

# Interview Qs

*again*



Written By

*Satya Kaveti*

Latest Revision  
18 - June - 2021

<b>INDEX .....</b>	<b>3</b>
<b>CORE JAVA .....</b>	<b>4</b>
BASICS .....	4
JVM INTERNALS AND GARBAGE COLLECTION.....	7
##### WHAT'S THE USE OF INNERCLASSES IN REALTIME? .....	ERROR! BOOKMARK NOT DEFINED.
###STRING IS IMMUTABLE. SO, STRING CLASS IS FINAL ?.....	23
DATA TYPES .....	24
JAVA.LANG PACKAGE.....	33
JAVA OOPS CONCEPTS .....	41
DESIGN PATTERNS.....	42
STRINGS.....	47
ENUM .....	55
EXCEPTION HANDLING.....	58
I/O STREAMS .....	62
THREADS.....	64
COLLECTIONS.....	85
CODING.....	105
<b>ARCHITECTURE .....</b>	<b>110</b>
<b>JDBC.....</b>	<b>125</b>
<b>SERVLETS.....</b>	<b>127</b>
<b>HIBERNATE.....</b>	<b>127</b>
<b>WEB SERVICES .....</b>	<b>151</b>
<b>SPRING .....</b>	<b>166</b>
SPRING CORE .....	166
SPRING MVC.....	170
SPRING DATA .....	172
SPRING SECURITY .....	172
<b>SPRING MVC .....</b>	<b>173</b>
<b>SPRINGBOOT IQ'S .....</b>	<b>178</b>
<b>SQL.....</b>	<b>190</b>
<b>ANGULARJS .....</b>	<b>192</b>
<b>JUNIT.....</b>	<b>197</b>
<b>MONGODB VS SQL.....</b>	<b>198</b>
<b>REAL.....</b>	<b>200</b>
<b>H R MAPPING .....</b>	<b>204</b>
<b>REFERENCES .....</b>	<b>205</b>

# Index

---

- 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>

**ClassNotFoundException**

# Core Java

## Basics

### What is a strongly typed programming language?

In a strongly typed language compiler ensure type correctness, for example, you **can not store the**

**Number in String / String in Number vice-versa.**

**Java is a strongly typed language**, that's why you have different data types

Ex `int, float, String, char, boolean etc.`

In weakly typed language, it won't enforce type checking at compile time and their values based upon context. **Python and Perl** are two popular example of weakly typed programming language, where you can store a numeric string in number type.

### Can you describe three different kinds of testing that might be performed?

**Unit Testing, Integration Testing and Smoke Testing.**



- **Unit testing** is used to test individual units to verify whether they are working as expected
- **Integration testing** is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.
- **Smoke Testing** is a way to test whether most common functionality of software is working properly or not e.g. in a flight booking website, you should be able to book, cancel or change flights.
- **Penetration testing** is also known as pen testing or ethical hacking. It describes the intentional launching of simulated cyberattacks that seek out exploitable vulnerabilities in computer systems, networks, websites, and applications.

### What is the difference between iteration and recursion? (detailed answer)

- **Iteration** uses a **loop** to perform the same step again and again.
- **Recursion calls the same method** itself to do the repetitive task

## What is test-driven development?

Test driven is one of the popular development methodologies in which tests are written before writing any function code. In fact, test drives the structure of your program. Purists never wrote a single line of application code without writing a test for that. It greatly improves code quality and often attributed as a quality of rockstar developers.

## How do you find a running Java process on UNIX?

You can use the combination of '**ps**' and '**grep**' command to find any process running on UNIX machine.  
**ps -ef** will list every process detail including PID. Will use PID to kill this process using **kill -9**.

```
ps -ef | grep 'java'  
kill -9 16906
```

## Difference between WeakReference vs SoftReference vs PhantomReference vs Strong reference in Garbage Collection?

there are four kind of reference in Java:

1. **Strong reference**
2. **Weak Reference**
  - a. Soft Reference
  - b. Phantom Reference

