Que.1. Why variables in interface are public, static, final?

: **Static:**We don't create instance of Interface, so for accessing this variables we have only interface name, for that purpose: variables in java are static.
final: To make them constants. If 2 classes implement the same interface and you give both of them the right to change the value, conflict will occur in the current value of the var, which is why only one time initialization is permitted.

Que.2. Why variables in interface don't need to declare in implementing classes?
:Because, variables in java are by default public static final, so if any class implementing this variables we can use this variables by using interface name only. Variables overridng is not in java.

Que. 3: What does the following Java program print?

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

Answer: 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.

Que. 4 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. 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 finally block will execute even if you put a return statement in the try block or catch block but finally block won't run if you call `System.exit()` from try or catch block.

Que. 5: Can you override a private or static method in Java?
Another popular Java tricky question, As I said method overriding is a good topic to ask trick questions in Java. Anyway, you can not override a private or static method in Java, if you create a similar method with same return type and same method arguments in child class then it will hide the superclass method, this is known as method hiding.

Similarly, you cannot override a private method in sub class because it's not accessible there, what you do is create another private method with the same name in the child class. See Can you override a private method in Java or more details.

Que. 6: What do the expression 1.0 / 0.0 will return? will it throw Exception? any compile time error?
Answer: This is another tricky question from Double class. Though Java developer knows about the 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. 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.

Que. 7: 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.

Que. 8: What will happen if we put a key object in a HashMap which is already there?
This tricky Java question is part of another frequently asked question, How HashMap works in Java. HashMap is also a popular topic to create confusing and tricky question in Java. Answer of this question is if you put the same key again then it will replace the old mapping because HashMap doesn't allow duplicate keys. The Same key will result in the same hashcode and will end up at the same position in the bucket.

Each bucket contains a linked list of Map.Entry object, which contains both Key and Value. Now Java will take the Key object from each entry and compare with this new key using `equals()` method, if that return true then value object in that entry will be replaced by new value. See How HashMap works in Java for more tricky Java questions from HashMap.Que. 8: What is the difference between factory and abstract factory pattern?

Abstract Factory provides one more level of abstraction. Consider different factories each extended from an Abstract Factory and responsible for creation of different hierarchies of objects based on the type of factory. E.g. AbstractFactory extended by AutomobileFactory, UserFactory, RoleFactory etc. Each individual factory would be responsible for creation of objects in that genre.

Que. 9: When do you override hashcode and equals() ?
Whenever necessary especially if you want to do equality check or want to use your object as key in HashMap.

Que. 10: What will be the problem if you don't override hashcode() method ?
You will not be able to recover your object from hash Map if that is used as key in HashMap.

Que 11: What does the following Java program print?

```
public class Test {
    public static void main(String[] args) throws Exception {
        char[] chars = new char[] {'\u0097'};
        String str = new String(chars);
        byte[] bytes = str.getBytes();
        System.out.println(Arrays.toString(bytes));
    }
}
```

Answer: The trickiness of this question lies on character encoding and how String to byte array conversion works. In this program, we are 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()`.

In simple word, to 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. On Linux, it is `UTF-8` and on 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.

Que 12: Is it better to synchronize critical section of getInstance() method or whole getInstance() method ?
Answer is critical section because if we lock whole method than every time some one call this method will have to wait even though we are not creating any object.

Que 13: Does not overriding hashcode() method has any performance implication ?
This is a good question and open to all , as per my knowledge a poor hashcode function will result in frequent collision in HashMap which eventually increase time for adding an object into Hash Map.

Que 14: What's wrong using HashMap in multithreaded environment? When get() method go to infinite loop ?
Another good question. His answer was during concurrent access and re-sizing.

Que 15: If a method throws NullPointerException in the superclass, can we override it with a method which throws RuntimeException?
One more tricky Java questions from the overloading and overriding concept. The answer is you can very well throw superclass of RuntimeException in overridden method, but you can not do same if its checked Exception. See Rules of method overriding in Java for more details.

Que 16: What do you understand by thread-safety ? Why is it required ? And finally, how to achieve thread-safety in Java Applications ?

