Over 500 Misc Java Interview Questions:

Basic OOPS questions, remember PIEA

1. What is POLYMORPHISM?

- **POLYMORPHISM** is the ability of an object to take on many forms. The most common use of **POLYMORPHISM** in OOP occurs when a parent class reference is used to refer to a child class object.
- Example of public Overriding or Runtime **POLYMORPHISM**

2. What is INHERITANCE?

- It is the process where one object acquires the properties of another. With the use of **INHERITANCE** the information is made manageable in a hierarchical order.

3. What is ENCAPSULATION ?

- It is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. Therefore **ENCAPSULATION** is also referred to as data hiding.

4. What is the primary benefit of ENCAPSULATION ?

- The main benefit of **ENCAPSULATION** is the ability to modify our implemented code without breaking the code of others who use our code. With this **ENCAPSULATION** gives maintainability, flexibility and extensibility to our code.

5. What is ABSTRACTION?

- ability to make a class abstract in OOP. It helps to reduce the complexity and also improves the maintainability of the system.

6. What is ABSTRACT CLASS

- These classes cannot be instantiated
- are either partially implemented or
- not at all implemented.
- contains one or more **ABSTRACT METHODS**, which are simply method declarations without a body.

7. When ABSTRACT METHODS are used?

- If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as abstract.

8. What does the following Java program print?

```
public class Test {
    public static void main(String[] args) {
        System.out.println(Math.min(Double.MIN_VALUE, 0.0d));
    }
}
```

- **Integer**, where `MIN_VALUE` is negative,
- **`Double` class** both the `MAX_VALUE` and `MIN_VALUE` are positive numbers.
- The `Double.MIN_VALUE is 2^(-1074)`, a double constant whose magnitude is the least among all double values.
- So unlike the obvious answer, this program will print 0.0 because Double.MIN_VALUE is greater than 0. I have asked this question to Java developer having experience up to 3 to 5 years and surprisingly almost 70% candidate got it wrong.
- Java has wrapper classes for primitive types (Byte, Short, Character, Integer, Long, Float, and Double)
  - SIZE
  - MIN_VALUE
  - MAX_VALUE
  - Short.TYPE, Short.SIZE, Short.MIN_VALUE, Short.MAX_VALUE)
  - Byte.TYPE, Byte.SIZE, Byte.MIN_VALUE, Byte.MAX_VALUE);
  - Character.TYPE, Character.SIZE, (int) Character.MIN_VALUE, (int) Character.MAX_VALUE);
  - Integer.TYPE, Integer.SIZE, Integer.MIN_VALUE, Integer.MAX_VALUE);
  - Long.TYPE, Long.SIZE, Long.MIN_VALUE, Long.MAX_VALUE);
  - Float.TYPE, Float.SIZE, Float.MIN_VALUE, Float.MAX_VALUE);
  - Double.TYPE, Double.SIZE, Double.MIN_VALUE, Double.MAX_VALUE);

```
type:byte   size:8  min:-128                 max:127
type:short  size:16 min:-32768               max:32767
type:char   size:16 min:0                    max:65535
type:int    size:32 min:-2147483648          max:2147483647
type:long   size:64 min:-9223372036854775808 max:9223372036854775807
type:float  size:32 min:1.4E-45              max:3.4028235E38
type:double size:64 min:4.9E-324             max:1.7976931348623157E308
```

Double has one bit reserved for the sign. Which makes these wrappers seem backwards for Doubles

double.max_value is the smallest possible number and

double.min_value is the largest number

9. What will happen if you put return statement or System.exit () on try or catch block? Will finally block execute?

- This is a very popular tricky Java question and it's tricky because many programmers think that no matter what, but the finally block will always execute EXCEPT with System.exit(). This question challenge that concept by putting a return statement in the try or catch block or calling `System.exit()` from try or catch block. Answer of this tricky question in Java is that

Try block
    return
    finally block will execute
catch block
    return
    finally block will execute

Try block
    `System.exit()`
    finally block will NOT execute
catch block
    `System.exit()`
    finally block will NOT execute
    catch block but finally block won't run if you call `System.exit()` from try or catch block.

10. What do the expression 1.0 / 0.0 will return? will it throw Exception? any compile time error?

- This is another tricky question from Double class. Though Java developer knows about the
- **Double primitive data type**: The double data type is a double-precision 64-bit.. For decimal values, this data type is generally the default choice. This data type should never be used for precise values, such as currency.
- **Double class**, while doing floating point arithmetic they don't pay enough attention to `Double.INFINITY`, NaN, and `-0.0` and other rules that govern the arithmetic calculations involving them.
- The simple answer to this question is that it will not throw `ArithmeticExcpetion` and return `Double.INFINITY`.
- Also, note that the comparison ~~`x == Double.NaN` always evaluates to false,~~ even if x itself is a NaN. To test if x is a NaN, one should use the method call `Double.isNaN(x)` to check if given number is `NaN` or not. If you know SQL, this is very close to NULL there.

11. What do you know about Java?

- Java is a high-level programming language originally developed by Sun Microsystems and released in 1995.
- Object Oriented,
- Platform Independent,
- Robust,
- Interpreted,
- Multi-threaded

12. Why is Java Architectural Neutral?

- its compiler generates an architecture-neutral object file format, which makes the compiled code to be executable on many processors, with the presence of Java runtime system.

13. How Java enabled High Performance?

- Java uses Just-In-Time compiler to enable high performance. Just-In-Time compiler is a program that turns Java bytecode, which is a program that contains instructions that must be interpreted into instructions that can be sent directly to the processor.

14. Why Java is considered dynamic?

- It is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

15. What is Java Virtual Machine and how it is considered in context of Java's platform independent feature?

- When Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code.
- This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.

16. List two Java IDE's?

- NetBeans, Eclipse, etc.

17. List some Java keywords(unlike C, C++ keywords)?

- Some Java keywords are import, super, finally, etc.

18. What do you mean by Object?

- Object is a runtime entity and its state is stored in fields and behavior is shown via methods.

19. Define class?

- A class is a blue print from which individual objects are created. A class can contain fields and methods to describe the behavior of an object.

20. What kind of variables a class can consist of?

- Local,
- instance and
- class.

21. What do you mean by Access Modifier?

- access levels for classes, variables, methods and **CONSTRUCTOR**s.
- Default package or default accessibility when no accessibility modifier is specified.

22. What is protected access modifier?

- Variables, methods and **CONSTRUCTOR**s that are declared protected in a superclass can be accessed only by the
  - subclasses in other package or
  - any class within the package of the protected members' class.

23. List the three steps for creating an Object for a class?

- An Object is

1. declared,
2. instantiated
3. initialized.

24. What is the Java default value of

- byte datatype = 0.
- float = 0.0f
- double = 0.0d
- different as compared to C/C++

25. When a byte datatype is used?

- This data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.

26. What is a static variable?

- Class variables also known as static variables
- declared with the static keyword in a class,
- however, outside a method, **CONSTRUCTOR** or a block.

27. According to Java Operator precedence, which operator is considered to be with highest precedence?

- Postfix operators i.e. () [] . is at the highest precedence.

28. Variables used in a switch statement can be used with which datatypes?

- Variables used in a switch statement can only be a byte, short, int, or char.

29. When parseInt() method can be used?

- This method is used to get the primitive data type of a certain String.

30. Which package is used for pattern matching with regular expressions?

- java.util.regex package is used for this purpose.

31. java.util.regex consists of which classes?

- java.util.regex consists of three classes: Pattern class, Matcher class and PatternSyntaxException class.

32. What is finalize() method?

- It is possible to define a method that will be called just before an object's final destruction by the garbage collector. This method is called finalize( ), and it can be used to ensure that an object terminates cleanly.

33. When super keyword is used?

- If the method overrides one of its superclass's methods, overridden method (parent method) can be invoked through the use of the keyword super. It can be also used to refer to a hidden field

34. Define Packages in Java?

- A Package can be defined as a grouping of related types(classes, **INTERFACES**, enumerations and annotations) providing access protection and name space management.

35. Why Packages are used?

- Packages are used in Java in-order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, **INTERFACES**, enumerations and annotations, etc., easier.

36. Define IMMUTABLE object?

- An **IMMUTABLE** object can't be changed once it is created.

37. Explain Set INTERFACE ?

- It is a collection of element, which cannot contain duplicate elements. The Set **INTERFACE** contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

38. Explain TreeSet?

- It is a Set implemented when we want elements in a sorted order.

39. Explain the following line used under Java Program:

- public static void main (String args[ ])
- The following shows the explanation individually:
  - public: it is the access specifier.
  - static: it allows main() to be called without instantiating a particular instance of a class.
  - void: it affirms the compiler that no value is returned by main().
  - main(): this method is called at the beginning of a Java program.
  - String args[ ]: args parameter is an instance array of class String

40. Define JRE i.e. Java Runtime Environment?

- Java Runtime Environment is an implementation of the Java Virtual Machine, which executes Java programs. It provides the minimum requirements for executing a Java application;

41. What is JAR file?

- JAR files are Java Archive files and it aggregates many files into one. It holds Java classes in a library. JAR files are built on ZIP file format and have .jar file extension.

42. What is a WAR file?

- This is Web Archive File and used to store XML, java classes and JavaServer pages. Which is used to distribute a collection of JavaServer Pages, Java Servlets, Java classes, XML files, static Web pages etc.?

43. Define JIT compiler?

- It improves the runtime performance of computer programs based on bytecode.

44. What is the difference between object oriented programming language and object based programming language?

- Object based programming languages follow all the features of OOPs except **INHERITANCE**.
- JavaScript is an example of object based programming languages

45. What is static block?

- It is used to initialize the static datamember; It is executed before main method at the time of classloading.

46. Define composition?

- Holding the reference of the other class within some other class is known as composition.

47. static binding in Java. occurs during compile time compiler bind method call to actual method

- fast because they are bonded during compile time and no check or binding is required during runtime.
- remember return type is not part of method signature
- most popular example of method overloading is `System.out.println()System.out.println()` method which is overloaded to accept all kinds of data types in Java.

48. Does Java support multiple INHERITANCES?

- This is the trickiest question in Java if C++ can support direct multiple **INHERITANCES** than why not Java is the argument Interviewer often give. Answer of this question is much more subtle then it looks like, because
- Java does support multiple **INHERITANCES** of Type by allowing an **INTERFACE** to extend other **INTERFACES**,
- Java doesn't support is multiple **INHERITANCES** of implementation.

49. Why Java doesn't support multiple INHERITANCE

- First reason is ambiguity around Diamond problem
- Second and more convincing reason to me is that multiple **INHERITANCES** does complicate the design and creates problem during casting, **CONSTRUCTOR** chaining etc

50. Does Java 8 support multiple INHERITANCE use default methods

- In Java 8, we can realize the concept of **multiple INHERITANCE through use of default methods**..
- default methods in **INTERFACES** are methods which will be invoked by default – if not overridden in implementing classes.
- Default methods enable you to add new functionality to the **INTERFACES** and ensure backward compatibility for existing classes which implement that **INTERFACE** . That means programmers who use the **INTERFACES** don't have to go back and rewrite the implementations.
- (Remember that an **ABSTRACT METHOD** is a method declared without an implementation.) With default method you can **add an implementation** and still ensure backward compatibility.

- `Moveable` **INTERFACE** is some existing **INTERFACE** and wants to add a new method `moveFast()`.
  - If it adds moveFast() method using old technique, then all classes implemeting Moveable will also be changed.
  - So, let's add moveFast() method as default method.

```
public INTERFACE Moveable
{
    default void moveFast()
    {
        System.out.println("I am moving fast, buddy !!");
    }
}
```

- If all classes implementing `Moveable` **INTERFACE** do not need change themselves (until some class specifically wants to override `moveFast()` method to add custom logic). All classes can directly call `instance.moveFast()` method. http://howtodoinjava.com/object-oriented/multiple-**INHERITANCE**-in-java/

```
public class Animal implements Moveable
{
    public static void main(String[] args)
    {
        Animal tiger = new Animal();

        //Call default method using instance reference
        tiger.moveFast();
    }
}
```

- All method declarations in an **INTERFACE**, including default methods, are implicitly `public`, so you can omit the `public` modifier.

51. What does the following Java program print?

```
public class Test {
    public static void main(String[] args) throws Exception {
        char[] characterArray = new char[] {'\u0097'}; // character array
        String str = new String(characterArray);        // Unicode characters
        byte[] byteArray = str.getBytes();         // Unicode characters
        System.out.println(Arrays.toString(byteArray));
    }
}
```

- character encoding and how String to byte array conversion works.
- char[] chars = new char[] {'\u0097'}; // character array first creating a String from a character array, which just has one character `'\u0097'`,
- after that we are getting the byte array from that String and printing that byte.
- Since `\u0097` is within the 8-bit range of byte primitive type, it is reasonable to guess that the `str.getBytes()` call will return a byte array that contains one element with a value of -105 `((byte) 0x97)`.
- However, that's not what the program prints and that's why this question is tricky. As a matter of fact, the output of the program is **operating system and locale dependent**. On a Windows XP with the US locale, the above program prints [63], if you run this program on Linux or Solaris, you will get different values.
- To answer this question correctly, you need to know about how Unicode characters are represented in Java char values and in Java strings, and what role character encoding plays in `String.getBytes()`.
- convert a string to a byte array, Java iterate through **all the characters** that the string represents and turn each one into a number of bytes and finally put the bytes together. The rule that maps each Unicode character into a byte array is called a character encoding. So It's possible that if same character encoding is not used during both encoding and decoding then retrieved value may not be correct. When we call `str.getBytes()` without specifying a character encoding scheme, the JVM uses the default character encoding of the platform to do the job.
- The default encoding scheme is operating system and **locale dependent**.
  - Linux, it is `UTF-8`
  - Windows with a US locale, the default encoding is `Cp1252`.
- This explains the output we get from running this program on Windows machines with a US locale. No matter which character encoding scheme is used, Java will always translate Unicode characters not recognized by the encoding to 63, which represents the character U+003F (the question mark, ?) in all encodings.

-

52. What is the issue with following implementation of compareTo() method in Java

```
public int compareTo(Object o){
   Employee emp = (Employee) o;
   return this.id - e.id;
}
```

- where an id is an integer number.
- Well, three is nothing wrong in this Java question until you guarantee that id is always positive. This Java question becomes tricky when you **can't guarantee that id is positive or negative. the tricky** part is, If id becomes negative than subtraction may overflow and produce an incorrect result.

53. How do you ensure that N thread can access N resources without deadlock?

- if you acquire resources in a particular order and release resources in the reverse order you can prevent deadlock.

54. What is difference between CyclicBarrier and CountDownLatch in Java

- from Java 5. The main difference between both of them is that you can reuse CyclicBarrier even if Barrier is broken, but you **can not** reuse CountDownLatch in Java.

55. What is the difference between StringBuffer and StringBuilder in Java?

- **StringBuffer** methods e.g. length(), capacity() or append() are **SYNCHRONIZED**
- StringBuilder are not synchronized.
- **concatenation** of String using **StringBuilder** is faster than

56. Can you access a non-static variable in the static context?

- No, you can not access a non-static variable from the static context in Java.
- If you try, it will give compile time error. This is actually a common problem beginner in Java face when they try to access instance variable inside the main method. Because main is static in Java, and instance variables are non-static, you can not access instance variable inside main.

http://www.java67.com/2012/09/top-10-tricky-java-interview-questions-answers.html#ixzz4H0IRWbGL

57. What is a Class? Object?

- Remember class is a blueprint
- objects are real instances

58. Difference between extends Thread vs implements Runnable in Java?

- Can only extend one class in Java, which means if you extend Thread class you lose your opportunity to extend another class,
- implement Runnable, you can still extend another class.

59. Difference between Runnable and Callable INTERFACE in Java?

- Old - Runnable run() was the only way to implement a task before JDK 1.5 adds
- Callable also defines a single call() method it can return values and throw exceptions.

60. Difference between ArrayList and LinkedList in Java?

- ArrayList - provides random search if you know the index,
- LinkedList is just collection of nodes, sequential search **adding and removing element from middle is efficient** because it only require to modify links and no other element is rearranged.

61. Difference between Association, Composition and Aggregation?

- composition is strongest.
- aggregation - If part can exists without whole than relationship between two class
- composition - if part cannot exists without whole than relationship between two class
- Between **INHERITANCE** and composition, later provides more flexible design.

62. What is difference between FileInputStream and FileReader in Java?

- FileInputStream - used to read binary data
- FileReader - used to read text data, which means later also consider character encoding while converting bytes to text in Java.

63. Why wait and notify methods are declared in Object class?

- owned by object not thread
- that's why it make sense to keep those method on java.lang.Object class.

64. What is difference between Iterator and Enumeration in Java?

- Iterator replaces Enumeration.
- Iterator allows you to remove elements from collection while traversing
- Iterator methods `hasNext()` and `next()` are also more concise then

65. Difference between static and dynamic binding in Java?

- static binding is related to overloaded method, compile time
- dynamic binding is related to overridden method, runtime
- private, final and static are resolved using static binding at compile time
- but virtual methods which can be overridden are resolved using dynamic binding at runtime.

66. Virtual Method

- Overridden method, **POLYMORPHISM**
- An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.
- Ex at compile time uses methodx() in Class A and runntime overriden and uses methodx() in Class B.

67. Difference between Comparator and Comparable in Java?

- Comparator
  - Diff - can create multiple Comparator to define multiple sorting order based upon different attribute of object.
  - Diff - to implement Comparable you must have access of the class or code, but you can use Comparator without having source code of class, all you need is the JAR file of particular object.
  - Thus, Powerful to implement custom sorting order
- Comparable
  - core classes in Java e.g. String, Integer implements Comparable to define their natural sorting order
  - if you define a value class or a domain object then you should also implement Comparable and define natural ordering of your object.

68. How do you sort ArrayList in descending order?

- You can use `Collections.sort()` method with reverse Comparator, which can sort elements in the reverse order of their natural order e.g.

```
List<String> listOfString = Arrays.asList("London", "Tokyo", "NewYork");
Collections.sort(listOfString, Collections.reverseOrder());
```

69. What is difference between PATH and CLASSPATH in Java?

- PATH environment variable points to Java binary which is used to run Java programs.
- CLASSPATH points to Java class files or JAR files. If a class is not found in CLASSPATH then Java throws ClassNotFoundException.

70. What is difference between Checked and Unchecked Exception in Java?

- Checked
  - ensures that handling of exception is provided and its verified by compiler also,
  - Requires special provision is needed e.g. throws clause.
- unchecked exception
  - can throw unchecked exception without any throw clause.

October 3, 2015 by Kasia Mikoluk

71. Rookie - What is JVM? Why is Java called the 'Platform Independent Programming Language'?

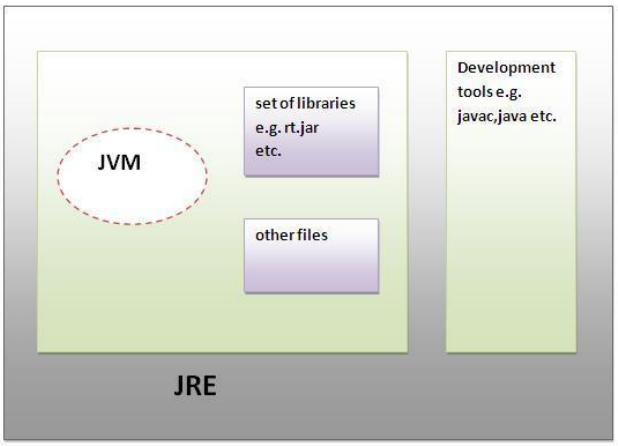
- JVM, or the Java Virtual Machine, is an interpreter which accepts 'Bytecode' and executes it.
- 'compile once, run everywhere'.
- The Java Compiler outputs Non-Executable Codes called 'Bytecode'.
- need is a JVM designed for that particular platform.
- Execute bytecode on machine using JVM.

72. Difference between JDK, JRE and JVM

- JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
- JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.
- The JVM performs following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

- JRE
  - JRE is an acronym for Java Runtime Environment.It is used to provide runtime environment.It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.
  - Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.
- JDK
  - JDK is an acronym for Java Development Kit.It physically exists.It contains JRE + development tools.



73. Rookie - What is the Difference between JDK and JRE?

- The "JDK" is the Java Development Kit. I.e., JDK is bundle of software that you can use to develop Java based software. **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**.
- "JRE" is the Java Runtime Environment. I.e., JVM which actually executes Java programs.

74. Types of Java Applications

- There are mainly 4 type of applications that can be created using java programming:

Standalone Application

- It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

Web Application

- An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

Enterprise Application

- An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

Mobile Application

- An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

75. Rookie - What does the 'static' keyword mean?

- 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

- The static keyword in java is used for memory management mainly.
- can apply java static keyword with variables, methods, blocks and nested class.
- The static keyword belongs to the class than instance of the class.
- The static can be:
    - variable (also known as class variable)
    - method (also known as class method)
    - block
    - nested class

76. What are the Data Types supported by Java? - Rookie

- The eight Primitive Data types supported by Java are:
- **Byte :** 8-bit signed two's complement integer. -128 127 (inclusive)
- **Short :** 16-bit signed min -32,768 maxi 32,767 (inclusive).
- **Int :** 32-bit signed min -2,147,483,648 max 2,147,483,647 (inclusive)
- **Long :** 64-bit signed min -9,223,372,036,854,775,808 max 9,223,372,036,854,775,807 (inclusive)
- Float
- Double

77. What is Autoboxing and Unboxing? - Rookie

- **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.
- int object = Integer, float object = Float, Double object = double
- **Unboxing**: The automatic transformation of wrapper types into their primitive equivalent is known as Unboxing.

78. What is the difference between STRINGBUFFER and STRING? - Rookie

- **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.

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

- **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.

80. 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];
        int bytesRead;
        while ((bytesRead = inputStream.read(buffer)) != -1)
            outputStream.write(buffer, 0, bytesRead);
        inputStream.close() ;
        outputStream.close() ;
    }
}
```

81. 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);
```

82. 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));
   }
}
```

Most Common, Top 25 Java Interview questions

83. Can you write IMMUTABLE object?

- **IMMUTABLE** classes are Java classes whose objects can not be modified once created.
- Any modification in **IMMUTABLE** object result in new object.
- For example String is **IMMUTABLE** in Java.
- Most IMMUTABLEs are also final in Java, in order to prevent sub class from overriding methods in Java which can compromise Immutability. You can achieve same functionality by making member as non final but private and not modifying them except in **CONSTRUCTOR**.

84. Length of the String without using java built in length method : Java code with example

The length of the string is easy to define , it is the number of characters in the string including white spaces .

Java make the things easy as it provides the built in length method for the String class (java.lang.String) which is a final class .

for example :

String demo = " This is java built in method example "

int length= demo.length();

But what happens , if we don't want to use the java built in method to calculate the length of the string .

Then there is also a way to calculate the length of the string

Logic:

Initialize two variables and keep the condition of the loop always true. Iterate the whole string using one variable ,while use other variable as counter to count the length of the string .When the string completes , then String.charAt() will throw an exception as array index out of range exception , which is caught in the catch block.

So in the below example , there are two variables i and c , here i is used to iterate through the string while c is used as a counter to calculate the length of the string .

Please note that Array index out of bounds and array index out of range are different exceptions

And the other point to be learn in the code is that we can not put the return statement in the catch block of the try catch block.

We need to put the return statement after the catch block completes.