### 1. Strong Reference

```
StringBuilder builder = new StringBuilder("Satya");
```

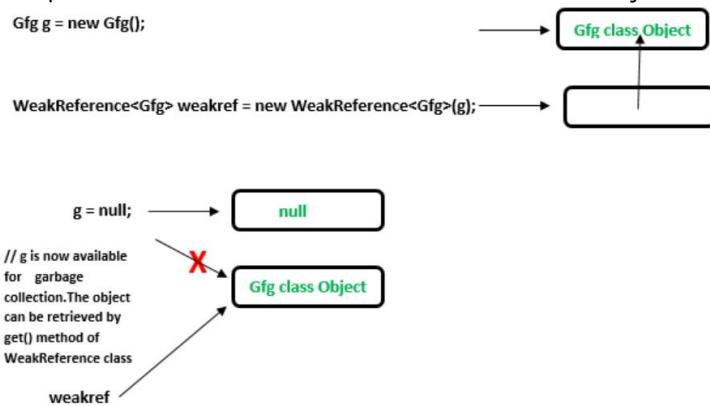
Here reference variable **builder** has strong reference to **StringBuilder** object "Satya". these are objects which is needed by Java program. Any object which has Strong reference attached to it is **not eligible for garbage collection**.

### 2. Weak Reference

To make Strong Reference to Weak reference, we have two ways. One is nullfing (**s = null**), second is using **java.lang.ref.WeakReference** class.

```
WeakReference<String> weakBuilder = new WeakReference<String>(name);
```

Weak Reference Objects are not the default available. we should be explicitly specified like in the above example. This kind of reference makes the reference object eligible for GC.



```

public class Example
{
    public static void main(String[] args)
    {
        // Strong Reference
        Gfg g = new Gfg();
        g.sumMethod();

        // Creating Weak Reference to Gfg-type object to which 'g' is also pointing.
        WeakReference<Gfg> weakref = new WeakReference<Gfg>(g);

        //Now, Gfg-type object to which 'g' was pointing earlier is available for garbage collection.
        //But, it will be garbage collected only when JVM needs memory.
        g = null;

        // You can retrieve back the object which has been weakly referenced.
        g = weakref.get();
    }
}

```

We have Two levels in **Weak Reference** – Soft Reference, Phantom Reference

**a. Soft References:** In Soft reference, even if the object is free for garbage collection, but it is not garbage collected until JVM is in need of memory badly. The objects gets cleared from the memory when JVM runs out of memory. To create such references `java.lang.ref.SoftReference` class is used.

```

public class MainClass
{
    public static void main(String[] args)
    {
        Gfg g = new Gfg();          //Strong Reference

        //Creating Soft Reference to Gfg-type object to which 'g' is also pointing
        SoftReference<Gfg> softGfg = new SoftReference<Gfg>(g);

        //Now, Gfg-type object to which 'g' is pointing earlier is eligible for garbage collection.
        //But it will be garbage collected only when JVM needs memory.
        g = null;

        //You can retrieve back the object which has been softly referenced
        g = softGfg.get();
    }
}

```

### b. Phantom References

The objects which are being referenced by phantom references are eligible for garbage collection. But, before removing them from the memory, JVM puts them in a queue called 'reference queue'. They are put in a reference queue after calling `finalize()` method on them. To create such references `java.lang.ref.PhantomReference` class is used.

```

public class Example
{
    public static void main(String[] args)
    {
        //Strong Reference
        Gfg g = new Gfg();
        g.x();

        //Creating reference queue
        ReferenceQueue<Gfg> refQueue = new ReferenceQueue<Gfg>();

        //Creating Phantom Reference to Gfg-type object to which 'g' is also pointing.
        PhantomReference<Gfg> phantomRef = new PhantomReference<Gfg>(g,refQueue);

        //Now, Gfg-type object to which 'g' was pointing earlier is available for garbage collection.
        //But, this object is kept in 'refQueue' before removing it from the memory.
    }
}

```

```

g = null;

//It always returns null.
g = phantomRef.get();

//It shows NullPointerException.
g.x();
}
}

```

## 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.

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

```

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. If you want to run JUnit from anywhere from cmdline you need 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 adds optimization to this process by analyzing the code in blocks and then prepares 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**, **OpenJDK**, **IBM J9**, **Azure Zulu** 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 operators 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?

It is exactly same as HashMap except following difference

- In the case of HashMap even though Object doesn't have any reference it is **NOT eligible** for garbage collection, if it is associated with HashMap.
- In case of **WeakHashMap** if Object doesn't have any references it is **eligible** for **gc()** even though object associated with WeakHashMap. that means Garbage collector dominates WeakHashMap.