Java Memory Model defines the legal interaction of threads with the memory in a real computer system. In a way, it describes what behaviors are legal in multi-threaded code. It determines when a Thread can reliably see writes to variables made by other threads. It defines semantics for volatile, final & synchronized, that makes guarantee of visibility of memory operations across the Threads.

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 and write barrier.

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.
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.
JMM semantics for synchronized
When a thread acquires monitor of an object, by entering into a synchronized block of code, it performs a read barrier (invalidates the local memory and reads from the heap instead). Similarly exiting from a synchronized block as part of releasing the associated monitor, it performs a write barrier (flushes changes to the main memory)

Thus modifications to a shared state using synchronized block by one Thread, is guaranteed to be visible to subsequent synchronized reads by other threads. This guarantee is provided by JMM in presence of synchronized code block.

JMM semantics for Volatile fields
Read & write to volatile variables have same memory semantics as that of acquiring and releasing a monitor using synchronized code block. So the visibility of volatile field is guaranteed by the JMM. Moreover afterwards Java 1.5, volatile reads and writes are not reorderable with any other memory operations (volatile and non-volatile both). Thus when Thread A writes to a volatile variable V, and afterwards Thread B reads from variable V, any variable values that were visible to A at the time V was written are guaranteed now to be visible to B.

Let's try to understand the same using the following code

Data data = null;
volatile boolean flag = false;

Thread A
----
data = new Data();
flag = true; <-- writing to volatile will flush data as well as flag to main memory

Thread B
----
if(flag==true){ <-- as="" barrier="" data.="" flag="" font="" for="" from="" perform="" read="" reading="" volatile="" well="" will="">
use data; <!--- data is guaranteed to visible even though it is not declared volatile because of the JMM semantics of volatile flag.
}

Que 17. 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, there 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. See How to override compareTo method in Java for the complete answer of this Java tricky question for an experienced programmer.

Que.18: How do you ensure that N thread can access N resources without deadlock?
If you are not well versed in writing multi-threading code then this is a real tricky question for you. This Java question can be tricky even for the experienced and senior programmer, who are not really exposed to deadlock and race conditions. The key point here is ordering, if you acquire resources in a particular order and release resources in the reverse order you can prevent deadlock. See how to avoid deadlock in Java for a sample code example.

Que.19: 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 can not reuse CountDownLatch in Java.

Que.20: What will below statements print?

```
long longWithL = 1000*60*60*24*365L;
long longWithoutL = 1000*60*60*24*365;
System.out.println(longWithL);
System.out.println(longWithoutL);
```

Que.21: What does this program do if you compile it on Windows

```
class Con {
 static final String hi = “\n\n\Hello World\n\n";
}
```

Que.22: Can you instantiate this class?

```
public class A
{
   A a = new A();
}
```

Que.23: What happens when you call methodOfA() in the below class?

```
class A
{
    int methodOfA()
    {
        return (true ? null : 0);
    }
}
```

Not possible. Because while instantiating, constructor will be called recursively.

Que.24: What will be the output of the below program?

```
public class MainClass
{
    public static void main(String[] args)
    {
        Integer i1 = 127;

        Integer i2 = 127;

        System.out.println(i1 == i2);

        Integer i3 = 128;

        Integer i4 = 128;

        System.out.println(i3 == i4);
    }
}
```

Que.25: Can we override method(int) as method(Integer) like in the below example?

```
class A
{
    void method(int i)
    {
        //method(int)
    }
}

class B extends A
{
    @Override
    void method(Integer i)
    {
        //method(Integer)
    }
}
```

No. It gives compile time error. Compiler treats int and Integer as two different types while overriding. Auto-boxing doesn’t happen here.

Que.26: Which statements in the below class shows compile time error (Line 5 or Line 7 or both)?

```
public class MainClass
{
    public static void main(String[] args)
    {
        Integer i = new Integer(null);

 7       String s = new String(null);
    }
}
```

Only Line 7 shows compile time error. Because, compiler will be in an ambiguous situation of which constructor to call. Because, String class has five constructors which take one argument of derived type . Where as Integer class has only one constructor which takes one argument of derived type.

