Table of Contents

difference between Iterator and Enumeration in Java?

Property	Enumeration	Iterator	ListIterator
1) Is it legacy?	Yes	no	no
2) It is applicable for?	Only legacy classes.	Applicable for any collection object.	Applicable for only list objects.
3) Moment?	Single direction cursor(forward)	Single direction cursor(forward	Bi-directional.
4) How to get it?	By usingelements() method.	By using iterator()method.	By using listIterator() method.
5) Accessibility?	Only read.	Both read and remove.	Read/remove/replace/add.
6) methods	hasMoreElement() nextElement()	hasNext() next() remove()	9 methods.

- ~~Java Fundamentals~~(Java fundamentals)
- ~~Object-Oriented Concepts~~ (questions)
- ~~Multithreading, concurrency, and thread basics~~ (questions)
- ~~Date type conversion and fundamentals (~~questions)
- ~~Garbage Collection~~ (questions)
- ~~Java Collections Framework~~ (questions)
- ~~Array~~ (questions)

- ~~String~~ (questions)
- ~~GOF Design Patterns~~ (questions)
- ~~SOLID design principles~~ (questions)
- ~~Abstract class and interface~~ (questions)
- ~~Java basics e.g. equals and hashcode~~ (questions)
- ~~Generics and Enum~~ (questions)
- ~~Java Best Practices~~ (codegeek, howtoinjava)

Read more: https://javarevisited.blogspot.com/2017/01/how-to-prepare-for-java-interviews.html#ixzz5foXyQowA

Core Java

JVM Internals and Garbage Collection

```
public class Zoo {
public static void main(String[] args) {
System.out.println(args[0]);
System.out.println(args[1]);
} }
```

The program correctly identifies the first two "words" as the arguments. Spaces are used to separate the arguments. If you want spaces inside an argument, you need to use quotes as in this example:

```
$ javac Zoo.java
$ java Zoo "San Diego" Zoo
```

All command-line arguments are treated as String objects, even if they represent another data type:

```
$ javac Zoo.java
$ java Zoo Zoo 2
```

Finally, what happens if you don't pass in enough arguments?

```
$ javac Zoo.java
$ java Zoo Zoo
Zoo
```

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1 at mainmethod.Zoo.main(Zoo.java:7)

```
public class Conflicts {
Date date;
// some more code
}
```

The answer should be easy by now. You can write either import java.util.*; or import java.util.Date;. The tricky cases come about when other imports are present:

```
import java.util.*;
import java.sql.*; // DOES NOT COMPILE

import java.util.Date;
import java.sql.*;
```

If you explicitly import a class name, it takes precedence over any wildcards present.

■ octal (digits 0–7), which uses the number 0 as a prefix—for example, 017

■ hexadecimal (digits 0–9 and letters A–F), which uses the number 0 followed by x or X as a prefix—for example, 0xFF

■ binary (digits 0–1), which uses the number 0 followed by b or B as a prefix—for example, 0b10

```
System.out.println(56); // 56
System.out.println(0b11); // 3
System.out.println(017); // 15
System.out.println(0x1F); // 31
```

added in Java 7. You can have underscores in numbers to make them easier to read:

```
int million1 = 1000000;
int million2 = 1_000_000;

double notAtStart = _1000.00; // DOES NOT COMPILE
double notAtEnd = 1000.00_; // DOES NOT COMPILE

double notByDecimal = 1000_.00; // DOES NOT COMPILE
double annoyingButLegal = 1_00_0.0_0; // this one compiles
```

Declaring Multiple Variables

```
int i1, i2, i3 = 0;
```

As you should expect, three variables were declared: i1, i2, and i3. However, only one of those values was initialized: i3. The other two remain declared but not yet initialized.

```
int num, String value; // DOES NOT COMPILE
```

This code doesn't compile because it tries to declare multiple variables of different types in the same statement.

```
double d1, double d2;  // DOES NOT COMPILE
```

If you want to declare multiple variables in the same statement, they must share the same type declaration and not repeat it. double d1, d2; would have been legal.

```
boolean b1, b2;
String s1 = "1", s2;
double d1, double d2;
int i1; int i2;
int i3; i4;
```

The first statement is legal. It declares two variables without initializing them. second statement is also legal. It declares two variables and initializes only one of them. The third statement is not legal. Variables d1 and d2 are the same type & breaks between them. The fourth statement is legal.The fifth statement is not legal.The second one is not a valid declaration because it omits the type.

Garbage Collection

The methods to request JVM to run Garbage Collector

System.gc() : 'System' class contains a static 'gc' method for requesting JVM to run Garbage Collector.

Runtime.getRuntime().gc() : gc() method available in Runtime class is an instance method.

1.Nullifying the reference Variable

```
Student s1 = new Student();
Student s2 = new Student();
//No Object eligible for Garbage Collector

s1 = null;
//One Object eligible for Garbage Collector

s2 = null;
Both Objects eligible for Garbage Collector
```

2.Reassigning the reference Variable

```
Student s1 = new Student();
Student s2 = new Student();

s1 = s2;
//One Object eligible for Garbage Collector
```

3.The Objects Created inside a method
The objects which are created in a method are by default eligible for Garbage Collector once the method completes

```
1: public class Scope {
2: public static void main(String[] args) {
3: String one, two;
4: one = new String("a");
5: two = new String("b");
6: one = two;
7: String three = one;
8: one = null;
9: } }
```

Difference between PATH and Classpath in Java?

Answer : PATH is a environment variable in Java which is used to help Java program to compile and run.To set the PATH variable we have to include JDK_HOME/bin directory in PATH environment variable and also we cannot override this variable.

On the other hand, ClassPath variable is used by class loader to locate and load compiled Java codes stored in .class file. We you want to run JUnit from any where from cmdline you eed to add Junit.jar in class path.

Difference between interpreter and JIT compiler?

The interpreter interprets the bytecode line by line and executes it sequentially. It results in poor performance. JIT compiler add optimization to this process by analyzing the code in blocks and then prepare more optimized machine code.

Difference between JRE and JVM?

JVM is the specification for runtime environment which executes the Java applications. Hotspot JVM is such one implementation of the specification. It loads the class files and uses interpreter and JIT compiler to convert bytecode into machine code and execute it.

Difference Between JVM & HotSpot VM

JVM : is a Specification, **HotSpot** : is a implementation of JVM.

HotSpot is an implementation of the JVM concept, originally developed by Sun and now owned by Oracle. There are other implementations of the JVM specification, like JRockit, IBM J9, among many others.

Is Java a pure object oriented language? (answer)

Java is not a pure object-oriented programming language e.g. There are many things in Java which are not objects e.g. primitive data types e.g. boolean, char, short, int, long, float, double, different kinds of arithmetic, logical and bitwise operator e.g. +, -. *, /, &&, || etc. Few pure OO languages are **Smalltalk** and **Eiffel**.

There are seven qualities to be satisfied for a programming language to be pure Object Oriented. They are:

1. **Encapsulation/Data Hiding**
2. **Inheritance**
3. **Polymorphism**
4. **Abstraction**
5. **All predefined types are objects**

6. All operations are performed by sending messages to objects

7. All user defined types are objects

How does WeakHashMap work?

WeakHashMap operates like a normal HashMap but uses WeakReference for keys. Meaning if the key object does not devise any reference then both key/value mapping will become appropriate for garbage collection.

How do you locate memory usage from a Java program?

Answer: You can use memory related methods from **java.lang.Runtime** class to get the free memory, total memory and maximum heap memory in Java.

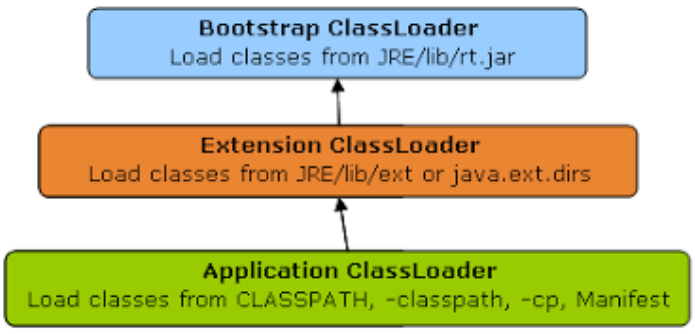
public static Runtime getRuntime()	returns the instance of Runtime class.
public void exit(int status)	terminates the current virtual machine.
public void addShutdownHook(Thread hook)	registers new hook thread.
public Process exec(String command)	executes given command in a separate process.
public int availableProcessors()	returns no. of available processors.
public long freeMemory()	returns amount of free memory in JVM.
public long totalMemory()	returns amount of total memory in JVM.

```
public class TestApp {
	public static void main(String[] args) {
		Runtime r = Runtime.getRuntime();
		System.out.println(r.totalMemory()); //16252928
		System.out.println(r.freeMemory()); //15709576
		System.out.println(r.availableProcessors());//24
		r.gc();
	}
}
```

What is ClassLoader in Java?

When a Java program is converted into **.class** file by Java compiler which is collection of byte code. **ClassLoader** is responsible to load that class file from file system, network or any other location

- Bootstrap ClassLoader - **JRE/lib/rt.jar**
- Extension ClassLoader - **JRE/lib/ext** or any directory denoted by java.ext.dirs
- Application ClassLoader - **CLASSPATH environment variable, -classpath or -cp option, Class-Path attribute of Manifest inside JAR file.**



Java heap memory

When a Java program started Java Virtual Machine gets some memory from Operating System.

whenever we create an object using new operator or by any another means the object is allocated memory from Heap and When object dies or garbage collected, memory goes back to Heap space.

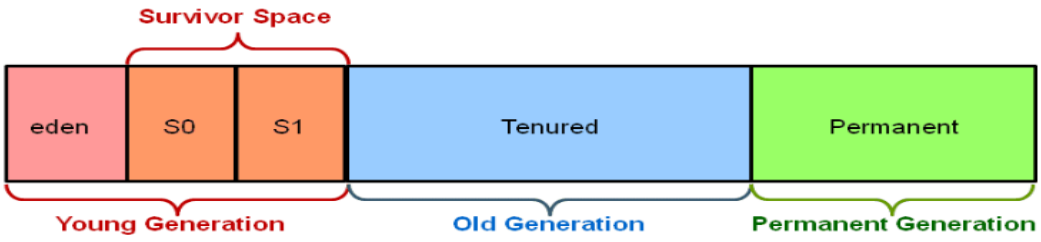
How to increase heap size in Java

Default size of Heap space in Java is 128MB on most of 32 bit Sun's JVM but its highly varies from JVM to JVM. change size of heap space by using JVM **options -Xms and -Xmx**. Xms denotes starting size of Heap while -Xmx denotes maximum size of Heap in Java.

Java Heap and Garbage Collection

As we know objects are created inside heap memory and Garbage Collection is a process which removes dead objects from Java Heap space and returns memory back to Heap in Java.

For the sake of Garbage collection Heap is divided into three main regions named as **New Generation, Old Generation, and Perm space**



- **New Generation** of Java Heap is part of Java Heap memory where a newly created object is stored,
- **Old Generation** During the course of application many objects created and died but those remain live they got moved to Old Generation by Java Garbage collector thread
- **Perm space** of Java Heap is where JVM stores Metadata about classes and methods, String pool and Class level details.
- Perm Gen stands for permanent generation which holds the meta-data information about the classes.
- Suppose if you create a class name A, it's instance variable will be stored in heap memory and class A along with static classloaders will be stored in permanent generation.
- Garbage collectors will find it difficult to clear or free the memory space stored in permanent generation memory. Hence it is always recommended to keep the permgen memory settings to the advisable limit.
- JAVA8 has introduced the concept called meta-space generation, hence permgen is no longer needed when you use jdk 1.8 versions.

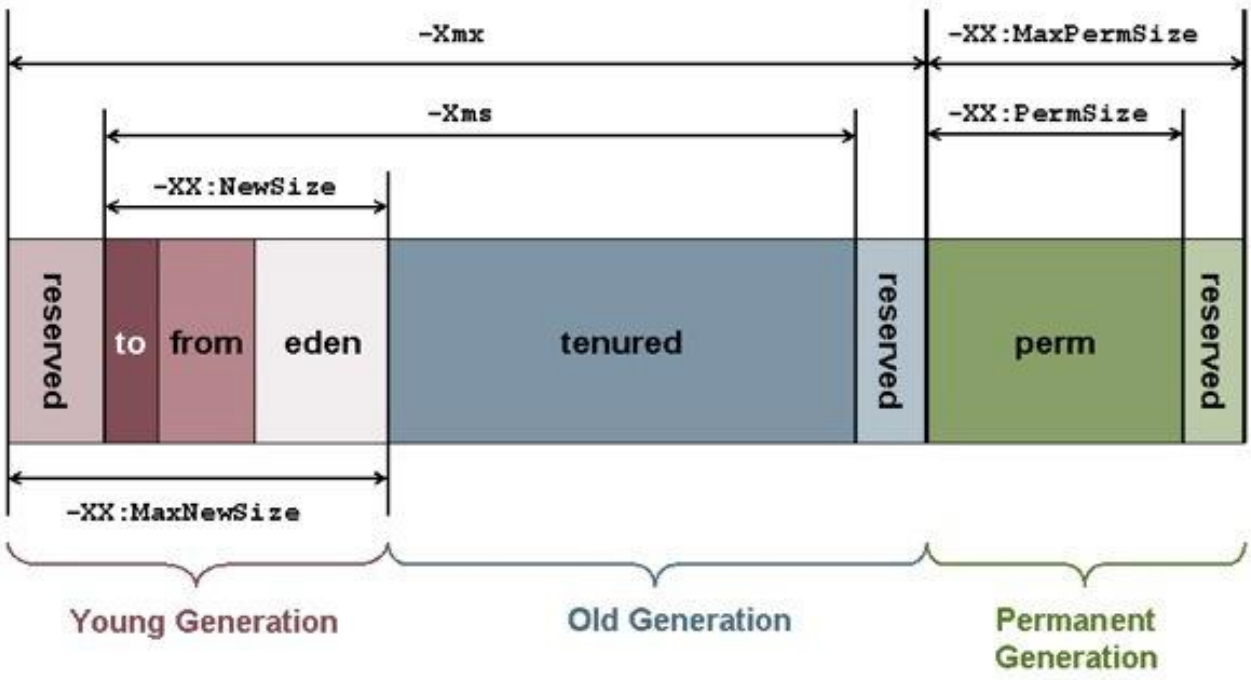
Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

Neither finalization nor garbage collection is guaranteed.

Does Garbage collection occur in permanent generation space in JVM?