```
This Temp class is common for HashMap & WeakHashMap Demo's
class Temp {
    @Override
    public String toString() {
        return "Temp";
    }
    @Override
    protected void finalize() throws Throwable {
        System.out.println("Finalize Called");
    }
}

HashMap Demo with GC
public class HashMapdemo {
    public static void main(String[] args) throws InterruptedException {
        HashMap m = new HashMap();
        Temp t = new Temp();
        m.put(t, "Satya");
        System.out.println(m);

        t=null;
        System.gc();
        Thread.sleep(5000);
        //main Thread Sleeping for 5 seconds, Garbage collector takes control for 5 seconds
        System.out.println(m);
    }
}

{Temp=Satya}
{Temp=Satya}
```

In the above example Temp object is not eligible for gc() because it is associated with HashMap.in this case output is {Temp=Satya} {Temp=Satya}

```
WeakHashMap Demo with GC
public class WeakHashMapdemo {
    public static void main(String[] args) throws InterruptedException {
        WeakHashMap m = new WeakHashMap();
        Temp t = new Temp();
        m.put(t, "Satya");
        System.out.println(m);
        t=null;
        System.gc();
        Thread.sleep(5000);
        System.out.println(m);
    }
}

{Temp=Satya}
Finalize Called
{}
```

In the above example Temp object is eligible for gc() because it is associated with HashMap.in this case output is {Temp=Satya} Finalize Called {}

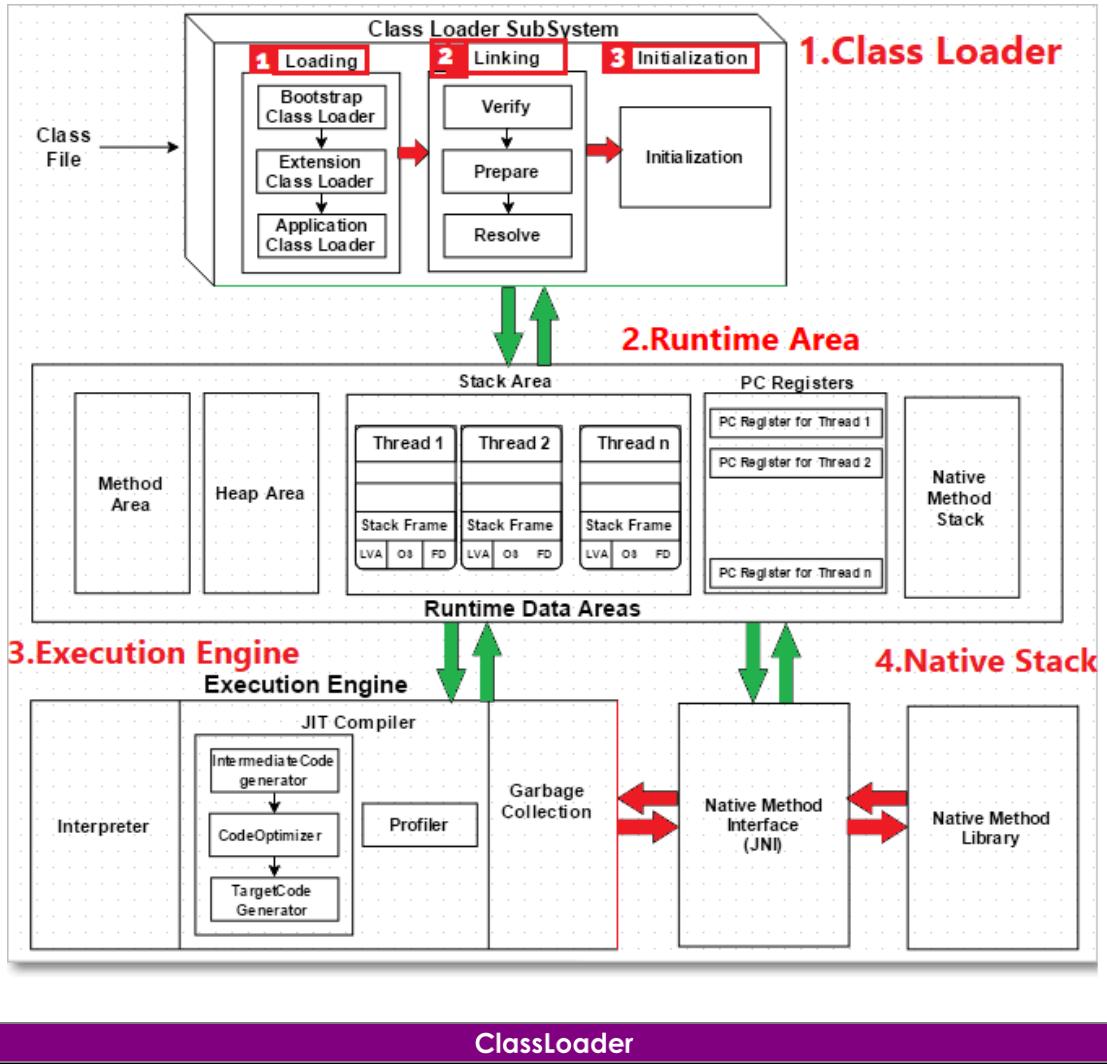
### How do you locate memory usage from a Java program?

Answer: You can use memory related methods from **java.lang.Runtime** class to get the **totalMemory()**, **freeMemory()** and Maximum Heap memory in Java.

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



### 1. Loading the Class:

When a Java program is converted into **.class** file by Java compiler **ClassLoader** is responsible to load that class file from file system or any other location. Our Java class is depending up on any other class, let's say **JdbcDriver.class**, it will search by following Class Loaders