Que.27: What will be the output of the following program?

```
public class MainClass
{
    public static void main(String[] args)
    {
        int i = 10 + + 11 - - 12 + + 13 - - 14 + + 15;

        System.out.println(i);
    }
}
```

Que.28: What will be the output of below program?

```
public class Testing {
    public static void main(String[] args)
     {
         // the line below this gives an output
         // \u000d System.out.println("comment executed");
     }
```

}

Que.29: How will you make Collections readOnly ?

We can make the Collection readOnly by using the following lines code:

General : Collections.unmodifiableCollection(Collection c)

Collections.unmodifiableMap(Map m)
Collections.unmodifiableList(List l)
Collections.unmodifiableSet(Set s)

Top 10 Tricky Java interview questions and Answers

What is a tricky question? Well, tricky Java interview questions are those questions which have some surprise element on it. If you try to answer a tricky question with common sense, you will most likely fail because they require some specific knowledge. Most of the tricky Java questions comes from confusing concepts like function overloading and overriding, Multi-threading which is really tricky to master, character encoding, checked vs unchecked exceptions and subtle Java programming details like Integer overflow. Most important thing to answer a tricky Java question is attitude and analytical thinking, which helps even if you don't know the answer. Anyway in this Java article we will see 10 Java questions which are real tricky and requires more than average knowledge of Java programming language to answer them correctly. As per my experience, there is always one or two tricky or tough Java interview question on any core Java or J2EE interviews, so it's good to prepare tricky questions from Java in advance.

If I take an interview, I purposefully put this kind of question to gauge the depth of candidate's understanding in Java. Another advantage of asking such question is the surprising element, which is a key factor to put the candidate on some pressure during interviews.

Since these questions are less common, there is good chance that many Java developer doesn't know about it. You won't find these questions even on popular Java interview books like Java Programming Interview exposed, which is nevertheless an excellent guide for Java interviews.

Btw, if you don't find these question tricky enough, then you should check Joshua Bloch's another classic book, Java Puzzlers for super tricky questions. I am sure you will find them challenging enough.

10 Tricky Java interview question - Answered

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 can not guess answers of this tricky

Java questions easily, you need some subtle details of Java programming language to answer these questions.

Question: 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));
    }
}
```

Answer: 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.

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. 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 finally block will execute even if you put a return statement in the try block or catch block but finally block won't run if you call `System.exit()` from try or catch block.

Question: Can you override a private or static method in Java?

Another popular Java tricky question, As I said method overriding is a good topic to ask trick questions in Java. Anyway, you can not override a private or static method in Java, if you create a similar method with same return type and same method arguments in child class then it will hide the superclass method, this is known as method hiding.

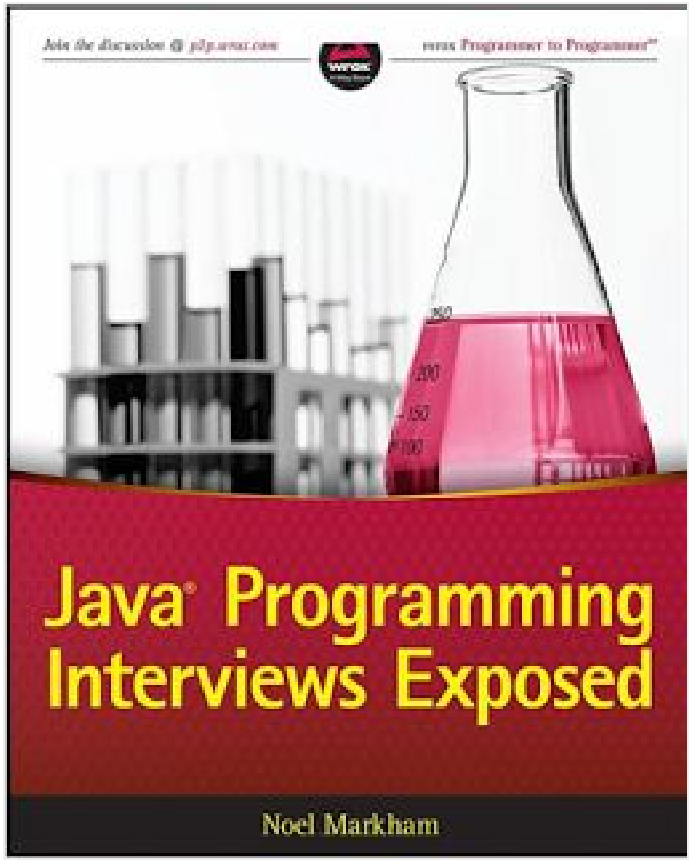
Similarly, you cannot override a private method in sub class because it's not accessible there, what you do is create another private method with the same name in the child class. See Can you override a private method in Java or more details.

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

Answer: This is another tricky question from Double class. Though Java developer knows about the 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. 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.

Btw, If you are running out of time for your interview preparation, you can also check out Java Programming Interviews exposed for more of such popular questions,


Join the discussion @ p2p.wrox.com
wrox Programmer to Programmer™
Java® Programming Interviews Exposed
Noel Markham

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, what Java doesn't support is multiple inheritances of implementation. This distinction also gets blur because of default method of Java 8, which now provides Java, multiple inheritances of behavior as well. See why multiple inheritances are not supported in Java to answer this tricky Java question.

What will happen if we put a key object in a HashMap which is already there?

This tricky Java question is part of another frequently asked question, How HashMap works in Java. HashMap is also a popular topic to create confusing and tricky question in Java. Answer of this question is if you put the same key again then it will replace the old mapping because HashMap doesn't allow duplicate keys. The Same key will result in the same hashcode and will end up at the same position in the bucket.

Each bucket contains a linked list of Map.Entry object, which contains both Key and Value. Now Java will take the Key object from each entry and compare with this new key using `equals()` method, if that return true then value object in that entry will be replaced by new value. See How HashMap works in Java for more tricky Java questions from HashMap.

Question: What does the following Java program print?

