would be: **Be Happy With Your Salary. Because Increments are a myth.**

6. What is Function Over-Riding and Over-Loading in Java?

This is a very important concept in **OOP (Object Oriented Programming)** and is a must-know for every Java Programmer.

Over-Riding: An override is a type of function which occurs in a class which inherits from another class. An override function “replaces” a function inherited from the base class, but does so in such a way that it is called even when an instance of its class is pretending to be a different type through polymorphism. That probably was a little over the top. The code snippet below should explain things better.

```
public class Car { public static void main (String [] args) { Car a = new Car(); Car b = new Ferrari(); //Car ref, but a Ferrari object a.start(); // Runs the Car version of start() b.start(); // Runs the Ferrari version of start() } } class Car { public void start() { System.out.println("This is a Generic start to any Car"); } } class Ferrari extends Car { public void start() { System.out.println("Lets start the Ferrari and go out for a cool Party."); } }
```

Over-Loading: Overloading is the action of defining multiple methods with the same name, but with different parameters. It is unrelated to either overriding or polymorphism. Functions in Java could be overloaded by two mechanisms ideally:

- Varying the number of arguments.

- Varying the Data Type.

```
class CalculateArea{       void Area(int
length){System.out.println(length*2);}       void Area(int length
, int width){System.out.println(length*width);}        public
static void main(String args[]){       CalculateArea obj=new
CalculateArea();       obj.Area(10);   // Area of a Square
obj.Area(20,20);  // Area of a Rectangle        } }
```

7. What is Constructors, Constructor Overloading in Java and Copy-Constructor?

Constructors form the basics of OOPs, for starters.

Constructor: The sole purpose of having Constructors is to create an instance of a class. They are invoked while creating an object of a class. Here are a few salient features of Java Constructors:

- Constructors can be public, private, or protected.
- If a constructor with arguments has been defined in a class, you can no longer use a default no-argument constructor – you have to write one.
- They are called only once when the class is being instantiated.
- They must have the same name as the class itself.
- They do not return a value and you do not have to specify the keyword void.
- If you do not create a constructor for the class, Java helps you by using a so called default no-argument constructor.

```
public class Boss{      String name;        Boss(String input) {
//This is the constructor         name = "Our Boss is also known
as : " + input;     }  public static void main(String args[]) {
```

```
Boss p1 = new Boss("Super-Man");     } }
```

Constructor overloading: passing different number and type of variables as arguments all of which are private variables of the class. Example snippet could be as follows:

```
public class Boss{    String name;     Boss(String input) {
//This is the constructor         name = "Our Boss is also known
as : " + input;    }    Boss() {          name = "Our Boss is a
nice man. We don't call him names.";    } public static void
main(String args[]) {       Boss p1 = new Boss("Super-Man");
Boss p2 = new Boss();    } }
```

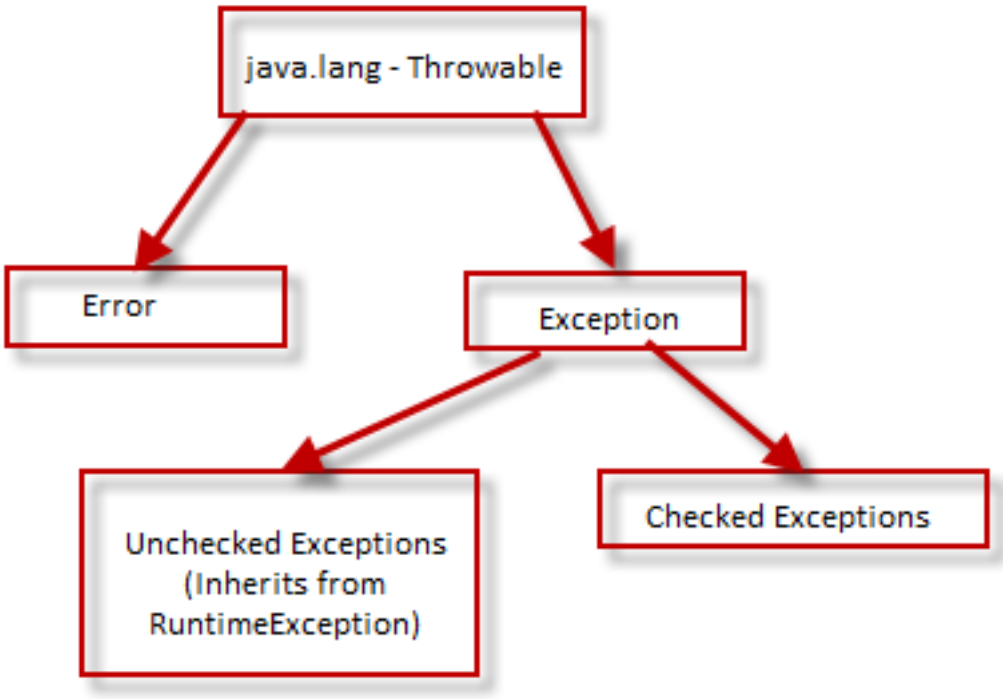
Copy Constructor: A copy constructor is a type of constructor which constructs the object of the class from another object of the same class. The copy constructor accepts a reference to its own class as a parameter.