- **Bootstrap ClassLoader - JRE/lib/rt.jar**  
First bootstrap class loader tries to find the class. It scans the **rt.jar** file in JRE **lib** folder.
- **Extension ClassLoader - JRE/lib/ext** or any directory denoted by **java.ext.dirs**  
If class is not found, then extension class loader searches the class file in inside **jre\lib\ext** folder
- **Application ClassLoader - CLASSPATH environment variable, -classpath or -cp option**  
Again, if class is not found then application ClassLoader searches all the Jar files and classes in **CLASSPATH** environment variable of system.

If class is found by any loader then class is loaded by class loader; else **ClassNotFoundException** is thrown

**2. Linking:** once Class is loaded it performs below operations

- **Bytecode verifier** will verify whether the generated bytecode is proper or not.
- **Prepare (memory allocation)**: allocates memory **to static variables & methods**.
- **Resolve** – All symbolic memory references are replaced with original references from Method Area.

**3. Initialization:** In prepare only memory is allocated, here all **static variables will be assigned** with the original values and the **static blocks will be executed**.

### Runtime area

**fields (Data members)** and **methods** are also known as **class members**.

- **Method Area:** all **Class level** Data members, Method definitions stored here.
- **Heap** All **Objects** & instance variable Data stored Here.
- **Stacks:** All **Methods executions & Thread Executions** done here. Stores local variables, and intermediate results. Each thread has its own JVM stack, created simultaneously as the thread is created. So, all such local variables are called **thread-local variables**.
- **PC registers:** store the physical memory address of the statements which is currently executing. In Java, each thread has its separate PC register.
- **Native Method Stack:** Java supports and uses **native code** as well. Many low-level codes is written in languages like C and C++. Native method stacks hold the instruction of native code.