Answer : This is a tricky Garbage collection interview question as many programmers are not sure whether PermGen space is part of Java heap space or not and since it maintains class Meta data and String pool, whether its eligible for garbage collection or not. By the way Garbage Collection does occur in PermGen space and if PermGen space is full or cross a threshold, it can trigger Full GC. If you look at output of GC you will find that PermGen space is also garbage collected. This is why correct sizing of PermGen space is important to avoid frequent full GC. You can control size of PermGen space by JVM options -XX:PermGenSize and -XX:MaxPermGenSize.



How to you monitor garbage collection activities?

just to check whether candidate has ever monitored GC activities or not. You can monitor garbage collection activities either offline or real-time. You can use tools like **JConsole** and **VisualVM** VM with its Visual GC plug-in to monitor real time garbage collection activities and memory status of JVM or you can redirect Garbage collection output to a log file for offline analysis by using -XlogGC=<PATH> JVM parameter. Anyway you should always enable GC options like -XX:PrintGCDetails -X:verboseGCand -XX:PrintGCTimeStamps as it doesn't impact application performance much but provide useful states for performance monitoring.

How do you identify minor and major garbage collection in Java?

Answer: Minor collection prints "GC" if garbage collection logging is enable using –verbose:gc or -XX:PrintGCDetails, while Major collection prints "Full GC". This Garbage collection interview question is based on understanding of Garbage collection output. As more and more Interviewer are asking question to check candidate's ability to understand GC output, this topic become even more important.

How do you find GC resulted due to calling System.gc()?

Answer : Another GC interview question which is based on GC output. Similar to major and minor collection, there will be a word "System" included in Garbage collection output.

Read more: https://javarevisited.blogspot.com/2012/10/10-garbage-collection-interview-question-answer.html#ixzz5fwmNzRHE

Read more: https://javarevisited.blogspot.com/2012/10/10-garbage-collection-interview-question-answer.html#ixzz5fwm9nzDa

Data Types

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.

```
String str = new String(bytes, "UTF-8");
```

How do you convert bytes to long in Java

The byte takes 1 byte of memory and long takes 8 bytes of memory. Assignment 1 byte value to 8 bytes is done implicitly by the JVM.

byte –> short –> int –> long –> float –> double

The left-side value can be assigned to any right-side value and is done implicitly. The reverse requires explicit casting.

```
byte b1 = 10;                // 1 byte
long l1 = b1;                //  one byte to 8 bytes, assigned implicitly
```

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.

What will this return 3*0.1 == 0.3? true or false?

Both are not equal, because floating point arithmetic has a certain precision. Check the difference (a-b) it should be really small.

In computer memory, floats and doubles are stored using IEEE 754 standard format.

- f1 = (0.1+0.1+0.1....11 times) = 1.0999999999999999
- f2 = 0.1*11 = 1.1

In **BigDecimal** class, you can specify the rounding mode and exact precision which you want to use. Using the exact precision limit, rounding errors are mostly solved. Best part is that BigDecimal numbers are immutable i.e. if you create a BigDecimal BD with value "1.23", that object will remain "1.23" and can never be changed. You can use it's .compareTo() method to compare to BigDecimal numbers

```
private static void testBdEquality()
{
     BigDecimal a = new BigDecimal("2.00");
     BigDecimal b = new BigDecimal("2.0");

     System.out.println(a.equals(b));           // false

     System.out.println(a.compareTo(b) == 0);   // true
}
```

Which one will take more memory, an int or Integer? (answer)

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.

How to convert Primitives to Wrapper & Wrapper to Primitive ??

```
// 1. using constructor
Integer i = new Integer(10);

// 2. using static factory method
Integer i = Integer.valueOf(10);

//3.wrapper to primitive
int val = i.intValue();
```

Autoboxing and Unboxing?

If a method requires Integer Object value, we can directly pass primitive value without issue. Autoboxing will take care about these.

But direct initializations it doesn't possible.

```
Integer i = 10;// Type mismatch: cannot convert from Integer to int
int j = i;
```

what if I make main() private/protected ?

if you do not make `main()` method `public`, there is no compilation error. You will **runtime error** because matching `main()` method is not present. Remember that whole syntax should match to execute `main()` method.

```
Error: Main method not found in class Main, please define the main method as:
   public static void main(String[] args)
```

How does Autoboxing of Integer works in Java? (answer)

Compiler uses valueOf() method to convert primitive to Object uses intValue(), doubleValue() etc to get primitive value from Object.

Can you store String in an array of Integer in Java? compile time error or runtime exception? [answer]

Answer is both yes and no. You cannot store an String in an array of primitive int, it will result in compile time error as shown below, but if you create an array of Object and assign String[] to it and then try to store Integer object on it. Compiler won't be able to detect that and it will throw ArrayStoreExcpetion at runtime

```
int[] primes = new int[10];
primes[0] = "a"; // compile time error

Object[] names = new String[3];
names[0] = new Integer(0);// ArrayStoreException at runtime
```

What is difference between ArrayIndexOutfOBounds and ArrayStoreException? [answer]

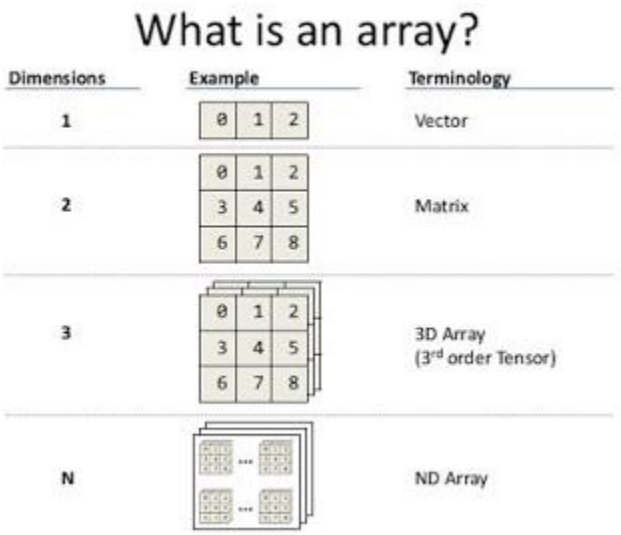
ArrayIndexOutOfBoundsException comes when your code tries to access an invalid index for a given array e.g. negative index or higher index than length - 1. While, ArrayStoreException comes when you have stored an element of type other than type of array, as shown in above example.

Is it legal to initialize an array int i[] = {1, 2, 3, 4, 5}; [answer]

Yes, it's perfectly legal. You can create and initialize array in same line in Java.

Where does array stored in memory? [answer]

Array is created in heap space of JVM memory. Since array is object in Java, even if you create array locally inside a method or block, object is always allocated memory from heap.



How do you reverse an array in Java?

org.apache.commons.lang.ArrayUtils class to reverse Array in Java. As discussed in our last post How to print array element in Java, We are using Arrays.toString() to print content of array.

```
int[] iArray = new int[] {101,102,103,104,105};
               String[] sArray = new String[] {"one", "two", "three", "four", "five"}; //
reverse int array using Apache commons ArrayUtils.reverse() method
```

```
System.out.println("Original int array : " + Arrays.toString(iArray));
ArrayUtils.reverse(iArray);
```

java.lang Package

We have mainly five classes in java.lang. Which are most commonly used in any java program

1. **Object**
2. **String**
3. **StringBuffer**
4. **StringBuilder**
5. **Wrapper Classes (AutoBoxing / AutoUnboxing)**

Object Class

The most common general methods which can be applicable on any java object are defined in object class. Object class is the parent class of any java class, whether it is predefined or programmer defined, hence all the object class methods are by default available to any java class.

Object class define the following 11 methods

1.toString():Returns a string representation of the object.

```
public String toString() {
        return getClass.getName() + '@' + Integer.toHexString(HashCode);
    }
```

2.hashCode():returns the integer representation of memory location which used by JVM while saving/adding Objects into Hashsets, Hashtables or Hashmap

3.equals(Object): Compares two Objects for equality.

4.clone(): Creates a new object of the same class as this object which implements Clonable interface.

```
Test t1 = new Test();
Test t2 = (Test)t1.clone();
```

5.finalize():Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

6.getClass():Returns the runtime class of an object. etClass(), or the class-literal - Foo.class return a Class object, which contains some metadata about the class:

- name
- package
- methods
- fields
- constructors
- annotations

we can create Class object by following ways

```
Class c = Class.forName("StudentBO")
Class c = StudentBO.class
Class c = a.getClass();
```

```
public static void main(String[] args) throws Exception {
		TestApp a = new TestApp();
		Class c1 = a.getClass();

		 Class c = Class.forName("java.lang.String");
		 System.out.print("Class represented by c : " + c.toString());

		 Object obj = c.newInstance();
	}
```

7.wait():Waits to be notified by another thread of a change in this object.

8.wait(long):Waits to be notified by another thread of a change in this object.

wait(long, int):Waits to be notified by another thread of a change in this object.

9.notify():Wakes up a single thread that is waiting on this object's monitor.

notifyAll():Wakes up all threads that are waiting on this object's monitor.

`equals(Object otherObject)` – As method name suggests, is used to simply verify the equality of two objects. It's default implementation simply check the object references of two objects to verify their equality. *By default, two objects are equal if and only if they are stored in the same memory address.*

`hashcode()` – Returns a unique integer value for the object in runtime. By default, integer value is mostly derived from memory address of the object in heap (but it's not mandatory always).

If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.

Whenever we override the equals() method, we should override hashcode() method

In String class(not StringBuilder, StringBuffer) & All Wrapper classes equals() method is overridden for Content Comparison

Compare two employee Objects based on Their Id?

```
public class Employe {
	int id;
	String name;
//Setters & Getters
	@Override
	public boolean equals(Object obj) {
		Employe e = (Employe) obj;
		boolean flag = false;
		if (this.getId() == e.getId()) {
			flag = true;
		}
		return flag;
	}
	public static void main(String[] args) {
		Employe e1 = new Employe();
		Employe e2 = new Employe();
		e1.setId(101);
		e2.setId(101);
		System.out.println(e1.equals(e2));//true
	}
}
```

So are we done? Not yet. Lets test again above modified `Employee` class in different way.

```
public static void main(String[] args) {
		Employe e1 = new Employe();
		Employe e2 = new Employe();
		e1.setId(101);
		e2.setId(101);

		Set<Employe> set = new HashSet<>();
		set.add(e1);
		set.add(e2);
		System.out.println(set); //[basic.Employe@15db9742, basic.Employe@6d06d69c]
	}
```

Above class prints two objects in the second print statement. If both employee objects have been equal, in a `Set` which stores only unique objects, there must be only one instance inside `HashSet`

We are missing the second important method `hashCode()`. As java docs say, if you override `equals()`method then you ***must*** override `hashCode()` method

```
public class Employe {
	int id;
	String name;

	@Override
	public boolean equals(Object obj) {
		Employe e = (Employe) obj;
		boolean flag = false;
		if (this.getId() == e.getId()) {
			flag = true;
		}
		return flag;
	}

	@Override
	public int hashCode() {
		return getId();
	}

	public static void main(String[] args) {
		Employe e1 = new Employe();
		Employe e2 = new Employe();
		e1.setId(101);
		e2.setId(101);

		Set<Employe> set = new HashSet<>();
		set.add(e1);
		set.add(e2);
		System.out.println(set); //[basic.Employe@65]
	}
}
```

Apache commons provide two excellent utility classes **EqualsBuilder** & **HashCodeBuilder** for generating hash code and equals methods.

Can a top-level class be private or protected?

Top level classes in java can't be private or protected, but inner classes in java can. The reason for not making a top-level class as private is very obvious, because nobody can see a private class and thus they cannot use it

What Happens if we compile Empty java file?

Compiles but Runtime Error.

```
C:\Users\kaveti_S\Downloads>javac Demo.java

C:\Users\kaveti_S\Downloads>java Demo
Error: Could not find or load main class Demo
```

Is it possible to make array volatile in Java?

Yes, it is possible to make an array volatile in Java, but only the reference which is pointing to an array, by reassigning it

What is a.hashCode() used for? How is it related to a.equals(b)?

According to the Java specification, two objects which are identical to each other using equals() method needs to have the same hash code

What is a compile time constant in Java? What is the risk of using it?

Answer: **Public static final variables** are also known as the compile time constant, the public is optional there. They are substituted with actual values at compile time because compiler recognizes their value up-front, and also recognize that it cannot be altered during runtime.

One of the issues is that if you choose to use a public static final variable from in-house or a third party library, and their value changed later, then your client will still be using the old value even after you deploy a new version of JARs.

Explain Liskov Substitution Principle.

According to the Liskov Substitution Principle, Subtypes must be appropriate for super type i.e. methods or functions which use super class type must be able to work with object of subclass without issues. Co-Variant return types are implemented based on this principle .

What is double checked locking in Singleton?

Singleton means we can create only one instance of that class

Rules:

- Create Singleton class Object make it as PRIVATE
- Create PRIVATE constructor
- Every Singleton class contains at least one factory method

```
class Student {
    private static Student st;
    private Student() {
        System.out.println("OBJECET Created FIRST TIME");
    }
    public static Student getObject() {
        if (st == null) {
            st = new Student();
        } else {
            System.out.println("OBJECET ALREDAY CREATED");
        }
        return st;
    }
}

public class Singleton {
```

```
    public static void main(String[] args) {
        Student s1 = Student.getObject();
        Student s2 = Student.getObject();
        System.out.println(s1.hashCode());
        System.out.println(s2.hashCode());
    }
}
```

Double checked locking in Singleton means, at any cost only one instance is created in multi-threaded environment.

In this case at null checking make Block as Synchronized.

```
public static Singleton getInstanceDC() {
        if (_instance == null) {                // Single Checked
            synchronized (Singleton.class) {
                if (_instance == null) {        // Double checked
                    _instance = new Singleton();
                }
            }
        }
        return _instance;
}
```

When to use volatile variable in Java?

- Volatile keyword is used with only variable in Java
- it guarantees that value of volatile variable will always be read from main memory and not from Thread's local cache.
- So, we can use volatile to achieve synchronization because its guaranteed that all reader thread will see updated value of volatile variable once write operation completed

Difference between Serializable and Externalizable in Java?

Serialization is a default process of serializing or persisting any object's state in Java. It's triggered by implementing Serializable interface which is a marker interface (an interface without any method). uses default implementation to handle the object serialization process.

Externalizable is used to user defined serialization process and control default serialization process which is implemented by application.

Externalizable interface extends Serializable interface. It consists of two methods

// to read object from stream
void readExternal(ObjectInput in)

// to write object into stream
void writeExternal(ObjectOutput out)

Difference between static and dynamic binding in Java? (detailed answer)

This is usually asked as follow-up of previous question, static binding is related to overloaded method and dynamic binding is related to overridden method. Method like 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.

Which design pattern have you used in your production code?

- **Dependency injection**
- **Factory pattern**
- **Adapter Design pattern**
- **Singleton**

Decorator design pattern is used to modify the functionality of an object at runtime.