Note: Java Doesn't support Copy Constructor. Nevertheless folks fromC/C++ background often get confused when asked about Java Copy Constructors.

8. What is Java Exception Handling? What is the difference between Errors, Unchecked Exception and Checked Exception?

Anything that's not Normal is an exception. Exceptions are the customary way in Java to indicate to a calling method that an abnormal condition has occurred.

In Java, exceptions are objects. When you throw an exception, you throw an object. You can't throw just any object as an exception, however — only those objects whose classes descend from Throwable. Throwable serves as the base class for an entire family of classes, declared in java.lang, that your program can instantiate and throw. Here's a hierarchical Exception class structure:



- An **Unchecked Exception** inherits from RuntimeException (which extends from Exception). The JVM treats RuntimeException differently as there is no requirement for the application-code to deal with them explicitly.
- A **Checked Exception** inherits from the Exception-class. The client code has to handle the checked exceptions either in a try-catch clause or has to be thrown for the Super class to catch the same. A Checked Exception thrown by a lower class (sub-class)

enforces a contract on the invoking class (super-class) to catch or throw it.
- **Errors (members of the Error family)** are usually thrown for more serious problems, such as OutOfMemoryError (OOM), that may not be so easy to handle.

Exception handling needs special attention while designing large applications. So we would suggest you to spend some time **brushing up your Java skills**.

9. What is the difference between Throw and Throws in Java Exception Handling (remember this queston?)

Throws: A throws clause lists the types of exceptions that a method might throw, thereby warning the invoking method – 'Dude. You need to handle this list of exceptions I might throw.' Except those of type Error or RuntimeException, all other Exceptions or any of their subclasses, must be declared in the throws clause, if the method in question doesn't implement a try...catch block. It is therefore the onus of the next-on-top method to take care of the mess.

```
public void myMethod() throws PRException {..}  This means the
super function calling the function should be equipped to handle
this exception. public void Callee() { try{    myMethod();
}catch(PRException ex) { ...handle Exception.... } }
```

Using the Throw: If the user wants to throw an explicit Exception, often customized, we use the Throw. The Throw clause can be used in any part of code where you feel a specific exception needs to be thrown to the calling method.

```
try{ if(age>100){throw new AgeBarException(); //Customized
ExceptioN }else{ ....} } }catch(AgeBarException ex){ ...handle
Exception..... }
```

10. What is the Difference between byte stream and Character streams?

Every **Java Programmer** deals with File Operations. To generate User reports, send attachments through mails and spill out data files from Java programs. And a sound knowledge on File Operation becomes even more important while dealing with Java questions.

byte stream : For reading and writing binary data, byte stream is incorporated. Programs use byte streams to perform byte input and output.

- Performing InputStream operations or OutputStream operations means generally having a loop that reads the input stream and writes the output stream one byte at a time.
- You can use buffered I/O streams for an overhead reduction (overhead generated by each such request often triggers disk access, network activity, or some other operation that is relatively expensive).

Character streams: Character streams work with the characters rather than the byte. In Java, characters are stored by following the Unicode (allows a unique number for every character) conventions. In such kind of storage, characters become the platform independent, program independent, language independent.

11. What are FileInputStream and FileOutputStream ? Explain with an example to read and write into files.

FileInputStream : It contains the input byte from a file and implements an input stream.