```
public class StringLength
{
    static int i,c,res;

    static int length(String s)
    {
        try
        {
            for(i=0,c=0;0<=i;i++,c++)
            s.charAt(i);
        }
        catch(Exception e)
        //Array index out of bounds and array index out of range are different exceptions
        {
            System.out.print("length is ");
            // we can not put return  statement in catch
        }
        return c;
    }

    public static void main (String args[])
    {
```

```
        System.out.println("Original String is : ");
        System.out.println("Alive is awesome ");
        res=StringLength.length("Alive is awesome ");
        System.out.println( res);
    }
```

85. How do you handle error condition while writing stored procedure or accessing stored procedure from java?

- This is one of the tough Java interview question and its open for all, my friend didn't know the answer so he didn't mind telling me. my take is that stored procedure should return error code if some operation fails but if stored procedure itself fail than catching SQLException is only choice.

86. What is difference between Executor.submit() and Executer.execute() method ?

- There is a difference when looking at exception handling. If your tasks throws an exception and if it was submitted with execute this exception will go to the uncaught exception handler (when you don't have provided one explicitly, the default one will just print the stack trace to System.err). If you submitted the task with submit any thrown exception, checked exception or not, is then part of the task's return status. For a task that was submitted with submit and that terminates with an exception, the Future.get will re-throw this exception, wrapped in an ExecutionException.
- that genre.

87. Let's first discuss about Memory Barrier which are the base for our further discussions.

- There are two type of memory barrier instructions in JMM
  - - read barriers
- A read barrier invalidates the local memory (cache, registers, etc) and then reads the contents from the main memory, so that changes made by other threads becomes visible to the current Thread.
  - - write barrier.
- A write barrier flushes out the contents of the processor's local memory to the main memory, so that changes made by the current Thread becomes visible to the other threads.

88. What will happen if you call return statement or System.exit on try or catch block ? will finally block execute?

- This is a very *popular tricky Java question* and its tricky because many programmer think t**hat finally block always executed.** This question challenge that concept by putting return statement in try or catch block or calling System.exit from try or catch block. Answer of this tricky question in
- Java is that finally block will execute even if you put return statement in try block or catch block but finally block won't run if you call System.exit form try or catch.

89. What is Retrieving Class Objects

reflection operations invoke appropriate methods on `Class`. There are several ways to get a `Class` depending on whether the code has access to an object, the name of class, a type, or an existing `Class`.

90. What is Object.getClass()

If an instance of an object is available, then the simplest way to get its `Class` is to invoke `Object.getClass()`. Of course, this only works for reference types which all inherit from `Object`. Some examples follow.

```
Class c = "foo".getClass();
Returns the Class for String
Class c = System.console().getClass();
```

Since arrays are `Objects`, it is also possible to invoke `getClass()` on an instance of an array. The returned `Class` corresponds to an array with component type `byte`.

```
import java.util.HashSet;
import java.util.Set;
Set<String> s = new HashSet<String>();
Class c = s.getClass();
```

91. What is The .class Syntax

If the type is available but there is no instance then it is possible to obtain a `Class` by appending `".class"` to the name of the type. This is also the easiest way to obtain the `Class` for a primitive type.

```
boolean b;
Class c = b.getClass();   // compile-time error
Class c = boolean.class;  // correct
```

Note that the statement `boolean.getClass()` would produce a compile-time error because a `boolean` is a primitive type and cannot be dereferenced. The `.class` syntax returns the `Class` corresponding to the type `boolean`.

```
Class c = java.io.PrintStream.class;
The variable c will be the Class corresponding to the type java.io.PrintStream.
Class c = int[][][].class;
```

The `.class` syntax may be used to retrieve a `Class` corresponding to a multi-dimensional array of a given type.

92. What is Class.forName()

If the fully-qualified name of a class is available, it is possible to get the corresponding `Class` using the static method `Class.forName()`. This cannot be used for primitive types. The syntax for names of array classes is described by `Class.getName()`. This syntax is applicable to references and primitive types.

```
Class c = Class.forName("com.duke.MyLocaleServiceProvider");
```

This statement will create a class from the given fully-qualified name.

```
Class cDoubleArray = Class.forName("[D");
Class cStringArray = Class.forName("[[Ljava.lang.String;");
```

The variable `cDoubleArray` will contain the `Class` corresponding to an array of primitive type `double` (i.e. the same as `double[].class`). The `cStringArray` variable will contain the `Class` corresponding to a two-dimensional array of `String` (i.e. identical to `String[][].class`).

93. What is the TYPE Field for Primitive Type Wrappers

- The `.class` syntax is a more convenient and the preferred way to obtain the `Class` for a primitive type; however there is another way to acquire the `Class`. Each of the primitive types and `void` has a wrapper class in `java.lang` that is used for boxing of primitive types to reference types. Each wrapper class contains a field named `TYPE` which is equal to the `Class` for the primitive type being wrapped.
- Class c = Double.TYPE;
- There is a class `java.lang.Double` which is used to wrap the primitive type `double` whenever an `Object` is required. The value of `Double.TYPE` is identical to that of `double.class`.

```
Class c = Void.TYPE;
```

`Void.TYPE` is identical to void.class.

94. What are the Methods that Return Classes

There are several Reflection APIs which return classes but these may only be accessed if a `Class` has already been obtained either directly or indirectly.

```
Class.getSuperclass()
```

Returns the super class for the given class.

Class c = javax.swing.JButton.class.getSuperclass();

```
The super class of javax.swing.JButton is javax.swing.AbstractButton.
Class.getClasses()
```

Returns all the public classes, **INTERFACES**, and enums that are members of the class including inherited members.

Class<?>[] c = Character.class.getClasses();

```
Character contains two member classes Character.Subset and Character.UnicodeBlock.
Class.getDeclaredClasses()
```

Returns all of the classes **INTERFACES**, and enums that are explicitly declared in this class.

```
Class<?>[] c = Character.class.getDeclaredClasses();
Character contains two public member classes Character.Subset and Character.UnicodeBlock
and one private class Character.CharacterCache.
```

95. Provide an example and explaination of a Anonymous Class

Anonymous classes in Java are more accurately known as anonymous inner classes – there's no such thing as anonymous classes without the "inner". That distinction is important, because the fact that they are anonymous

inner classes means that they are defined inside another class.

An anonymous inner class is an inner class that is declared without using a class name at all – and that of course is why it's called an anonymous class. An anonymous inner class also has some pretty unusual syntax.

Anonymous inner class example:

```
class ProgrammerInterview  {
 public void read() {
  System.out.println("Programmer Interview!");
 }
}

class Website {
   /*  This creates an anonymous inner class: */
   ProgrammerInterview pInstance = new    ProgrammerInterview() {
      public void read() {
         System.out.println("anonymous ProgrammerInterview");
      }
   };
}
```

In the code above, you can see that we have two classes – one called Website and another called ProgrammerInterview. The ProgrammerInterview class is pretty straightforward – there's just a simple method called "read()" that prints the text "Programmer Interview!" when called.

It might look like we are creating an instance of the ProgrammerInterview class called pInstance in that code, but what's actually happening in that code is that an instance of an anonymous class is being created.

An anonymous inner class is a subclass

Pay special attention to the fact that inside the curly braces – after the "new ProgrammerInterview()" code – there is actually a method definition for a method named "read()". This certainly does not look like we are creating a normal instance of a class – because you don't normally see methods being defined at the same time that an instance of a class is created.

What's actually happening in the code above is that we are creating an instance of a subclass (also known as a child class) of the ProgrammerInterview class. And, the most important thing to understand here is that this instance (pInstance) is actually an instance of an anonymous subclass of the ProgrammerInterview class.

96. Why is it called an anonymous inner class?

- The reason it's called an anonymous inner class is because the class that we have created clearly has no name! We jump straight to creating an instance of the class, but we do not even give the class a name – all we have is a reference variable (pInstance, in our example above) for the anonymous inner class.

97. What is the issue with following implementation of compareTo() method in Java

```
public int compareTo(Object o){
   Employee emp = (Employee) emp;
   return this.id - o.id;
}
```

End of top 25 popular Java interview question

98. Can you access non static variable in static context?

- Another tricky Java question from Java fundamentals. No you can not access static variable in non static context in Java. Read why you can not access non-static variable from static method to learn more about this tricky Java questions.

99. Is 'abc' a primitive value?

The String literal 'abc' is not a primitive value. It is a String object .

100. What restrictions are placed on the values of each case of a switch statement?

During compilation, the values of each case of a switch statement must evaluate to a value that can be promoted to an int value.

101. Is a class a subclass of itself?

A class is a subclass of itself.

102. What is the difference between a while statement and a do statement?

A while statement checks at the beginning of a loop to see whether the next loop iteration should occur. A do statement checks at the end of a loop to see whether the next iteration of a loop should occur. The do statement will always execute the body of a loop at least once.

103. What modifiers can be used with a local inner class?

A local inner class may be final or abstract.

inner class **= Class defined within another class**

nested class = Class defined within another class **AND the modifier is Static**

104. What is the purpose of the File class?

- The File class is used to create objects that provide access to the files and directories of a local file system.

105. Can an exception be rethrown? Yes.

106. If a method is declared as protected, where may the method is accessed?

- A protected method may only be accessed by classes or **INTERFACES** of the same package or by subclasses of the class in which it is declared.

107. Which non-Unicode letter characters may be used as the first character of an identifier?

- The non-Unicode letter characters $ and _ may appear as the first character of an identifier $identifier or _identifier

108. What is casting?

- There are two types of casting, casting between primitive numeric types and casting between object references.. Casting between object references is used to refer to an object by a compatible class, **INTERFACE** or array type reference.

109. What is the return type of a program's main() method? -

- A program's main() method has a void return type.

110. What class of exceptions is generated by the Java run-time system?

- The Java runtime system generates RuntimeException and Error exceptions.

111. What class allows you to read objects directly from a stream?

- The ObjectInputStream class supports the reading of objects from input streams.

112. What is the difference between a field variable and a local variable?

- field variable = variable that is declared as a member of a class.
- local variable = variable that is declared local to a method.

113. What is "super" keyword in java?

- The super keyword in java is a reference variable that is used to refer parent class objects.
- Various scenarios of using java super Keyword:
    - super is used to refer immediate parent instance variable
    - super is used to call parent class method
    - super() is used to call immediate parent **CONSTRUCTOR**

114. What is the relationship between a

- method's throws clause and the
- exceptions that can be thrown during the method's execution?
- method's throws clause must declare any checked exceptions that are not caught within the body of the method.

115. What are the components of j2ee application?

- The components of j2ee application –
- • Servlet and JSP technology are web components
- • Business components (JavaBeans)
- • Resource adapter components
- • Application client's components.

116. Why are the methods of the Math class static?

- So they can be invoked as if they are a mathematical code library.

117. What are the legal operands of the instanceof operator?

- The left operand is an object reference or null value and the right operand is a class, **INTERFACE** or array type.
- instanceof Operator: used only for objectreference variables. The operator checks whether the obj is of a part type (class type or **INTERFACE** type) written as
- (Object ref var) instanceof (class/**INTERFACE** type)

```
ex.
String name = = "Dinesh";
boolean result = name instanceOf String;
```

- // This will return true since name is type of String

118. What an I/O filter?

- An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another.

119. If an object is garbage collected, can it become reachable again?

- Once an object is garbage collected, it ceases to exist. It can no longer become reachable again.

120. What are E and PI?

- E is the base of the natural logarithm and PI is mathematical value pi.

121. Are true and false keywords? NO

122. What is the difference between the File and RandomAccessFile classes?

- **File class** - encapsulates the files and directories of the local file system.
- **RandomAccessFile class** provides the methods needed to directly access data contained in any part of a file.

123. What happens when you add a double value to a String?

- The result is a String object.

124. What is your platform's default character encoding?

- Java on English Windows platforms, it is probably **Cp1252**.
- Java on English Solaris platforms, it is most likely 8859_1.

125. Which package is always imported by default? The java.lang

126. What INTERFACE must an object implement before it can be written to a stream as an object?

- An object must implement the Serializable or Externalizable **INTERFACE** before it can be written to a stream as an object.

127. How can my application get to know when an HttpSession is removed?

- Define a Class HttpSessionNotifier which implements HttpSessionBindingListener and implement the functionality what you need in valueUnbound() method. Create an instance of that class and put that instance in HttpSession.

128. What's the difference between notify() and notifyAll()?

- notify() is used to unblock one waiting thread;

- notifyAll() is used to unblock all of them.
- Using notify() is preferable (for efficiency) when only one blocked thread can benefit from the change (for example, when freeing a buffer back into a pool).
- notifyAll() is necessary (for correctness) if multiple threads should resume (for example, when releasing a

129. What is -XX:+UseCompressedOops JVM option? Why use it?

- When you go migrate your Java application from 32-bit to 64-bit JVM, the heap requirement suddenly increases, almost double, due to increasing size of ordinary object pointer from 32 bit to 64 bit. This also adversely affect how much data you can keep in CPU cache, which is much smaller than memory. Since main motivation for moving to 64-bit JVM is to specify large heap size, you can save some memory by using compressed OOP. By using -XX:+UseCompressedOops, JVM uses 32-bit OOP instead of 64-bit OOP.

130. What is the difference when String is gets created using literal or new() operator ?

- create string with new() its created in heap and **not** added into string pool while explicitly. It's only when you create String object as String literal e.g. `String s = "Test"`
- String created using literal are created in String pool itself which exists in Perm area of heap.

131. Why can't I say just abs() or sin() instead of Math.abs() and Math.sin()?

- The import statement does not bring methods into your local name space. It lets you abbreviate class names, but not get rid of them altogether.

132. Why are there no global variables in Java?

- Global variables are considered bad form for a variety of reasons:
- Adding state variables breaks referential transparency (you no longer can understand a statement or expression on its own: you need to understand it in the context of the settings of the global variables),
- State variables lessen the cohesion of a program: you need to know more to understand how something works.
- major point of Object-Oriented programming is to break up global state into more easily understood collections of local state, When you add one variable, you limit the use of your program to one instance.
- What you thought was global, someone else might think of as local: they may want to run two copies of your program at once. For these reasons, Java decided to ban global variables.

133. Why main method is public static in Java

- JVM search for **public static void main(String args[])** method in that class and if it doesn't find that method it throws error **NoSuchMethodError:main** and terminates.
- If main method were not declared static than JVM has to create instance of main Class and since **CONSTRUCTOR** can be overloaded and can have arguments there would not be any certain and consistent way for **JVM to find main method in Java**.
- The method is static because otherwise there would be **ambiguity**: which **CONSTRUCTOR** should be called?

134. What happens if you remove static modifier from the main method?

- Program compiles successfully . But at runtime throws an error "NoSuchMethodE

135. How are Observer and Observable used?

- Objects that subclass the Observable class maintain a list of observers.
- When an Observable object is updated it invokes the update() method of each of its observers to notify the observers that it has changed state. The Observer **INTERFACE** is implemented by objects that observe Observable objects.Bottom of Form

136. Can a lock be acquired on a class?

- Yes, a lock can be acquired on a class. This lock is acquired on the class Class object.

137. What state does a thread enter when it terminates its processing?

- enters the **dead state**.

138. How does Java handle integer overflows and underflows?

- uses those low order bytes of the result that can fit into the size of the type allowed by the operation.

139. What is the difference between the >> and >>> operators?

- >> operator carries the sign bit when shifting right
- >>> zero-fills bits that have been shifted out.

140. Is sizeof a keyword? -

- It's an operator not a keyword.

141. Does garbage collection guarantee that a program will not run out of memory?

- Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection

142. If you are extending ABSTRACT CLASS or implementing INTERFACE than you need to override all ABSTRACT METHOD unless your class is not abstract. ABSTRACT METHOD can only be used by using method overriding.

- Overriding ex

```
class Animal{
   public void move(){
 System.out.println("Animals can move");
class Dog extends Animal{
   public void move(){
 System.out.println("Dogs can walk and run");

public class TestDog{
   public static void main(String args[]){
 Animal a = new Animal(); // Animal reference and object
 Animal b = new Dog(); // Animal reference but Dog object
 a.move();// runs the method in Animal class
 b.move();//Runs the method in Dog class
```

- Animals can move
- Dogs can walk and run
- Thread finished executing

143. What is final class?

- Final classes are created so the methods implemented by that class cannot be overridden. It can't be inherited.

144. What is NullPointerException?

- A NullPointerException is thrown when calling the instance method of a null object, accessing or modifying the field of a null object etc.

145. What are the ways in which a thread can enter the waiting state?

- A thread can enter the waiting state by invoking its sleep() method, by blocking on IO, by unsuccessfully attempting to acquire an object's lock, or by invoking an object's wait() method. It can also enter the waiting state by invoking its (deprecated) suspend() method.

146. How does multi-threading take place on a computer with a single CPU?

- The operating system's task scheduler allocates execution time to multiple tasks. By quickly switching between executing tasks, it creates the impression that tasks execute sequentially.

147. What invokes a thread's run() method?

- After a thread is started, via its start() method of the Thread class, the JVM invokes the thread's run() method when the thread is initially executed.

148. Does it matter in what order catch statements for FileNotFoundException and IOException are written?

- Yes, it does. The FileNotFoundException is inherited from the IOException. Exception's subclasses have to be caught first.

149. What is the difference between yielding and sleeping?

- When a task invokes its yield() method, it returns to the ready state. When a task invokes its sleep() method, it returns to the waiting state.

150. How many bits are used to represent Unicode, ASCII, UTF-16, and UTF-8 characters?

- Unicode requires 16 bits
- ASCII requires 7 bits. usually represented as 8 bits.
- UTF-8 represents characters using 8, 16, and 18 bit
- UTF-16 uses 16-bit and larger

151. What are Wrapper classes?

- These are classes that allow primitive types to be accessed as objects. Example: Integer, Character, Double, Boolean etc.

152. What is the difference between a Window and a Frame?

- The Frame class extends Window to define a main application window that can have a menu bar.

153. Which package has lightweight components?

- javax.Swing package. All components in Swing, except JApplet, JDialog, JFrame and JWindow are lightweight components.

154. What is the difference between the paint() and repaint() methods?

- The paint() method supports painting via a Graphics object.
- The repaint() method is used to cause paint() to be invoked by the AWT painting thread.

155. What is the purpose of File class?

- It is used to create objects that provide access to the files and directories of a local file system.

156. What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

- Reader/Writer class hierarchy is character-oriented
- InputStream/OutputStream class hierarchy is byte-oriented.

157. Which class should you use to obtain design information about an object?

- The Class class is used to obtain information about an object's design and java.lang.Class class instance represent classes, **INTERFACES** in a running Java application.

158. What is the difference between static and non-static variables?

- A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.What is Serialization and deserialization?

159. What are use cases?

- It is part of the analysis of a program and describes a situation that a program might encounter and what behavior the program should exhibit in that circumstance.

160. Explain the use of subclass in a Java program?

- Sub class inherits all the public and protected methods and the implementation. It also inherits all the default modifier methods and their implementation.

161. How to add menushortcut to menu item?

- If there is a button instance called b1, you may add menu short cut by calling b1.setMnemonic('F'), so the user may be able to use Alt+F to click the button.

162. What is the difference between Swing and AWT components?

- AWT components are heavyweight, whereas Swing components are lightweight. Heavy weight components depend on the local windowing toolkit. For example, java.awt.Button is a heavy weight component, when it is running on the Java platform for Unix platform, it maps to a real Motif button.

163. When is the ArrayStoreException thrown?

- When copying elements between different arrays, if the source or destination arguments are not arrays or their types are not compatible, an ArrayStoreException will be thrown.

164. What's the difference between the methods sleep() and wait()?

- The code sleep(2000); puts thread aside for exactly two seconds. The code wait(2000), causes a wait of up to two second. A thread could stop waiting earlier if it receives the notify() or notifyAll() call. The method wait() is defined in the class Object and the method sleep() is defined in the class Thread.

165. When ArithmeticException is thrown?

- The ArithmeticException is thrown when integer is divided by zero or taking the remainder of a number by zero. It is never thrown in floating-point operations**:** What is a transient variable?

166. What is the Collections API?

- The Collections API is a set of classes and **INTERFACES** that support operations on collections of objects.

167. Which Java operator is right associative?

- = operator i.

168. What is the difference between a break statement and a continue statement?

- A break statement results in the termination of the statement to which it applies (switch, for, do, or while).
- continue statement is used to end the current loop iteration and return control to the loop statement.

169. If a variable is declared as private, where may the variable be accessed?

- A private variable may only be accessed within the class in which it is declared.

170. What is the purpose of the System class?

- The purpose of the System class is to provide access to system resources.

171. List primitive Java types?

- The eight primitive types are byte, char, short, int, long, float, double, and boolean.

172. What is the relationship between clipping and repainting under AWT?

- When a window is repainted by the AWT painting thread, it sets the clipping regions to the area of the window that requires repainting.

173. Which class is the immediate superclass of the Container class?

- Component class is the immediate super class.

174. What class of exceptions is generated by the Java run-time system?

- The Java runtime system generates RuntimeException and Error exceptions.

175. Which arithmetic operations can result in the throwing of an ArithmeticException?

- Integer / and % can result in the throwing of an ArithmeticException.

176. Variable of the boolean type is automatically initialized as?

- The default value of the boolean type is false.

177. What are ClassLoaders?

- A class loader is an object that is responsible for loading classes. The class ClassLoader is an **ABSTRACT CLASS**.

178. ClassLoader in Java works on three principle: `delegation`, `visibility` and `uniqueness`.

Java class loaders are used to load classes at runtime.

Delegation principle forward request of class loading to parent class loader and only loads the class, if parent is not able to find or load class.

Visibility principle allows child class loader to see all the classes loaded by parent ClassLoader, but parent class loader can not see classes loaded by child.

Uniqueness principle allows to load a class exactly once, which is basically achieved by delegation and ensures that child ClassLoader doesn't reload the class already loaded by parent.

4. What is ClassLoader in Java- MetLife

179. ClassLoader in Java is a class which is used to load class files in Java.

There are three default class loader used in Java,

Bootstrap ,

Extension

System or Application

where they loads class files

Bootstrap ClassLoader is responsible for loading standard JDK class files from `rt.jar` and it is parent of all class loaders in Java. Bootstrap class loader don't have any parents,

Extension ClassLoader delegates class loading request to its parent, `Bootstrap` and if unsuccessful, loads class form `jre/lib/ext` directory or any other directory pointed by `java.ext.dirs` system property.

System or Application class loader and it is responsible for loading application specific classes from CLASSPATH environment variable, `-classpath` or `-cp` command line option, `Class-Path` attribute of Manifest file inside JAR.

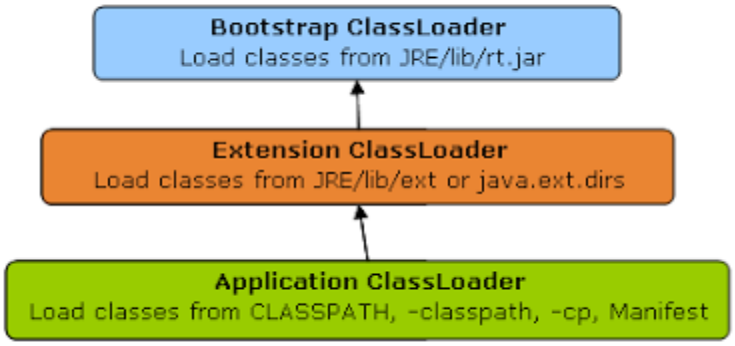
Application class loader is a child of Extension ClassLoader

In short here is the location from which Bootstrap, Extension and Application ClassLoader load Class files.

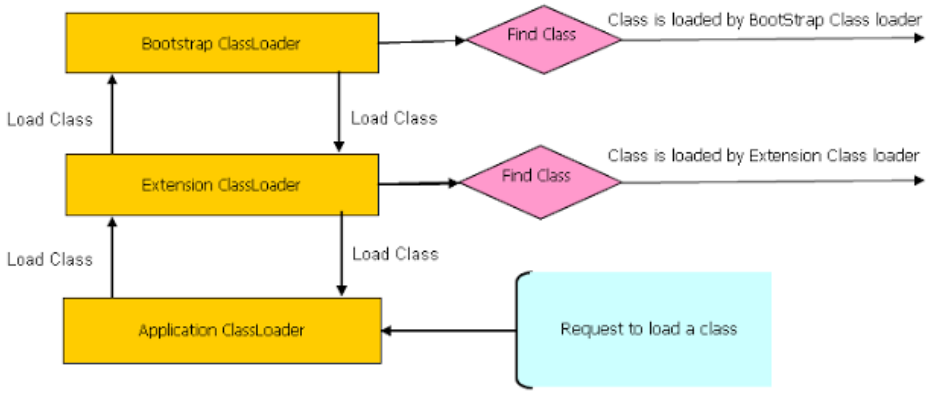
1) Bootstrap ClassLoader - **JRE/lib/rt.jar**

2) Extension ClassLoader - **JRE/lib/ext or any directory denoted by `java.ext.dirs`**

3) Application ClassLoader - **`CLASSPATH` environment variable,** `-classpath` or `-cp` option, Class-Path attribute of Manifest inside JAR file.



180. How ClassLoader works in Java



181. Why is access to non-static variables not allowed from static methods in Java

- because non-static variables are associated with a particular instance of object while Static is not associated with any instance. You can also see my post why non-static variable are not accessible in static context for more detailed discussion.

182. Give example of decorator design pattern in Java ? Does it operate on object level or class level ?

- Decorator pattern enhances capability of individual object.
- Ex
- Buffered classes like BufferedReader and BufferedWriter, which enhances Reader and Writer objects to perform Buffer level reading and writing for improved performance.

183. What is decorator pattern in Java? Can you give an example of Decorator pattern?

- Decorator pattern is another popular java design pattern question, which is common because of its heavy usage in java.io package. BufferedReader and BufferedWriter are good example of decorator pattern in Java. See How to use Decorator pattern in Java for more details.

184. When to use Composite design Pattern in Java? Have you used previously in your project?

- This design pattern question is asked on Java interview not just to check familiarity with Composite pattern but also, whether candidate has real life experience or not. Composite pattern is also a core Java design pattern, which allows you to treat both whole and part object to treat in similar way. Client code, which deals with Composite or individual object doesn't differentiate on them, it is possible because Composite class also implement same **INTERFACE** as there individual part. One of the good example of Composite pattern from JDK is JPanel class, which is both Component and Container. When paint() method is called on JPanel, it internally called paint() method of individual components and let them draw themselves. On second part of this design pattern interview question, be truthful, if you have used then say yes, otherwise say that you are familiar with concept and used it by your own. By the way always remember, giving an example from your project creates better impression.

185. When to use Template method design Pattern in Java?

- Template pattern is another popular core Java design pattern interview question. I have seen it appear many times in real life project itself. Template pattern outlines an algorithm in form of template method and let subclass implement individual steps. Key point to mention, while answering this question is that template method should be final, so that subclass cannot override and change steps of algorithm, but same time individual step should be abstract, so that child classes can implement them.

186. What is Factory pattern in Java? advantage to create object?

- Factory pattern used to create object by providing static factory methods. There are many advantage of providing factory methods e.g. caching **IMMUTABLE** objects, easy to introduce new objects etc.

187. Difference between Decorator and Proxy pattern in Java?

- both Decorator and Proxy implements **INTERFACE** of the object they decorate or encapsulate.
- Decorator pattern is used to implement functionality on already created object, Decorator doesn't create object, instead it get object in its **CONSTRUCTOR**,
- Proxy pattern is used for controlling access to object. actually creates objects.

188. When to use Setter and CONSTRUCTOR Injection in Dependency Injection pattern?

- Use Setter injection to provide optional dependencies of an object,
- use **CONSTRUCTOR** injection to provide mandatory dependency of an object, w

189. When to use Adapter pattern in Java?

- Use Adapter pattern when you need to make two-class work with incompatible **INTERFACES**. Adapter pattern can also be used to encapsulate third party code, so that your application only depends upon Adapter, which can adapt itself when third party code changes or you moved to a different third party library. By the way this Java design pattern question can also be asked by providing actual scenario.

190. Can you write code to implement producer consumer design pattern in Java?

- Producer consumer design pattern is a concurrency design pattern in Java, which can be implemented using multiple ways.

191. What is Open closed design principle in Java?

- Open closed design principle is one of the SOLID principle defined by Robert C. Martin, popularly known as Uncle Bob. This principle advices that a code should be open for extension but close for modification.

192. What is Builder design pattern in Java? When do you use Builder pattern ?

- creational design pattern and often asked in Java interviews because of its specific use when you need to build an object which requires multiple properties some optional and some mandatory.

193. Create a file in a specified directory

- File.createTempFile() method of File class