Can you explain Liskov Substitution principle?(Comes under inheritance)

According to Liskov Substitution Principle, Subtypes must be substitutable for supertype i.e. methods or functions which uses superclass type must be able to work with the object of subclass without any issue

How to create an instance of any class without using new keyword

Using newInstance method of Class class

```
Class c = Class.forName("StudentBo");
StudentBo bo = (StudentBo) c.newInstance();
```

Using clone() of java.lang.Object

```
NewClass obj = new NewClass();
NewClass obj2 = (NewClass) obj.clone();
```

Java OOPs Concepts

Can we prevent overriding a method without using the final modifier? (answer)

Yes, you can prevent the method overriding in Java without using the final modifier. In fact, there are several ways to accomplish it e.g. you can mark the method **private or static, those cannot be overridden.**

Can we override a private method in Java? (answer)

No, you cannot. Since the private method is only accessible and visible inside the class they are declared, it's not possible to override them in subclasses. Though, you can override them inside the inner class as they are accessible there

Can we change the return type of method to subclass while overriding? (answer)

Yes, you can, but only from Java 5 onward. This feature is known as **covariant method** overriding and it was introduced in JDK 5 release. This is immensely helpful if original method return super-class e.g. clone() method return java.lang.Object. By using this, you can directly return the actual type, preventing client-side type casting of the result.

Can we make a class both final and abstract at the same time? (answer)

No, you cannot apply both final and abstract keyword at the class same time because they are exactly opposite of each other. A final class in Java cannot be extended and you cannot use an abstract class without extending and make it a concrete class. As per Java specification, the compiler will throw an error if you try to make a class abstract and final at the same time.

Can we overload or override the main method in Java? (answer)

No, since main() is a static method, you can only overload it, you cannot override it because the static method is resolved at compile time without needing object information hence we cannot override the main method in Java.

Design Patterns

SOLID design principles and GOF design patterns which take advantage of OOPS concept discussed here.

What are SOLID Design principles

S.O.L.I.D. Class Design Principles

Principle Name	What it says? howtodoinjava.com
Single Responsibility Principle	One class should have one and only one reasonability
Open Closed Principle	Software components should be open for extension, but closed for modification
Liskov's Substitution Principle	Derived types must be completely substitutable for their base types
Interface Segregation Principle	Clients should not be forced to implement unnecessary methods which they will not use
Dependency Inversion Principle	Depend on abstractions, not on concretions

1. Single Responsibility Principle

"One class should have one and only one responsibility"

In other words, we should write, change and maintain a class for only one purpose. If it is model class then it should strictly represent only one actor/ entity. This will give we the flexibility to make changes in future without worrying the impacts of changes for another entity.

2. Open Closed Principle

"Software components should be open for extension, but closed for modification"

If we take a look into any good framework like struts or spring, we will see that we can not change their core logic and request processing, but we modify the desired application flow just by extending some classes and plugin them in configuration files.

For example, spring framework has class `DispatcherServlet`. This class acts as **front controller** for String based web applications. To use this class, we are not required to modify this class. All we need is to pass initialization parameters and we can extend it's functionality the way we want.

3. Liskov's Substitution Principle

"Derived types must be completely substitutable for their base types"

4. Interface Segregation/Separation Principle

This principle is my favorite one. It is applicable to interfaces as single responsibility principle holds to classes. ISP says:

"Clients should not be forced to implement unnecessary methods which they will not use"

Take an example. Developer Alex created an interface `Reportable` and added two methods `generateExcel()` and `generatedPdf()`. Now client 'A' wants to use this interface but he intend to use reports only in PDF format and not in excel. Will he be able to use the functionality easily?

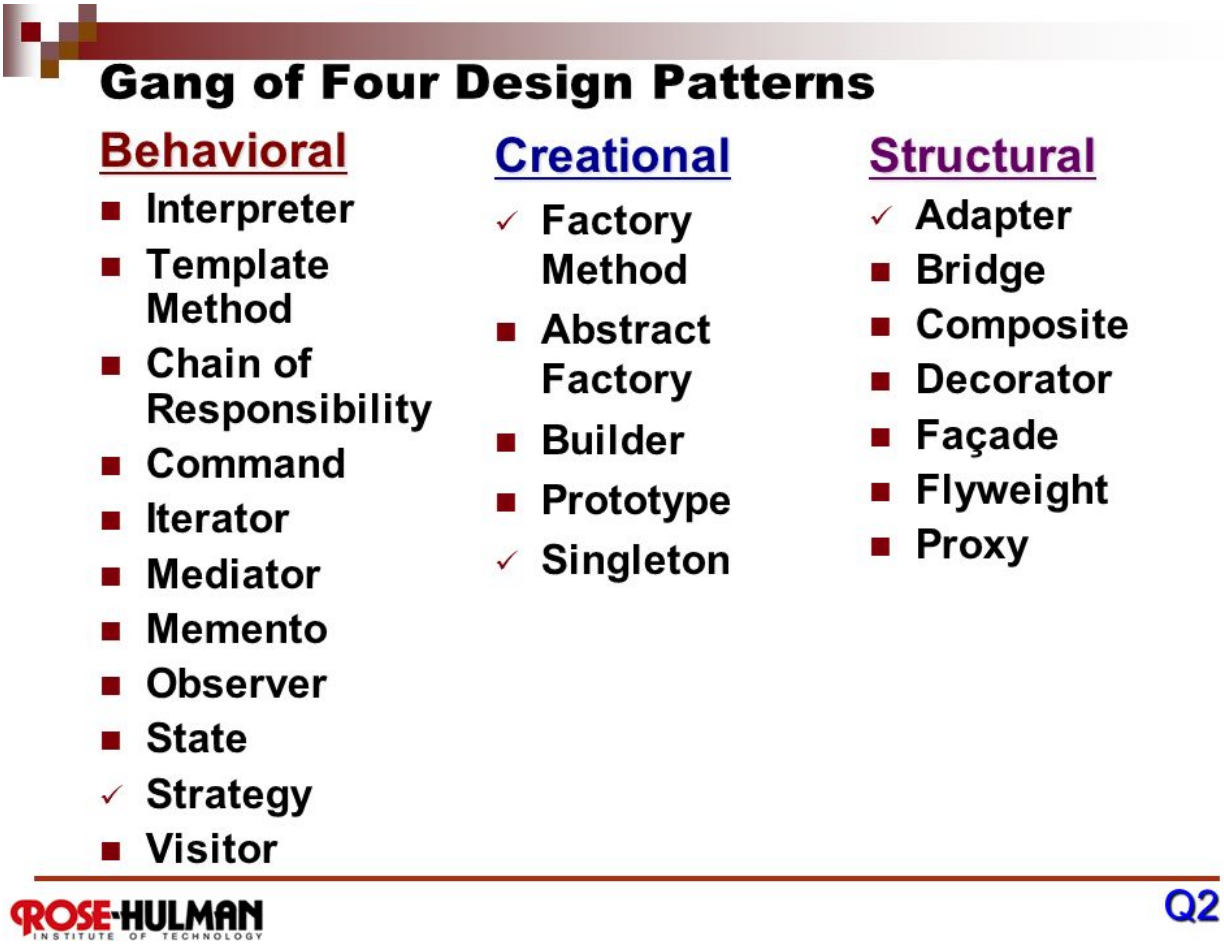
NO. He will have to implement both the methods, out of which one is extra burden put on him by designer of software. Either he will implement another method or leave it blank. This is not a good design.

5. Dependency Inversion/Injection Principle

Remove dependency from classes

In spring framework, all modules are provided as separate components which can work together by simply injected dependencies in other module. This dependency is managed externally in XML files.

What are GOF(Gang of Four) design patterns?



What is Strategy pattern in Java?

Strategy pattern allows you to introduce new strategy without changing the code.

For example, the **Collections.sort()** method which sorts the list of the object uses the Strategy pattern to compare object. Since every object uses different comparison strategy you can compare various object differently without changing sort method.

What is Decorator Design Pattern?

Decorator pattern provides new features without modifying the original class. Inheritance is the example.

What is the difference between Decorator, Proxy and Adapter pattern in Java? (answer)

Again they look similar because their structure or class diagram is very similar but their intent is quite different. Decorator adds additional functionality without touching the class, Proxy provides access control and Adapter is used to make two incompatible interfaces work together.

Strings

1.What is immutable object? Can you write immutable object?

Don't confuse over SingleTon class

Immutable classes are Java classes whose objects can not be modified once created.

1. Declare **the class as final** so it can't be extended.
2. Make all **fields private & final** so that direct access is not allowed & it's values can be assigned only once..
3. **Initialize** all the fields via a **constructor**
4. Write getters only**, not setters**.

```
// An immutable class
public final class Student {
	final String name;
	final int regNo;

	public Student(String name, int regNo) {
		this.name = name;
		this.regNo = regNo;
	}
	public String getName() {
		return name;
	}
	public int getRegNo() {
		return regNo;
	}
}

// Driver class
class Test {
	public static void main(String args[]) {
		Student s = new Student("ABC", 101);
		System.out.println(s.name);
		System.out.println(s.regNo);

		// Uncommenting below line causes error
		// s.regNo = 102;
	}
}
```

2.What is Singleton? Can you write critical section code for singleton?

A Singleton class is one which allows us to create only one object for JVM.

Rules:

- Create Singleton class **Object make it as PRIVATE**
- Create **PRIVATE contrcutor**
- Every Singleton class contains **at least one factory method**

```
class Student {
	private static Student st;
```

```
        private Student() {
                System.out.println("OBJECET Created FIRST TIME");
        }

        public static Student getObject() {
                if (st == null) {
                        st = new Student();
                } else {
                        System.out.println("OBJECET ALREDAY CREATED");
                }
                return st;
        }
}

public class Singleton {
        public static void main(String[] args) {
                Student s1 = Student.getObject();
                Student s2 = Student.getObject();
                System.out.println(s1.hashCode());
                System.out.println(s2.hashCode());
        }
}
```

In above code, it will create multiple instances of Singleton class if called by more than one thread parallel

Double checked locking of Singleton is a way to ensure only one instance of Singleton class is created through application life cycle.

This will bring us to **double checked locking pattern**, where only critical section of code is locked. Programmer call it double checked locking because there are two checks for _instance == null, one without locking and other with locking (inside synchronized) block. Here is how double checked locking looks like in Java

```
public static Singleton getInstanceDC() {
        if (_instance == null) {                // Single Checked
            synchronized (Singleton.class) {
                if (_instance == null) {        // Double checked
                    _instance = new Singleton();
                }
            }
        }
        return _instance;
}
```

How do you reverse a String in Java without using StringBuffer?

The Java library provides String Buffer and StringBuilder class with **reverse()** method, which can be used to reverse String in Java.

```
String reverse = "";
String source= "My Name is Khan";
        for(int i = source.length() -1; i>=0; i--){
            reverse = reverse + source.charAt(i);
        }
```

How to Print duplicate characters from String?

```
public class RepreatedChar {
        public static void main(String[] args) {
                String a = "success";

                // 1.convert into char array
                char[] c = a.toCharArray();

                // 2.create Hashmap store key as character, count as value
```

```
                HashMap map = new HashMap<>();
                for (char ch : c) {

                        // 3.Check if Map contains given Char as <key> or not
                        if (map.containsKey(ch)) {
                                // if their, get the value & increment it
                                int i = (int) map.get(ch);
                                i++;
                                // add updated value to it
                                map.put(ch, i);
                        } else {
                                // if not their , add key & value as 1
                                map.put(ch, 1);
                        }
                }

                 Set  set =  map.entrySet();
                 Iterator iterator = set.iterator() ;
                 while (iterator.hasNext()) {
                        Map.Entry entry = (Entry) iterator.next();
                        System.out.println(entry.getKey()+" : "+entry.getValue());

                }
        }
}
s : 3
c : 2
u : 1
e : 1
```

Reverse String in Java

1. Get String length
2. Iterate by using charAt() in reverse & append to new String

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

        String s = "satyam";
        String rev="";
        int len = s.length();

        for(int i=(len-1);i>=0;i--){

                rev = rev+s.charAt(i);
        }

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

Is String contains Number or not

```
public class RegEx {
        public static void main(String[] args) {
                // Regular expression in Java to check if String is number or not
                Pattern pattern = Pattern.compile(".*[^0-9].*");
                String[] inputs = { "123", "-123", "123.12", "abcd123" };
                /* Matches m = pattern.match(input);
                 * boolean ch = m.match();    */

                for (String input : inputs) {
System.out.println("does " + input + " is number : " + !pattern.matcher(input).matches());
                }
```

```
	// Regular expression in java to check if String is 6 digit number or  not
		String[] numbers = { "123", "1234", "123.12", "abcd123", "123456" };
		Pattern digitPattern = Pattern.compile("\\d{6}");
		// Pattern digitPattern = Pattern.compile("\\d\\d\\d\\d\\d\\d");

		for (String number : numbers) {
			System.out.println("does " + number + " is 6 digit number : " +
digitPattern.matcher(number).matches());
		}
	}
}
```

Reverse Words in a String

```
public class RevWords {
	public static void main(String[] args) {
		// using s.split("\\s");
		String s = "My name is Satya";
		String words[] = s.split("\\s");
		String rev = "";
		int len = words.length;
		for (int i = (len - 1); i >= 0; i--) {
			rev = rev + words[i];
		}
		System.out.println(rev);

		// using Collections.reverse(str)
		List<String> word = Arrays.asList(s.split("\\s"));
		Collections.reverse(word);
		System.out.println(word);
	}
}
```

What does the intern() method of String class do? (answer)

The intern() method of String class put the String on which it has called into String pool e.g. str.intern() will put the String str into the pool. Once the String is the pool it can be reused and improve performance.

How to convert String to Date in Java? (answer)

Prior to Java 8, you can use DateFormat or SimpleDateFormat class to convert a String to Date In Java or vice-versa. From Java 8 onwards, when you use the new Date and Time API, you can also use the DateTimeFormatter class to convert String to LocalDate, LocalTime, or LocalDateTime class in Java.

```
String string = "February 6, 2014";
	date = new SimpleDateFormat("MM/dd/yyyy").parse(string);
```

Difference between format() and printf() method in Java? (answer)

Even though both methods can be used to format String and they have same rules the key difference is that format() method returns a formatted String while printf() method print formatted String to console. So, if you need a formatted String, use format method and if you want to print, then use the printf() method.