### Execution Engine

All code assigned to JVM is executed by an **execution engine**. The execution engine reads the byte code and executes line by line. It uses two inbuilt tools – **interpreter** and **JIT compiler to convert the bytecode to machine code and execute it**.

1. **Interpreter** converts each byte-code instruction to native instruction. It directly executes the bytecode only one instruction at a time and **does not perform any optimization**.
2. **JIT Compiler takes a block of code** (not one statement at a time as interpreter), optimize the code and then translate it to optimized machine code. **To improve performance, it will Optimizes the bytecode**
3. **Garbage Collection:** Once code Execution done, it will clear the memory.

### Java Native Interface

- **Java Native Interface (JNI):** It is an interface which interacts with the Native Method Libraries and provides the native libraries (C, C++) required for the execution.
- **Native Method Libraries:** It is a collection of the Native Libraries which are required by the Execution Engine.

## **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.

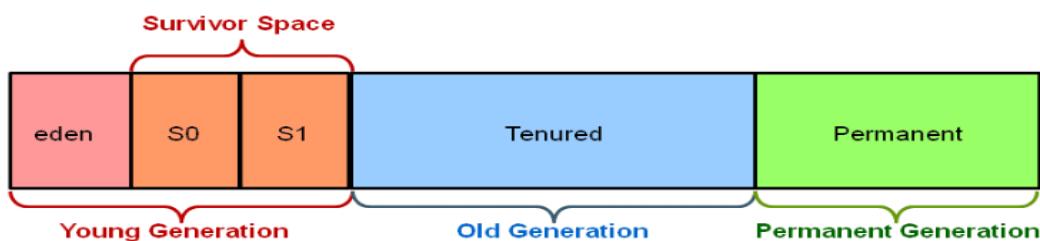
## **How to increase heap size in Java**

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**

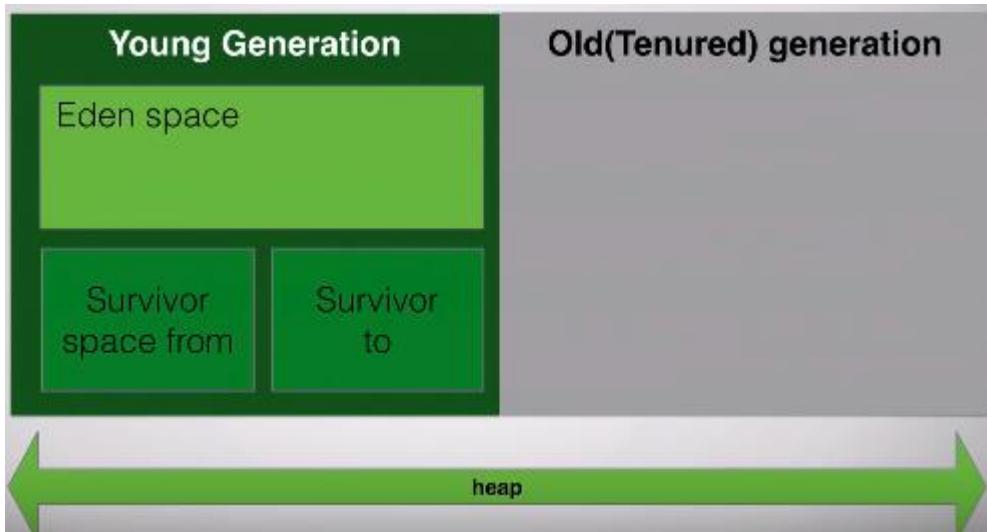


- **New/Young Generation** of Java Heap is part of Java Heap memory where a newly created object is stored,
- **Old Generation** During the course of application many objects created and died but those remain live they got moved to Old Generation by Java Garbage collector thread
- **Perm space** of Java 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.

### **1.New Generation has two parts**

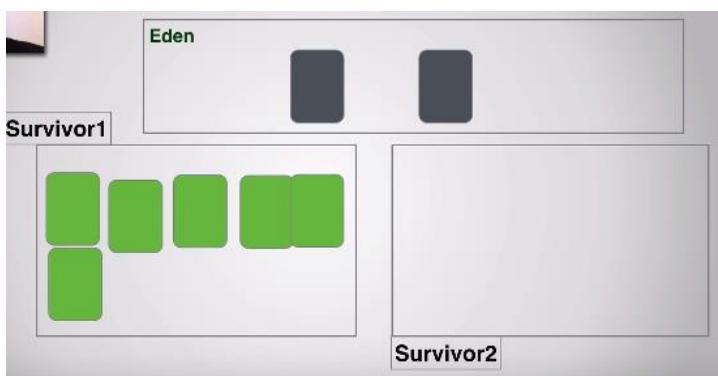
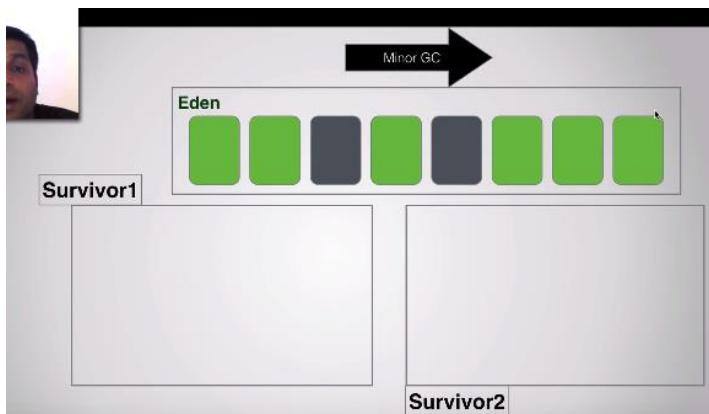
- **Eden Space** – New Objects are Created
- **Survivor Space** – Once Eden space is full, Minor thread try to clear the Space, the objects which are survived by GC will be placed in Survivor space.