```
public class Test {
    public static void main(String[] args) throws Exception {
        char[] chars = new char[] {'\u0097'};
        String str = new String(chars);
        byte[] bytes = str.getBytes();
        System.out.println(Arrays.toString(bytes));
    }
}
```

Answer: The trickiness of this question lies on character encoding and how String to byte array conversion works. In this program, we are 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()`.

In simple word, to 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. On Linux, it is `UTF-8` and on 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.

If a method throws NullPointerException in the superclass, can we override it with a method which throws RuntimeException?

One more tricky Java questions from the overloading and overriding concept. The answer is you can very well throw superclass of RuntimeException in overridden method, but you can not do same if its checked Exception. See Rules of method overriding in Java for more details.

/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. See How to override compareTo method in Java for the complete answer of this Java tricky question for an experienced programmer.

How do you ensure that N thread can access N resources without deadlock?If you are not well versed in writing multi-threading code then this is a real tricky question for you. This Java question can be tricky even for the experienced and senior programmer, who are not really exposed to deadlock and race conditions. The key point here is ordering, if you acquire resources in a particular order and release resources in the reverse order you can prevent deadlock. See how to avoid deadlock in Java for a sample code example.

Question: Consider the following Java code snippet, which is initializing two variables and both are not volatile, and two threads T1 and T2 are modifying these values as following, both are not synchronized

```
int x = 0;
boolean bExit = false;

Thread 1 (not synchronized)
x = 1;
bExit = true;