How do you append leading zero to a numeric String? (answer)

You can use the format() method of String to append leading zeros to a numeric String in Java. See the link for an example

Enum

Enumeration in Java is supported by keyword enum. enums are a special type of class that always extends java.lang.Enum.enums are **Compiletime Constants, because they are "public static final**

Logically, **each enum is an instance of enum type** itself. So given enum can be seen as below declaration. **JVM internally adds ordinal and value methods** to this class which we can call while working with enum.

```
public enum Direction
{
   EAST, WEST, NORTH, SOUTH;
}
```

```
final class Direction extends Enum<Direction>
{
    public final static Direction EAST = new Direction();
    public final static Direction WEST = new Direction();
    public final static Direction NORTH = new Direction();
    public final static Direction SOUTH = new Direction();
}
```

The `ordinal()` method returns the order of an enum instance. It represents the sequence in the enum declaration, where the initial constant is assigned an ordinal of `'0'`. It is very much like array indexes.

```
Direction.EAST.ordinal();     //0
Direction.NORTH.ordinal();    //2
```

The **enum values()** method returns all the enum values in an **enum array**.

```
Direction[] directions = Direction.values();
```

By default, **enums don't require constructor** definitions and their default values are always the string used in the declaration. you can give define your own values by constructors to initialize.

```
public enum Direction
{
    // enum fields
    EAST(0), WEST(180), NORTH(90), SOUTH(270);
    // internal state
    private int angle;

    // constructor
    private Direction(final int angle) {
        this.angle = angle;
    }

    public int getAngle() {
        return angle;
    }
Public Static void Main(){
Direction north = Direction.NORTH;

System.out.println( north );                      //NORTH

System.out.println( north.getAngle() );           //90
System.out.println( Direction.NORTH.getAngle() ); //90
}
}
```

Remember that enum is basically a special class type, and can have methods and fields just like any other class. You can add methods which are **abstract** as well as **concrete methods** as well. Both methods are allowed in enum.

Two classes have been added to `java.util` package in support of enums – EnumSet and EnumMap

```
public class Test
{
   public static void main(String[] args)
   {
     Set enumSet = EnumSet.of(  Direction.EAST,
                                Direction.WEST,
                                Direction.NORTH,
                                Direction.SOUTH
                             );
Map enumMap = new EnumMap(Direction.class);

    //Populate the Map
    enumMap.put(Direction.EAST, Direction.EAST.getAngle());
    enumMap.put(Direction.WEST, Direction.WEST.getAngle());
    enumMap.put(Direction.NORTH, Direction.NORTH.getAngle());
    enumMap.put(Direction.SOUTH, Direction.SOUTH.getAngle());
   }
 }
```

1) Can Enum implement interface in Java?

Yes, Enum can implement interface in Java. Since enum is a type, similar to class and interface, it can implement interface. This gives a lot of flexibility to use Enum as specialized implementation in some cases

Can Enum extends class in Java?

No, Enum can not extend class in Java. Surprised, because I just said it's a type like a class or interface in Java. Since all Enum by **default extend abstract base class java.lang.Enum**, obviously they cannot extend another class, because Java doesn't support multiple inheritance for classes. Because of extending java.lang.Enum class, all enum gets methods like ordinal(), values() or valueOf().

Can we create instance of Enum outside of Enum itself? If Not, Why?

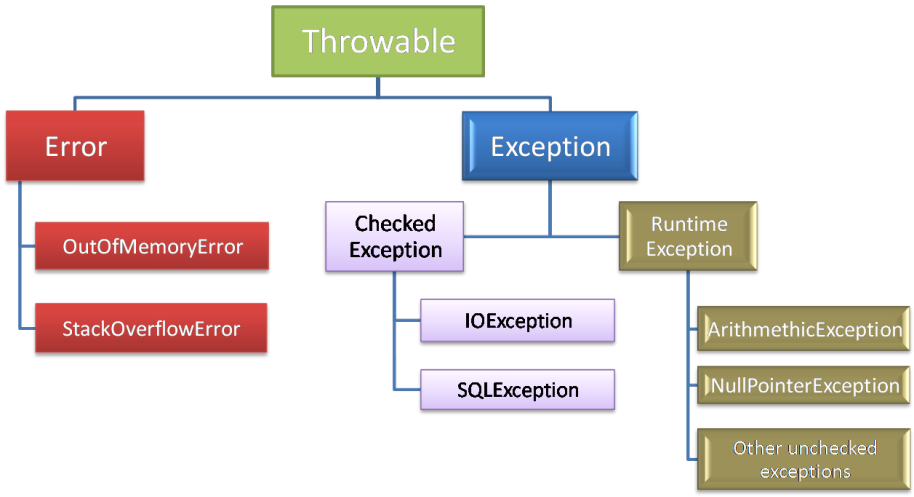
No, you cannot create enum instances outside of Enum boundary, because **Enum doesn't have any public constructor**, and compiler doesn't allow you to provide any public constructor in Enum.

Since compiler generates lot of code in response to enum type declaration, it doesn't allow public constructors inside Enum, which enforces declaring enum instances inside Enum itself.

Can we declare Constructor inside Enum in Java?

Yes, you can, but remember you **can only declare either private or package-private constructor** inside enum. public and protected constructors are not permitted inside enum. See here for a code example.

Exception Handling



1.Checked Exceptions: They Cheked by Compiler, the given resource is existed or not, They are usually occur interacting with outside resources/ network resources e.g. database problems, network connection errors, missing files etc. **Java forces you to handle these error scenarios in some manner in your application code**

2.Unchecked Exceptions: occurrences of which are not checked by the compiler like coding, initialization, Primitive data errors. They usually result of bad code in your system.

3.Error : JVM +Memory+ OS level issues . OutofMemory, StatckOverFlow

What will happen if you put System.exit(0) on try or catch block?

In normal Finally block will always execute. The only case finally block is not executed is **System.exit(0).** In advanced case it will execute in following case.

By Calling System.exit(0) in try or catch block, its stops execution & throws **SecurityException**.

- If System.exit(0) **NOT throws security exception, then finally block Won't be executed**
- But, if **System.exit(0) throws security exception then finally block will be executed**.

java.lang.System.exit() will terminates the currently executing program by JVM.

- **exit(0)** : Generally used to indicate successful termination.
- **exit(1) or exit(-1) or any other non-zero value** –indicates unsuccessful termination.

What happens if we put return statement on try/catch? Will finally block execute?

Yes, finally block will execute even if you put a return statement in the try block or catch block.

```
try {
    //try block
    ...
    return success;
}
catch (Exception ex) {
    //catch block
    .....
    return failure;
}
```

```
finally {
    System.out.println("Inside finally");
}
```
The answer is yes. **finally** block will execute. The only case where it will not execute is when it encounters **System.exit().**

What happens when a finally block has a return statement?

Finally block overrides the value returned by try and catch blocks.

```
public static int myTestingFuncn(){
  try{
     ....
     return 5;
  }
  finally {
     ....
     return 19;
   }
}
```
This program would return value 19 since the value returned by try has been overridden by finally.

Why do you think Checked Exception exists in Java, since we can also convey error using RuntimeException?

Most of checked exceptions are in java.io package, which make sense because if you request any system resource and its not available, than a robust program must be able to handle that situation gracefully.

By declaring **IOException** as checked Exception, Java ensures that your provide that gracefully exception handling. Another possible reason could be to ensuring that system resources like file descriptors, which are limited in numbers, should be released as soon as you are done with that using catch or finally block

Have you faced OutOfMemoryError in Java? How did you solved that?

OutOfMemoryError in Java is a subclass of java.lang.VirtualMachineError and JVM throws java.lang.OutOfMemoryError when it **ran *out of memory in the heap*.**

An easy way to solve OutOfMemoryError in java is to increase the maximum heap size by using JVM options "-Xmx512M", this will immediately solve your OutOfMemoryError.

I/O Streams

1.ByteStreams(1 byte at a time) : read image, audio, video etc

```
FileOutputStream outputStream = new FileOutputStream("c:\a.txt");
for (int i = 0; i < 10; i++) {
    outputStream.write(i);
}
FileInputStream inputStream = new FileInputStream("c:\a.txt");
int i;
while ((i = inputStream.read()) != -1) {
    System.out.println("I : " + i);
}
```

2.CharacterStreams(2 Bytes at a time) : Charater file data

```
char[] ch ={ 'a', 'b', 'c', 'd', 'e' };
       FileWriter w = new FileWriter(filepath);
       w.write(ch);
       w.close();

       FileReader r = new FileReader(filepath);
       int i;
       while ((i = r.read()) != -1) {
              System.out.println(i + ":" + (char) i);
       }
```

3.Buffered Streams(1024 bytes at a time): Rather than read one byte at a time,it reads a larger block at a time into an internal buffer

```
        // 1.Create Stream Object
FileOutputStream fos = new FileOutputStream(filepath);
// 2.pass Stream object to BufferStream constructor
BufferedOutputStream bos = new BufferedOutputStream(fos);
String s = "SmlCodes.com -Programmimg Simplified";
byte[] b = s.getBytes();
bos.write(b);
bos.flush();

// 1.Create Stream Object
FileInputStream fis = new FileInputStream(filepath);
// 2.pass Stream object to BufferStream constructor
BufferedInputStream bis = new BufferedInputStream(fis);
int i;
while((i=bis.read())!=-1){
 System.out.println((char)i);
}
```

4.Data streams: I/O of primitive data type values (int, long, float, and double)

```
DataOutputStream dos = new DataOutputStream(new       FileOutputStream("sml.bin"));
dos.writeInt(10);
dos.writeUTF("Satya");

DataInputStream dis = new DataInputStream(new FileInputStream("sml.bin"));
System.out.println("Int : " + dis.readInt());
System.out.println("String : " + dis.readUTF());
```

5.**Object Streams** : object streams support I/O of objects. Serialization.

- Choose the appropriate class name whose object is participating in serialization.
- This class must implement java.io.Serializable interface

```
class Student implements Serializable {
    // Exception in thread "main" java.io.NotSerializableException: io.Student
    private int sno;
    private String name;
    private String addr;
}
public class Serialization {
       public static void main(String[] args) throws  Exception {
           Student student = new Student();
           student.setSno(101);
           student.setName("Satya Kaveti");
           student.setAddr("VIJAYAWADA");

           FileOutputStream fos = new FileOutputStream("student.txt");
           ObjectOutputStream oos = new ObjectOutputStream(fos);
           oos.writeObject(student);

           FileInputStream fis = new FileInputStream("student.txt");
```

```
                ObjectInputStream ois = new ObjectInputStream(fis);
                Student st = (Student)ois.readObject();
                System.out.println(st.getSno());
                System.out.println(st.getName());
                System.out.println(st.getAddr());
              }
              }
```

Can a Serializable class contain a non-serializable field in Java? (answer)

Yes, but you need to make it either **static or transient**.

A static variable cannot be serialized.

Static variables belong to a class and not to any individual instance. The concept of **serialization** is concerned with the object's current state. Only data associated with a specific instance of a class is **serialized**, therefore **static** member fields are ignored during **serialization**

- The answer is very simple, only instance variables will be participated in Serialization process
- static variables doesn't participate in Serialization process
- **Reason :** static variable isn't part of Object's state
- So, by declaring static data member with transient doesn't have any impact
- There won't be any compile-time or run-time error

Threads

What happens if we starts same Thread(ob) Twice?

```
public class ThreadDemo extends Thread {
    @Override
    public void run() {
        System.out.println("Iam Running");
    }
    public static void main(String[] args) {
        ThreadDemo ob = new ThreadDemo();
        ob.start();
        ob.start();
    }
}
Exception in thread "main" java.lang.IllegalThreadStateException
at java.lang.Thread.start(Thread.java:705)
at threads.ThreadDemo.main(ThreadDemo.java:11)
Iam Running
```

What guarantee volatile variable provides?

volatile provides the guarantee, changes made in one thread is visible to others.

What is busy spin?

Busy spinning or busy wait in a multi-threaded environment is a technique where other threads loop continuously waiting for a thread to complete its task and signal them to start.

```
while(spinningFlag){
```

```
 System.out.println("Waiting busy spinning");
}
```

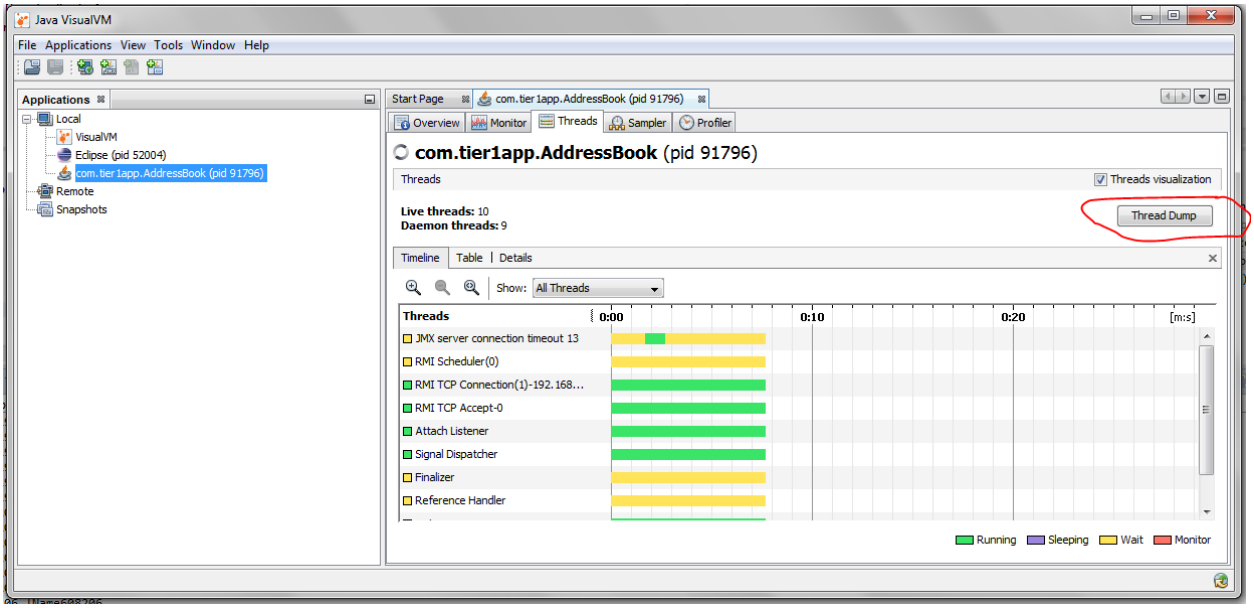
What is race condition in Java? Given one example? (answer)

Race condition are cause of some subtle programming bugs when Java programs are exposed to concurrent execution environment. As the name suggests, a race condition occurs due to race between multiple threads, if a thread which is supposed to execute first lost the race and executed second, behaviour of code changes, which surface as non-deterministic bugs. This is one of the hardest bugs to find and re-produce because of random nature of racing between threads. One example of race condition is out-of-order processing, see this answer for some more example of race conditions in Java programs.

What is Thread Dump? How do you take thread dump in Java?