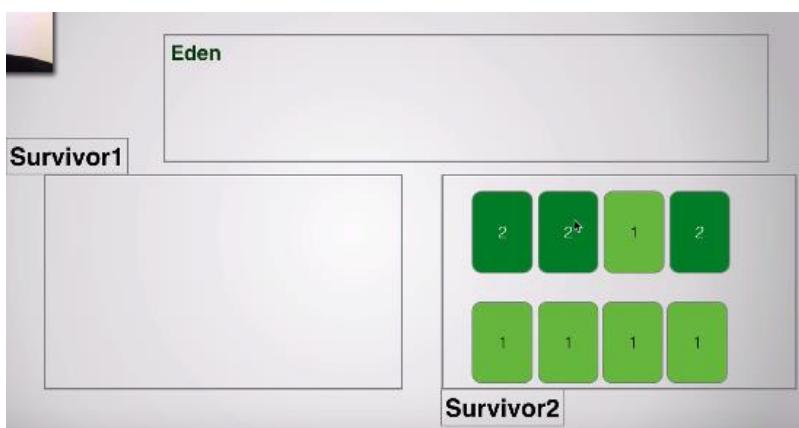
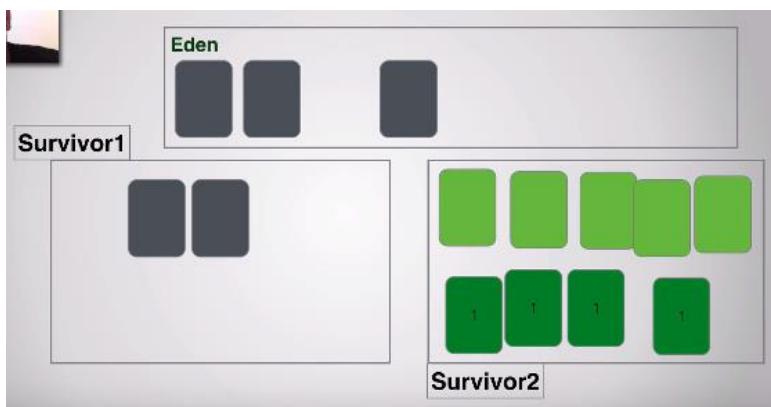
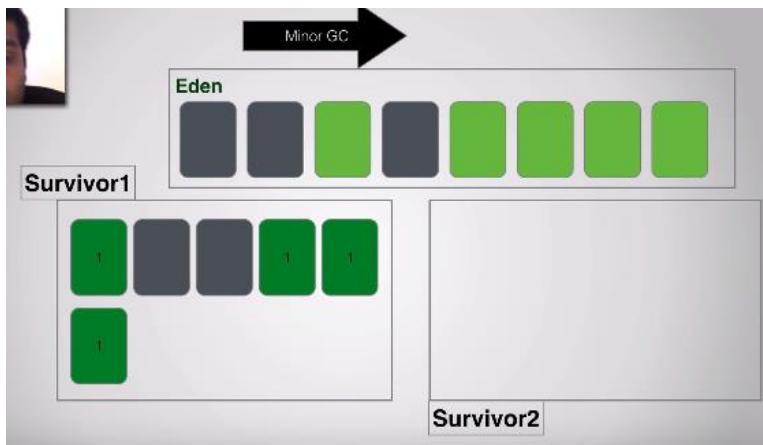
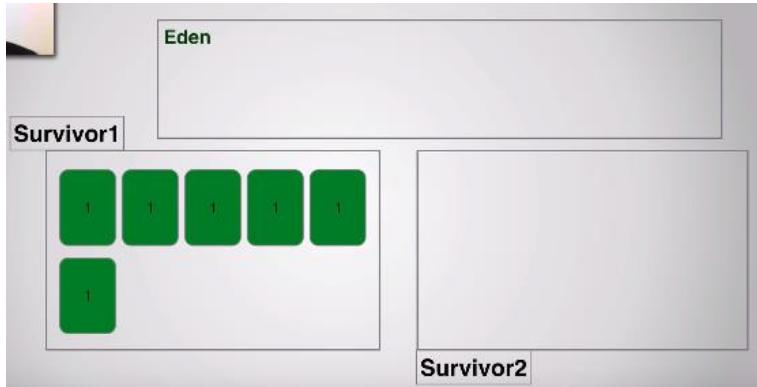
**2.Old Gen** – Objects which are survived from a long time, let say 16 Cycles (GC cycle Threshold) of Minor Garbage Collector, those objects moved to Old Gen. The Old Gen space finally collected by Main Garbage collector.

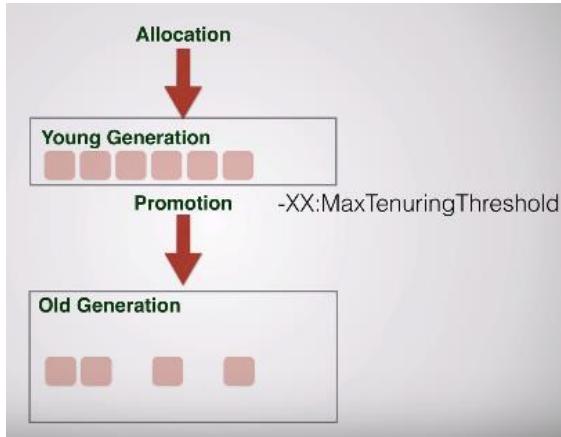


Finally,

- **Minor Garbage collector** will run only on **Young Gen**,
- **Main Garbage Collector** will run on **whole heap space**







**Does Garbage collection occur in permanent generation space in JVM?**

**YES, Garbage Collection occur in PermGen space as well. and if PermGen space is full or cross a threshold, it can trigger Full GC(Main Thread).**

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.

### Types of Garbage Collectors

When an object is no longer used, the garbage collector reclaims the underlying memory and reuses it for future object allocation. This means there is no explicit deletion, and no memory is given back to the operating system.

#### Involves

**Mark:** Starts from root node of your application(main), walks the object graph, marks objects that are reachable as live.

**Delete/sweep:** Delete unreachable objects

**Compacting:** Compact the memory by moving around the objects and making the allocation contiguous than fragmented.  
**(Arranging in order)**

#### Throughput

Throughput focuses on maximizing the amount of work by an application in a specific period of time. Examples of how throughput might be measured include:

The number of transactions completed in a given time.  
The number of jobs that a batch program can complete in an hour.  
The number of database queries that can be completed in an hour.  
High pause times are acceptable for applications that focus on throughput. Since high throughput applications focus on benchmarks over longer periods of time, quick response time is not a consideration.