Thread 2 (not synchronized)
if (bExit == true)
System.out.println("x=" + x);
```

Why finalize() method should be avoided?

We all know the basic statement that finalize() method is called by garbage collector thread before reclaiming the memory allocated to the object. See this program which prove that finalize() invocation is not guaranteed at all. Other reasons can be:

1. finalize() methods do not work in chaining like constructors. It means like when you call a constructor then constructors of all super classes will be invokes implicitly. But, in case of finalize methods, this is not followed. Super class's finalize() should be called explicitly.
2. Any Exception thrown by finalize method is ignored by GC thread and it will not be propagated further, in fact it will not be logged in your log files. So bad, isn't it?
3. Also, There is some performance penalty when finalize() in included in your class. In Effective java (2nd edition) Joshua bloch says,
4. "Oh, and one more thing: there is a severe performance penalty for using finalizers. On my machine, the time to create and destroy a simple object is about 5.6 ns. Adding a finalizer increases the time to 2,400 ns. In other words, it is about 430 times slower to create and destroy objects with finalizers."

When doesn't Singleton remain Singleton in Java?

1. Multiple Singletons in Two or More Virtual Machines
2. Multiple Singletons Simultaneously Loaded by Different Class Loaders
3. Singleton Classes Destroyed by Garbage Collection, then Reloaded
4. Purposely Reloaded Singleton Classes
5. Copies of a Singleton Object that has Undergone Serialization and Deserialization

What happens if your Serializable class contains a member which is not serializable? How do you fix it?

In this case, NotSerializableException will be thrown at runtime. To fix this issue, a very simple solution is to mark such fields transient. It means these fields will not be serialized. If you want to save the state of these fields as well then you should consider reference variables which already implements Serializable interface.

You also might need to use readResolve() and writeResolve() methods. Lets summarize this:

- First, make your non-serialisable field transient.
- In writeObject(), first call defaultWriteObject() on the stream to store all the non-transient fields, then call other methods to serialize the individual properties of your non-serializable object.
- In readObject(), first call defaultReadObject() on the stream to read back all the non-transient fields, then call other methods (corresponding to the ones you added to writeObject) to deserialise your non-serializable object.

Dynamic loading is a technique for programmatically invoking the functions of a classloader at runtime. Let us look at how to load classes dynamically.

```
//static method which returns a Class

Class clazz = Class.forName ("com.Car");  // The value "com.Car" can be evaluated
at runtime & passed in via a variable

```

The above static method “forName” & the below instance (i.e. non-static method) “loadClass”

```
ClassLoader classLoader = MyClass.class.getClassLoader();
Class clazz = classLoader.loadClass(("com.Car"); //Non-static method that returns
a Class
```

return the class object associated with the class name.Once the class is dynamically loaded the following method returns an instance of the loaded class. It’s just like creating a class object with no arguments.

```
// A non-static method, which creates an instance of a
// class (i.e. creates an object).
Car myCar = (Car) clazz.newInstance ( );
```

The string class name like “com.Car” can be supplied dynamically at runtime. Unlike the static loading, the dynamic loading will decide whether to load the class “com.Car” or the class “com.Jeep” at runtime based on a runtime condition.

```

public void process(String classNameSupplied) {
   Object vehicle = Class.forName (classNameSupplied).newInstance();
   //......
}
```

Static class loading throws NoClassDefFoundError if the class is NOT FOUND whereas the dynamic class loading throws ClassNotFoundException if the class is NOT FOUND.

What will be the output of these statement?

```
class output {
    public static void main(String args[])
    {
      double a, b,c;
      a = 3.0/0;
      b = 0/4.0;
      c=0/0.0;

     System.out.println(a);
       System.out.println(b);
       System.out.println(c);
     }
   }
```

What is the output of this program?

```
  class increment {
    public static void main(String args[])
    {
      int g = 3;
      System.out.print(++g * 8);
    }
  }
```

What is the output of this program?

```
class conversion {
    public static void main(String args[])
    {
        double a = 295.04;
        int  b = 300;
        byte c = (byte) a;
        byte d = (byte) b;
        System.out.println(c + " "  + d);
    }
  }
```

What is the output of this program?

```
class San
{
    San()throws IOException
    {

    }

}
class Foundry extends San
  {
      Foundry()
      {

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

If parent class constructor throws any checked exception, compulsory child class constructor should throw the same checked exception as its parent, otherwise code won't compile.

Which of these class can encapsulate an entire executing program?

a) Void b) Process c) Runtime d) System

Which of these class holds a collection of static methods and variables? a) Void b) Process c) Runtime d) System

http://www.sanfoundry.com/java-mcqs-url-class/

Which of these process occur automatically by java run time system?

a) Serialization b) Garbage collection c) File Filtering d) All of the mentioned

How to call private method from another class in java

You can call the private method from outside the class by changing the runtime behaviour of the class.

By the help of **java.lang.Class** class and **java.lang.reflect.Method** class, we can call private method from any other class.

What is the purpose of the System class?

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