Process has multiple Threads. **Thread dump** is a summary of the state of all **threads** of the process

- **'jstack'** is an effective command line tool to capture thread dumps
- **Java VisualVM** is a GUI tool that provides detailed information about the applications



Why Swing is not thread-safe in Java?

User can't click two buttons at a time right ? Since GUI screens are mostly updated in response of user action e.g. when user click a button, and since events are handled in the same Event dispatcher thread, it's easy to update GUI on that thread.

What is a ThreadLocal variable in Java?

Thread-local variables are variables restricted to a thread, it's like thread's own copy which is not shared between multiple threads. Java provides a **ThreadLocal** class to support thread-local variables, It extends Object class.

- Basically it is an another way to achieve thread safety apart from writing immutable classes.
- Since Object is no more shared there is no requirement of Synchronization which can improve scalability and performance of application.
- ThreadLocal provides thread restriction which is extension of local variable. ThreadLocal are visible only in single thread. No two thread can see each others thread local variable.
- These variable are generally **private static** field in classes and maintain its state inside thread.
- **void set(Object value), Object get(), void remove()** methods are available

```
public class ThreadLocalExample {
	public static class MyRunnable implements Runnable {
		private ThreadLocal<Integer> threadLocal = new ThreadLocal<Integer>();
		public void run() {
			threadLocal.set((int) (Math.random() * 100D));
			System.out.println(threadLocal.get());
		}
	}

	public static void main(String[] args) throws InterruptedException {
		MyRunnable sharedRunnableInstance = new MyRunnable();
		Thread thread1 = new Thread(sharedRunnableInstance);
		Thread thread2 = new Thread(sharedRunnableInstance);
		thread1.start();
		thread2.start();

		thread1.join(); // wait for thread 1 to terminate
		thread2.join(); // wait for thread 2 to terminate
	}
}
-----------
36
16
```

This example creates a single MyRunnable instance which is passed to two different threads. Both threads execute the run() method, and thus sets different values on the ThreadLocal instance. If the access to the set() call had been synchronized, and it had not been a ThreadLocal object, the second thread would have overridden the value set by the first thread

Write code for thread-safe Singleton in Java?

When we say thread-safe, which means Singleton should remain singleton even if initialization occurs in the case of multiple threads.

```
public class DoubleCheckedLockingSingleton {
	private volatile DoubleCheckedLockingSingleton INSTANCE;

	private DoubleCheckedLockingSingleton() {
	}

	public DoubleCheckedLockingSingleton getInstance(){
	 if(INSTANCE == null){
	    synchronized(DoubleCheckedLockingSingleton.class){
	        //double checking Singleton instance
	        if(INSTANCE == null){
	            INSTANCE = new DoubleCheckedLockingSingleton();
	        }
	    }
	 }
	 return INSTANCE;
    }
}
```

When to use Runnable vs Thread in Java? (Think Inheritance)

it's better to implement Runnable then extends Thread if you **also want to extend another class**

Difference between Runnable and Callable in Java?

Callable was added on JDK 1.5. Main difference between these two is that Callable's **call()** method can return value and throw Exception, which was not possible with Runnable's run() method. Callable return **Future** object, which can hold the result of computation.

```
public class CallableDemo {
   public static void main(String[] args) throws Exception {
            ExecutorService service =  Executors.newSingleThreadExecutor();
            SumTask sumTask = new SumTask(20);
            Future<Integer> future = service.submit(sumTask);
            Integer result = future.get();
            System.out.println(result);
   }
}
class SumTask implements Callable<Integer> {
         private int num = 0;
        public SumTask(int num){
                this.num = num;
        }
         @Override
         public Integer call() throws Exception {
                 int result = 0;
                 for(int i=1;i<=num;i++){
                         result+=i;
                 }
                 return result;
         }
}
```

How to stop a thread in Java?

There was some control methods in JDK 1.0 e.g. **stop(), suspend() and resume()** which are deprecated. To manually stop, programmers either take advantage of volatile boolean variable and check in every iteration if run method has loops or **interrupt** threads to abruptly cancel tasks.

Why wait, notify and notifyAll are not inside thread class?

Java provides lock at object level not at thread level. Every object has lock, which is acquired by thread. Now if thread needs to wait for certain lock it make sense to call wait() on that object rather than on that thread. Had wait() method declared on Thread class, it was not clear that for which lock thread was waiting. In short, since wait, notify and notifyAll operate at lock level, it make sense to defined it on object class because lock belongs to object.

What is the difference between livelock and deadlock in Java?

A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time. In short, the main difference between livelock and deadlock is that in former state of process change but no progress is made.

How do you check if a Thread holds a lock or not?

There is a method called **holdsLock()** on java.lang.Thread, it returns true if and only if the current thread holds the monitor lock on the specified object.

```
Thread t = Thread.currentThread();
System.out.println(Thread.holdsLock(t));
```

What is Semaphore in Java?

Semaphore in Java is a new kind of synchronizer. It's a counting semaphore. Conceptually, a semaphore maintains a set of permits. Each **acquire()** blocks if necessary until a permit is available, and then takes it. Each **release()** adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly. Semaphore is used to protect an expensive resource which is available in fixed number e.g. database connection in the pool.

What is the difference between the submit() and execute() method thread pool in Java?

- **execute(Runnable command)** is defined in Executor interface and executes given task in future, but more importantly, it does not return anything.
- **submit()** is an overloaded method, it can take either **Runnable or Callable** task and can return Future object which can hold the pending result of computation. This method is defined on ExecutorService interface, which extends Executor interface, and every other thread pool class e.g. ThreadPoolExecutor or ScheduledThreadPoolExecutor gets these methods.

Which method of Swing API are thread-safe in Java?

I know about **repaint(),** and revalidate() being thread-safe but there are other methods on different swing components e.g. **setText() method of JTextComponent, insert() and append()** method of JTextArea class.

What is the difference between the volatile and atomic variable in Java?

For example count++ operation will not become atomic just by declaring count variable as volatile. On the other hand AtomicInteger class provides atomic method to perform such compound operation atomically e.g. getAndIncrement() is atomic replacement of increment operator. It can be used to atomically increment current value by one. Similarly, you have atomic version for other data type and reference variable as well.

What happens if a thread throws an Exception inside synchronized block?

To answer this question, no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired while entering that synchronized block.

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

Key point here is order, if you acquire resources in a particular order and release resources in reverse order you can prevent deadlock.

What is busy spin, and why should you use it?

Busy spinning is a waiting strategy in which one thread loop continuously to check certain condition and waiting for other thread to change this condition to break the loop without releasing CPU so that waiting thread can proceeds its work further

What's the difference between Callable and Runnable?

Both of these are interfaces used to carry out task to be executed by a thread. The main difference between the two interfaces is that

- Callable can **return a value**, while Runnable cannot.
- Callable can throw a checked exception, while Runnable cannot.
- Runnable has been around since Java 1.0, while Callable was introduced as part of Java 1.5.

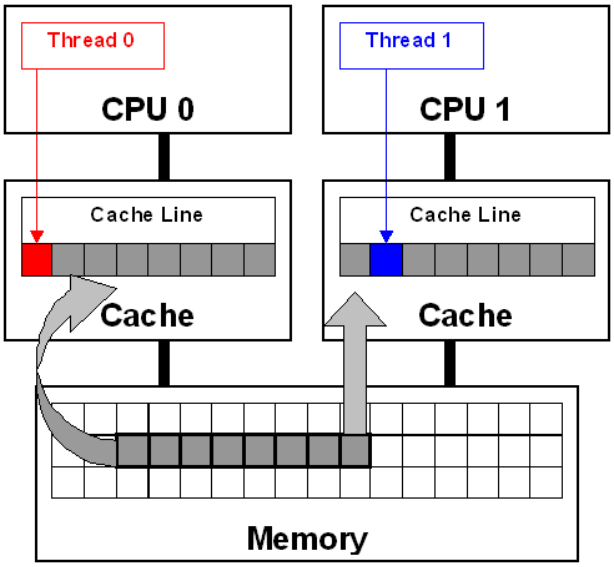
The *Callable* interface is a generic interface containing a single *call()* method – which returns a generic value *V*:

```
public interface Callable<V> {
    V call() throws Exception;
}
class CallableExample implements Callable
{

    public Object call() throws Exception
    {
        Random generator = new Random();
        Integer randomNumber = generator.nextInt(5);
        Thread.sleep(randomNumber * 1000);

        return randomNumber;
```

```
    }
}
```

What is false sharing in the context of multi-threading?

false sharing is one of the well-known performance issues on multi-core systems, where each process has its local cache.



False sharing is very hard to detect because the thread may be accessing completely different global variables that happen to be relatively close together in memory. Like many concurrency issues, the primary way to avoid false sharing is careful code review and aligning your data structure with the size of a cache line

Object level and Class level locks in Java

Object level lock : Every object in java has a unique lock. Whenever we are using synchronized keyword, then only lock concept will come in the picture. If a thread wants to execute synchronized method on the given object. First, it has to get lock of that object. Once thread got the lock then it is allowed to execute any synchronized method on that object. Once method execution completes automatically thread releases the lock. Acquiring and release lock internally is taken care by JVM and programmer is not responsible for these activities. Lets have a look on the below program to understand the object level lock:

```
class Geek implements Runnable {
    public void run()
    {
        Lock();
    }
    public void Lock()
    {
        System.out.println(Thread.currentThread().getName());
        synchronized(this)
        {
            System.out.println("in block "
                + Thread.currentThread().getName());
            System.out.println("in block " +
                Thread.currentThread().getName() + " end");
        }
    }

    public static void main(String[] args)
    {
        Geek g = new Geek();
        Thread t1 = new Thread(g);
        Thread t2 = new Thread(g);
        Geek g1 = new Geek();
        Thread t3 = new Thread(g1);
        t1.setName("t1");
        t2.setName("t2");
        t3.setName("t3");
        t1.start();
        t2.start();
        t3.start();
    }
}
```

```
Run on IDE
Output:
t1
in block t1
in block t1 end
t2
in block t2
in block t2 end
t3
in block t3
in block t3 end
```

Class level lock : Every class in java has a unique lock which is nothing but class level lock. If a thread wants to execute a static synchronized method, then thread requires class level lock. Once a thread got the class level lock, then it is allowed to execute any static synchronized method of that class. Once method execution completes automatically thread releases the lock. Lets look on the below program for better understanding:

```
// Java program to illustrate class level lock
class Geek implements Runnable {
    public void run()
    {
        Lock();
    }

    public void Lock()
    {
        System.out.println(Thread.currentThread().getName());
        synchronized(Geek.class)
        {
            System.out.println("in block "
                + Thread.currentThread().getName());
            System.out.println("in block "
```

```
                + Thread.currentThread().getName() + " end");
        }
    }

    public static void main(String[] args)
    {
        Geek g1 = new Geek();
        Thread t1 = new Thread(g1);
        Thread t2 = new Thread(g1);
        Geek g2 = new Geek();
        Thread t3 = new Thread(g2);
        t1.setName("t1");
        t2.setName("t2");
        t3.setName("t3");
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Producer-Consumer solution using threads in Java

- The producer's job is to generate data, put it into the buffer, and start again.
- same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.
- producer won't try to add data into the buffer if it's full & consumer won't try to remove data from an empty buffer

```
class Producer extends Thread {

	List buffer;
	int maxsize;

	public Producer(List buffer, int maxsize) {
		this.buffer = buffer;
		this.maxsize = maxsize;
	}

	@Override
	public void run() {
	int i = 1;
	  while (true) {
	     synchronized (buffer) {
		try {
			if (buffer.size() == maxsize) {
			System.out.println("Maximum Size Reached, wait until consume");
			buffer.wait();
			} else {
			buffer.add(i++);
			System.out.println(i + " : Produced, notify wating COnsumer Thread");
			buffer.notifyAll();

			}
		} catch (InterruptedException e) {
				// TODO Auto-generated catch block
			e.printStackTrace();
		}
		}

		}

	}
```

```
}

class Consumer extends Thread {

	List buffer;
	int maxsize;

	public Consumer(List buffer, int maxsize) {
		this.buffer = buffer;
		this.maxsize = maxsize;
	}

	@Override
	public void run() {

	while (true) {
		try {
		synchronized (buffer) {
		if (buffer.isEmpty()) {
			System.out.println("Consumer : Buffer Empty, wait untill produce");
			buffer.wait();
			} else {
			Object ob = buffer.remove(0);
		System.out.println(ob + " : Removed, notify Producer waiting for Removing for
maxsize");
		buffer.notifyAll();
			}
		}
		} catch (Exception e) {
				// TODO: handle exception
		}

	}

   }

}

public class ProducerConsumer {

	public static void main(String[] args) {

		List buffer = new LinkedList<>();
		Producer producer = new Producer(buffer, 10);
		Consumer consumer = new Consumer(buffer, 10);
		producer.start();
		consumer.start();

	}

}
28054 : Produced, notify wating COnsumer Thread
28055 : Produced, notify wating COnsumer Thread
28056 : Produced, notify wating COnsumer Thread
28057 : Produced, notify wating COnsumer Thread
28058 : Produced, notify wating COnsumer Thread
28059 : Produced, notify wating COnsumer Thread
28060 : Produced, notify wating COnsumer Thread
Maximum Size Reached, wait until consume
28050 : Removed, notify Producer waiting for Removing for maxsize
28051 : Removed, notify Producer waiting for Removing for maxsize
28052 : Removed, notify Producer waiting for Removing for maxsize
28053 : Removed, notify Producer waiting for Removing for maxsize
28054 : Removed, notify Producer waiting for Removing for maxsize
```