```
import java.io.File;
public class Main {
   public static void main(String[] args)
   throws Exception {
 File file = null;
 File dir = new File("C:/");
 file = File.createTempFile("JavaTemp", ".javatemp", dir);
 System.out.println(file.getPath());
   }
}
```

Result:

```
C:\JavaTemp37056.javatemp
```

194. Read a file

- using readLine method of BufferedReader class.

```
import java.io.*;
public class Main {
   public static void main(String[] args)  {
 try {
    BufferedReader in = new BufferedReader
    (new FileReader("c:\\filename"));
    String str;
    while ((str = in.readLine()) != null) {
  System.out.println(str);
    }
    System.out.println(str);
    }
    catch (IOException e) {
    }
 }
   }
}
```

- Result:
- The above code sample will produce the following result.

```
aString
```

195. Write to a file

- write to a file using write method of BufferedWriter.

```
import java.io.*;
public class Main {
   public static void main(String[] args)  {
 try {
    BufferedWriter out = new
    BufferedWriter(new FileWriter("outfilename"));
    out.write("aString");
    out.close();
    System.out.println
    ("File created successfully");
 }
 catch (IOException e) {
 }
   }
}
```

Result:

- The above code sample will produce the following result.

```
File created successfully.
```

196. What will happen if static modifier is removed from the signature of the main method?

- Program throws "NoSuchMethodError" error at runtime .

197. What is the default value of an object reference declared as an instance variable?

- Null, unless it is defined explicitly.

198. Can a top-level class be private or protected?

- No, a top-level class cannot be private or protected. It can have either "public" or no modifier.

199. Why do we need wrapper classes?

- We can pass them around as method parameters where a method expects an object. It also provides utility methods.

200. What is the difference between error and an exception?

- An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. Exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist.

201. Is it necessary that each try block must be followed by a catch block?

- It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block or a finally block.

202. When a thread is created and started, what is its initial state?

- A thread is in the ready state as initial state after it has been created and started.

203. What is the Locale class?

- The Locale class is used to tailor program output to the conventions of a particular geographic, political or cultural region.

204. What is runtime POLYMORPHISM or dynamic method dispatch?

- Runtime **POLYMORPHISM** or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass.

205. What is Dynamic Binding(late binding)?

- Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time.

206. What are the advantages of ArrayList over arrays?

- ArrayList can grow dynamically and provides more powerful insertion and search mechanisms than arrays.

207. When to use Array

- Use Array when have group of objects that won't be changing and want high performance.

208. When to use ArrayList

- Use ArrayList when have group of objects that could or will be changing.

209. When to use HashMap

- Use HashMap when index isn't an integer.

210. Why deletion in LinkedList is fast than ArrayList?

- Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

211. How do you decide when to use ArrayList and LinkedList?

- If you need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList should be used. If you need to support random access, without inserting or removing elements from any place other than the end, then ArrayList should be used.

212. What is a Values Collection View ?

- It is a collection returned by the values() method of the Map **INTERFACE** , It contains all the objects present as values in the map.

213. What is dot operator?

- The dot operator (.) is used to access the instance variables and methods of class objects. It is also used to access classes and sub-packages from a package.

214. What is type casting?

- Type casting means treating a variable of one type as though it is another type.

215. Describe life cycle of thread?

- A thread is an execution in a program. The life cycle of a thread include:
- Newborn state
- Runnable state
- Running state
- Blocked state
- Dead state

216. What is the difference between the >> and >>> operators?

- The >> operator carries the sign bit when shifting right. The >>> zero-fills bits that have been shifted out.

217. Which method of the Component class is used to set the position and size of a component?

- setBounds() method is used for this purpose.

218. What is the range of the short type?

- The range of the short type is -(2^to 2^15 - 1.

219. What is the immediate superclass of Menu?

- MenuItem class

220. Does Java allow Default Arguments?

- No, Java does not allow Default Arguments.

221. Which number is denoted by leading zero in java?

- Octal Numbers are denoted by leading zero in java, example: 06

222. Which number is denoted by leading 0x or 0X in java?

- Hexadecimal Numbers are denoted by leading 0x or 0X in java, example: 0XF

223. Break statement can be used as labels in Java?

- Yes, an example can be break one;

224. Where import statement is used in a Java program?

- Import statement is allowed at the beginning of the program file after package statement.

225. Explain suspend() method under Thread class>

- It is used to pause or temporarily stop the execution of the thread.

226. Explain isAlive() method under Thread class?

- It is used to find out whether a thread is still running or not.

227. What is currentThread()?

- It is a public static method used to obtain a reference to the current thread.

228. Explain main thread under Thread class execution?

- The main thread is created automatically and it begins to execute immediately when a program starts. It is a thread from which all other child threads originate.

229. What is an applet?

- An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

230. An applet extends which class?

- An applet extends **java.applet.Applet** class.
- provides interactive features to web applications, like capture mouse input, buttons or checkboxes
- code downloaded from server
- small, fast files, similar func and exec is JavaScript
- A Java Servlet is sometimes informally compared to be "like" a server-side applet, but it is different in its language, functions, and in each of the characteristics described here about applets.

```
import java.applet.Applet;
import java.awt.*;

// Applet code for the "Hello, world!" example.
// This should be saved in a file named as "HelloWorld.java".
public class HelloWorld extends Applet {

    // Print a message on the screen (x=20, y=10).
    public void paint(Graphics g) {
   g.drawString("Hello, world!", 20, 10);

   // Draws a circle on the screen (x=40, y=30).
   g.drawArc(40, 30, 20, 20, 0, 360);
    }
}
```

231. Life cycle of an applet includes which steps?

1. Initialization
2. Starting
3. Stopping
4. Destroying
5. Painting

232. Why is the role of init() method under applets?

- It initializes the applet and is the first method to be called.

233. Which method is called by Applet class to load an image?

- getImage(URL object, filename) is used for this purpose.

234. Define code as an attribute of Applet?

- It is used to specify the name of the applet class.

235. Can you write a Java class that could be used both as an applet as well as an application?

- Yes, just add a main() method to the applet.

236. What is the difference between an Applet and a Servlet?

Applet	Servlet
• transmitted over the network • Uses the **INTERFACE** classes like AWT or Swing • • **Client** side java program, • in **browser** on **client** • • Life Cycle Methods Init, stop, paint, start, destroy	• analog to CGI programs • implemented by means of servlet container associated with an HTTP server • **Server side** component • runs on the **web server** • No UI • Methods doGet, doPost

237. Define canvas?

- It is a simple drawing surface, which are used for painting images or to perform other graphical operations.

238. Define Network Programming?

- It refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

239. What is a Socket?

- Sockets provide the communication mechanism between two computers using TCP.
- **Client** end - A client program **java.net.Socket creates a socket** on its end of the communication and attempts to connect that socket to a server,
- **Server** end -The **java.net.ServerSocket** class is used by server applications to **obtain a port** and listen for client requests and establish connections with them
- Each socket has both an OutputStream and an InputStream.
- TCP is a two way communication protocol, so data can be sent across both streams at the same time.

240. Advantages of Java Sockets?

- Sockets are flexible and sufficient. Efficient socket based programming can be easily implemented for general communications. It cause low network traffic.

241. Disadvantages of Java Sockets?

- Socket based communications allows only to send packets of raw data between applications. Both the client-side and server-side have to provide mechanisms to make the data useful in any way.

242. Why Generics are used in Java?

- Generics provide compile-time type safety that allows programmers to catch invalid types at compile time. Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods or, with a single class declaration, a set of related types.

243. What environment variables do I need to set on my machine in order to be able to run Java programs?

- are the two variables.

244. Is there any need to import java.lang package?

- No, there is no need to import this package. It is by default loaded internally by the JVM.

245. Is it possible to import same package or class twice? Will the JVM load the package twice at runtime?

- *YES*: It is possible to import the same package or class more than once. But It will not have any effect on the compiler or JVM. But, The JVM will NOT load the class more than once, irrespective of the number of times you import the same class.

246. What is Nested top-level class?

247. If System.exit (0); is written at the end of the try block, will the finally block still execute?

- No in this case the finally block will not execute because when you say System.exit (0); the control immediately goes out of the program, and thus finally never executes.

248. Which method must be implemented by all threads?

All tasks must implement the run() method

249. What is the GregorianCalendar class?

- The GregorianCalendar provides support for traditional Western calendars

250. What is the SimpleTimeZone class?

- The SimpleTimeZone class provides support for a Gregorian calendar .

251. What is an enumeration?

- An enumeration is an **INTERFACE** containing methods for accessing the underlying data structure from which the enumeration is obtained. It allows sequential access to all the elements stored in the collection.
- location of .class files.

252. Can a class declared as private be accessed outside its package?

- No, it's not possible to access outside its package.

253. What is the restriction imposed on a static method or a static block of code?

- A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

254. What is “this” keyword in java?

- reference to the current object —
- Usage of this keyword
    - Used to refer current class instance variable.
    - To invoke current class **CONSTRUCTOR**.
    - It can be passed as an argument in the method call.
    - It can be passed as argument in the **CONSTRUCTOR** call.
    - Used to return the current class instance.
    - Used to invoke current class method (implicitly)

255. What is Downcasting?

- It is the casting from a general to a more specific type, i.e. casting down the hierarchy.

256. What are order of precedence and associativity and how are they used?

- Order of precedence determines the order in which operators are evaluated in expressions. Associativity determines whether an expression is evaluated left-to-right or right-to-left.

257. What is the difference between inner class and nested class?

- When a class is defined within a scope of another class, then it becomes inner class. If the access modifier of the inner class is static, then it becomes nested class.

258. What is the scope of variables in Java in following cases?

- **Member Variables** (Class Level Scope) : The member variables must be declared inside class (outside any function). They can be directly accessed anywhere in class
- **Local Variables** (Method Level Scope) : Variables declared inside a method have method level scope and can't be accessed outside the method.
- **Loop Variables** (Block Scope) : A variable declared inside pair of brackets "{" and "}" in a method has scope withing the brackets only.

259. What restrictions are placed on method overriding?

- Overridden methods must have the same name, argument list, and return type. The overriding method may not limit the access of the method it overrides.

260. Can a double value be cast to a byte?

- Yes, a double value can be cast to a byte.

261. How does a try statement determine which catch clause should be used to handle an exception?

- When an exception is thrown within the body of a try statement, the catch clauses of the try statement are examined in the order in which they appear. The first catch clause that is capable of handling the exception is executed. The remaining catch clauses are ignored.

262. What will be the default values of all the elements of an array defined as an instance variable?

- If the array is an array of primitive types, then all the elements of the array will be initialized to the default value corresponding to that primitive type.

263. In Java, how does System.out.println() work?

- This question is an excellent example of how just some very basic knowledge of Java can lead you to the correct answer. Most interviewers would not expect you to know the answer to do this right away – but would like to see how you think and arrive at an answer.
- Marcus Aurelius (a Roman emperor) once said: "Of each particular thing ask: what is it in itself? What is its nature?". This problem is an excellent example of how that sort of thinking can help one arrive at an answer

with only some basic Java knowledge.

- With that in mind, let's break this down, starting with the dot operator. In Java, the dot operator can only be used to call methods and variables so we know that 'out' must be either a method or a variable. Now, how do we categorize 'out'? Well, 'out' could not possibly be a method because of the fact that there are no parentheses – the '( )' – after 'out', which means that out is clearly not a method that is being invoked. And, 'out' does not accept any arguments because only methods accept arguments – you will never see something like "System.out(2,3).println". This means 'out' must be a variable.

264. What is "out" in System.out.println()?

- We now know that 'out' is a variable, so we must now ask ourselves what kind of variable is it. There are two possibilities – it could be a static or an instance variable. Because 'out' is being called with the 'System' class name itself, and not an instance of a class (an object), then we know that 'out' must be a static variable, since only static variables can be called with just the class name itself. So now we know that 'out' is a static member variable belonging to the System class.

265. Is "out" in System.out.println() an instance variable?

- A:Noticing the fact that 'println()' is clearly a method, we can further classify the 'out' in System.out.println(). We have already reasoned that 'out' is a static variable belonging to the class System. But now we can see that 'out' must be an instance of a class, because it is invoking the method 'println ()'.
- The thought process that one should use to arrive at an answer is purposely illustrated above. Without knowing the exact answer beforehand, you can arrive at an approximate one by applying some basic knowledge of Java. Most interviewers wouldn't expect you to know how System.out.println() works off the top of your head, but would rather see you use your basic Java knowledge to arrive at an answer that's close to exact.

266. When and where is the "out" instantiated in System.out.println?

- When the JVM is initialized, the method **initializeSystemClass()** is called that does exactly what its name says – it initializes the System class and sets the out variable. The **initializeSystemClass()** method actually calls another method to set the out variable – this method is called **setOut()**.

267. The final answer to how system.out.println() works

- The more exact answer to the original question is this: inside the System class is the declaration of 'out' that looks like: 'public static final PrintStream out', and inside the Prinstream class is a declaration of 'println()' that has a method signature that looks like: 'public void println()'.
- Here is what the different pieces of System.out.println() actually look like:

```
//the System class belongs to java.lang package
class System {
  public static final PrintStream out;
  //...
}
//the Prinstream class belongs to java.io package
class PrintStream{
public void println();
//...
}
```

268. JVM translates bytecode into machine language

- Every Java program is first compiled into an intermediate language called Java bytecode. The JVM is used primarily for 2 things: the first is to translate the bytecode into the machine language for a particular computer, and the second thing is to actually execute the corresponding machine-language instructions as well. The JVM and bytecode combined give Java its status as a "portable" language – this is because Java bytecode can be transferred from one machine to another.

269. Machine language is OS dependent

- Given the previous information, it should be easier to figure out an answer to the original question. Since the JVM must translate the bytecode into machine language, and since the machine language depends on the operating system being used, it is clear that the JVM is platform (operating system) dependent – in other words,

the JVM is **not** platform independent.

270. The JVM is not platform independent

- The key here is that the JVM depends on the operating system – so if you are running Mac OS X you will have a different JVM than if you are running Windows or some other operating system. This fact can be verified by trying to download the JVM for your particular machine – when trying to download it, you will be given a list of JVM's corresponding to different operating systems, and you will obviously pick whichever JVM is targeted for the operating system that you are running.

271. However, An inner class can access private members of its outer class. YES

- because the private member wasn't overridden. **So when you call the extended object method you're really calling the outer/base method**

```
/* Java program to demonstrate whether we can override private method
   of outer class inside its inner class */
class Outer {
private String msg = "GeeksforGeeks";
private void fun() {
System.out.println("Outer fun()");
}

class Inner extends Outer {
    private void fun()  {
    System.out.println("Accessing Private Member of Outer: " + msg);
    }
}

public static void main(String args[])  {

// In order to create instance of Inner class, we need an Outer
// class instance. So, first create Outer class instance and then
// inner class instance.
Outer o = new Outer();
Inner  i   = o.new Inner();

// This will call Inner's fun, the purpose of this call is to
// show that private members of Outer can be accessed in Inner.
i.fun();

o = i;
o.fun();calls Outer's fun, No run-time POLYMORPHISM
}
}
```

Comparison With C++

- In Java, inner Class is allowed to access private data members of outer class. This behavior is same as C++ (See this).
- In Java, methods declared as private can never be overridden, they are in-fact bounded during compile time. T**his behavior is different from C++. In C++, we can have virtual private methods** (See this).

272. In Java, what's the difference between an object and a class?

- **class** is used to define the behavior, attributes like the method definitions within a class.
- **object** actual **instance** of a class.

273. In Java, what does the 'final' modifier mean when applied to a method, class, and an instance variable?

- Final applied to
- Final class - cannot derive any class from it.
- Final method - not be overridden in a derived class.
- Final instance variable - can't be changed
- good to point out in an interview that the final modifier, when applied to either a class or a method, turns off late

binding, and thus prevents **POLYMORPHISM**.

274. In Java, what does the finally block do?

- placed after a try block and the catch blocks that follow it.
- contains code that will be **run whether or not an exception** is thrown in a try block.:

```
public void someMethod{
    Try {
    // some code
    }
    Catch(Exception x) {
    // some code
    }
    Catch(ExceptionClass y) {
    // some code
    }
    Finally{
    //this code will be executed whether or not an exception
    //is thrown or caught
    }
}
```

- There are 4 potential scenarios here:
- 1. The try block runs to the end, and no exception is thrown. In this scenario, **the finally block will be**
- executed after the try block.
- 2. An exception is thrown in the try block, which is then caught in one of the catch blocks. In this scenario, **the finally block will execute right after the catch block executes.**
- 3.An exception is thrown in the try block and there's no matching catch block in the method that can catch
- the exception. In this scenario, the call to the method ends, and the exception object is thrown to the enclosing method - as in the method in which the try-catch-finally blocks reside. But, before the method ends, **the finally block is executed.**
- 4. Before the try block runs to completion it returns to wherever the method was invoked. But, before it returns to the invoking method, the code in the finally block is still executed. So, remember that the code **in the finally block will still be executed even if there is a return statement somewhere in the try block.**
- 5. finally block be called and run after a return statement is executed
- yes – the code in a finally block will take precedence over the return statement.
- 6. Very unique situations when finally will not run after return
- if System.exit() is called first, or if the JVM crashes.
- What if there is a return statement in the finally block as well?
- If you have a return statement in both the finally block and the try block, then you could be in for a surprise. Anything that is returned in the finally block will actually **override** any exception or returned value that is inside the try/catch block. Here is an example that will help clarify what we are talking about:

```
public static int getANumber(){
    try{
   return 7;
    } finally {
   return 43;
    }
}
```

- The code above will actually return the “43″ instead of the “7″, because the return value in the finally block (“43″) will override the return value in the try block (“7″).
- Also, if the finally block returns a value, it will override any exception thrown in the try/catch block. Here is an example:

```
public static int getANumber(){
    try{
   throw new NoSuchFieldException();
    } finally {
   return 43;
    }
}
```

- A return statement in the finally block is a bad idea
- Running the method above will return a “43″ and the exception in the try block will not be thrown. This is why it

is considered to be a very bad idea to have a return statement inside the finally block.

275. Does Java pass by reference or by value?

- Java passes **everything** by value, and **not by reference** – make sure you remember that. And when we say everything, we mean everything – objects, arrays (which are objects in Java), primitive types (like ints and floats), etc. – these are **all passed by value** in Java. What is the difference between pass by value and pass by reference? When passing an argument (or even multiple arguments) to a method, Java will create a **copy or copies** of the values inside the original variable(s) and pass that to the method as arguments – and that is why it is called pass by **value**. The key with pass by value is that the method will not receive the actual variable that is being passed – but just a copy of the value being stored inside the variable. So, how does this affect the code you write? Well, this is best illustrated by a simple and easy to understand example.
- Example of pass by value in Java
- Suppose we have a method that is named "Receiving" and it expects an integer to be passed to it:

```
public void Receiving (int var)
{
  var = var + 2;
}

Note that the "var" variable has 2 added to it. Now, suppose that we have some code which
calls the method Receiving:

public static void main(String [] args)
{
  int passing = 3;
  Receiving (passing);