FileOutputStream : It uses for writing data to a file and also implements an output stream.

```
public class FileHandling { public static void main(String [ ] args) throws IOException { FileInputStream inputStream = new FileInputStream ("Input.txt") ; FileOutputStream outputStream = new FileOutputStream("Output.txt",true) ;  byte[] buffer = new byte[1024]; //For larger files we specify a buffer size which defines the chunks size for data int bytesRead; while ((bytesRead = inputStream.read(buffer)) != -1) outputStream.write(buffer, 0, bytesRead); inputStream.close() ; outputStream.close() ; } }
```

12. What are FileReader and FileWriter ? Explain with an example to read and write into files.

FileReader : The FileReader class makes it possible to read the contents of a file as a stream of characters. It works much like the FileInputStream, except the FileInputStream reads bytes, whereas the FileReader reads characters. The FileReader is intended to read text, in other words. One character may correspond to one or more bytes depending on the character encoding scheme.The FileReader object also lets web applications asynchronously read the contents of files (or raw data buffers) stored on the user's computer, using File or Blob objects to specify the file or data to read.

FileWriter : This class is used to write to character files. Creation of a FileWriter is not dependent on the file already existing. FileWriter will create the file before opening it for output when you create the object. Its write() methods allow you to write character(s) or Strings to a file. FileWriters are usually wrapped by higher-level Writer objects such as BufferedWriters or PrintWriters, which provide better performance and higher-level, more flexible methods to write data.

Usage of FileWriter can be explained as follows :

```
File file = new File("fileWrite2.txt"); FileWriter fw = new
FileWriter(file); for(int i=0;i<10;i++){ fw.write("Soham is Just
Awesome : "+i); fw.flush(); } fw.close();
```

Usage of FileWriter and FileReader used in conjunction is as follows:

```
int c; FileReader fread = new FileReader("xanadu.txt");
FileWriter fwrite = new FileWriter("characteroutput.txt"); while
((c = fread.read()) != -1)        fwrite.write(c);
```

13. What is the difference between ArrayList and LinkedList ?

Please pay special attention as this is probably one of the most widely asked interview questions.

We aren't going to state the properties of each in this question. What we are looking for are the differences. The prime areas where the two stand apart are as follows :

Arraylist	**Linklist**
Random access.	Sequential access.The control traverses from the first node to reach the indexed node.
Only objects can be added.	The LinkedList is implemented using nodes linked to each other. Each node contains a previous node link, next node link, and value,

which contains the actual data

14. Explain the difference between ITERATOR AND ENUMERATION INTERFACE with example.

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Iterator actually adds one method that Enumeration doesn't have: remove ().

```
public class IteratorAndEnumeration {
    public static void main(String[] args) {
        List<String> linkList=new LinkedList<String>();
        linkList.add("Hello1");
        linkList.add("Hello2");
        linkList.add("Hello3");
        linkList.add("Hello4");
        linkList.add("Hello5");
        linkList.add("Hello6");

        //Iterator : dynamic removal of elements from the list.
        Iterator<String> iterList=linkList.iterator();
        while(iterList.hasNext())
        {
            if(((String)iterList.next()).equals("Hello3")){
                iterList.remove(); //Allowed
            }
        }
```

Whereas in case of Enumeration:

```
/*
 * Enumeration : dynamic removal of elements from the
 * list not present.Enumeration with non-synchronized
 * concrete classes like LinkedList or ArrayList.
 */

Enumeration<String> enumList=Collections.enumeration(linkLis
while(enumList.hasMoreElements())
{
    System.out.println((String)enumList.nextElement());
}
```

15. What is the use of the 'SimpleDateFormat' and how can you use it to display the current system date in 'yyyy/MM/DD HH:mm:ss' format?

SimpleDateFormat is one such concrete class which is widely used by Java developers for parsing and formatting of dates. This is also used to convert Dates to String and vice-versa.

Literally every Enterprise level Java Application invariably uses the SimpleDateFormat for handling user dates. We ofcourse aren't expecting Java interviewees to be absolutely spectacular with the syntaxes. But a basic know-how of this class is mandatory.

```
public class CurrentSystemDate { public static void main(String[]
args) {         SimpleDateFormat sysForm = new
SimpleDateFormat("yyyy/MM/DD HH:mm:ss");         Date curdate=
new Date();         System.out.println(sysForm.format(curdate));
```

```
} }
```

The best way to brush up your Java knowledge is to **open up an eclipse** and write loads of codes and sample programs. Get into the habit of remembering syntaxes and applying the best coding standards possible.

And there's your top 15! Best of luck on your interview.