```
28055 : Removed, notify Producer waiting for Removing for maxsize
28056 : Removed, notify Producer waiting for Removing for maxsize
28057 : Removed, notify Producer waiting for Removing for maxsize
28058 : Removed, notify Producer waiting for Removing for maxsize
28059 : Removed, notify Producer waiting for Removing for maxsize
Consumer : Buffer Empty, wait untill produce
```

Thread. yield ()

yield() method: Theoretically, **to 'yield' means to let go, to give up, to surrender**. A yielding thread tells the virtual machine that it's willing to let other threads be scheduled in its place.

This indicates that it's not doing something too critical. Note that *it's only a hint*, though, and not guaranteed to have any effect at all.

- Yield is a Static method and Native too.
- Yield tells the currently executing thread to give a chance to the threads that have equal priority in the Thread Pool.
- There is no guarantee that Yield will make the currently executing thread to runnable state immediately.
- It can only make a thread from Running State to Runnable State, not in wait or blocked state.

What do you understand about Thread Priority?

Every thread has a priority, usually higher priority thread gets precedence in execution but it depends on Thread Scheduler implementation that is OS dependent. We can specify the priority of thread but it doesn't guarantee that higher priority thread will get executed before lower priority thread.

How can we make sure main() is the last thread to finish in Java Program?

We can use Thread join() method to make sure all the threads created by the program is dead before finishing the main function.

Why wait(), notify() and notifyAll() methods have to be called from synchronized method or block?

When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait state until any other thread call notify() on this Object. Similarly when a thread calls notify() on any Object, it leaves the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized method or block.

How can we achieve thread safety in Java?

There are several ways to achieve thread safety in java – **synchronization, atomic concurrent classes, implementing concurrent Lock interface, using volatile keyword**, using immutable classes and Thread safe classes.

What is volatile keyword in Java

When we use volatile keyword with a variable, all the threads read it's value directly from the memory and don't cache it. This makes sure that the value read is the same as in the memory.

What is ThreadLocal?

Java ThreadLocal is used to create thread-local variables. We know that all threads of an Object share it's variables, so if the variable is not thread safe, we can use synchronization but if we want to avoid synchronization, we can use ThreadLocal variables.
Every thread has it's own ThreadLocal variable and they can use it's get() and set() methods to get the default

value or change it's value local to Thread. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread

What is Java Thread Dump, How can we get Java Thread dump of a Program?

Thread dump is list of all the threads active in the JVM, thread dumps are very helpful in analyzing bottlenecks in the application and analyzing deadlock situations. There are many ways using which we can generate Thread dump – Using Profiler, Kill -3 command, jstack tool etc. I prefer jstack tool to generate thread dump of a program because it's easy to use and comes with JDK installation

What is atomic operation? What are atomic classes in Java Concurrency API?

Atomic operations are performed in a single unit of task .int++ is not an atomic operation. So by the time one threads read it's value and increment it by one, other thread has read the older value leading to wrong result.

To solve this issue, we will have to make sure that increment operation on count is atomic, we can do that using Synchronization but Java 5 java.util.concurrent.atomic provides wrapper classes for int and long that can be used to achieve this atomically without usage of Synchronization

What is BlockingQueue? implement Producer-Consumer using Blocking Queue?

- java.util.concurrent.BlockingQueue is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.
- BlockingQueue doesn't accept null values and throw NullPointerException if you try to store null value in the queue.
- BlockingQueue implementations are thread-safe. All queuing methods are atomic in nature and use internal locks or other forms of concurrency control.
- BlockingQueue interface is part of java collections framework and it's primarily used for implementing producer consumer problem.
  Check this post for producer-consumer problem implementation using BlockingQueue.

What is Executors Class?

Executors class provide utility methods for Executor, ExecutorService, ScheduledExecutorService, ThreadFactory, and Callable classes.

Executors class can be used to easily create Thread Pool in java, also this is the only class supporting execution of Callable implementations.

What happens when an Exception occurs in a thread?

Thread.UncaughtExceptionHandler is an interface, defined as nested interface for handlers invoked when a Thread abruptly terminates due to an uncaught exception.

When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for its UncaughtExceptionHandler using Thread.getUncaughtExceptionHandler() and will invoke the handler's uncaughtException() method, passing the thread and the exception as arguments.

Why wait, notify and notifyAll are not inside thread class?

One reason which is obvious is that Java provides lock at object level not at thread level.

How do you check if a Thread holds a lock or not?

Boolean Thread.*holdsLock*(Obj)

What is FutureTask in Java? (answer)

This class provides a base implementation of Future, it retrieve the result of the computation. It will get the results from Feature Object.

What is the concurrency level of ConcurrentHashMap in Java? (answer)

ConcurrentHashMap achieves it's scalability and thread-safety by partitioning actual map into a number of sections. This partitioning is achieved using concurrency level. Its optional parameter of ConcurrentHashMap constructor and it's default value is 16. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. To learn more about concurrency level and internal resizing

What happens if a thread throws an Exception inside synchronized block?

To answer this question, no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired while entering that synchronized block. This is actually one of the reasons I like synchronized block over lock interface, which requires explicit attention to release lock, generally this is achieved by releasing the lock in a finally block.

Collections

Java Collections class

Java collection class is used exclusively with static methods that operate on or return collections. It inherits Object class.

boolean addAll(Collection c, T... elements): This method adds all of the provided elements to the specified collection at once. The elements can be provided as a comma-separated list.

```
List list = new ArrayList();
Collections.addAll(list, "Apples", "Oranges", "Banana");
list.forEach(System.out::println);
```

void sort(List list, Comparator c): This method sorts the provided list according to the natural ordering. We can also pass in s Comparator, if we want some custom ordering.

```
Collections.sort(list);
Collections.sort(list,comparator);
```

int binarySearch (list,"elemet") : This method searches the key using binary search in the specified list. The list should be sorted by natural ordering, before calling this method, otherwise, the result will be undefined

```
System.out.println(Collections.binarySearch(fruits, "Banana"));
System.out.println(Collections.binarySearch(fruits, "Grapes"));
```

- Collections.*copy*(list, fruits);
- Collections.*fill*(list, "filled with dummy data"); : replaces all of the elements of the specified list with the specified element.
- Collections.max(fruits): returns the maximum element in collection according to the natural ordering of elements.
- Collections.reverse(list);
- Collections.unmodifiableList(band)
- Collections.synchronizedCollection(fruits)
    - synchronizedSet
    - synchronizedSortedSet
    - synchronizedMap
    - synchronizedSortedMap

Java9 Collection Static Factory Methods

```
List<String> list= List.of("apple","bat");
List<String> list= List.of();

Set<String> set= Set.of("apple","bat");
Set<String> set= Set.of()

Map<Integer,String> emptyMap = Map.of()
Map<Integer,String> map = Map.of(1, "Apple", 2, "Bat", 3, "Cat")

Map<Integer,String> emptyEntry = Map.ofEntries()
Map.Entry<Integer,String> mapEntry1 = Map.entry(1,"Apple")
Map.Entry<Integer,String> mapEntry2 = Map.entry(2,"Bat")
Map.Entry<Integer,String> mapEntry3 = Map.entry(3,"Cat")
Map<Integer,String> mapEntry = Map.ofEntries(mapEntry1,mapEntry2,mapEntry3)
```

Arrays Class

```
public static <T> List<T> asList(T… a)
```

```
public static void sort(int[] a)
public static int binarySearch(int[] a, int k)
public static boolean equals(int[] a, int[] a2)

Arrays.toString(ar);
static int[] copyOf(int[] original, int newLength);
public static void fill(int[] a, int val)
```

Comparable and Comparator

Comparators and comparable in Java are two interfaces which is used to implement sorting in Java.

Comparable object is capable of comparing itself(this) with another object. The class itself must implements the **java.lang.Comparable** interface to compare its instances

Comparator is external to the element type we are comparing. It's a separate class. We create multiple separate classes (that implement Comparator) to compare by different members.

Comparable interface

It provide single sorting sequence only i.e. you can sort the elements on based on single data member only. For example it may be rollno, name, age or any one of them, not all else.

We use **public int compareTo(Object obj):** is used to compare the current object with the specified object.

☑ **String class and Wrapper classes implements Comparable interface by default. So if you store the objects of string or wrapper or Date classes in list, set or map, it will be Comparable by default.**

Collection.sort(EmpBo): if we pass employee list Objects to the Collections.sort() method it will throws

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
	The method sort(List<T>) in the type Collections is not applicable for the arguments
(List<Employee>)
```

Comparable is an interface defining a strategy of comparing an object with other objects of the same type. This is called the class's "natural ordering".so we need to define CompareTo() method

```
public class Employee implements Comparable<Employee> {

	private int id;
	private String name;
	private double salary;
//Setters/getters

	public Employee(int id, String name, double salary) {
		super();
		this.id = id;
		this.name = name;
		this.salary = salary;
	}


	@Override
	public int compareTo(Employee o) {
		if (this.id < o.id) {
			return -1;
		} else if (this.id > o.id) {
```

```
                        return 1;
                } else {
                        return 0;
                }
        }

        @Override
        public String toString() {
                return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";
        }

        public static void main(String[] args) {

                List<Employee> employees = new ArrayList<Employee>();
                employees.add(new Employee(105, "Satya", 3000));
                employees.add(new Employee(102, "RAJ", 2000));
                employees.add(new Employee(104, "Madhu", 5000));
                employees.add(new Employee(101, "Srini", 1000));
                employees.add(new Employee(103, "Vinod", 4000));

                System.out.println("Before : " + employees);
                Collections.sort(employees);
                System.out.println("After : " + employees);
        }
}
Before : [Employee [id=105, name=Satya, salary=3000.0], Employee [id=102, name=RAJ,
salary=2000.0], Employee [id=104, name=Madhu, salary=5000.0], Employee [id=101, name=Srini,
salary=1000.0], Employee [id=103, name=Vinod, salary=4000.0]]
After : [Employee [id=101, name=Srini, salary=1000.0], Employee [id=102, name=RAJ,
salary=2000.0], Employee [id=103, name=Vinod, salary=4000.0], Employee [id=104, name=Madhu,
salary=5000.0], Employee [id=105, name=Satya, salary=3000.0]]
```

Now, suppose we want sort Employees by their salary and names & also Default comparable method also Should there. In this case using comparable we get only one chance to implement the compareTo() method. The solution is using **Comparator**

Comparator Interface

Comparator interface ***compare(Object o1, Object o2)*** method need to be implemented that takes two Object argument

```
class EmpName implements Comparator<Employee> {
        public int compare(Employee o1, Employee o2) {
                return o1.getName().compareTo(o2.getName());
        };
}

class EmpSalary implements Comparator<Employee> {
        public int compare(Employee o1, Employee o2) {
                if (o1.getSalary() < o2.getSalary()) {
                        return -1;
                } else if (o1.getSalary() > o2.getSalary()) {
                        return 1;
                }
                return 0;
        }
}

public class Employee implements Comparable<Employee> {
        private int id;
```

```
	private String name;
	private double salary;
//Setters & Getters
	public Employee(int id, String name, double salary) {
		super();
		this.id = id;
		this.name = name;
		this.salary = salary;
	}

	@Override
	public int compareTo(Employee o) {
		if (this.id < o.id) {
			return -1;
		} else if (this.id > o.id) {
			return 1;
		} else {
			return 0;
		}
	}

	@Override
	public String toString() {
		return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";
	}

	public static void main(String[] args) {

		List<Employee> employees = new ArrayList<Employee>();
		employees.add(new Employee(105, "AAA", 3000));
		employees.add(new Employee(102, "ZZZ", 2000));
		employees.add(new Employee(104, "BBB", 5000));
		employees.add(new Employee(101, "DDD", 1000));
		employees.add(new Employee(103, "CCC", 4000));

		System.out.println("Before : " + employees);
		Collections.sort(employees);
		System.out.println("ByID :\n " + employees);

		//Now we can Sort our Emplyees based on Multiple Sorting(EmpName, EmpSaltry)
		Collections.sort(employees, new EmpName());
		System.out.println("EmpName : \n "+employees);

		Collections.sort(employees, new EmpSalary());
		System.out.println("EmpSalary : \n "+employees);

	}

}
----------------------------------------
Before : [Employee [id=105, name=AAA, salary=3000.0], Employee [id=102, name=ZZZ,
salary=2000.0], Employee [id=104, name=BBB, salary=5000.0], Employee [id=101, name=DDD,
salary=1000.0], Employee [id=103, name=CCC, salary=4000.0]]
ByID :
[Employee[id=101, name=DDD, salary=1000.0], Employee [id=102, name=ZZZ, salary=2000.0],
Employee [id=103, name=CCC, salary=4000.0], Employee [id=104, name=BBB, salary=5000.0],
Employee [id=105, name=AAA, salary=3000.0]]
EmpName :
[Employee[id=105, name=AAA, salary=3000.0], Employee [id=104, name=BBB, salary=5000.0],
Employee [id=103, name=CCC, salary=4000.0], Employee [id=101, name=DDD, salary=1000.0],
Employee [id=102, name=ZZZ, salary=2000.0]]
EmpSalary :
```

```
[Employee[id=101, name=DDD, salary=1000.0], Employee [id=102, name=ZZZ, salary=2000.0],
Employee [id=105, name=AAA, salary=3000.0], Employee [id=103, name=CCC, salary=4000.0],
Employee [id=104, name=BBB, salary=5000.0]]
```

- **Comparable** interface can be used to provide **single way of sorting** whereas **Comparator** interface is used to provide **different ways of sorting**.
- For using Comparable, Class needs to implement it whereas for using Comparator we don't need to make any change in the class, **we can implement it in outside**.
- **Comparable** interface is in **java.lang** package whereas **Comparator** interface is present in **java.util** package.
- We don't need to make any code changes at client side for using Comparable, Arrays.sort() or Collection.sort() methods automatically uses the compareTo() method of the class. For Comparator, client needs to provide the Comparator class to use in compare() method.

PriorityQueue : https://www.callicoder.com/java-priority-queue/

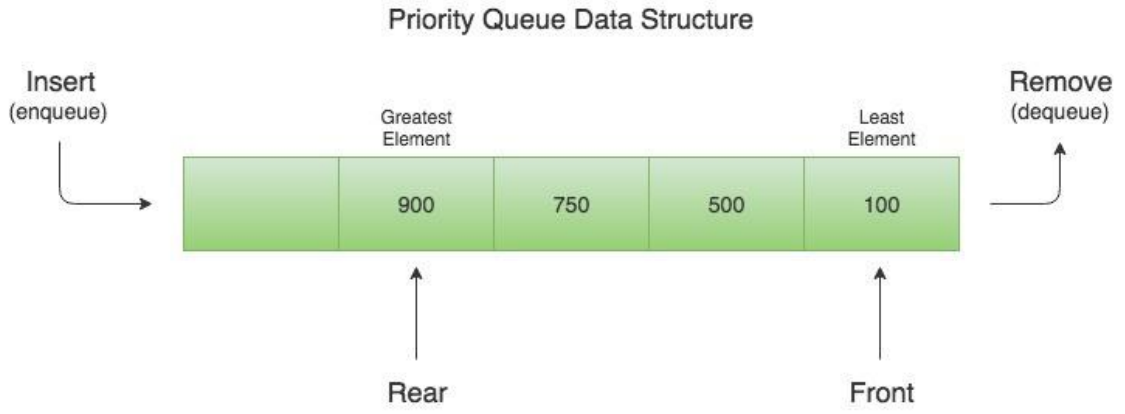
A Queue in Java is just an interface. We need a concrete implementation of the Queue interface to work with, in our programs. LinkedList class implements the Queue interface and therefore it can be used as a Queue.

The process of adding an element at the end of the Queue is called **Enqueue**, and the process of removing an element from the front of the Queue is called **Dequeue**.

A priority queue in Java is a special type of queue wherein all the elements are **ordered**

- as per their **natural ordering** using Comparable or
- based on a **custom Comparator** supplied at the time of creation.