  System.out.println("The value of passing is: " + passing);
}


3
```

- The reason it prints out a "3" is because Java passes arguments by value – which means that when the call to "Receiving" is made, Java will just create a copy of the "passing" variable and that copy is what gets passed to the Receiving method – and **not** the original variable stored in the "main" method. This is why whatever happens to "var" inside the Receiving method does not affect the "passing" variable inside the main method.

http://www.programmerinterview.com/index.php/java-questions/

276. How does pass by value work?

- Java basically creates another integer variable with the value of 3, and passes that to the "Receiving" method. That is why it is called "pass by value" – because the value of 3 is what is being passed to the "Receiving" method, not a reference to the "passing" variable.

277. Are objects passed by reference in Java?

- As we said in the beginning of this article – everything is passed by value in Java. So, objects are **not** passed by reference in Java. Let's be a little bit more specific by what we mean here: objects are passed by reference – meaning that a reference/memory address is passed when an object is assigned to another – **BUT** (and this is what's important) that reference is actually passed by value. The reference is passed by value because a **copy** of the reference value is created and passed into the other object – read this entire article and this will make a lot more sense. So objects are still passed by value in Java, just like everything else.
- One other very important thing to know is that a variable of a class type stores an address of the object, and not the object itself. The object itself is stored at the address to which the variable points. Let's suppose we have a class called SomeClass and that we instantiate it with a variable of the SomeClass type. So, suppose we have the following very simple code:
- SomeClass someVar;
- The someVar variable above is a variable of a class type – which is commonly referred to as an **object**. But, what actually happens behind the scenes is that someVar will just contain a reference – which is basically a memory address that points to the real object and all the class variables for that object actually lives. So,

remember that a variable of a class type – or what is commonly referred to as an object of a class – really just stores a memory address that points to the real object. Bottom line: anytime you see a variable of a class type like the someVar variable above, just remember that it is basically a reference that stores an address in memory.

- Confused yet? An example and some diagrams will help make this very clear. Suppose we have a class called PersonClass, which has a method called set that just sets the name and age for a given Person object. The details of the class itself don't matter for the purpose of this example so we won't bother to show the implementation of PersonClass. Now, let's say we have the following code:

```
//create an object by passing in a name and age:
PersonClass variable1 = new PersonClass("Mary", 32);
PersonClass variable2;

// Both variable2 and variable1 now both name the same object
variable2 = variable1;

/*this also changes variable1, since variable 2 and variable1
   name the same exact object: */
variable2.set("Jack", 22);
System.out.println(variable1);
```

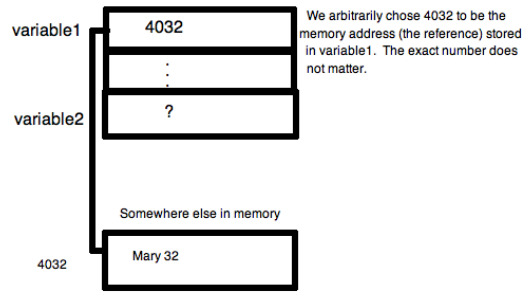
- Let's just assume that System.out.println method above will print the name and age of a PersonClass object – even though this is not correct, it does keep the example simple and easy to understand. So, if we run the code above, the output will be:

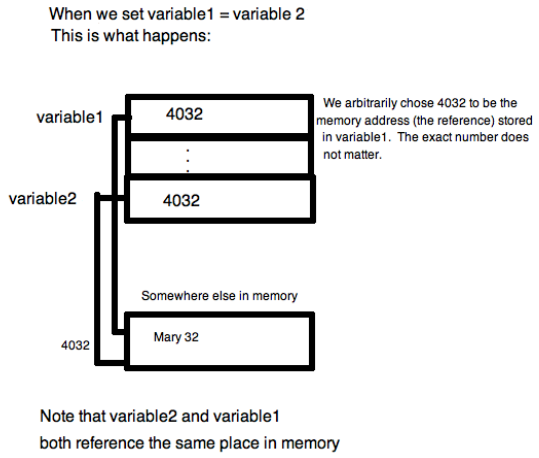
```
Jack 22
```

- In the code above, you can see that when we made a change to variable2, that it also changed variable1. **This might confuse you into thinking that because of that fact Java is pass by reference – and you would be very WRONG**. Both variable1 and variable2 hold a reference to the same object in memory – and this happened when we ran this line of code "variable2 = variable1". When we use the assignment operator (the "=") with variables of a class type, then we are copying that reference value – in other words variable2 is given the memory address (or reference) that is being stored in variable1. Remember that the word reference means the same thing as memory address. This assignment of references is best displayed by actual drawings, which we show below:

- An illustration of pass by value in Java

PersonClass variable1 = new PersonClass("Mary", 32);
PersonClass variable2;





- So, looking at the images above it should be clear that variable1 and variable2 are just two different references to the same exact spot in memory. Because of the fact that the statement "variable2 = variable1" essentially just **copies** the reference from variable1 into variable2, this is **pass by value**. You should think of the **reference (which is a memory address) as the value**, and in this case that reference is what is being copied. And, that is exactly what passes by value means – a copy of a value is passed to another variable. You should also read our article on the differences between primitive types and reference types here: Primitive type vs Reference type.
- Let's go through one last example. Let's say we have the following code:

```
//create an object by passing in a name and age:
PersonClass variable1 = new PersonClass("Mary", 32);
PersonClass variable2;
//Both variable2 and variable1 now reference the same object
variable2 = variable1;
PersonClass variable3 = new PersonClass("Andre", 45);
// variable1 now points to variable3
variable1 = variable3;
//WHAT IS OUTPUT BY THIS?
System.out.println(variable2);
System.out.println(variable1);
```

- If we run the code above it will output the code below:

```
Mary 32
Andre 45
```

- The key to understanding the code above is the fact that changing the object that variable1 points to does not change variable2. So, even though variable1 is changed to point to a different object, that has no effect whatsoever on variable2. Hopefully that is clear to you – that variable1 and variable2 are not interchangeable – they are different variables that just store the same value – at least until the "variable1 = variable3" statement. And that should prove to you that Java passes objects by value, and everything else for that matter.

278. Is it possible to pass an object by reference in Java? No

- No, there are no "extra" language features in Java that would allow for you to pass an object by reference in Java. Java is strictly pass by value, and does not give the programmer the option of passing anything by reference.

279. Are arrays passed by reference in Java?

- Arrays in Java are also objects. And what did you learn about objects in Java? Well, that they are passed by value and not passed by reference. And the same is true for arrays in Java – they are passed by value and not by reference. Of course, like any other class object, when an array is passed to another method that method can still change the contents of the array itself. But, what is being passed to the method is a copy of the reference address that points to the array object. Just take a look at our example above for the PersonClass – the same exact principles apply to arrays since they are objects in Java, and are passed by value.

280. What's the difference between a primitive type and a class type in Java?

- Every variable in Java has a **type**, which essentially tells Java how that variable should be treated, and how

much memory should be allocated for that variable. Java has basic types for characters, different kinds of integers, and different kinds of floating point numbers (numbers with a decimal point), and also types for the values true and false – char, int, float, and bool. All of these basic types are known as **primitive** types.

281. What is a class type in Java?

- Java classes, as you probably already know, are created to solve object-oriented problems. Any object of a class has the type "class". Because an object of a class is more complex than a simple integer, boolean, or other "primitive" type, a variable naming an object is known to be a **class type**.

282. Class types vs Object types vs Reference types

- You might be confused by all the different terminology used. So just to clarify,
  class types, object types, and reference types all mean the exact same thing – an object of a class.

283. The differences between class and primitive types in Java

- A variable of a class type – like a String – stores objects of its class differently from how variables of primitive types – like int or char – store their values. Every variable, whether it's of a primitive type or of a class type, is implemented as a location in computer memory.
- For a variable of a primitive type, the value of the variable is stored in the memory location assigned to the variable. So, if an integer variable is declared as "int x = 3″, then when we look at the memory location of "x", there will be a "3″ stored there just as expected.
- However, a variable of a class type only stores the memory address of where the object is located – not the values inside the object. So, if we have a class called "SomeClass", when we create an object like this: "SomeClass anObject", then when we look at "anObject" in memory, we will see that it does not store any of the variables that belong to that object in memory. Instead, the variable "anObject" just stores an address of another place in memory where all the details of "anObject" reside. This means that the object named by the variable is stored in some other location in memory and the variable contains only the memory address of where the object is stored. This memory address is called a **reference** to the object. If this is confusing, you will see an example of this further down.

284. Different memory requirements for variables of class types and primitive types

- A value of a primitive type – like a type int – will always require the same amount of memory to store one value. There is a maximum value of type int – which means that values of type int will have a limit on their size. But – the big difference between a primitive type and a class type is that an
- object of a class type, like an object of the class String, can be of any size.
- variable of type String (a class type) is of a **fixed size**, so it cannot store a gigantic string like one that is 3,000 characters long. But, it can and does store the address of any string since there is a limit to the size of an address.
- **So, remember that a variable of a class type stores an address of the object, and not the object values themselves – which are stored at that address in memory.**

285. Two class type variables may contain same reference (memory address)

- Because variables of a class type contain a reference (memory address), two different variables can hold the **same** reference, and in that situation both variables name the same object. Any change to the object named by one of these variables will produce a change to the object named by the other variable, since them, both reference the same object. This is best shown by an example, so take a look at the following simple code, assuming we have a class called PersonClass which has a method called set that just sets the name and age for a given Person object. This code can be understood without looking at the definition of PersonClass, which is also presented below for convenience:

```
PersonClass variable1 = new PersonClass("Mary", 32);
PersonClass variable2;
```

- // Both variable2 and variable1 now both name the same object

```
variable2 = variable1;
```

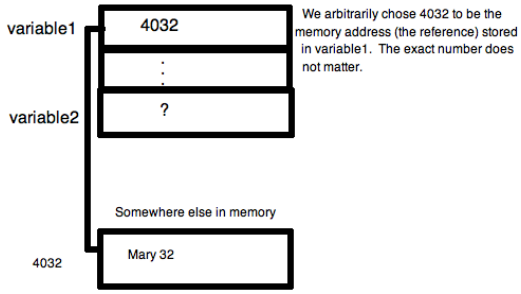
- /*this also changes variable1, since variable 2 and variable1

- name the same exact object */

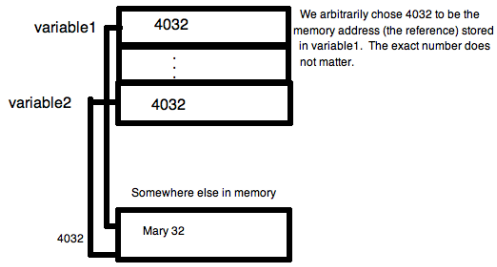
```
variable2.set("Jack", 22);
```

- System.out.println(variable1);
- And the output of this code is:
- Jack 22
- In the code above, you can see that when we made a change to variable2, that it also changed variable1. This is because both variable1 and variable2 reference the same object in memory – and this happened when we ran this line of code “variable2 = variable1″. When we use the assignment operator (the “=”) with variables of a class type, then we are assigning a reference – in other words variable2 is given the memory address (or reference) of variable1. Remember that the word reference means the same thing as memory address. This assignment of references is best displayed by actual drawings, which we provide below so that you can actually visualize what’s going on when an object is assigned to another in Java :

PersonClass variable1 = new PersonClass("Mary", 32);
PersonClass variable2;



When we set variable1 = variable 2
This is what happens:



Note that variable2 and variable1
both reference the same place in memory

- And here is the definition of the class PersonClass:

```
public class PersonClass
{
  private String name;
  private int number;
  public PersonClass(String initialName, int initialNumber)
  {
    name = initialName;
    number = initialNumber;
  }
  public void set(String newName, int newNumber)
  {
    name = newName;
    number = newNumber;
  }
}
```

- Generally, when you look at a declaration like this: “PersonClass y”, then you would refer to “y” as an object.

Although, for the sake of keeping things simple when communicating it is good, but in reality “y” is a variable that stores an address. That address is where the object is really stored, **not** in the variable “y”. This is an important distinction, and something that you should understand.

286. Class types are also reference types

- A type whose variables contain references is called a reference type. This means that in Java, class types are also reference types.
- But, primitive types (like ints, Booleans, etc.) are not reference types since primitive type variables do not hold references (which are basically addresses).

287. Does Java have pointers?

- No, Java does **not** have pointers. This was an intentional decision by the creators of Java, because most people would agree that having pointers creates a lot of potential for bugs in the code – pointers can be quite confusing, especially to new programmers. Because arrays and strings are provided as class types in Java, there is no need for pointers to those constructs. By not allowing pointers, Java provides effectively provides another level of **ABSTRACTION** to the programmer.
- Java has references, but not pointers
- But, what Java does have is **references**, which are different from pointers. Here are some of the differences between references in Java and pointers in C++:
- 1. References store an address. That address is the address in memory of the object. So, when a class is declared like so:
- "PersonClass y = new PersonClass();",
- the "y" variable actually stores an address in memory. If you were to look at that address in memory you would see the details of the PersonClass object. Pointers in C++, however, point directly to the object.
- 2. You cannot perform arithmetic operations on references. So, adding 1 to a pointer is not possible, but is possible in C++.

Design pattern interview questions for Advanced

288. Design a Vending Machine which can accept different coins, deliver different products?

- This is an open design question which you can use as exercise, try producing **design document**, **code** and **Junit test** rather just solving the problem and check how much time it take you to come to solution and produce require artifacts, Ideally this question should be solve in 3 hours, at least a working version.

289. You have a Smartphone class and will have derived classes like IPhone, AndroidPhone,WindowsMobilePhone

- can be even phone names with brand, how would you design this system of Classes.
- This is another design pattern exercise where you need to apply your object oriented design skill to come with a design which is flexible enough to support future products and stable enough to support changes in existing model.

290. Design ATM Machine ?

- We all use `ATM (Automated Teller Machine)` , Just think how will you design an ATM ? for designing financial system one must requirement is that they should work as expected in all situation. so no matter whether its power outage ATM should maintain **correct state (transactions**), think about **locking**, **transaction**, **error condition**, **boundary condition** etc. even if you not able to come up exact design but if you be able to point out nonfunctional requirement, raise some question , think about boundary condition will be good progress.

291. You are writing classes to provide Market Data and you know that you can switch to different vendors overtime like `Reuters`, `wombat` and may be even to direct exchange feed , how do you design your Market Data system.

- This is very interesting design interview question and actually asked in one of big investment bank and rather common scenario if you have been writing code in Java. Key point is you will have a `MarketData` **INTERFACE** which will have methods required by client e.g. `getBid()`, `getPrice()`, `getLevel()` etc. and `MarketData` should be composed with a `MarketDataProvider` by using **dependency injection**. So when you change your `MarketData` provider Client won't get affected because they access method form `MarketData` **INTERFACE** or class.

292. Design a Concurrent Rule pipeline in Java?

- `Concurrent programming` or `concurrent design` is very hot now days to leverage power of ever-increasing cores in advanced processor and Java being a multi-threaded language has benefit over others. Do design a concurrent system key point to note is thread-safety, `immutability`, local variables and avoid using static or instance variables. You just to think that one class can be executed by multiple threads a same time, so best approach are that every thread work on its own data, doesn't interfere on other data and have minimal synchronization preferred at start of pipeline. This question can lead from initial discussion to full coding of classes and **INTERFACE** but if you remember key points and issues around concurrency e.g. race condition, deadlock, memory interference, atomicity, ThreadLocal variables etc. you can get around it.

Java point Questions and Answers!!!!!!!!!!!

293. **What is the full form of MIME type**

Multiple Internet Mail Extension

294. **Which of the following interprets html code and renders webpages to user?**

Browser

295. **HTML is part of HttpRequest**

True

296. **Which http method send by browser ask the server to get the page only?**

Get

297. **How to send data in get method?**

Through URL

298. **Which http method sent to browser gives the server what user data typed into the form?**

Post

299. **When we are sending data in URL in get method, how are the parameters separated?**

Ampersand

300. **When we are sending data in URL in get method, how to separate the path and parameter**

Question mark

301. **Which of these are MIME types?**

Application /java etc.

302. **What is true about MIME types**

Tells the browser what type of data the browser will receive

303. **Assignment statement**

Assigning a value to a variable

304. **A UML Association is**

Implemented as a Java attribute member

305. **Correct syntax for java main method**

Public static void main

306. **What is the appropriate data type for this field: isSwimmer**

Boolean

307. **Public static void main() or public static void main(String[] args)**

- `void` is the return type, so it must go last
- Public and static have no particular order

308. **Size of Char in Java**

16 bits or 2 bytes

309. Is Empty.java file name a valid source file name

Yes

- .

10 Object Oriented Design Principles know

310. What is DRY (Don't repeat yourself)

- Our first object oriented design principle is DRY, as name suggest **DRY (don't repeat yourself)** means don't write duplicate code, instead use **ABSTRACTION** to abstract common things in one place. If you have block of code in more than two place consider making it a separate method, or if you use a hard-coded value more than one time make them public final constant. Benefit of this Object oriented design principle is in maintenance. It's important not to abuse it, duplication is not for code, but for functionality . It means, if you used common code to validate `OrderID` and `SSN` it doesn't mean they are same or they will remain same in future. By using common code for two different functionality or thing you closely couple them forever and when your OrderID changes its format , your SSN validation code will break. So beware of such coupling and just don't combine anything which uses similar code but are not related.

311. When to Encapsulate

- Only one thing is constant in software field and that is "Change", So encapsulate the code you expect or suspect to be changed in future. Benefit of this OOPS Design principle is that It's easy to test and maintain proper encapsulated code. If you are coding in Java then follow principle of making variable and methods private by default and increasing access step by step e.g. from private to protected and not public. Several of **design patterns in Java** uses **ENCAPSULATION** , Factory design pattern is one example of **ENCAPSULATION** which encapsulate object creation code and provides flexibility to introduce new product later with no impact on existing code.

312. What is the Open Closed Design Principle

- Classes, methods or functions should be Open for extension (new functionality) and Closed for modification. This is another beautiful SOLID design principle, which prevents some one from changing already tried and tested code. Ideally if you are adding new functionality only than your code should be tested and that's the goal of Open Closed Design principle. By the way, Open Closed principle is "O" from SOLID acronym.

313. What is the Single Responsibility Principle (SRP)

- Single Responsibility Principle is another SOLID design principle, and represent "S" on SOLID acronym. As per SRP, there should not be more than one reason for a class to change, or a class should always handle single functionality. If you put more than one functionality in one Class in Java it introduce **coupling** between two functionality and even if you change one functionality there is chance you broke coupled functionality, which require another round of testing to avoid any surprise on production environment.

314. What is the Dependency Injection or Inversion principle

- Don't ask for dependency it will be provided to you by framework. This has been very well implemented in Spring framework, beauty of this **design principle** is that any class which is injected by DI framework is easy to test with mock object and easier to maintain because object creation code is centralized in framework and client code is not littered with that. There are multiple ways to implemented **Dependency injection** like using byte code instrumentation which some AOP (Aspect Oriented programming) framework like AspectJ does or by using proxies just like used in Spring. See this example of IOC and DI design pattern to learn more about this SOLID design principle. It represents "D" on SOLID acronym.

315. Why Favor Composition over INHERITANCE

- Always favor composition over **INHERITANCE** ,if possible. Some of you may argue this, but I found that Composition is lot more flexible than **INHERITANCE**. Composition allows changing behavior of a class at

runtime by setting property during runtime and by using **INTERFACES** to compose a class we use **POLYMORPHISM** which provides flexibility of to replace with better implementation any time. Even Effective Java advice to favor composition over **INHERITANCE**.

316. What is the Liskov Substitution Principle (LSP)

- According to `Liskov Substitution Principle`, Subtypes must be substitutable for super type i.e. methods or functions which uses super class type must be able to work with object of sub class without any issue". LSP is closely related **to Single responsibility principle** and **INTERFACES egregation Principle**. If a class has more functionality than subclass might not support some of the functionality ,and does violated LSP. In order to follow **LSP SOLID design principle**, derived class or sub class must enhance functionality, but not reduce them. LSP represent "L" on SOLID acronym.

317. What is the INTERFACE Segregation principle (ISP)

- **INTERFACE S**`egregation Principle` stats that, a client should not implement an **INTERFACE** , if it doesn't use that. This happens mostly when one **INTERFACE** contains more than one functionality, and client only need one functionality and no other. **INTERFACE** design is tricky job because once you release your **INTERFACE** you cannot change it without breaking all implementation. Another benefit of this design principle in Java is, **INTERFACE** has disadvantage to implement all method before any class can use it so having single functionality means less method to implement.

318. What is Programming for INTERFACE not implementation

- Always program for **INTERFACE** and not for implementation this will lead to flexible code which can work with any new implementation of **INTERFACE** . So use **INTERFACE** type on variables, return types of method or argument type of methods in Java. This has been advised by many Java programmer including in Effective Java and headfirst design pattern book.

319. What is the Delegation principle

- Don't do all stuff by yourself, delegate it to respective class. Classical example of delegation design principle is equals() and hashCode() method in Java. In order to compare two object for equality we ask class itself to do comparison instead of Client class doing that check. Benefit of this design principle is no duplication of code and pretty easy to modify behavior.

- serves as Controller on many web MVC frameworks

320. What is J2EE

- J2EE stands for java 2 Enterprise Edition which is used to develop the multitier web-based applications. It consists of application programming **INTERFACES** (APIs), set of services and protocols.

321. What are the phases of the servlet life cycle?

- It has following phases –
- • Servlet class loading
- • Servlet instantiation
- • the init method
- • Request handling (call the service method)
- • Removal from service (call the destroy method)

322. In web.xml file <load-on-startup>1</load-on-startup> is defined between <servlet></servlet> tag what does it means.

- Whenever we request any servlet, the **servlet container will initialize the servlet and load it**, which is defined in our config file called web.xml
- by default it will not initialize when our context is loaded
- .defining like this <load-on-startup>1</load-on-startup> is also known as pre initialization of servlet means now the servlet for which we have define this tag has been initialized in starting when context is loaded before getting any request.

323. How can we create deadlock condition on our servlet?

- inside doGet () call doPost ()
- and inside doPost () call doGet ()
- It will create deadlock situation for a servlet.

324. For initializing a servlet can we use CONSTRUCTOR in place of init ().

- No, we cannot use **CONSTRUCTOR** for initializing a servlet because for initialization we need an object of servletConfig using this object we get all the parameter which are defined in deployment descriptor for initializing a servlet and in servlet class we have only default **CONSTRUCTOR** according to older version of java so if we want to pass a Config object we don't have parametrized **CONSTRUCTOR** and apart from this servlet is loaded and initialized by container so it's a job of container to call the method according to servlet specification they have lifecycle method so init() method is called firstly.

325. Why super.init (config) will be the first statement inside init (config) method.

- This will be the first statement if we are overriding the init (config) method by this way we will store the config object for future reference and we can use by getServletConfig () to get information about config object if will not do this config object will be lost and we have only one way to get config object because servlet pass config object only in init method. Without doing this if we call the servletConfig, method will get **NullPointerException.**

326. Can we call destroy () method inside the init () method is yes what will happen?

- Yes, we can call like this but if we have not override this method container will call, the default method and nothing will happen. After calling this if, any we have override the method then the code written inside is executed.

327. How can we refresh servlet on client and server side automatically?

On client side, we can use Meta http refresh and server side we can use server push.

328. How can you get the information about one servlet context in another servlet?

In context object, we can set the attribute, which we want on another servlet, and we can get that attribute using their name on another servlet.

- Context.setAttribute ("name"," value")
- Context.getAttribute ("name")

329. Why we need to implement Single Thread model in case of Servlet.

- In J2EE by either:
- 1. Single Thread Model
- 2. Multithread Model
- Single thread means only one instance is going handle one request at a time no two threads will concurrently execute.

330. What is servlet collaboration?

- communication between two servlets achieved by 3 ways.
- RequestDispatchers include () and forward () method.
- Using **sendRedirect ()** method of Response object.
- Using servlet Context methods

331. What is the difference between ServletConfig and ServletContext?

-
- ServletConfig
- Information about configuration of a servlet, which is defined inside the web.xml
- A specific object for each servlet.
- ServletContext
- Shared by all the servlet
- Belongs to one application
- All the servlet access application specific data
- Communicate with container.

332. How do you solve producer consumer problem in Java?

- One of my favorite questions during any Java multithreading interview, almost half of the concurrency problems can be categorized in producer consumer pattern. There are two ways to solve this problem in Java, One by using wait and notify method and other by using `BlockingQueue` in Java. Later is easy to implement and good choice if you are coding in Java 5. Key points to mention, while answering this question is `threadsafety` and blocking nature of BlockingQueue and how that helps, while writing concurrent code. You can also expect many follow-up questions including, what happen, if you have multiple producer or multiple consumer, what will happen if producer is faster than consumer thread or vice-versa. You can also see this link for example of how to code producer consumer design in Java using blocking queue

333. What is difference between submit () and execute () method of Executor and ExecutorService in Java?

- Main difference between submit and execute method from `ExecutorService` **INTERFACE** is that former return a result in form of Future object, while later doesn't return result. By the way, both are used to submit task to thread pool in Java but one is defined in `Executor` **INTERFACE**, while other is added into `ExecutorService` **INTERFACE**. This multithreading interview question is also asked at first round of Java interviews.

334. What is ReadWriteLock in Java? What is benefit of using ReadWriteLock in Java?

- This is usually a follow-up question of previous Java concurrency questions. `ReadWriteLock` is again based upon lock striping by providing separate lock for reading and writing operations. If you have noticed before, reading operation can be done without locking if there is no writer and that can hugely improve performance of any application. `ReadWriteLock` leverage this idea and provide policies to allow maximum concurrency level. Java Concurrency API also provides an implementation of this concept as `ReentrantReadWriteLock`. Depending upon Interviewer and experience of candidate, you can even expect to provide your own implementation of `ReadWriteLock`, so be preparing for that as well.

335. What are differences between wait and sleep method in java?

- Another frequently asked thread interview question in Java mostly appears in phone interview. Only major difference is wait release the lock, monitor while sleep doesn't release any lock, or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution. See my post wait vs sleep in Java for more differences

336. Write code to implement blocking queue in Java?

- This is relatively tough java multi-threading interview question which servers many purpose, it checks whether candidate can actually write Java code using thread or not, it sees how good candidate is on understanding concurrent scenarios and you can ask lot of follow-up question based upon his code. If he uses `wait()` and `notify()` method to implement blocking queue, Once interviewee successfully writes it you can ask him to write it again using new java 5 concurrent classes etc.

337. Write code to solve the Produce consumer problem in Java?

- Similar to above questions on thread but more classic in nature, sometime interviewer ask follow up questions How do you solve producer consumer problem in Java, well it can be solved in multiple way, I have shared one way to solve producer consumer problem using BlockingQueue in Java , so be prepare for surprises. Some time they even ask to implement solution of dining philosopher problem as well.

338. Write a program, which will result in deadlock? How will you fix deadlock in Java?

- This is my favorite java thread interview question because even though deadlock is quite common while writing multi-threaded concurrent program many candidates not able to write deadlock free code and they simply struggle. Just ask them you have n resources and n thread and to complete an operation you require all resources. Here n can be replace with 2 for simplest case and higher number to make question more intimidating. See How to avoid deadlock in java for more information on deadlock in Java.

339. What is race condition? How will you find and solve race condition?

- Another multi-threading question in Java, which appear mostly on senior level interviews. Most interviewer grill on recent race condition you have faced and how did you solve it and some time they will write sample code and ask you detect race condition. See my post on Race condition in Java for more information. In my opinion, this is one of the best java thread interview question and can really test the candidate's experience on solving race condition or writing code, which is free of data race or any other race condition. Best book to get mastery of this topic is "`Concurrency practices in Java`".

340. How will you take thread dump in Java? How will you analyze Thread dump?

- In UNIX you can use **kill -3** and then thread dump will print on log on windows you can use **"CTRL+Break".** Rather simple and focus thread interview question but can get tricky if he ask how you analyze it. Thread dump can be useful to analyze deadlock situations as well.

341. Why we call `start ()` method which in turns calls `run ()` method, why not we directly call `run ()` method?

- Another classic java multi-threading interview question this was my original doubt when I started programming in thread. Now days mostly asked in phone interview or first round of interview at mid and junior level java

interviews. Answer to this question is that, when you call `start ()` method it creates new Thread and execute code declared in `run ()` while directly calling `run ()` method doesn't create any new thread and execute code on same calling thread. Read my post Difference between starts and run method in Thread for more details.

342. How will you awake a blocked thread in java?

- This is tricky question on threading, blocking can result on many ways, if thread is blocked on IO then I don't think there is a way to interrupt the thread, let me know if there is any, on the other hand if thread is blocked due to result of calling `wait (), sleep () or join ()` method you can interrupt the thread and it will awake by throwing `InterruptedException`. See my post How to deal with blocking methods in Java for more information on handling blocked thread.

343. What is Busy Spinning? Why you will use Busy Spinning as wait strategy?

- This is really an advanced concurrency interview questions in Java and only asked to experienced and senior Java developers, with lots of concurrent coding experience under belt. By the way concept of busy spinning is not new, but it's usage with multi core processor has risen recently. It's a wait strategy, where one thread wait for a condition to become true, but instead of calling wait or sleep method and releasing CPU, it just spin. This is particularly useful if condition is going to be true quite quickly i.e. in millisecond or microsecond. Advantage of not releasing CPU is that, all cached data and instruction are remained unaffected, which may be lost, had this thread is suspended on one core and brought back to another thread. If you can answer this question, that rest assure of a good impression.

344. What is difference between `CyclicBarriar` and `CountdownLatch` in Java?

- New java thread interview questions mostly to check familiarity with JDK 5 concurrent packages. One difference is that you can reuse `CyclicBarrier` once barrier is broken but you cannot reuse `ContdownLatch`.

345. What is IMMUTABLE object? How does it help on writing concurrent application?

- Another classic interview questions on multi-threading, not directly related to thread but indirectly helps a lot. This java interview question can become more tricky if ask you to write an **IMMUTABLE** class or ask you Why String is **IMMUTABLE** in Java as follow-up.

346. What are some common problems you have faced in multi-threading environment? How did you resolve it?

- Memory-interference, race conditions, deadlock, live lock and starvation are example of some problems comes in multi-threading and concurrent programming. There is no end of problem if you get it wrong and they will be hard to detect and debug. This is mostly experienced based interview question on java thread instead of fact based.

Core Java Interview Questions Answers in Finance domain

347. What is IMMUTABLE Object? Can you write IMMUTABLE Class?

- **IMMUTABLE** classes are Java classes whose objects cannot be modified once created. Any modification in **IMMUTABLE** objects result in new object. For example, String is **IMMUTABLE** in Java. Mostly **IMMUTABLE** classes are also final in Java, in order to prevent sub classes from overriding methods, which can compromise Immutability. You can achieve same functionality by making member as non-final but private and not modifying them except in **CONSTRUCTOR**. Apart from obvious, you also need to make sure that, you should not expose internals of **IMMUTABLE** object, especially if it contains a mutable member. Similarly, when you accept value for mutable member from client e.g. `java.util.Date`, use clone () method keep separate copy for yourself, to prevent risk of malicious client modifying mutable reference after setting it. Same precaution needs to be taken while returning value for a mutable member, return another separate copy to client, never return original reference held by **IMMUTABLE** class.

348. Does all property of IMMUTABLE Object need to be final?

- Not necessary, as stated above you can achieve same functionality by making member as non-final but private and not modifying them except in **CONSTRUCTOR**. Don't provide setter method for them and if it is a mutable object, then don't ever leak any reference for that member. Remember referring variable final, only ensures that it will not be reassigned a different value, but you can still change individual properties of object, pointed by that reference variable. This is one of the key points, Interviewer like to hear from candidates. See my post on Java final variables, to learn more about them.

349. How does substring () inside String works?

- Another good Java interview question, I think answer is not sufficient but here it is “Substring creates new object out of source string by taking a portion of original string”. This question was mainly asked to see if developer is familiar with risk of memory leak, which sub-string can create. Until Java 1.7, substring holds reference of original character array, which means even a substring of 5 characters long, can prevent 1GB character array from garbage collection, by holding a strong reference.

350. How do you handle error condition while writing stored procedure or accessing stored procedure from java?

- This is one of the tough Java interview question and its open for all; my friend didn't know the answer so he didn't mind telling me. My take is that stored procedure should return error code if some operation fails but if stored procedure itself fail than catching SQLException is only choice.

351. What is difference between Executor.submit () and Executer.execute () method?

- This Java interview question is from my list of Top 15 Java multi-threading question answers, its getting popular day by day because of huge demand of Java developer with good concurrency skill. Answer of this Java interview question is that former returns an object of Future, which can be used to find result from worker thread)

352. Give a simplest way to find out the time a method takes for execution without using any profiling tool?

- This questions is suggested by @Mohit
- Read the system time just before the method is invoked and immediately after method returns. Take the time difference, which will give you the time taken by a method for execution.