Java has **four types of garbage collectors**,

- **Serial Garbage Collector**
- **Parallel Garbage Collector**
- **CMS Garbage Collector**
- **G1 Garbage Collector**

Each of these four types has its own advantages and disadvantages. Most importantly, we the programmers can choose the type of garbage collector to be used by the JVM. We can choose them by passing the choice as JVM argument

**1. Serial Garbage Collector**

- It is designed for **the single-threaded environments**.
- It **uses just a single thread for garbage collection**.
- It freezes(stops) all the application threads while performing garbage collection.
- it may not be suitable for a server environment. It is best suited for simple command-line programs.

Turn on the **-XX:+UseSerialGC** JVM argument to use the serial garbage collector.

**2. Parallel Garbage Collector**

- It is the **default garbage** collector of the JVM.
- It uses **multiple threads for garbage collection**.
- Similar to serial garbage collector this also freezes(stops) all the application threads while performing garbage collection.

**3. Concurrent Mark& Sweep Garbage Collector**

- Concurrent Mark & Sweep (CMS) garbage collector uses **multiple threads to scan the heap memory** to **mark instances** for eviction and **then sweep the marked instances**.
- It runs along with our Application. Uses multiple cores(cpus) to run multiple GC's concurrently.
- Here your application won't pass/stop

Turn on the **XX:+UseConcMarkSweepGC** JVM argument to use the CMS garbage collector.

**Use concurrent collector(CMS) when**

- There is more memory
- There is high number of CPUs
- Application demands short pauses

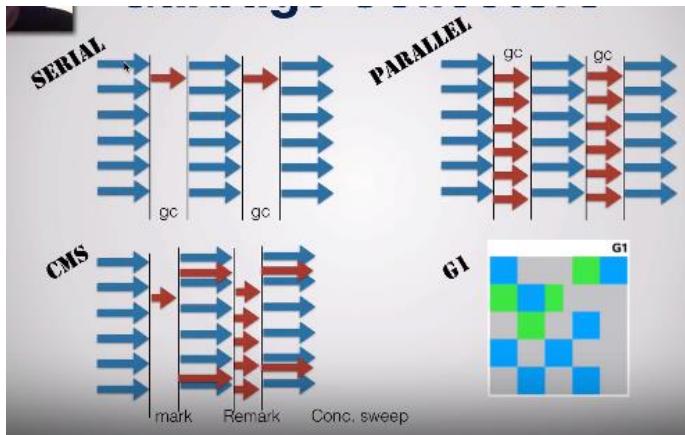
**Use a parallel collector when**

- There is less memory
- There is lesser number of CPUs
- Application demands high throughput and can withstand pauses.

**4. G1 Garbage Collector**

- **G1 garbage collector is used for large heap memory areas**.