The ***front*** of the priority queue contains the least element according to the specified ordering, and the ***rear*** of the priority queue contains the greatest element.



So when you remove an element from the priority queue, the least element according to the specified ordering is removed first.

```
public class Demo {
public static void main(String[] args) {
Queue<Integer> q = new PriorityQueue<>();
q.offer(400);
q.add(200);
q.add(700);
q.add(100);
```

```
q.add(500);
 while (!q.isEmpty()) {
       System.out.println(q.remove());
 }
}
}
```

```
100
200
400
500
700
```

Let's say that we need to create a priority queue of String elements in which the String with the smallest *length* is processed first.

We can create such a priority queue by passing a custom Comparator that compares two Strings by their length

Since a priority queue needs to compare its elements and order them accordingly, the user defined class must implement the Comparable interface, or you must provide a Comparator while creating the priority queue. **Otherwise, the priority queue will throw a ClassCastException when you add new objects to it.**

Difference between poll() and remove() method?

Both poll() and remove() take out the object from the Queue but if **poll() fails then it returns null** but if **remove() fails it throws Exception**.

Ways that you could sort a collection?

use the Sorted collection like **TreeSet** or **TreeMap** or you can sort using the ordered collection like a list and using **Collections.sort()** method

How do you print Array in Java?

array doesn't implement toString() by itself, just passing an array to System.out.println() will not print its contents but **Arrays.toString()** will print each element

```
String a[] =  {"a","b","c"};
System.out.println(a);
System.out.println(Arrays.toString(a));
```

What is the difference between ArrayList and Vector ?

Synchronization and Thread-Safe
Vector is synchronized while ArrayList is not synchronized

Performance
Vector is slow as it is thread safe . In comparison ArrayList is fast

Automatic Increase in Capacity
A Vector defaults to **doubling size** ,ArrayList ,it increases its Array size by **(curr.capcity*3)/2 + 1**

Enumeration & iterator
Vector is the only other class which uses **both Enumeration and Iterator** .While ArrayList can only use Iterator for traversing an ArrayList

Difference between Hashtable and ConcurrentHashMap in Java?

Answer : Both Hashtable and ConcurrentHashMap is used in multi-threaded environment because both are therad-safe but main difference is on performance Hashtable's performance become poor if the size of Hashtable become large because it will be locked for long time during iteration but in case of concurrent HaspMap only specific part is locked because concurrent HaspMap works on segmentation and other thread can access the element without iteration to complete. To learn more about how ConcurrentHashMap achieves it's thread-safety, scalability using lock stripping and non blocking algorithm

Read more: http://www.java67.com/2014/07/21-frequently-asked-java-interview-questions-answers.html#ixzz5fofpCIHh

Is it possible for two unequal objects to have the same hashcode?

Yes, two unequal objects can have the same hashcode. This is why collision can occur in hashmap. The equal hashcode contract only says **that two equal objects must have the identical hashcode**, but there is no indication to say anything about the unequal object.

Differences between HashMap and HashTable in Java.

- HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code whereas Hashtable is **synchronized**.
- **HashMap allows one null key and multiple null values** whereas **Hashtable doesn't allow any null key or value.**

Which two method you need to implement for key Object in HashMap ?

In order to use any object as Key in HashMap, it must implements equals and hashcode method in Java.

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

if you put the same key again than it will replace the old mapping because HashMap doesn't allow duplicate keys

difference between Iterator and Enumeration in Java?

Property	Enumeration	Iterator	ListIterator
1) Is it legacy?	Yes	no	no
2) It is applicable for?	Only legacy classes.	Applicable for any collection object.	Applicable for only list objects.
3) Moment?	Single direction cursor(forward)	Single direction cursor(forward	Bi-directional.
4) How to get it?	By usingelements() method.	By using iterator()method.	By using listIterator() method.
5) Accessibility?	Only read.	Both read and remove.	Read/remove/replace/add.
6) methods	hasMoreElement() nextElement()	hasNext() next() remove()	9 methods.

What is the difference between fail-fast and fail-safe Iterators?

1. Can come if two threads trying to modify one list at same time e.g. one Thread is iterating over it and other is removing elements from it.
2. But, more commonly, it also comes when you use ArrayList's remove() method while iterating over List.
3. Always use Iterator's remove() method to delete elements when traversing.
4. Don't delete elements when looping over ArrayList using advanced for loop, because it doesn't expose iterator's remove method, but will throw ConcurrentModificationException if you delete using ArrayList's remove() method.

.The Collection specicific remove() method throws Exception, but not Iterator based remove() method

Collection interface defines remove(Object obj) method to remove objects from Collection. List interface adds another method remove(int index), which is used to remove object at specific index. You can use any of these method to remove an entry from Collection, while not iterating.

If we traversing a if we use Iterator's remove() method, it will removes current element from Iterator's perspective. If you use Collection's or List's remove() method during iteration then will throw **ConcurrentModificationException**.

```
 public class FailFastExample {
       public static void main(String args[]){
        List<String> myList = new ArrayList<String>();

        myList.add("1");
        myList.add("2");
        myList.add("3");

        Iterator<String> it = myList.iterator();
        while(it.hasNext()){
            String value = it.next();
            System.out.println("List Value:"+value);
            if(value.equals("2"))
               myList.remove(value); // ThrowsException
            //it.remove(value);     // Not Throws Exception

        }
    }
}
----
List Value:1
List Value:2
Exception in thread "main" java.util.ConcurrentModificationException
       at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:901)
       at java.util.ArrayList$Itr.next(ArrayList.java:851)
       at theads.FailFastExample.main(FailFastExample.java:21)
```

Avoid ConcurrentModificationException in multi-threaded environment

- You can lock the list while iterating by putting it in a synchronized block.
- you can use **ConcurrentHashMap** and **CopyOnWriteArrayList**classes

in single-threaded environment,You can use the iterator **remove()** function to remove the object from underlying collection object.

How do you Sort objects on the collection? (solution)

Sorting is implemented using Comparable and Comparator in Java and when you call Collections.sort() it gets sorted based on the natural order specified in compareTo() method while Collections.sort(Comparator) will sort objects based on compare() method of Comparator.

Can we replace Hashtable with ConcurrentHashMap? (answer)

Yes, we can replace Hashtable with ConcurrentHashMap and that's what suggested in Java documentation of ConcurrentHashMap. but you need to be careful with code which relies on locking behavior of Hashtable.

Since **Hashtable locks whole Map** instead of a portion of Map, compound operations like if(Hashtable.get(key) == null) put(key, value) works in Hashtable but not in concurrentHashMap. instead of this use putIfAbsent() method of ConcurrentHashMap

What is CopyOnWriteArrayList, how it is different than ArrayList and Vector? (answer)

Answer: CopyOnWriteArrayList is new List implementation introduced in Java 1.5 which provides better concurrent access than Synchronized List. better concurrency is achieved by Copying ArrayList over each write and replace with original instead of locking. Also CopyOnWriteArrayList doesn't throw any ConcurrentModification Exception. Its different than ArrayList because its thread-safe and ArrayList is not thread-safe and it's different than Vector in terms of Concurrency. CopyOnWriteArrayList provides better Concurrency by reducing contention among readers and writers. Here is a nice table which compares performance of three of popular List implementation ArrayList, LinkedList and CopyOnWriteArrayList in Java

Read more: https://javarevisited.blogspot.com/2011/11/collection-interview-questions-answers.html#ixzz5fwos8T00

JDBC

What is JNDI?

JNDI is the Java Naming and Directory Interface. It's used to separate the concerns of the application *developer* and the application *deployer*. When you're writing an application which relies on a database, you shouldn't need to worry about the user name or password for connecting to that database. JNDI allows the developer to give a name to a database, and rely on the deployer to map that name to an actual instance of the database.

Add a file `META-INF/context.xml` into the root of your web application folder, which defines database connection details

```
Context>
```

```
  <Resource name="jdbc/mkyongdb" auth="Container" type="javax.sql.DataSource"
                maxActive="50" maxIdle="30" maxWait="10000"
                username="mysqluser" password="mysqlpassword"
                driverClassName="com.mysql.jdbc.Driver"
                url="jdbc:mysql://localhost:3306/mkyongdb"/>

</Context>
```

In web.xml, defines your MySQL datasource again :

```
  <resource-ref>
        <description>MySQL Datasource example</description>
        <res-ref-name>jdbc/mkyongdb</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
  </resource-ref>
```

get the datasource via context lookup service

```
private DataSource ds;

        public CustomerBean(){
          try {
                Context ctx = new InitialContext();
                ds = (DataSource)ctx.lookup("java:comp/env/jdbc/mkyongdb");
          } catch (NamingException e) {
                e.printStackTrace();
          }
        }
```

What are the steps to connect to the database in java?

```
public class JDBC {
        public static void main(String[] args) throws Exception {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con = DriverManager.getConnection
                                ("jdbc:mysql://localhost:3306/mydb", "root", "root");

                Statement stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT * FROM customer");

                while (rs.next())
                System.out.println(rs.getInt(1) + ": " + rs.getString(2));
        }
}
```

What are the JDBC statements?

There are 3 types of JDBC Statements, as given below:

- **Statement**: It will execute SQL query (static SQL query) against the database.
- **Prepared Statement:** Used when we want to execute SQL statement repeatedly. Input data is dynamic and taken input at the run time.
- **Callable Statement:** Used when we want to execute stored procedures.

```
public CallableStatement prepareCall("{ call procedurename(?,?...?)}");
CallableStatement cs=con.prepareCall("{call myprocedure(?,?)}");
```

Explain the difference between RowSet vs. ResultSet in JDBC?

RowSet extends the ResultSet interface, so it holds all methods from ResultSet. RowSet is serialized.

What is the difference between executing, executeQuery, executeUpdate in JDBC?

- **boolean** execute(): it can be used for any kind of SQL Query.
- **ResultSet** executeQuery() : it can be used for select query.
- **int** executeUpdate(): it can be used to change/update table.

What is JDBC database Connection Pool? How to setup in Java?

JDBC connection pool maintains pool of JDBC connection which is used by application to query database. Since JDBC connection are expensive it take time to create them which can slow response time of server if created during request time. Creating them on application start-up and reusing them result in better performance.

What is use of setAutoCommit(false) in JDBC ?

By **default** setAutoCommit() is **TRUE** . making setAutoCommit(false) saves a lot of performance as it doesn't commit transaction automatically after each query and we do batch update. It allows you to handle it using commit() and rollback().

Batch Processing?

Instead of executing a single query, we can execute a group of queries.The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing

- **void addBatch(String query) –** It adds query into batch.
- **int[] executeBatch() –** It executes the batch of queries.

```
Statement stmt=con.createStatement();
stmt.addBatch("insert into user420 values(190,'abhi',40000)");
stmt.addBatch("insert into user420 values(191,'umesh',50000)");
stmt.executeBatch();//executing the batch
```

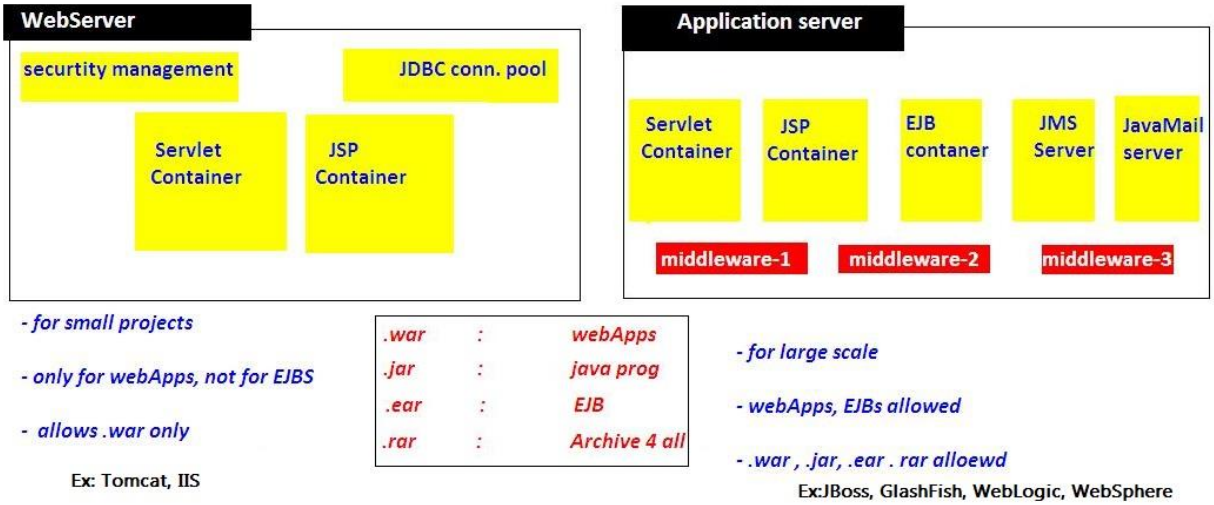
Difference between java.util.Date and java.sql.Date in Java? (answer)

java.util.Date contains both date and time while **java.sql.Date** contains only date part

Read more: http://www.java67.com/2018/03/top-50-core-java-interview-questions.html#ixzz5fuYL91FG

Servlets

Web Server VS Application Server?



Servlet Lifecycle & execution flow?

- When ever we deploys the application, container loads the application & creates **ServletContext** Object & waits for the Request
- if we give **<load-on-startup>**1**</load-on-startup>**container will creates ServletConfig Object when the time of Deploying application
- when we give the url : http://localhost:8080/Servlets/hello **,** request goes to container, and it searches for **/hello**url pattern in web.xml
- xml searches for **/hello** , in **<servlet-mapping>** and gets **Servelt-name**
- container loads HelloServlet class and creates creates **ServletConfig** Object and calls inti() method
- for every request it will calls **service(req,res)** method, for 100 requests it will execute 100 times
- **destroy()** method will be called before servlet is removed from the container, and finally it will be garbage collected as usual.

HttpServlet flow of execution?

- Container first calls **public Service(req,res)** method
- Public Service() method internally calls **protected Service(req,res)** method
- Protected Service() method will internally calling **doGet() or doPost() or doXXX()** depends on the type of http method used by the client
- If the client is **not specifying the type of Http** method then Http protocol by **default consider GET method**,
- so **finally** the client request is processed at **doGet() method**

ServletRequest ?

ServletRequest is send to Server to process particular request. It can send following details to servlet by submitting FORM or by URL.we can get these details at server side

- public String **getParameter("paramname");**
- public Enumeration **getParameterNames();**

- public String[] **getParamterValues("paramname");**

How can we create deadlock condition on our servlet? (detailed answer)

Ans: one simple way to call doPost() method inside doGet() and doGet()method inside doPost() it will create deadlock situation for a servlet.