```
To put it in code…

long start = System.currentTimeMillis ();

method ();

long end = System.currentTimeMillis ();

System.out.println ("Time taken for execution is " + (end - start));
```

- Remember that if the time taken for execution is too small, it might show that it is taking zero milliseconds for execution. Try it on a method, which is big enough, in the sense the one that is doing considerable amount of processing

JSP – Java Server Pages

10 XML Interview questions and answers for Java Programmer

XML Interview questions are very popular in various programming job interviews, including Java interviews for web developer. XML is a matured technology and often used as standard for transporting data from one platform other. *XML Interview questions* contains questions from various XML technologies like XSLT which is used to transform XML files, XPATH, XQuery and fundamentals of XML e.g. `DTD` or `Schema`. In this article we will see 10 frequently asked *XML Interview questions and answers* from above topics. These questions are mostly asked in various Java interviews but they are equally useful in other programming interviews like C, C++, Scala or any other programming language. Since XML is not tied with any programming language and like SQL its one of the desired skill in programmer, it make sense to practice some XML questions before appearing in any technical job interview

XML Interview Questions and Answers

 Here is my list of some common and frequently asked Interview questions on XML technologies. Questions on this list is not very tough but touches some important areas of XML technologies e.g. `DTD`, XML Schema, XSLT transformations, XPATH evaluation, XML binding, XML parsers and fundamentals of XML e.g. namespace, validation, attribute, elements etc.

What is XML ?

XML stands for Extensible Markup language which means you can extend XML based upon your needs. You can define custom tags like `<books>`, `<orders>` etc in XML easily as opposed to other mark-up language like HTML where you need to work with predefined tags e.g. `<p>` and you can not use user defined tag. Though structure of XML can be standardize by making use of DTD and XML Schema. XML is mostly used to transfer data from one system to another e.g. between client and server in enterprise applications.

Difference between DTD and XML Schema?

There are couple of differences between DTD and XML Schema e.g. DTD is not written using XML while XML schema are xml documents in itself, which means existing XML tools like XML parsers can be used to work with XML schema. Also XML schema is designed after DTD and it offer more types to map different types of data in XML documents. On the other hand DTD stands for Document Type definition and was a legacy way to define structure of XML documents.

What is XPath ?

XPath is an XML technology which is used to retrieve element from XML documents. Since XML documents are structured, XPath expression can be used to locate and retrieve elements, attributes or value from XML files. XPath is similar to SQL in terms of retrieving data from XML but it has it's own syntax and rules. See here to know more about How to use XPath to retrieve data from XML documents.

What is XSLT?

XSLT is another popular XML technology to transform one XML file to other XML, HTML or any other format. XSLT is like a language which specifies its own syntax, functions and operator to transform XML documents. Usually transformation is done by XSLT Engine which reads instruction written using XSLT syntax in XML style sheets or XSL files. XSLT also makes extensive use of recursion to perform transformation. One of the popular example of using XSLT is for displaying data present in XML files as HTML pages. XSLT is also very handy to transforming one XML file into another XML document.

What is element and attribute in XML?

This can be best explained by an example. let's see a simple XML snippet

```
<Orders>
  <Order id="123">
     <Symbol> 6758.T</Symbol>
     <Price> 2300</Price>
  <Order>
<Orders>
```

In this sample XML id is an attribute of `<Order>` element. Here `<Symbol>`, `<Price>` and `<Orders>` are also other elements but they don't have any attribute.

What is meaning of well formed XML ?

Another *interesting XML interview question* which most appeared in telephonic interviews. A well formed XML means an XML document which is syntactically correct e.g. it has a root element, all open tags are closed properly, attributes are in quotes etc. If an XML is not well formed, it may not be processed and parsed correctly by various XML parsers.

What is XML namespace? Why it's important?

XML namespace are similar to package in Java and used to provide a way to avoid conflict between two xml tags of same name but different sources. XML namespace is defined using xmlns attribute at top of the XML document and has following syntax xmlns:prefix="URI". later that prefix is used along with actual tag in XML documents. Here is an example of using XML namespace :

```
<root xmlns:inst="http://instruments.com/inst"
  <inst:phone>
      <inst:number>837363223</inst:number>
   </inst:phone>
</root>
```

Difference between DOM and SAX parser ?

This is another very popular XML interview question, not just in XML world but also on Java world. Main difference between DOM and SAX parser is the way they parse XML documents. DOM creates an in memory tree representation of XML documents during parsing while SAX is a event driven parser. See Difference between DOM and SAX parser for more detailed answer of this question.

What is a CDATA section in XML?

I like this XML Interview questions for its simplicity and importance, yet many programmer doesn't know much about it. CDATA stands for character data and has special instruction for XML parsers. Since XML parser parse all text in XML document e.g. `<name>`This is name of person`</name>` here even though value of tag `<name>` will be parsed because it may contain XML tags e.g. `<name><firstname>First Name</firstname></name>`. CDATA section is not parsed by XML parser. CDATA section starts with "`<![CDATA[`" and finishes with "`]]>`".

What is XML data Binding in Java?

XML binding in Java refers to creating Java classes and object from XML documents and then modifying XML documents using Java programming language. JAXB , Java API for XML binding provides convenient way to bind XML documents with Java objects. Other alternatives for XML binding is using open source library e.g. XML Beans. One of the biggest advantage of XML binding in Java is to leverage Java programming capability to create and modify XML documents.

This list of **XML Interview questions and answers** are collected from programmers but useful to anyone who is working in XML technologies. Important of XML technologies like `XPath`, `XSLT`, `XQuery` is only going to increase because of platform independent nature of XML and it's popularity of transmitting data over cross platform. Though XML has disadvantage like verbosity and size but its highly useful in web services and transmitting data from one system to other where bandwidth and speed is of secondary conce

353. Write a Java program to replace certain characters from String like

public String replaces(String str, char ch)

- solved in multiple way e.g. by using
  - charAt() or
  - subString() method,
- however, any approach throws couple of follow-up question e.g.
- you may be asked to write two versions to solve this coding exercise, one by using recursion and other by using Iteration. They may also ask you to write JUnit test for this function, which means handling null, empty string etc. By the way this programming question is quite common on technical interviews not just Java but also C, C++ but knowing API definitely helps to produce better solution quickly.

354. Write a Java program to print Fibonacci series up to 100?

This is one of the most *popular coding interview question asked in Java programming language*. Even though, Writing program for Fibonacci series is one of the basic Java program, not every Java developer get it right in interview.

using recursion or

Iteration.

```
// Arup Guha
// 7/24/07
// Done in class to illustrate dynamic programming.
public class fib {

   public static void main(String[] args) {
       // Test out both versions of the binomial coefficients.
       for (int i=0; i<=30; i++)
```

```
            System.out.print(binomial(30, i)+ " ");
        System.out.println();

        for (int i=0; i<=30; i++)
            System.out.print(binomialiter(30, i)+ " ");
        System.out.println();
    }

    // Recursive calculation of binomial coefficients.
    public static int binomial(int n, int k) {
        if (k == 0 || n == k)
            return 1;
        else
            return binomial(n-1, k) + binomial(n-1, k-1);
    }

    // Iterative binomial coefficient calculation.
    public static int binomialiter(int n, int k) {

        // Make space to store Pascal's triangle.
        int[][] pascaltri = new int[n+1][n+1];

        // Set the start and end of each row to 1.
        for (int i=0; i<=n; i++) {
            pascaltri[i][0] = 1;
            pascaltri[i][i] = 1;
        }

        // For each value, add the two values on the previous row.
        for (int i=2; i<=n; i++) {
            for (int j=1; j<i; j++)
                pascaltri[i][j] = pascaltri[i-1][j]+pascaltri[i-1][j-1];
        }

        // Return the answer.
        return pascaltri[n][k];
    }

    // Recursive fibonacci...
    public static int fib(int n) {
        if (n < 2)
            return n;
        else
            return fib(n-+ fib(n-2);
    }

    // Iterative fibonacci...
    public static int fibiter(int n) {
        int[] ans = new int[n+1];

        // Store first two fibonacci numbers.
        ans[0] = 0;
        ans[1] = 1;

        // add the last two to get the next one.
        for (int i=2; i<=n; i++)
            ans[i] = ans[i-1] + ans[i-2];
        return ans[n];
    }

}
```

FizzBuzz problem : Write a Java program that prints the numbers from 1 to However, for multiples of three print "Fizz" instead of the number and for the multiples of five prints "Buzz". For numbers, which are, multiples of both three and five print "FizzBuzz"?

classical programming questions,

used to differentiate programmers who can do coding and who can't.

```
public class FizzBuzzTest{

    public static void main(String args[]){

   for(int i = 1; i <= 50; i++) {
  if(i % (3*== System.out.println("FizzBuzz");
  else if(i % 5 == System.out.println("Buzz");
  else if(i % 3 == System.out.println("Fizz");
  else System.out.println(i);
   }
    }
```

```
}
```

355. Write a Comparator in Java to compare two employees based upon their name, departments and age?

- What is comparator in Java and
- How to use compare method in Java for sorting Object.
- Sorting is one of the most logical and practical question on technical interview and ability to sort Java object is must to code in Java. .

356. Design vending machine in Java, which vends Item, based upon four denominations of coins and return coin if there is no Item.

This kind of Java coding interview question appears in written test and I believe if you get it right, you are almost through the Interview. This kind of problem solving questions in Java are not easy, you need to design , developer and write JUnit test within 2 to 3 hours and only good Java developers, with practical coding experience can solve this kind of Java programming question. What helps you is to keep practicing your coding skill even before interview. See this programming exercise in Java to get you going. I personally like to ask programming questions, which test your object oriented design skills e.g. designing ATM machine, designing parking lot or implementing logic for Traffic Signal controller.

357. Write a Java program to check if a number is Armstrong or not ?

Another popular *logical coding interview questions* in Java, which is based on programming logic. In order to answer this programming question, you need to know what Armstrong number is, but that is not a problem because question may specify that and even provide sample input and output. Key thing to demonstrate is logic to check if a number is Armstrong or not. In most cases, you cannot use utility methods defined by logic and you need to produce logic by yourself by using basic operators and methods. By the way this is also one of the basic programming questions and I have already provided a solution for this. I suggest seeing this Java program to find Armstrong Number in Java to answer this coding question

358. Write a Java program to prevent deadlock in Java ?

Some of the programming or coding interview question is always based on fundamental feature of Java programming language e.g. multi-threading, synchronization etc. Since writing deadlock proof code is important for a Java developer, programming questions, which requires knowledge of concurrency constructs, becomes popular coding question asked in Java Interviews. Deadlock happens if four conditions is true e.g. mutual exclusion, no waiting, circular wait and no preemption. If you can break any of this condition than you can create Java programs, which are deadlock proof. One easy way to avoid deadlock is by imposing an ordering on acquisition and release of locks. You can further check How to fix deadlock in Java to answer this Java programming questions with coding in Java

359. Write Java program to reverse String in Java without using API functions ?

Another classic *Java programming or coding exercise* mostly asked on 2 to 5 years experienced Java interviews. Despite being simple answering this coding question is not easy, especially if you are not coding frequently. It's best to prepare this programming question in advance to avoid any embarrassment during interviews. I suggest seeing this post, which shows How to reverse String using recursion in Java

360. Write a Java program to find if a number is prime number or not

One more basic Java program, which made its way to Interviews. One of the simplest coding question and also a very popular Java programming exercise. Beauty of these kinds of logical questions is that, they can really test basic programming skills or a coder, programmer or developer. Not just problem solving, you can also check there coding style and thought process. By the way. you can check this article for answer of this Java coding interview question.

361. How to Swap two numbers without using third variable in Java?

This Java program might require just four lines to code, but it's worth preparing. Most of the programmers make same kind of mistakes, while writing solution for this program e.g. Integer overflow, they tend to forget that integer can overflow if its limit exceeded, which is not very big. Sure shot way to answer these programming questions is to use XOR trick to swap numbers, as mentioned in that blog post.

362. Create a Java program to find middle node of linked list in Java in one pass?

Any list of programming questions is incomplete without any questions from linked list, arrays and string, these three forms bulk of coding questions asked on Java interviews. Trick to solve this problem is to remember that last node of linked list points to null and you can trade memory with speed. Sometime your approach to come to two pointer solution really matters, by taking rational steps as

mentioned above, you can sound more intelligent, problem solver and genuine. Quick solution of this programming question can be found here.

363. How to find if a linked list contains cycle or not in Java?

Another programming question based on linked list. By the way this coding question is bit tricky than previous one, but this can also be solved using two pointer approach. If linked list has cycle, than fast pointer will catch either slow pointer or point to null. See Java program to check if linked list contains loop in Java for complete solution of this coding interview question.

364. Implement Producer Consumer design Pattern in Java using wait, notify and notify All method in Java?

Similar to deadlock related programming interview question, this is also used to test programmer's ability to write bug free concurrent programs in Java. These coding questions can be difficult if you haven't used wait and notify before, you can confuse yourself as hell on different aspect e.g. which condition to check, on which lock you should **SYNCHRONIZED** etc. I suggest following here to answer this multithreading based programming interview question.

365. Write a Java program to calculate Factorial of a number in Java?

This Java coding interview questions is also based on list of basic Java programs for beginners. As usual, you had better remember how to calculate factorial and how to code solution-using loop and recursive method calls. For complete code solution of this programming question, see Java program to calculate factorial

These are some of the **Java coding interview questions and answers**, which appears frequently on Java Programming interviews. I have included links, with some of my blog posts, which discusses answers of these Java coding question, but you can also find answers by doing google yourself. Please share what kind of Programming, logical, Problem solving or coding related questions, asked to you in Java interviews?

10 Tricky Java interview question

Here is my list of 10 tricky Java interview questions, Though I have prepared and shared lot of difficult core Java interview question and answers, But I have chosen them as Top 10 tricky questions because you cannot guess answers of this tricky Java questions easily, you need some subtle details of Java programming language to answer these questions.

366. What does the following Java program print?

```
public class Test {
    public static void main(String[] args) {
   System.out.println(Math.min(Double.MIN_VALUE, 0.0d));
    }
}
```

- This question is tricky because unlike the Integer, where `MIN_VALUE` is negative, both the `MAX_VALUE` and `MIN_VALUE` of the `Double` class are positive numbers. The `Double.MIN_VALUE is 2^(-1074)`, a double constant whose magnitude is the least among all double values. So unlike the obvious answer, this program will print 0.0 because `Double.MIN_VALUE` is greater than 0. I have asked this question to Java developer having experience up to 3 to 5 years and surprisingly almost 70% candidate got it wrong.

367. Question : What will happen if you put return statement or System.exit () on try or catch block ? Will finally block execute?

- This is a very *popular tricky Java question* and it's tricky because many programmers think that no matter what, but finally block will always execute. This question challenge that misconception by putting `return` statement in try or catch block or calling `System.exit` from try or catch block. Answer of this tricky question in Java is that `finally` block will execute even if you put `return` statement in try block or catch block but finally block won't run if you call `System.exit` form try or catch.

368. What does the expression 1.0 / 0.0 will return? will it throw Exception? any compile time error?

- This is another tricky question from Double class. Though Java developer knows about double primitive type and Double class, while doing floating point arithmetic they don't pay enough attention to `Double.INFINITY`, NaN, and `-0.0` and other rules that govern the arithmetic calculations involving them. Simple answer to this question is that it will not throw `ArithmeticExcpetion` and return `Double.INFINITY`. Also note that the comparison `x == Double.NaN` always evaluates to false, even if x itself is a NaN. To test if x is a NaN, one

should use the method call `Double.isNaN(x)` to check if given number is `NaN` or not. If you know SQL, this is very close to NULL there.

369. What is the issue with following implementation of compareTo() method in Java

```
public int compareTo(Object o){
   Employee emp = (Employee) emp;
   return this.id - o.id;
}
```

370. Now tell us, is it possible for Thread 2 to print "x=0"?

- It's impossible for a list of tricky Java questions to not contain anything from multi-threading. This is the simplest one I can get. Answer of this question is Yes, It's possible that thread T2 may print x=0.Why? because without any instruction to compiler e.g. **SYNCHRONIZED** or volatile, `bExit=true` might come before x=1 in compiler reordering. Also x=1 might not become visible in Thread 2, so Thread 2 will load x=0. Now, how do you fix it? When I asked this question to couple of programmers they answer differently, one suggest to make both thread **SYNCHRONIZED** on a common mutex, another one said make both variable volatile. Both are correct, as it will prevent reordering and guarantee visibility. But best answer is you just need to make `bExit` as volatile, then Thread 2 can only print "x=1". x does not need to be volatile because x cannot be reordered to come after `bExit=true` when `bExit` is volatile.

371. What is difference between CyclicBarrier and CountDownLatch in Java

- Relatively newer Java tricky question, only been introduced form Java 5. Main difference between both of them is that you can reuse `CyclicBarrier` even if Barrier is broken but you cannot reuse `CountDownLatch` in Java. See CyclicBarrier vs CountDownLatch in Java for more differences.

372. Can you access non-static variable in static context?

- Another tricky Java question from Java fundamentals. No you cannot access non-static variable from static context in Java. If you try, it will give compile time error. This is actually a common problem beginner in Java face, when they try to access instance variable inside main method. Because main is static in Java, and instance variables are non-static, you cannot access instance variable inside main. Read why you cannot access non-static variable from static method to learn more about this tricky Java questions.

373. difference between Path and Classpath

Path	Classpath
environment variable which is used to locate JDK binaries like "`java`" or "`javac`" command used to run java program and compile java source file	environment variable is used by System or Application ClassLoader to locate and load compile Java bytecodes stored in .class file.
include JDK_HOME/bin directory	all dirs w/ your .class file or JAR file
cannot be overridden	**can be overridden** by providing command line option **`-classpath` or `-cp`** to both "`java`" and "`javac`" commands or by using Class-Path attribute in Manifest file inside JAR archive.

Command to set PATH in Windows
`set PATH=%PATH%;C:\Program Files\Java\JDK1.6.20\bin`
Command to set PATH in UNIX/Linux
`export PATH =`
	Command to set CLASSPATH in windows
	`set CLASSPATH=%CLASSPATH%;C:\Program Files\Java\source\compiled`
	Command to set CLASSPATH in Unix/Linux
	`export CLASSPATH=`

${PATH}:/opt/Java/JDK1.6.18/bin	${CLASSPATH}:/opt/Java/JDK1.6.18/lib
Look at the difference between two commands, in Linux use colon(:) as separator and in Windows use semi-colon(;) as separator.	

374. What is Difference between Iterator and Enumeration in Java?

- One of the classic interview Questions asked on Java collection framework, This is pretty old and programmer who has been working in Java for 4 to 6 years must have seen this question before. Well Iterator and ListIterator in Java is a new way to iterator collection in Java and provide ability to remove object while traversing while Enumeration doesn't allow you to remove object. See Iterator vs Enumeration in Java for more differences between both of them.

375. What is Difference between fail-safe and fail-fast Iterator in Java?

- This is relatively new Java collection interview question because concept of fail-safe iterator is come along with `ConcurrentHashMap` and `CopyOnWriteArrayList`. See Difference between fail-safe and fail-fast Iterator in Java for answer of this Java collection question.

376. Can you write code to traverse Map in Java on 4 ways?

- Another Java collection question, which appear as part of Java Coding interview question and appeared in many interviews. As you know there are multiple ways to traverse or iterate Map in Java e.g. for loop, while loop using Iterator etc. 4 ways to iterator Map in Java has detailed explanation and sample code which is sufficient to answer this Java collection framework interview question.

377. What is difference between ArrayList and LinkedList in Java?

- A follow-up question, which is asked in response to previous Java collection interview question. Here also both `LinkedList` and `ArrayList` are List implementation but there internal data-structure is different, one is derived from Array while other is derived from LinkedList. See LinkedList vs ArrayList in Java to answer this Java Collection interview question.

378. What is difference between List and Set in Java ?

- `List` vs `Set` is one of the most important concepts to understand in Java Collection framework and this *Java collection interview question* focus on that. Most important difference between them is that List allows duplicates and maintains insertion order while Set doesn't allow duplicates and doesn't maintain any order. See Difference between Set and List in Java to see more differences between them

379. How do you find if ArrayList contains duplicates or not ?

- Since `List` allows duplicates this becomes a follow-up question of earlier Java collection framework interview question. See How to check if ArrayList contains duplicates or not for answer of this Java collection question.

Code writing questions and answers

```
1
2  public class Question
3  {
4      public static void JavaHungry(String s)
5      {
6          System.out.println("String");
7      }
8
9      public static void JavaHungry(Object o)
10     {
11         System.out.println("Object");
12     }
13
14     public static void main (String [] args)
15     {
16         JavaHungry(null);
17     }
18 }// end class
```

```
1  public class Question {
2
3  public static void JavaHungry(String s)
4  {
5    System.out.println("String");
6  }
7
8  public static void JavaHungry(Object o)
9  {
10   System.out.println("Object");
11 }
12
13 public static void JavaHungry(Integer s)
14 {
15   System.out.println("Integer");
16 }
17
18
19
20
21 public static void main (String args[])
22 {
23     JavaHungry(null);
24 }
25
26 }
27
```

```
1
2  public class Question
3  {
4      public static void JavaHungry(Exception e)
5      {
6          System.out.println("Exception");
7      }
8
9      public static void JavaHungry(ArithmeticException ae)
10     {
11         System.out.println("ArithemticException");
12     }
13
14     public static void JavaHungry(Object o)
15     {
16         System.out.println("Object");
17     }
18
19     public static void main (String [] args)
20     {
21         JavaHungry(null);
22     }
23 }// end class
```

380. Logic for the first three questions :

- The method lookup used by javac chooses the most specific valid overload.
- It will go to the most specific case as possible. Since null matches every possible subclass of object and object itself, it will always go to the most specific method possible. But if you have 2 classes that have split paths in the **INHERITANCE** tree, it will throw the compiler time exception that it is too ambiguous.

```
1
2  public class JavaHungry
3  {
4
5     public static void main(String[] args)
6      {
7          if(null==null)
8          {
9              System.out.println("Java Hungry Blogspot");
10         }
11     }
12
13 }
14
```

- The above question is valid one , there is no compile time error . The most important thing is that the expression in the *if block* should return a boolean value , so above null==null returns *true* , which is boolean so , the if block body executes and Java Hungry Blogspot prints

---

```
1
2  public class JavaHungry
3  {
4
5      public  void  m1(String arg1)
6      {
7          arg1 = "Am I going to disappear?";
8      }
9
10     public static void main (String[] args)
11     {
12         JavaHungry test = new JavaHungry();
13         String iAmOfAnArgumentativeNature = "I am born new";
14         test.m1(iAmOfAnArgumentativeNature);
15         System.out.print(iAmOfAnArgumentativeNature);
16     }
17 }// end class
18
19
```

- Java passes objects by reference and not by value. In other words , Java is always pass-by-value.
- The difficult thing can be to understand that Java passes objects as references and those references are passed by value.
- In Java strings can never change, because they're IMMUTABLE.
- arg1 initially is a copy of the reference pointing to the "I am born new!" string.
- "Am I going to disappear?" is simply a different string whose reference gets assigned to arg1.
- Changing arg1 has no effect on the original, since arg1 is just a copy of the iAmOfAnArgumentativeNature reference.

```
1  public class JavaHungry {
2
3  public static void main(String args [])
4
5  {
6      short s = 0;
7      int x = 07;
8      int y = 08;
9      int z = 123456;
10
11     s += z;
12     System.out.println("" + x + y + s);
13
14 }
15
16
17 }
18
```

There are a number of things going on in the question.

1. Line 12 The "" in the println causes the numbers to be automatically cast as strings. So it doesn't do addition, but appends together as string.
2. Line11 the += does an automatic cast to a short. However the number 123456 can't be contained within a short, so you end up with a negative value (-7616)
3. Those other two are red herrings however as the code will never compile due to line 8. Any number beginning with zero is treated as an octal number (which is 0-7).

```
1  public class Question
2  {
3      public static void main (String args[])
4      {
5          int x=2;
6          if(x=2)
7          {
8              System.out.println("Both number are equal");
9          }
10     }
11 }
12
```

- *If block* expression should always return a boolean value . In line 6 , x=2 , assign value 2 to the variable x. x is an int variable . So , *if block* expression ends up with int value while it is expecting boolean value hence compiler error .

```
class Job extends Thread {
private int counter;

@Override
public void run() {
synchronized(this) {
for(int i = 0; i < 100000; i++)
counter++;

this.notifyAll();
System.out.println("Completed Counting.....");
}
}
public static void main(String[] args) throws InterruptedException {
Job job = new Job();
job.start();
Thread.sleep(10000);
System.out.println("Waiting to get End.....");
synchronized(job) {
job.wait();
}

System.out.println(job.counter);
}
}
```

if the Thread cquires the monitor first, then it will call notifyAll() before main even starts wait()ing. When the thread eleases the monitor, main will start wait()ing forever and never get to print the result.

Reinvestigate GR - ----------

```
public class Question
{
    public static void main (String[] args)
    {
        javahungrymethod();
    }

    public static int javahungrymethod()
    {
       try
       {
            System.out.println("try");
            return 10 ;
       }
       catch(Exception e)
       {
           System.out.println("catch");
       }
       finally
       {
          System.out.println("finally");
          return 88;
       }
}
}// end class
```

```
public class JavaHungry
{
    public static void main (String[] args)
    {
        javahungrymethod();
    }

    public static int javahungrymethod()
    {
       try
       {
            System.out.println("try");
            int x =4/0;
            return 10 ;
       }
       catch(Exception e)
       {
           System.out.println("catch");
       }
       finally
       {
          System.out.println("finally");
          return 88;
       }
}
}// end class
```

Logic for the above questions : Main reason to how above is that finally block always execute and it will execute last in try/catch/finally block , rrespective of flow of exception generation and handling . ----------

--------- ----------

```
1  public class JavaHungry
2  {
3      int x=3;
4
5      public static void main (String args[])
6      {
7        new JavaHungry().go1();
8      }
9      void go1()
10     {
11       int x;
12       go2(++x);
13     }
14     void go2(int y)
15     {
16         int x= ++y;
17         System.out.println(x);
18     }
19 }
20
```

In line 11 , the must know thing in java, he ocal variable always need to be initialized before using it in the code . Eclipse will not compile Java with his error

---------- -

```
1  public class JavaHungry
2  {
3
4      public static void main(String[] args)
5      {
6            String s1 = "abc";
7            String s2 = s1;
8            s1 += "d";
9            System.out.println(s1 + " " + s2 + " " + (s1==s2));
10
11           StringBuffer sb1 = new StringBuffer("abc");
12           StringBuffer sb2 = sb1;
13           sb1.append("d");
14           System.out.println(sb1 + " " + sb2 + " " + (sb1==sb2));
15     }
16
17 }
18
```

- Above example shows that , String is **IMMUTABLE** and string buffer is mutable . So string s2 and s1 both pointing to the same string bc . And , after making the changes the string s1 points to abcd and s2 points to abc , hence false . While in string buffer , both sb1 and sb2 both point to the same object , As string buffer are utable , so making changes in one string also make changes to the other string . So both string still pointing to the same object after making the changes to the object (here sb2)

381. Count number of words in the String with Example : Java Program Code

- To find the number of words in the given String in java is easy . Here we are using arrays and string class to achieve our goal .
- For example :
- Original String : "Alive is awesome "
- If we pass the above string in the *wordcount* method then it will return

Output : 3as the answer.

- Here we are passing String to the wordcount() function which return the int value that is the number of words in the string .

```
public class StringDemo
{
    static int i,c=0,res;
    static int wordcount(String s)
    {
   char ch[]= new char[s.length()]; //in string especially we have to mention the () after
length
   for(i=0;i<s.length();i++)
   {
  ch[i]= s.charAt(i);
  if( ((i>0)&&(ch[i]!=' ')&&(ch[i-1]==' ')) || ((ch[0]!=' ')&&(i==0)) )
  c++;
   }
   return c;
    }

    public static void main (String args[])
    {
   res=StringDemo.wordcount("   manchester united is also known as red devil ");
   //string is always passed in double quotes

   System.out.println("The number of words in the String are :  "+res);
    }
}
```

Java Hungry Copyright © 2015 · All Rights Reserved. Designed by studiopress, Oracle docs ,

Top 50 J2EE interview questions

382. Define Hash table

- **HashTable** is just like Hash Map,Collection having key(Unique),value pairs. **HashTable** is a collection Synchronozed object .It does not allow duplicate values but it allows null values.

383. A simple method is to create a Hash Table.

- Calculate the hash value of each word in such a way that all anagrams have the same hash value. Populate the Hash Table with these hash values. Finally, print those words together with same hash values. - MetLife
- A simple hashing mechanism can be modulo sum of all characters. With modulo sum, two non-anagram words may have same hash value. This can be handled by matching individual characters. - MetLife

384. What is equals() and hashCode() contract in Java? Where does it used?

- `equals()` and `hashCode()` the key point of contract is that if two objects are equal by `equals()` method then they must have same hashcode
- but unequal object can also have same hashcode, which is the cause of collision on hash table based collection e.g HashMap.
- When you override `equals()` you must remember to override `hashCode()` method to keep the contract valid.

385. Difference between save and saveorupdate

- **save()** – This method in hibernate is used to stores an object into the database. It insert an entry if the record doesn't exist, otherwise not.
- **saveorupdate ()** -This method in the hibernate is used for updating the object using identifier. If the identifier is missing this method calls save(). If the identifier exists, it will call update method.

386. Difference between load and get method?

- **load**() can't find the object from cache or database, an exception is thrown and the load() method never returns null.
- **get**() method returns null if the object can't be found. The load() method may return a proxy instead of a real persistent instance get() never returns a proxy.

387. How to invoke stored procedure in hibernate?

- { ? = call thisISTheProcedure() }

388. What are the benefits of ORM?

- Productivity
- Maintainability
- Performance
- Vendor independence

389. What the Core INTERFACES are of hibernate framework?

- Session INTERFACE
- SessionFactory INTERFACE
- Configuration INTERFACE
- Transaction INTERFACE
- Query and Criteria INTERFACE

390. What is the file extension used for hibernate mapping file?

- The name of the file should be like this : filename.hbm.xml

391. What is the file name of hibernate configuration file?

- The name of the file should be like this : hibernate.cfg.xml

392. How hibernate is database independent explain?

- Only changing the property

```
 <property name="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</property> and

<property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
```

- full database can be replaced.

393. How to add hibernate mapping file in hibernate configuration file?

- By <mapping resource=" filename.hbm.xml"/>

394. Define connection pooling?

- Connection pooling is a mechanism reuse the connection.which contains the number of already created object connection. So whenever there is a necessary for object, this mechanism is used to directly get objects without creating it.

395. What is the hibernate proxy?

- An object proxy is just a way to avoid retrieving an object until you need it. Hibernate 2 does not proxy objects by default.

396. What do you create a SessionFactory?

```
Configuration cfg = new Configuration(); cfg.addResource("dir/hibernate.hbm.xml");
cfg.setProperties( System.getProperties() ); SessionFactory sessions =
cfg.buildSessionFactory();
```

397. What is HQL?

- HQL stands for Hibernate Query Language. Hibernate allows to the user to express queries in its own portable SQL extension and this is called as HQL. It also allows the user to express in native SQL.

398. What are the Collection types in Hibernate ?

- Set, List, Array, Map, Bag

399. What is a thin client?

- A thin client is a program **INTERFACE** to the application that does not have any operations
- like query of databases, execute complex business rules, or connect to legacy applications.

400. Differentiate between .ear, .jar and .war files.

- **.jar files:**contains the libraries, resources and accessories files like property files.
- **.war files:** contains jsp, html, javascript and other files for necessary for the development of web applications.
- **.ear files:** contains the EJB modules of the application.

401. What are the JSP tag?

- In JSP tags can be divided into 4 different types.
  - Directives
  - Declarations
  - Scriplets
  - Expressions

402. How to access web.xml init parameters from jsp page?

- For example, if you have:
- <context-param> <param-name>Id</param-name> <param-value>this is the value</param-value></context-param>
- You can access this parameter
- Id: <h:outputText value="#{initParam['Id']}"/>

403. What are JSP Directives?

- 1.page Directives <%@page language="java" %>
- 2. include Directives: <%@ include file="/header.jsp" %>
- 3. taglib Directives <%@ taglib uri="tlds/taglib.tld" prefix="html" %>

404. What will happen when you compile and run the following code?

```
public class MyClass { public static void main(String argv[]){ int array[]=new int[]{1,2,3};
System.out.println(array [1]); } }
```

- Compiled and shows output : 2

405. What is Struts?

- Struts framework is a Model-View-Controller(MVC) architecture for designing large scale applications. Which is combines of Java Servlets, JSP, Custom tags, and message. Struts helps you to create an extensible development environment for your application, based on published standards and proven design patterns. Model in many applications represent the internal state of the system as a set of one or more JavaBeans.The *View* is most often constructed using JavaServer Pages (JSP) technology.The Controller is focused on receiving requests from the client and producing the next phase of the user **INTERFACE** to an appropriate View component. The primary component of the Controller in the framework is a servlet of class `ActionServlet`. This servlet is configured by defining a set of `ActionMappings`.

406. What is ActionErrors?

- ActionErrors object that encapsulates any validation errors that have been found. If no errors are found, return null or an ActionErrors object with no recorded error messages.The default implementation attempts to forward

to the HTTP version of this method. Holding request parameters mapping and request and returns set of validation errors, if validation failed; an empty set or null

407. What is ActionForm?

- ActionForm is a Java bean that associates one or more ActionMappings. A java bean become FormBean when extend org.apache.struts.action.ActionForm class. ActionForm object is automatically populated on the server side which data has been entered by the client from UI. ActionForm maintains the session state for web application.

408. What is action mapping??

- In action mapping we specify action class for particular url ie path and diffrent target view ie forwards on to which request response will be forwarded.The **ActionMapping** represents the information that the **ActionServlet** knows about the mapping of a particular request to an instance of a particular **Action** class.The ***mapping*** is passed to the ***execute()*** method of the **Action** class, enabling access to this information directly.

409. What is the MVC on struts.

MVC stands Model-View-Controller.

Model: Model in many applications represent the internal state of the system as a set of one or more JavaBeans.

View: The *View* is most often constructed using JavaServer Pages (JSP) technology.

Controller: The Controller is focused on receiving requests from the client and producing the next phase of the user **INTERFACE** to an appropriate View component. The primary component of the Controller in the framework is a servlet of class `ActionServlet`. This servlet is configured by defining a set of `ActionMappings`.

410. What are different modules in spring?

- There are seven core modules in spring
  - The Core container module
  - O/R mapping module (Object/Relational)
  - DAO module
  - Application context module
  - Aspect Oriented Programming
  - Web module
  - MVC module

411. How to Create Object without using the keyword “new” in java?

- Without new the Factory mehtods are used to create objects for a class. For example

```
Calender c=Calender.getInstance();
```

- here Calender is a class and the method getInstance() is a Factory method which can create object for Calender class.

Servlets

412. What is servlet?

- Servlets is a server side components that provide a powerful mechanism for developing server side programs.
- Servlets is a server as well as platform-independent
- Servlets are designed for a various protocols. Most commonly used HTTP protocols.
- Servlets uses the classes in the java packages javax.servlet, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse, javax.servlet.http.HttpSession;.
- All servlets must implement the Servlet **INTERFACE** , which defines life-cycle methods.

413. Servlet is pure java object or not?

- Yes, pure java object.

414. What are the phases of the servlet life cycle?

- The life cycle of a servlet consists of the following phases:
  - Servlet class loading
  - Servlet instantiation
  - the init method
  - Request handling (call the service method)
  - Removal from service (call the destroy method)

415. What must be implemented by all Servlets?

- The Servlet **INTERFACE** must be implemented by all servlets

http://career.guru99.com/top-50-j2ee-interview-questions/

416. In web.xml file <load-on-startup>1</load-on-startup> is defined between <servlet></servlet> tag what does it means.

- Ans**:** whenever we request for any servlet the servlet container will initialize the servlet and load it which is defined in our config file called web.xml by default it will not initialize when our context is loaded .defining like this <load-on-startup>1</load-on-startup> is also known as pre-initialization of servlet means now the servlet for which we have defined this tag has been initialized in starting when context is loaded before getting any request.When this servlet question was asked to me in an interview few years back ,

417. How can we create deadlock condition on our servlet?

- call `doPost()` method inside `doGet()` OR
- doGet()method inside doPost()
- It will create deadlock situation for a servlet.

418. For initializing a servlet can we use a CONSTRUCTOR in place of init()?

- No, we can not use **CONSTRUCTOR** for initializing a servlet because for initialization we need an object of servletConfig using this object we get all the parameter which are defined in deployment descriptor for initializing a servlet and in servlet class we have only default **CONSTRUCTOR** according to older version of java so if we want to pass a Config object we don't have parametrized **CONSTRUCTOR** and apart from this servlet is loaded and initialized by container so it's a job of container to call the method according to servlet specification they have lifecycle method so `init()` method is called firstly.
- More important Java doesn't allow **INTERFACES** to declare **CONSTRUCTOR**s.

419. Why super.init (config) is the first statement inside init(config) method.

- Ans: This will be the first statement if we are overriding the `init(config)` method by this way we will store the config object for future reference and we can use by `getServletConfig()` to get information about config object if will not do this config object will be lost and we have only one way to get config object because servlet pass config object only in init method . Without doing this if we call the ServletConfig method will get **NullPointerException.**

420. Can we call destroy() method inside the init() method is yes what will happen?

- Yes we can call like this but if we have not overridden this method container will call the default method and nothing will happen.after calling this if any we have overridden the method then the code written inside is executed.

421. How can we refresh servlet on client and server side automatically?

- Ans: On the client side we can use Meta HTTP refresh and server side we can use server push.

422. How can you get the information about one servlet context in another servlet?

- YES In context object we can set the attribute which we want on another servlet and we can get that attribute using their name on another servlet.