Difference between DOM and SAX parser in Java? (answer)

DOM loads whole XML File in memory while SAX doesn't. It is an event based parser and can be used to parse a large file, but DOM is fast and should be preferred for small files.

Read more: http://www.java67.com/2018/03/top-50-core-java-interview-questions.html#ixzz5fuXtTqen

Hibernate

What are advantages of Hibernate?

- Lazy Loading
- Caching
- You do not need to maintain JDBC code , Hibernate takes care of it.
- You need to write less code
- It provides high level object oriented API

What are some core interfaces of hibernate?

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

Difference between get() vs load() method in Hibernate? (detailed answer)

The key difference between get() and load() method is that

- **load() will throw an exception** if an object with id passed to them is not found
- **get() will return null.**

Another important difference is that load can return proxy without hitting the database unless required (when you access any attribute other than id) but get() always go to the database, so sometimes using load() can be faster than the get() method. It makes sense to use the load() method if you know the object exists but get() method if you are not sure about object's existence.

Parameter	get	load
Database retrieval	It always hits the database	It does not hit database
If null	If it does not get the object with id, it returns null	If it does get the object with id, it throws ObjectNotFoundException
Proxy	It returns real object	It returns proxy object
Use	If you are not sure if object with id exists or not, you can use get	If you are sure about existence of object, you can use load

What is the difference between save() and persist() method in Hibernate?

- **Serializable Object save()** returns a Serializable object
- **void persist()** method is void, so it doesn't return anything.

Does SessionFactory is thread-safe in Hibernate? (detailed answer)

SessionFactory is both Immutable and thread-safe and it has just one single instance in Hibernate application. It is used to create Session object and it also provide caching by storing SQL queries stored by multiple session. The second level cache is maintained at SessionFactory level.

Does Hibernate Session interface is thread-safe in Java? (detailed answer)

No, Session object is not thread-safe in Hibernate and intended to be used with-in single thread in the application.

What is difference between getCurrentSession() and openSession() in Hibernate?

openSession() When you call SessionFactory.openSession, it always create new Session object afresh and give it to you. As session objects are not thread safe, you need to create one session object per request in multithreaded environment and one session per request in web applications too.

getCurrentSession() When you call SessionFactory. getCurrentSession , it creates a new Session if not exists , else use same session which is in current hibernate context. It automatically flush and close session when transaction ends, so you do not need to do externally.If you are using hibernate in single threaded environment , you can use getCurrentSession, as it is faster in performance as compare to creating new session each time.

You need to add following property to hibernate.cfg.xml to use getCurrentSession method

```
<session-factory>
<!--  Put other elements here -->
<property name="hibernate.current_session_context_class"></property>
</session-factory>
```

If you do not configure above property, you will get error as below.

Exception in thread "main" org.hibernate.HibernateException: No CurrentSessionContext configured!

Can you declare Entity(Bean) class as final in hibernate?

Yes, you can declare entity class as final but it is not considered as a good practice because hibernate uses proxy pattern for lazy initialisation, If you declare it as final then hibernate won't be able to create sub class and won't be able to use proxy pattern, so it will limit performance and improvement options.

Does entity class in hibernate require no arg constructor?

Yes, Entity class in hibernate requires no arg constructor because Hibernate use reflection to create instance of entity class and it mandates no arg constructor in Entity class

How do you log SQL queries issued by the Hibernate framework in Java application?

You can procedure the **show_sql** property to log SQL queries delivered by the Hibernate framework

What is named SQL query in Hibernate?

Named queries are SQL queries which are defined in mapping document using **<sql-query>** tag and called using **Session.getNamedQuery()** method.

```
<sql-query name="findStudentByRollNumber">
         <!--[CDATA[
            select * from Student student where student.rollNumber = :rollNumber
          ]]-->
</sql-query>
```

you can define named query in hibernate either by using annotations or XML mapping file, as I said above. **@NameQuery** is used to define single named query and **@NameQueries** is used to define multiple named query in hibernate.

```
@NamedQueries({
          @NamedQuery(
          name = "findStockByStockCode",
          query = "from Stock s where s.stockCode = :stockCode"
          )
})
```

```
Query query = session.getNamedQuery("findStockByStockCode")
.setString("stockCode", "7277");
```

What is query cache in Hibernate?

.Query cache can be used along with second level cache for improved performance. QueryCache actually stores the result of SQL query for future calls. Hibernate support various open source caching solution to implement Query cache e.g. EhCache

What are two types of Collections in hibernate?

- Sorted Collection
- Ordered Collection

Parameter	Sorted Collection	Ordered Collection
Sorting	Sorted collection uses java's sorting API to sort the collection.	Ordered Collections uses order by clause while retrieval of objects
Default	It is enabled by default	It is not enabled by default, you need to enable it explicitly

What is lazy loading in hibernate?

Sometimes you have two entities and there's a relationship between them. For example, you might have an entity called University and another entity called Student

```
public class University {
 private String id;
 private String name;
 private String address;
 private List<Student> students;
 // setters and getters
}
```

Now when you load a University from the database, JPA loads its id, name, and address fields for you. But you have two options for students: to load it together with the rest of the fields (i.e. eagerly) or to load it on-demand (i.e. lazily) when you call the university's getStudents() method.

```
@OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER)
        @JoinColumn(name="countryId")
        private List<Student> students;
```

FetchType.LAZY: It fetches the child entities lazily, that is, at the time of fetching parent entity it just fetches proxy (created by cglib or any other utility) of the child entities and when you access any property of child entity then it is actually fetched by hibernate.

FetchType.EAGER: it fetches the child entities along with parent.

Lazy initialization improves performance by avoiding unnecessary computation and reduce memory requirements.

Eager initialization takes more memory consumption and processing speed is slow.
`lazy="true/false in xml`

Spring

What is IOC or inversion of control?

Inversion of the control means now we have transfer the control of creating the object from our own using new operator to container or framework. Now it's the responsibility of container to create an object as required.

Explain the Spring Bean-LifeCycle?

1. The container will look the bean definition inside configuration file (e.g. bean.xml).
2. using reflection container will create the object and if any property is defined inside the bean definition then it will also be set.
3. If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean's ID.
4. If the bean implements the BeanFactoryAware interface, the factory calls setBeanFactory(), passing an instance of itself.
5. If there are any BeanPostProcessors associated with the bean, their post- ProcessBeforeInitialization() methods will be called before the properties for the Bean are set.
6. If an init() method is specified for the bean, it will be called.
7. If the Bean class implements the DisposableBean interface, then the method destroy() will be called when the Application no longer needs the bean reference.
8. If the Bean definition in the Configuration file contains a 'destroy-method' attribute, then the

What are the difference between BeanFactory and ApplicationContext

ApplicationContext.	**BeanFactory**
Here we can have more than one config files possible	In this only one config file or .xml file
Application contexts can publish events to beans that are registered as listeners	Don't support.
Support internationalization (I18N) messages	It's not
Support application life-cycle events, and validation.	Doesn't support.
Supports many enterprise services such JNDI access, EJB integration, remoting	Doesn't support.

Spring MVC

What is default scope of bean in Spring framework & Soring WebApplication? (answer)

The default scope of a Spring bean is the **Singleton** scope and in the web application default scope of a spring bean is **request** scope. Singleton bean means the same instance of a bean is shared with all other beans, while request scope means a **bean is alive only for a request.**

What is the difference between @Controller and @RestController?

@RestController is better when you are developing RESTful web services using Spring MVC framework. It's a combination of @Controller + @ResponseBody annotation which allows the controller to directly write the response and bypassing the view resolution process, which is not required for RESTful web service.

It also instructs DispatcherServlet to use different HttpMessageConverters to represent the response in the format client is expecting e.g. HttpMessageJackson2Convert to represent response in JSON format and JAXB based message converts to generate XML response

What does @RequestMapping annotation do? (answer)

The @RequestMapping annotation is used to map web requests to Spring Controller methods. You can map request based upon HTTP methods e.g. GET and POST and various other parameters. For examples, if you are developing RESTful Web Service using Spring then you can use produces and consumes property along with media type annotation to indicate that this method is only used to produce or consumers JSON as shown below:

```
@RequestMapping (method = RequestMethod.POST, consumes="application/json")
public Book save(@RequestBody Book aBook) {
   return bookRepository.save(aBook);
}
```

When do you need @ResponseBody annotation in Spring MVC?

The @ResponseBody annotation can be put on a method to indicates that the return type should be written directly to the HTTP response body (and not placed in a Model, or interpreted as a view name).

```
@RequestMapping(path = "/hello", method = RequestMethod.PUT)
@ResponseBody
public String helloWorld() {
   return "Hello World";
}
```

Alternatively, you can also use @RestController annotation instead of @Controller annotation. This will remove the need for using @ResponseBody because as discussed in the previous answer, it comes automatically with @RestController annotation.

What does @PathVariable do in Spring MVC? Why it's useful in REST with Spring?

For example, in the URL http://myapp.com/books/101 if you want to extract 101 the id, then you can use @PathVariable annotation of Spring MVC

Where do you need @EnableWebMVC? (answer)

The @EnableWebMvc annotation is required to enable Spring MVC when Java configuration is used to configure Spring MVC instead of XML. It is equivalent to <mvc: annotation-driven> in XML configuration.

How to Call Stored procedure in Spring Framework?

To call a Stored procedure in Spring framework you need to create Class which will should extends **StoredProcedure** class. You just need to call the execute method from the DAO layer.

```
public class EmployeeInfo extends StoredProcedure
{
   private static final String EMP_ID = "EMP_ID";
   private static final String EMP_NAME = "EMP_NAME";
   private static final String JOIN_DATE = "JOIN_DATE";
   public SnapshotSearchStoredProcedure(DataSource dataSource, String procedureName)
   {
       super(dataSource, procedureName);
       declareParameter(new SqlParameter(EMP_ID, Types.NUMERIC));
       declareParameter(new SqlOutParameter(EMP_NAME, Types.VARCHAR));
       declareParameter(new SqlOutParameter(JOIN_DATE, Types.VARCHAR));
```

```
        compile ();
    }
    public Map execute(Integer empId)
    {
        Map<String, Object> inputs = new HashMap<String, Object>();
        inputs.put(P_CLD_IDR, empId);
        Map<String, Object> result = execute (inputs);
        return result;
    }
}
```

Spring Security

If we think about the meaning of authentication, it seems that it is all about a client identifying itself to the server. After client identification is done, the server can remember the client each time the request comes from the client. There are two common approaches to authentication mechanisms: one of them is called "Session Cookie Based" and the other one is "Token Based".

Session Cookie based

The most common approach we probably all know is to use a server generated secret token (Session key) in the form of a JSESSIONID cookie. Initial setup for this is near nothing these days perhaps making you forget you have a choice to make here in the first place. Even without further using this "Session key" to store any other state "in the session", the key itself is in fact *state* as well. I.e. without a shared and persistent storage of these keys, no successful authentication will survive a server reboot or requests being load balanced to another server.

OAuth2 / API keys

Whenever talking about REST APIs and Security; OAuth2 and other types of API keys are mentioned. Basically they involve sending custom tokens/keys within the HTTP Authorization header. When used properly both relieve clients from dealing with Cookies using the header instead. This solves CSRF vulnerabilities and other Cookie related issues. One thing they do not solve however is the need for the server to check the presented authentication keys, pretty much demanding some persistent and maintainable shared storage for linking the keys to users/authorizations.

```
        private HttpHeaders createHeaders(final String userId, final String password) {
                String auth = userId + ":" + password;
                byte[] encodedAuth =
Base64.encodeBase64(auth.getBytes(StandardCharsets.US_ASCII));
                String authHeader = "Basic " + new String(encodedAuth);

                HttpHeaders headers = new HttpHeaders();
                headers.set("Authorization", authHeader);
                return headers;
        }


private ResponseEntity<String> makeRestCall(String url, String userId,
                        String password) {
                // Basic Auth only.
                if (!"".equals(userId) && !"".equals(password)) {
```

```
                return restOperations.exchange(url, HttpMethod.GET,
                                new HttpEntity<>(createHeaders(userId, password)),
                                String.class);

        } else {
                return restOperations.exchange(url, HttpMethod.GET, null,
                                String.class);
        }

    }
```

Web services

What do you understand by payload in RESTFul?

Payload means data which passed inside request body also payload is not request parameters. So only you can do payload in POST and not in GET and DELTE method.

Can you do payload in HTTP DELETE?

This is again similar to previous REST interview question, answer is No. You can only pass payload using HTTP POST method.

How much maximum pay load you could do in POST method?

Answer : If you remember difference between GET and POST request then you know that unlike GET which passes data on URL and thus limited by maximum URL length, POST has no such limit. So, theoretically you can pass unlimited data as payload to POST method but you need to take practical things into account e.g. sending POST with large payload will consume more bandwidth, take more time and present performance challenge to your server.

References

1.https://javarevisited.blogspot.com/2017/01/how-to-prepare-for-java-interviews.html

- 133 Java Interview Questions from last 5 years (list) **(Done)**
- 50 Java Concurrency Interview Questions (list) **(Done)**
- 25 Java Collection Interview Questions (list) **(Done)**
- 10 Spring Framework Interview Questions (list) **(Done)**
- 20 Hibernate Framework Interview Questions with Answers (list)) **(Done)**
- 10 RESTful Web Service interview Questions for Java developers (list) **(Done)**
    - Top 10 Spring Framework Interview Questions with Answers (see here)
    - 20 Great Java Design Pattern Questions asked on Interviews (see here)
    - 10 popular Struts Interview Questions for Java developers (list)
    - 10 frequently asked Servlet Interview Questions with Answers (see here)
    - 20 jQuery Interview Questions for Java Web Developers (list)
    - 10 Great Oracle Interview Questions for Java developers (see here)
    - Top 10 JSP Questions from J2EE Interviews (read here)
    - 12 Good RESTful Web Services Questions from Interviews (read here)
    - Top 10 EJB Interview Questions and Answers (see here)
    - Top 10 JMS and MQ Series Interview Questions and Answers (list)
    - 10 Great Hibernate Interview Questions for Java EE developers (see here)
    - 10 Great JDBC Interview Questions for Java Programmers (questions)
    - 15 Java NIO and Networking Interview Questions with Answers (see here)

- Top 10 XSLT Interview Questions with Answers (read more)
- 15 Data Structure and Algorithm Questions from Java Interviews (read here)
- Top 10 Trick Java Interview Questions and Answers (see here)
- Top 40 Core Java Phone Interview Questions with answers (list)

2.https://www.pearsonfrank.com/blog/java-interview-questions/

3.https://howtodoinjava.com/java-interview-questions/

4.https://snowdream.github.io/

https://javaconceptoftheday.com/

http://www.thejavageek.com/core-java/