```
Context.setAttribute ("name"," value")
Context.getAttribute ("name")
```

423. Why we need to implement Single Thread model in the case of Servlet.

- Ans: In J2EE we can implement our servlet in two different ways either by using:

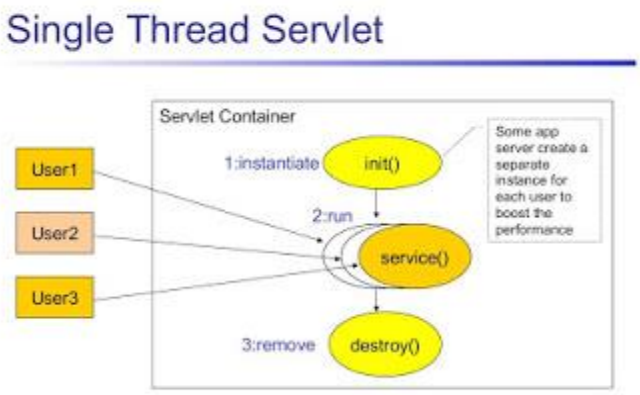
1. Single Thread Model
2. Multithread Model

Single Thread

- Depending upon our scenario, if we have implemented single thread means only one instance is going handle one request at a time no two thread will concurrently execute service method of the servlet.
- **The example** in banking accounts where sensitive data is handled mostly this scenario was used this **INTERFACE**is deprecated in Servlet API version 2.4.

Multithread

- As the name signifies multi-thread means a servlet is capable of handling multiple requests at the same time. This servlet interview question was quite popular few years back on entry level but now it's losing its shine.



424. What is servlet collaboration?

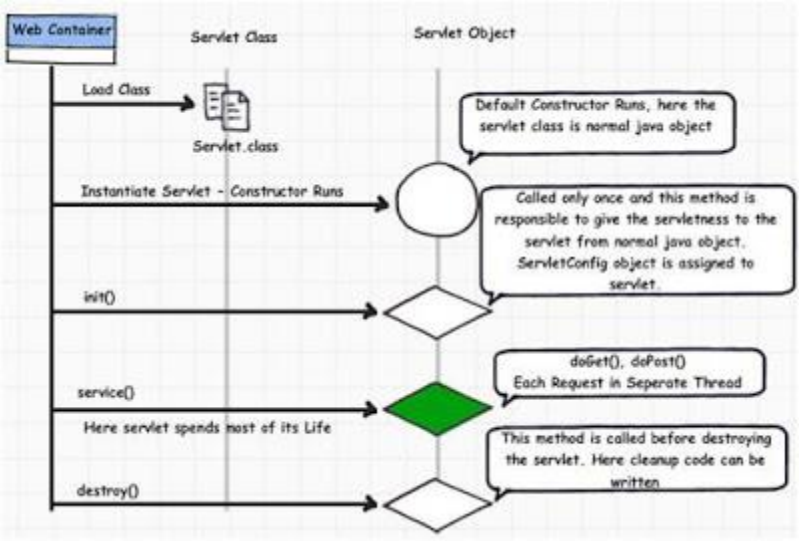
- Ans communication between two servlets is called servlet collaboration which is achieved by 3 ways.

1. RequestDispatchers include () and forward() method .
2. Using sendRedirect()method of Response object.
3. Using servlet Context methods

425. What is the difference between ServletConfig and ServletContext?

- **ServletConfig** as the name implies provide the information about the configuration of a servlet which is defined inside the web.xml file or we can say deployment descriptor.its a specific object for each servlet.
- **ServletContext** is an application specific object which is shared by all the servlet belongs to one application in one JVM .this is a single object which represents our application and all the servlet access application specific data using this object.servlet also use their method to communicate with the container.

426. Explain Servlet Life Cycle in Java EE environment?



427. What is the difference between HttpServlet and GenericServlet in Servlet API?

- GenericServlet provides framework to create a Servlet for any protocol e.g. you can write Servlet to receive content from FTP, SMTP etc,
- HttpServlet is built-in Servlet provided by Java for handling HTTP requests..

428. Java 8 – Convert a Stream to List

By mkyong | August 10, 2016

- A Java 8 example to show you how to convert a Stream to a List via Collectors.toList
- Java8Example1.java

```
package com.mkyong.java8;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;
public class Java8Example1 {
    public static void main(String[] args) {
        Stream<String> language = Stream.of("java", "python", "node");
        //Convert a Stream to List
        List<String> result = language.collect(Collectors.toList());

        result.forEach(System.out::println);
    }
}
```

output

java

python

node

429. filter a number 3 and convert it to a List.

Java8Example2.java

```
package com.mkyong.java8;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;
public class Java8Example2 {
    public static void main(String[] args) {
        Stream<Integer> number = Stream.of(1, 2, 3, 4, 5);
        List<Integer> result2 = number.filter(x -> x!= 3).collect(Collectors.toList());
        result2.forEach(x -> System.out.println(x));
    }
}
```

output

1

2

4

5

430. Java 8 – Filter a null value from a Stream

By mkyong | August 9, 2016 | Updated : August 10, 2016

- Review a `Stream` containing `null` values.
- Java8Examples.java

```
package com.mkyong.java8;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;
public class Java8Examples {
    public static void main(String[] args) {
        Stream<String> language = Stream.of("java", "python", "node", null, "ruby",
null, "php");
        List<String> result = language.collect(Collectors.toList());
        result.forEach(System.out::println);
    }
}
```

output

java

python

node

null // <--- NULL

ruby

null // <--- NULL

php

- Solution

To solve it, uses `Stream.filter(x -> x!=null)`

Java8Examples.java

```
package com.mkyong.java8;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;
public class Java8Examples {
    public static void main(String[] args) {
        Stream<String> language = Stream.of("java", "python", "node", null, "ruby",
null, "php");
        //List<String> result = language.collect(Collectors.toList());
        List<String> result = language.filter(x ->
x!=null).collect(Collectors.toList());
        result.forEach(System.out::println);
    }
}
```

output

java

python

node

ruby

php

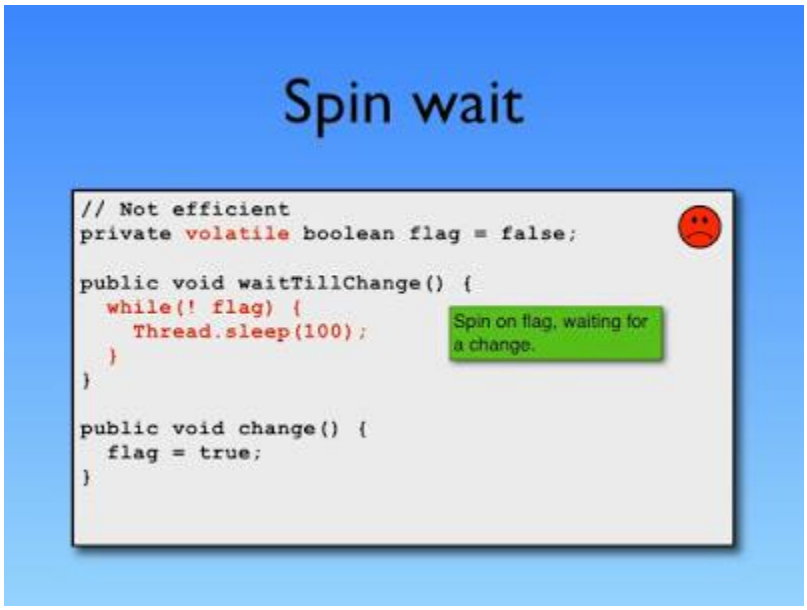
431. Alternatively, filter with `Objects::nonNull`

```
import java.util.List;
   List<String> result =
language.filter(Objects::nonNull).collect(Collectors.toList());
```

15 Core Java Questions For 5 to 6 Years Experienced

432. What is Busy Spinning? Why Should You Use It in Java?

- busy spinning is a *waiting strategy*, in which a thread just wait in a loop, without releasing the CPU for going to sleep. This is a very advanced and specialized waiting strategy used in the high-frequency trading application when wait time between two messages is very minimal.
- By not releasing the CPU or suspending the thread, your thread retains all the cached data and instruction, which may be lost if the thread was suspended and resumed back in a different core of CPU.
- This question is quite popular in high-frequency low latency programming domain, where programmers are trying for extremely low latency in the range of micro to milliseconds.



433. How to Make an Object IMMUTABLE in Java? Why Should You Make an Object IMMUTABLE?

- Immutability offers several advantages including thread-safety, ability to cache and result in more readable multithreading code. See here to learn how to make object IMMUTABLE. Once again, this question can also go into more detail and depending on your answer, can bring several other questions e.g. when you mention Spring is IMMUTABLE, be ready with some reasons on Why String is **IMMUTABLE** in Java.

434. Do you know about Open Closed Design Principle or Liskov Substitution Principle?

Design patterns are based on object-oriented design principles, which I strongly felt every object-oriented developer and the programmer should know, or, at least, have a basic idea of what are these principles and how they help you to write better object oriented code. I

f you don't know the answer to this question, you can politely say No, as it's not expected from you to know the answer to every question, but by answering this question, you can make your claim stronger as many experienced developers fail to answer basic questions like this. See Clean Code to learn more about object-oriented and SOLID design principles.

435. Which Design Pattern Will You Use to Shield Your Code From a Third Party library Which Will Likely to be Replaced by Another in Couple of Months?

This is just one example of the scenario-based design pattern interview question. In order to test the practical experience of Java

developers with more than 5 years experience, companies ask this kind of questions. You can expect more real-world design problems in different formats, some with more detail explanation with context, or some with only intent around.

One way to shield your code from third party library is to code against an **INTERFACE** rather than implementation and then use dependency injection to provide a particular implementation. This kind of questions is also asked quite frequently to experienced and senior *Java developers with 5 to 7 years of experience*.

436. How do you prevent SQL Injection in Java Code?

This question is more asked to J2EE and Java EE developers than core Java developers, but, it is still a good question to check the JDBC and Security skill of experienced Java programmers.

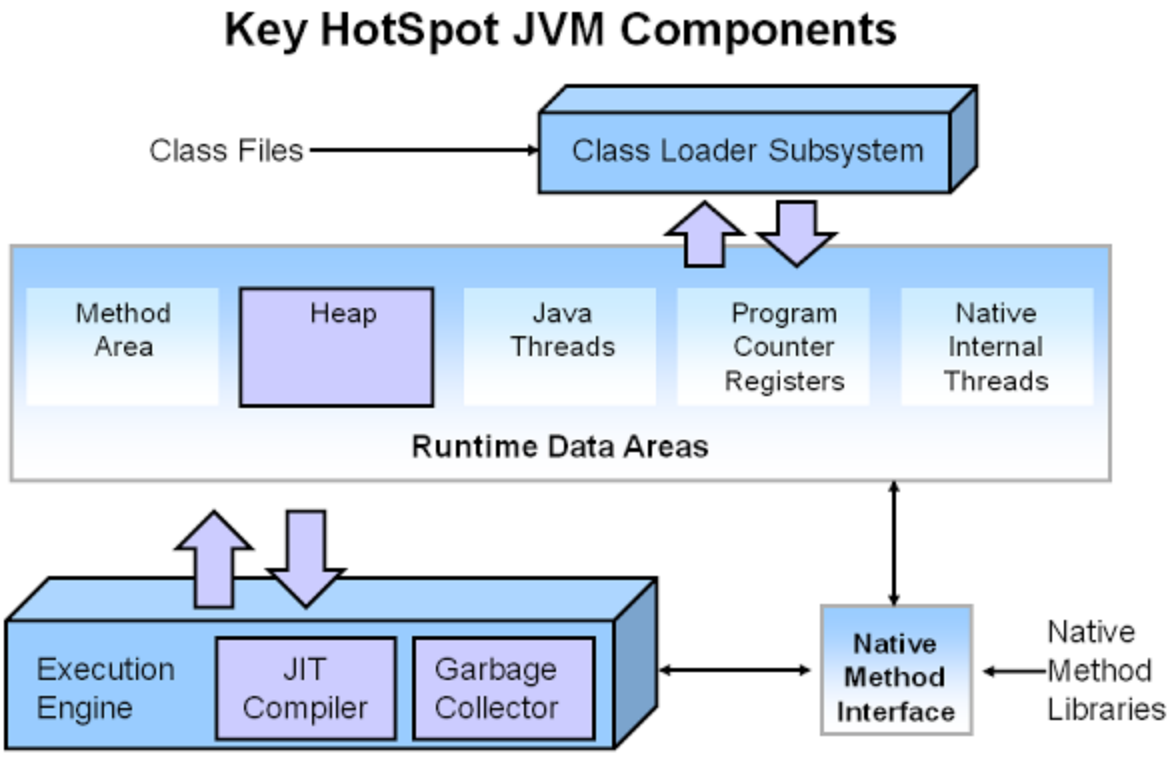
You can use PreparedStatement to avoid SQL injection in Java code. Use of the `PreparedStatement` for executing SQL queries not only provides better performance but also shield your Java and J2EE application from SQL Injection attack.

On a similar note, If you are working more on Java EE or J2EE side, then you should also be familiar with other security issues including *Session Fixation attack* or *Cross Site Scripting* attack and how to resolve them. These are some fields and questions where a good answer can make a lot of difference on your selection.

437. Tell me about different Reference types available in Java, e.g. WeakReference, SoftReference or PhantomReference? and Why should you use them?

Well, they are different reference types coming from `java.lang.ref` package and provided to assist Java Garbage Collector in a case of low memory issues. If you wrap an object with WeakReference than it will be eligible for garbage collected if there are o strong reference. They can later be reclaimed by Garbage collector if JVM is running low on memory.

The `java.util.WeakHashMap` is a special Map implementation, whose keys are the object of WeakReference, so if only Map contains the reference of any object and no other, those object can be garbage collected if GC needs memory. See Java Performance The Definitive Guide learn more about how to deal with performance issues in Java.



438. How does ConcurrentHashMap achieves its Scalability?

Sometimes this multithreading + collection interview question is also asked as, the difference between ConcurrentHashMap and **HashTable** in Java. The problem with **SYNCHRONIZED** HashMap or **HashTable** was that whole Map is locked when a thread performs any operation with Map.

The `java.util.ConcurrentHashMap` class solves this problem by using *lock stripping* technique, where the whole map is locked at different segments and only a particular segment is locked during the write operation, not the whole map. The `ConcurrentHashMap` also achieves it's scalability by allowing lock-free reads as read is a thread-safe operation. See here for more advanced multi-threading and concurrency questions in Java.

439. How do you share an object between threads? or How to pass an object from one thread to another?

There are multiple ways to do that e.g. Queues, Exchanger etc, but BlockingQueue using Producer Consumer pattern is the easiest way to pass an object from thread to another.

440. How do find if your program has a deadlock?

By taking thread dump using kill -3, using JConsole or VisualVM), I suggest to prepare this core java interview question in more detail, as Interviewer definitely likes to go with more detail e.g. they will press with questions like, have you really done that in your project or not?

441. How do you avoid deadlock while coding?

By ensuring locks are acquire and released in an ordered manner, see here for detail answer of this question.

Top 120 Java Interview Questions Answers

Date types and Basic Java Interview Questions

What is the right data type to represent a price in Java?

BigDecimal if memory is not a concern and Performance is not critical, otherwise double with predefined precision.

How do you convert bytes to String?

you can convert bytes to the string using string **CONSTRUCTOR** which accepts `byte[]`, just make sure that right character encoding otherwise platform's default character encoding will be used which may or may not be same.

How do you convert bytes to long in Java?

This questions if for you to answer :-)

Can we cast an int value into byte variable? what will happen if the value of `int` is larger than `byte`?

Yes, we can cast but int is 32 bit long in java while byte is 8 bit long in java so when you cast an int to byte higher 24 bits are lost and a byte can only hold a value from -128 to 128.

There are two classes B extends A and C extends B, Can we cast B into C e.g. C = (C) B;

Which class contains clone method? Cloneable or Object?

java.lang.Cloneable is marker **INTERFACE** and doesn't contain any method clone method is defined in the object class. It is also knowing that clone() is a native method means it's implemented in C or C++ or any other native language.

Is ++ operator is thread-safe in Java?

No it's not a thread safe operator because its involve multiple instructions like reading a value, incriminating it and storing it back into memory which can be overlapped between multiple threads.

Difference between a = a + b and a += b ?

The += operator implicitly cast the result of addition into the type of variable used to hold the result. When you add two integral variable e.g. variable of type byte, short, or int then they are first promoted to int and them addition happens. If result of addition is more than maximum value of a then a + b will give compile time error but a += b will be ok as shown below

```
byte a = 127;

byte b = 127;

b = a + b; // error : cannot convert from int to byte

b += a; // ok
```

Can I store a double value in a long variable without casting?

No, you cannot store a double value into a long variable without casting because the range of double is more that long and you we need to type cast. It's not dificult to answer this question but many develoepr get it wrong due to confusion on which one is bigger between double and long in Java.

What will this return 3*0.1 == 0.3? true or false?

This is one of the really tricky questions. Out of 100, only 5 developers answered this question and only of them have explained the concept correctly. The short answer is false because some floating point numbers can not be represented exactly.

Which one will take more memory, an int or Integer?

An Integer object will take more memory an Integer is the an object and it store meta data overhead about the object and int is primitive type so its takes less space.

442. Why String is IMMUTABLE in Java? - MetLife

- **IMMUTABLE** because String objects are cached in String **pool**. it can safely shared between many threads ,which is very important for **multithreaded programming** and to avoid any **synchronization** issues in Java,
- HashMap. Since Strings are very popular as **HashMap key**, it's important for them to be **IMMUTABLE** so that they can retrieve the value object which was stored in HashMap. Since HashMap works in principle of hashing, which requires same has value to function properly

443. Can we use String in the switch case?

Yes from Java 7 onward we can use String in switch case but it is just syntactic sugar. Internally string hash code is used for the switch. See the detaiedl answer for more explanation and discussion.

JVM Internals and Garbage Collection Interview Questions

444. What is the size of int in 64-bit JVM?

- The size of an int variable is constant in Java, it's always 32-bit irrespective of platform. Which means the size of primitive `int` is same in both 32-bit and 64-bit Java virtual machine.

445. The difference between Serial and Parallel Garbage Collector?

- Even though both the serial and parallel collectors cause a stop-the-world pause during Garbage collection. The main difference between them is that a serial collector is a default copying collector which uses only one GC thread for garbage collection while a parallel collector uses multiple GC threads for garbage collection.

446. What is the size of an int variable in 32-bit and 64-bit JVM?

- The size of int is same in both 32-bit and 64-bit JVM, it's always 32 bits or 4 bytes.

447. A difference between WeakReference and SoftReference in Java?

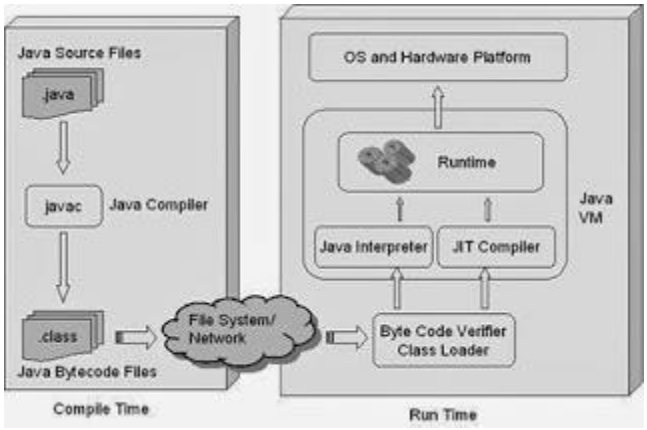
- Though both WeakReference and SoftReference helps garbage collector and memory efficient,
- WeakReference becomes eligible for garbage collection as soon as last strong reference is lost
- SoftReference even thought it can not prevent garbage collector, it can delay it until JVM absolutely need memory.

448. How do you find if JVM is 32-bit or 64-bit from Java Program?

- You can find that by checking some system properties like sun.arch.data.model or os.arch

449. What is the difference between JRE, JDK, JVM and JIT?

- JRE stands for Java run-time and it's required to run Java application. JDK stands for Java development kit and provides tools to develop Java program e.g. Java compiler. It also contains JRE. The JVM stands for Java virtual machine and it's the process responsible for running Java application. The JIT stands for Just In Time compilation and helps to boost the performance of Java application by converting Java byte code into native code when the crossed certain threshold i.e. mainly hot code is converted into native code.



Basic Java concepts Interview Questions

450. What's the difference between "a == b" and "a.equals(b)"?

- The a = b does object reference matching if both a and b are an object and only return true if both are pointing to the same object in the heap space, on the other hand, a.equals(b) is used for logical mapping and its expected from an object to override this method to provide logical equality. For example, String class overrides this equals() method so that you can compare two Strings, which are the different object but contains same letters.

451. What is a.hashCode() used for? How is it related to a.equals(b)?

- hashCode() method returns an int hash value corresponding to an object. It's used in hash based collection classes e.g **HashTable**, HashMap, LinkedHashMap and so on. It's very tightly related to equals() method. According to Java specification, two objects which are equal to each other using equals() method must have same hash code.

452. What is a compile time constant in Java? What is the risk of using it?

- public static final variables are also known as a compile time constant, the public is optional there. They are replaced with actual values at compile time because compiler know their value up-front and also knows that it cannot be changed during run-time. One of the problem with this is that if you happened to use a public static final variable from some in-house or third party library and their value changed later than your client will still be using old value even after you deploy a new version of JARs. To avoid that, make sure you compile your program when you upgrade dependency JAR files.
-

Date, Time and Calendar Interview questions in Java

453. Does SimpleDateFormat is safe to use in the multi-threaded program?

- No, unfortunately, DateFormat and all its implementations including SimpleDateFormat is not thread-safe, hence should not be used in the multi-threaded program until external thread-safety measures are applied e.g. confining SimpleDateFormat object into a `ThreadLocal` variable. If you don't do that, you will get an incorrect result while parsing or formatting dates in Java. Though, for all practical date time purpose, I highly recommend **`joda-time`** library.

454. How do you format a date in Java? e.g. in the ddMMyyyy format?

- You can either use `SimpleDateFormat` class or joda-time library to format date in Java. DateFormat class allows you to format date on many popular formats. Please see the answer for code samples to format date into different formats e.g. `dd-MM-yyyy` or `ddMMyyyy`.

Override

455. Can you override private or static method in Java,

- NO, you cannot override private or static method

456. if you create similar method with same return type and same method arguments that's called method hiding What is function overriding?

- If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding.
- based on **POLYMORPHISM** OOPS concept which allows programmer to create two methods with same name and method
- only `override` method in sub class. You cannot override method in same class.
- overriding method cannot reduce accessibility of overridden method in Java. For example if overridden method is public than overriding method cannot be protected, private or package-private; But opposite is true overriding method can increase accessibility of method in Java, i.e. if overridden method is protected than overriding method can be protected or public.
- Overridden method is called using dynamic binding in Java at runtime.

457. Can we override private methods in Java? NO

1. **Example of public Overriding or Runtime POLYMORPHISM.**

```
class Base {
  public void fun() {
System.out.println("Base fun");
  }
}

class Derived extends Base {
  public void fun() {  // overrides the Base's fun()
System.out.println("Derived fun");
  }
  public static void main(String[] args) {
 Base obj = new Derived();
 obj.fun();
  }
}
```

2. **Example of Private Overriding or Runtime POLYMORPHISM.**

```
class Base {
  private void fun() {
System.out.println("Base fun");
  }
}

class Derived extends Base {
  private void fun() {   compiler error "fun() has private access in Base"
System.out.println("Derived fun");
  }
  public static void main(String[] args) {
 Base obj = new Derived();
 obj.fun();
  }
}
```

- So the compiler tries to call base class function, not derived class, means fun() is not overridden.

458. Can we override static method in Java is also a popular Java question asked in interviews.

```
public class TradingSystem {
    public static void main(String[] args) {
   TradingSystem system = new DirectMarketAccess();
```

```
   DirectMarketAccess dma = new DirectMarketAccess();

   // static method of Instrument class will be called,
   // even though object is of sub-class DirectMarketAccess
   system.printCategory();

   //static method of EquityInstrument class will be called
   dma.printCategory();
    }

    public static void printCategory(){
   System.out.println("inside super class static method");
    }
}

class DirectMarketAccess extends TradingSystem{
    public static void printCategory(){
   System.out.println("inside sub class static method");
    }
}
Output:
inside super class static method
inside sub class static method
```

- This shows that static method cannot be overridden in Java and concept of method overloading doesn't apply to static methods. Instead declaring same static method on Child class is known as method hiding in Java.
- **If you try to override a static method with a non-static method in sub class you will get compilation error.**
- Be careful while using `static` keyword in multi-threading or concurrent programming because most of the issue arises of concurrently modifying a static variable by different threads resulting in working with stale or incorrect value if not properly synchronized. most common issue is race condition, which occurs due to poor synchronization or no synchronization of static variable.

459. If a method throws NullPointerException in super class, can we override it with a method which throws RuntimeException?

- YES. you can throw super class of RuntimeException in overridden method but you can not do same if its checked Exception.
-

460. If a method throws NullPointerException in super class, can we override it with a method, which throws RuntimeException?

- One more tricky Java questions from overloading and overriding concept. Answer is you can very well throw super class of RuntimeException in overridden method but you cannot do it if it's checked Exception. See Rules of method overriding in Java for more details.

Overload

461. What restrictions are placed on method overloading?

- Two methods may not have the same name and argument list but different return types.

462. What is function overloading?

- Single Class 2+ methods have the same name
- AND either of these 2 are true
- - The **number** of parameters is different for the methods.
- - The parameter **types** are different (like changing a parameter that was a float to an int).

```
public void  show(String message){
public void  show(String message, boolean show){ OK number of argument is different
 public void  show(Integer message){    OK type of argument is different
 int changeDate(int Year, int Month) ;
 int changeDate(int Year);    OK number of argument is different
 int changeDate(float Year) ;
 int changeDate(int Year);    OK type of argument is different
```

463. Which object oriented Concept is achieved by using overloading and overriding?

- **POLYMORPHISM**

464. Describe overloading and overriding in Java?

Both overloading and overriding allow you to write two methods of different functionality but with the same name, but overloading is compile time activity while overriding is run-time activity. Though you can overload a method in the same class, but you can only override a method in child classes. **INHERITANCE** is necessary for overriding.

465. Can you override private or static method in Java ?

- No, you can not override static method in Java,
- Though you can declare method with same signature in sub class. It won't be overridden in exact sense, instead that is called **method hiding**.
- But at same time, you can overload static methods in Java, there is nothing wrong declaring static methods with same name, but different arguments. Some time interviewer also ask,
- *Why you can not override static methods in Java*? Answer of this question lies on time of resolution. As I said in difference between static and dynamic binding , static method are bonded during compile time using Type of reference variable, and not Object. If you have using IDE like Netbeans and Eclipse, and If you try to access static methods using an object, you will see warnings. As per Java coding convention,
- static methods should be accessed by class name rather than object.

466. Can we overload main() method?

- The normal main method acts as an entry point for the JVM to start the execution of program.
- We can overload the main method in Java. But the program doesn't execute the overloaded main method when we run your program, we need to call the overloaded main method from the actual main method only.

Override vs Overload

467. When do you overload a method in Java and when do you override it ?

- different implementation of a class has different way of doing certain thing than overriding is the way to go
- overloading is doing same thing but with different input. method signature varies in case of overloading but not in case of overriding in java.

468. Describe overloading and overriding in Java?

- Both overloading and overriding allow you to write two methods of different functionality but with the same name, but overloading is compile time activity while overriding is run-time activity. Though you can overload a method in the same class, but you can only override a method in child classes. **INHERITANCE** is necessary for overriding.

469. Can we Overload or Override static methods in java ?

- NO you can not override a private or static method in Java,
- **Overriding :** Overriding is related to run-time **POLYMORPHISM**. A subclass (or derived class) provides a specific implementation of a method in superclass (or base class) at runtime.
- **Overloading:** Overloading is related to compile time (or static) **POLYMORPHISM**. This feature allows different methods to have same name, but different signatures, especially number of input parameters and type of input paramaters.
- **Can we overload static methods?** The answer is **'Yes'**. We can have two ore more static methods with same name, but differences in input parameters
- **Can we Override static methods in java? NO** 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**. Hence the answer is **'No'**. Static methods cannot be overridden because method overriding only occurs in the context of dynamic (i.e. runtime) lookup of methods. Static methods (by their name) are looked up statically (i.e. at

compile-time).

470. If a method throws NullPointerException in the superclass, can we override it with a method which throws RuntimeException?

- **Yes**, throw superclass of RuntimeException in overridden method, but you **can not do same if its checked Exception.**

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

- **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**.

```
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
 }
}
```

-

472. The difference between nested public static class and a top level class in Java?

- You can have more than one nested public static class inside one class, but you can only have one top-level public class in a Java source file and its name must be same as the name of Java source file.

473. Difference between Composition, Aggregation and Association in OOP?

- If two objects are related to each other, they are said to be associated with each other. Composition and Aggregation are two forms of association in object-oriented programming. The composition is stronger association than Aggregation. In Composition, one object is OWNER of another object while in Aggregation one object is just USER of another object. If an object A is composed of object B then B doesn't exist if A ceased to exists, but if object A is just an aggregation of object B then B can exists even if A ceased to exist.

Patterns Java Interview questions from OOP and Design Patterns

474. Which design pattern have you used in your production code? apart from Singleton?

- This is something you can answer from your experience. You can generally say about dependency injection, factory pattern, decorator pattern or observer pattern, whichever you have used. Though be prepared to answer follow-up question based upon the pattern you choose.

475. Can you explain Liskov Substitution principle?

- This is one of the toughest questions I have asked in Java interviews. Out of 50 candidates, I have almost asked only 5 have managed to answer it. I am not posting an answer to this question as I like you to do some research, practice and spend some time to understand this confusing principle well.

476. What is Law of Demeter violation? Why it matters?

- Believe it or not, Java is all about application programming and structuring code. If you have good knowledge of common coding best practices, patterns and what not to do than only you can write quality code. Law of Demeter suggests you "talk to friends and not stranger", hence used to reduce coupling between classes.

477. What is Adapter pattern? When to use it?

- Another frequently asked Java design pattern questions. It provides **INTERFACE** conversion. If your client is using some **INTERFACE** but you have something else, you can write an Adapter to bridge them together. This is good for Java software engineer having 2 to 3 years experience because the question is neither difficult nor tricky but requires knowledge of OOP design patterns.

478. Which one is better CONSTRUCTOR injection or setter dependency injection?

- Each has their own advantage and disadvantage. **CONSTRUCTOR** injection guaranteed that class will be initialized with all its dependency, but setter injection provides flexibility to set an optional dependency. Setter injection is also more readable if you are using an XML file to describe dependency. Rule of thumb is to use **CONSTRUCTOR** injection for mandatory dependency and use setter injection for optional dependency.

479. What is difference between dependency injection and factory design pattern?

- Though both patterns help to take out object creation part from application logic, use of dependency injection results in cleaner code than factory pattern. By using dependency injection, your classes are nothing but Plain old java object (POJO) which only knows about dependency but doesn't care how they are acquired. In the case of factory pattern, the class also needs to know about factory to acquire dependency. hence, DI results in more testable classes than factory pattern. Please see the answer for a more detailed discussion on this topic.

480. Difference between Adapter and Decorator pattern?

- Though the structure of Adapter and Decorator pattern is similar, the difference comes on the intent of each pattern. The adapter pattern is used to bridge the gap between two **INTERFACES**, but Decorator pattern is used to add new functionality into the class without the modifying existing code.

481. Difference between Adapter and Proxy Pattern?

- Similar to the previous question, the difference between Adapter and Proxy patterns is in their intent. Since both Adapter and Proxy pattern encapsulate the class which actually does the job, hence result in the same structure, but Adapter pattern is used for **INTERFACE** conversion while the Proxy pattern is used to add an extra level of indirection to support distribute, controlled or intelligent access.

482. What is Template method pattern?

- Template pattern provides an outline of an algorithm and lets you configure or customize its steps. For examples, you can view a sorting algorithm as a template to sort object. It defines steps for sorting but let you configure how to compare them using Comparable or something similar in another language. The method which outlines the algorithms is also known as template method.

483. When do you use Visitor design pattern?

- The visitor pattern is a solution of problem where you need to add operation on a class hierarchy but without

touching them. This pattern uses double dispatch to add another level of indirection.

484. When do you use Composite design pattern?

- Composite design pattern arranges objects into tree structures to represent part-whole hierarchies. It allows clients treat individual objects and container of objects uniformly. Use Composite pattern when you want to represent part-whole hierarchies of objects.

485. The difference between INHERITANCE and Composition?

- Though both allows code reuse, Composition is more flexible than **INHERITANCE** because it allows you to switch to another implementation at run-time. Code written using Composition is also easier to test than code involving **INHERITANCE** hierarchies.

486. Give me an example of design pattern which is based upon open closed principle?

- Open closed design principle asserts that your code should be open for extension but closed for modification. Which means if you want to add new functionality, you can add it easily using the new code but without touching already tried and tested code. There are several design patterns which are based upon open closed design principle e.g.
- Strategy pattern if you need a new strategy, just implement the **INTERFACE** and configure, no need to modify core logic.
- One working example is `Collections.sort()` method which is based on Strategy pattern and follows the open-closed principle, you don't modify `sort()` method to sort a new object, what you do is just implement Comparator in your own way.

487. When do you use Flyweight pattern?

- share object to support large numbers without actually creating too many objects.
- need to make your object **IMMUTABLE** so that they can be safely shared.
- String pool and pool of Integer and Long object JDK examples

Miscellaneous Java Interview Questions

- It contains XML Processing in Java Interview question, JDBC Interview question, Regular expressions Interview questions, Java Error and Exception Interview Questions, Serialization,

488. The difference between nested static class and top level class?

- One of the fundamental questions from Java basics. I ask this question only to junior Java developers of 1 to 2 years of experience as it's too easy for an experience Java programmers. The answer is simple, a public top level class must have the same name as the name of the source file, there is no such requirement for nested static class. A nested class is always inside a top level class and you need to use the name of the top-level class to refer nested static class e.g. HashMap.Entry is a nested static class, where HashMap is a top level class and `Entry` is nested static class.

489. Can you write a regular expression to check if String is a number? (solution)

- If you are taking Java interviews then you should ask at least one question on the regular expression. This clearly differentiates an average programmer with a good programmer. Since one of the traits of a good developer is to know tools, regex is the best tool for searching something in the log file, preparing reports etc. Anyway, answer to this question is, a numeric String can only contain digits i.e. 0 to 9 and + and - sign that too at start of the String, by using this information you can write following regular expression to check if given String is number or not

490. The difference between DOM and SAX parser in Java?

- Another common Java question but from XML parsing topic. It's rather simple to answer and that's why many interviewers prefers to ask this question on the telephonic round. DOM parser loads the whole XML into

memory to create a tree based DOM model which helps it quickly locate nodes and make a change in the structure of XML while SAX parser is an event based parser and doesn't load the whole XML into memory. Due to this reason DOM is faster than SAX but require more memory and not suitable to parse large XML files.

491. What is the difference between Maven and ANT in Java?

- Though both are build tool and used to create Java application build, Maven is much more than that. It provides standard structure for Java project based upon "convention over configuration" concept and automatically manage dependencies (JAR files on which your application is dependent) for Java application.

492. What is the difference between System.out, System.err, and System.in?

Your answer: System.out and System.err represent the monitor by default and thus can be used to send data or results to the monitor. System.out is used to display normal messages and results. System.eerr is used to display error messages. System.in represents InputStream object which by default represents standard input device, i.e., keyboard.

493. What is object cloning?

- Object cloning means to create an exact copy of the original object. If a class needs to support cloning, it must implement java.lang.Cloneable **INTERFACE** and override clone() method from Object class. Syntax of the clone() method is :

```
protected Object clone() throws CloneNotSupportedException
```

494. How to convert JSON to XML

- **view online https://codebeautify.org/jsontoxml**
- in code **JSON-Java library from json.org**
- `JSONObject json = new JSONObject(str);`
- `String xml = XML.toString(json);`
- `toString` can take a second argument to provide the name of the XML root node.

495. How to convert XML to JSON

- Use the (excellent) **JSON-Java library from json.org** then
- XML to JSON using `XML.toJSONObject(java.lang.String string)`

496. Write JSON Parser. What are the Data Structures used for this? - MetLife

- JSON: JavaScript Object Notation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.
- Are you talking about receiving and parsing the JSON string from a server request?
- For that you can use:

```
import org.json.JSONArray;
import org.json.JSONObject;
```

- Using these, I read through my JSON array from my POST request and store the resulting information in Class objects in my project.
- For each item in JSONArray, you can extract the JSONObject and attributes like this:
- JSON Parser

```
for (int i = 0; i < jsonArray.length(); i++) {
    JSONObject jsonObject = jsonArray.getJSONObject(i);
    jsonObject.getString("text");
}
```

As far as actually storing the data

store the data using `Map<String, Object>`

497. How to Send Data – Use JSON

- If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server: ex

```
<!DOCTYPE html>
<html>
<body>
<h2>Convert a JavaScript object into a JSON string, and send it to the server.</h2>
<script>
var myObj = { "name":"John", "age":31, "city":"New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
</script>
</body>
</html>
```

- output on the screen is

```
demo_json.php:
John from New York is 31
```

498. How to receive Data – Use Json

- If you receive data in JSON format, you can convert it into a JavaScript object:
- Example

```
var myJSON = '{ "name":"John", "age":31, "city":"New York" }';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myobj.name;
```

499. How to Store JSON data received - Storing Data

- When storing data, the data has to be a certain format, and regardless of were you choose to store it, *text* is always one of the legal formats.
- JSON makes it possible to store JavaScript objects as text.
- Storing data in local storage

```
//Storing data:
myObj = { "name":"John", "age":31, "city":"New York" };
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
//Retrieving data:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

- JSON Data - A Name and a Value
- JSON data is written as name/value pairs.

```
"name":"John"
```

- JSON names require double quotes. JavaScript names don't.
- JSON - Evaluates to JavaScript Objects
- JSON Values
- In JSON, *values* must be one of the following data types:
    - a string
    - a number
    - an object (JSON object)
    - an array
    - a boolean
    - null
- In JavaScript values can be all of the above, plus any other valid JavaScript expression, including:
    - a function
    - a date
    - undefined
- In JSON, *string values* must be written with double quotes:

```
{ "name":"John" }
```

- In JavaScript, you can write string values with double *or* single quotes:

```
{ name:'John' }
```

500. Similarities JSON vs XML

- Both JSON and XML can be used to receive data from a web server.
- The following JSON and XML examples both defines an employees object, with an array of 3 employees:
- JSON Example

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

- XML Example

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

- JSON is Like XML Because
  - Both JSON and XML are "self describing" (human readable)
  - Both JSON and XML are hierarchical (values within values)
  - Both JSON and XML can be fetched with an XMLHttpRequest

501. Diff JSON and XML

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays
- The biggest difference is:

XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

http://stackoverflow.com/questions/13866039/what-is-the-most-suitable-java-data-structure-for-representing-json

502. CI – Continuous Integration tools, CI is part of CD

- Jenkins - Blue Ocean
- Continuous Integration tools a.k.a build servers
- Continuous integration – the practice of frequently integrating one's new or changed code with the existing code repository
- Normal practice is to trigger these builds by every commit to a repository, rather than a periodically scheduled build. The practicalities of doing this in a multi-developer environment of rapid commits are such that it is usual to trigger a short time after each commit, then to start a build when either this timer expires, or after a rather longer interval since the last build.
- Travis CI is the pioneer in this field of cloud Continuous Integration tools and is leading the way in popularity still. CircleCI seems to have gained some traction in the recent months and is climbing its way up.
- Continuous deployment means that every change is automatically deployed to production.

503. Name CD – Continuous Development tools

- teams produce software in short cycles, ensuring that the software can be reliably released at any time.[1] It aims at building, testing, and releasing software faster and more frequently

- Jenkins too
- Continuous Development is part of Continuous Delivery
- Continuous delivery is enabled through the deployment pipeline

504. Name Version Control tools

- GIT
- Subversion
- Team Foundation Server

Deutsche Bank Interview Questions following:

- https://www.glassdoor.com/Interview/Deutsche-Bank-Software-Developer-Interview-Questions-EI_IE3150.0,13_KO14,32.htm

505. Then a coding question - reverse last n nodes of a linked list.

```
/* 1) Divide the list in two parts - first node and rest of the linked list.
   2) Call reverse for the rest of the linked list.
   3) Link rest to first.
   4) Fix head pointer.
*/

package com.java2novice.ds.linkedlist;

public class SinglyLinkedListImpl<T> {

    private Node<T> head;

    public void add(T element){

        Node<T> nd = new Node<T>();
        nd.setValue(element);
        System.out.println("Adding: "+element);
        Node<T> tmp = head;
        while(true){
            if(tmp == null){
                //since there is only one element, both head and
                //tail points to the same object.
                head = nd;
                break;
            } else if(tmp.getNextRef() == null){
                tmp.setNextRef(nd);
                break;
            } else {
                tmp = tmp.getNextRef();
            }
        }
    }

    public void traverse(){

        Node<T> tmp = head;
        while(true){
            if(tmp == null){
                break;
            }
            System.out.print(tmp.getValue()+"\t");
            tmp = tmp.getNextRef();
        }
    }

    public void reverse(){

        System.out.println("\nreversing the linked list\n");
        Node<T> prev = null;
        Node<T> current = head;
        Node<T> next = null;
        while(current != null){
            next = current.getNextRef();
            current.setNextRef(prev);
            prev = current;
            current = next;
        }
        head = prev;
    }

    public static void main(String a[]){
        SinglyLinkedListImpl<Integer> sl = new SinglyLinkedListImpl<Integer>();
        sl.add(3);
        sl.add(32);
        sl.add(54);
        sl.add(89);
        System.out.println();
        sl.traverse();
        System.out.println();
        sl.reverse();
```

```
        sl.traverse();
    }
}

class Node<T> implements Comparable<T> {

    private T value;
    private Node<T> nextRef;

    public T getValue() {
        return value;
    }
    public void setValue(T value) {
        this.value = value;
    }
    public Node<T> getNextRef() {
        return nextRef;
    }
    public void setNextRef(Node<T> ref) {
        this.nextRef = ref;
    }
    @Override
    public int compareTo(T arg) {
        if(arg == this.value){
            return 0;
        } else {
            return 1;
        }
    }
```

- Using object notation (JavaScript Object Notation (or JSON), how would you solve this problem. Then the problem was posed

506. private constructors impact

Private constructors prevent a class from being explicitly instantiated by its callers.

There are some common cases where a private constructor can be useful:

- classes containing only static utility methods
- classes containing only constants
- type safe enumerations
- singletons

These examples fall into two categories.

1. **Object construction is entirely forbidden**
   - No objects can be constructed, either by the caller or by the native class. This is only suitable for classes that offer only *static* members to the caller.
   - In these cases, the lack of an accessbile constructor says to the caller: "There are no use cases for this class where you need to build an object. You can only use static items. I am preventing you from even *trying* to build an object of this class." See class for constants for an illustration.
   - If the programmer does not provide a constructor for a class, then the system will always provide a default, *public* no-argument constructor. To disable this default constructor, simply add a *private* no-argument constructor to the class. This private constructor may be empty. Somewhat paradoxically, creation of objects by the caller is in effect disabled by the addition of this private no-argument constructor.
2. **Object construction is private only**
   - Objects can be constructed, but only internally. For some reason, a class needs to prevent the caller from creating objects.
   - In the case of a singleton, the policy is that only one object of that class is supposed to exist. Creation of multiple objects of that class is forbidden.
   - In the case of JDK 1.4 type-safe enumerations, more than one object can be created, but again, there is a limitation. Only a specific set of objects is permitted, one for each element of the

enumeration. (For JDK 1.5 enums, the creation of such objects is done implicitly by the Java language, not explicitly by the application programmer.)

- database joins types of joins – see SQL notes
- some UML questions
- JMS questions – see JMS file
- You are expected to know JMM better that its authors. see JVM, heap and stack

507. how do you call other webservices in you api.

- Answer: Using apache httpclient. Restful web services, SOAP web services
- what is segregation and aggregation - see Difference between Association, Composition and Aggregation?

508. what is guava cache? why and how you used it?how you invalidated it?

- Guava is an open source, Java-based library developed by Google. It facilitates best coding practices and helps reduce coding errors. It provides utility methods for collections, caching, primitives support, concurrency, common annotations, string processing, I/O, and validations. This tutorial adopts a simple and intuitive way to describe the basic-to-advanced concepts of Guava and how to use its APIs.
- Benefits of Guava
  - Standardized − The Guava library is managed by Google.
  - Efficient − It is a reliable, fast, and efficient extension to the Java standard library.
  - Optimized − The library is highly optimized.
  - Functional Programming − It adds functional processing capability to Java.
  - Utilities − It provides many utility classes which are regularly required in programming application development.
  - Validation − It provides a standard failsafe validation mechanism.
  - Best Practices − It emphasizes on best practices.
- Explicit Removals
- At any time, you may explicitly invalidate cache entries rather than waiting for entries to be evicted. This can be done:
  - individually, using `Cache.invalidate(key)`
  - in bulk, using `Cache.invalidateAll(keys)`
  - to all entries, using `Cache.invalidateAll()`

509. what is distributed caching?

- in computing, a **distributed cache** is an extension of the traditional concept of cache used in a single locale. A distributed cache may span multiple servers so that it can grow in size and in transactional capacity. It is mainly used to store application data residing in database and web session data. The idea of distributed caching[1] has become feasible now because main memory has become very cheap and network cards have become very fast, with 1 Gbit now standard everywhere and 10 Gbit gaining traction. Also, a distributed cache works well on lower cost machines usually employed for web servers as opposed to database servers which require expensive hardware.[2] An emerging internet architecture known as Information-centric networking (ICN) is one of the best example of distributed cache network. The ICN is a network level solution hence the existing distributed network cache management schemes are not well suited for ICN.[3]
- Oracle Coherence

510. mutable class?

- Mutable objects have fields that can be changed, immutable objects have no fields that can be changed after

the object is created.

- A very simple immutable object is a object without any field. (For example a simple Comparator Implementation).

```
class Mutable{
  private int value;

  public Mutable(int value) {
     this.value = value;
  }

  getter and setter for value
}

class Immutable {
  private final int value;

  public Immutable(int value) {
     this.value = value;
  }

  only getter
}
```

- annotations of rest and spring - see rest doc

511. how to return pojo from rest service

- What you're doing with `get(String.class)` is saying that the response stream should be deserialized into a Sting type response. This is not a problem, as JSON is inherently just a String. But if you want to unmarshal it to a POJO, then you need to have a `MessageBodyReader` that knows how to unmarshal JSON to your POJO type. Jackson provides a `MessageBodyReader` in it's `jackson-jaxrs-json-provider` dependency

```
<dependency>
  <groupId>com.fasterxml.jackson.jaxrs</groupId>
  <artifactId>jackson-jaxrs-json-provider</artifactId>
  <version>2.4.0</version>
</dependency>
```

- Most implementations will provider a wrapper for this module, like `jersey-media-json-jackson` for Jersey or `resteasy-jackson-provider` for Resteasy. But they are still using the underlying `jackson-jaxrs-json-provider`.
-

512. Question based on String tokeniser.

public class StringTokenizer

extends Object

implements Enumeration<Object>

The string tokenizer class allows an application to break a string into tokens. The tokenization method is much simpler than the one used by the `StreamTokenizer` class. The `StringTokenizer` methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments.

The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.

An instance of `StringTokenizer` behaves in one of two ways, depending on whether it was created with the `returnDelims` flag having the value `true` or `false`:

- If the flag is `false`, delimiter characters serve to separate tokens. A token is a maximal sequence of consecutive characters that are not delimiters.
- If the flag is `true`, delimiter characters are themselves considered to be tokens. A token is thus either one delimiter character, or a maximal sequence of consecutive characters that are not delimiters.

A `StringTokenizer` object internally maintains a current position within the string to be tokenized. Some operations

advance this current position past the characters processed.

A token is returned by taking a substring of the string that was used to create the `StringTokenizer` object.

The following is one example of the use of the tokenizer. The code:

```
    StringTokenizer st = new StringTokenizer("this is a test");
    while (st.hasMoreTokens()) {
        System.out.println(st.nextToken());
    }
```

prints the following output:

```
this
is
a
test
```

`StringTokenizer` is a legacy class that is retained for compatibility reasons although its use is discouraged in new code. It is recommended that anyone seeking this functionality use the `split` method of `String` or the java.util.regex package instead.

The following example illustrates how the **`String.split`** method can be used to break up a string into its basic tokens:

```
String[] result = "this is a test".split("\\s");
    for (int x=0; x<result.length; x++)
        System.out.println(result[x]);
```

prints the following output:

```
this
is
a
test
```

- Check for Compile-time/Runtime errors in the programs given in online test.
- Questions in computer basics on array , algo , BSt traversal, prefix n postfix, n mostly engineering basics .

Deutsche Bank Interview Questions ends