- It separates the heap memory into regions and does collection within them in parallel.
- G1 also does compacts the free heap space on the go just after reclaiming the memory.
- G1 collector prioritizes the region based on most garbage first.

Turn on **the -XX:+UseG1GC JVM** argument to use the G1 garbage collector.



<https://www.youtube.com/watch?v=UnaNQgzw4zY>

#### **A serial collector**

Basic garbage collector that runs in single thread, can be used for basic applications.

#### **A concurrent collector**

A thread that performs GC along with application execution as the application runs, does not wait for the old generation to be full - Stops the world only during mark/re-mark.

#### **A parallel collector**

Uses multiple CPUs to perform GC. Multiple threads doing mark/sweep etc. Does not kick in until heap is full/near-full. "stops-the-world" when it runs.

### **Java 8 Improvement**

Turn on the **-XX:+UseStringDeduplication** JVM argument while using G1 garbage collector. **This optimizes the heap memory by removing duplicate String values to a single char[] array.** This option is introduced in [Java 8 u 20](#).

Given all the above four types of Java garbage collectors, which one to use depends on the application scenario, hardware available and the throughput requirements.

#### **Garbage Collection JVM Options**

##### **Type of Garbage Collector to run**

Option	Description
<b>-XX:+UseSerialGC</b>	<b>Serial Garbage Collector</b>
<b>-XX:+UseParallelGC</b>	<b>Parallel Garbage Collector</b>
<b>-XX:+UseConcMarkSweepGC</b>	<b>CMS Garbage Collector</b>
<b>-XX:ParallelCMSThreads=</b>	<b>CMS Collector – number of threads to use</b>
<b>-XX:+UseG1GC</b>	<b>G1 Garbage Collector</b>

#### GC Optimization Options

Option	Description
-Xms	Initial heap memory size
-Xmx	Maximum heap memory size
-Xmn	Size of Young Generation
-XX:PermSize	Initial Permanent Generation size
-XX:MaxPermSize	Maximum Permanent Generation size

**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.**

#### How String Literals Garbage Collected?

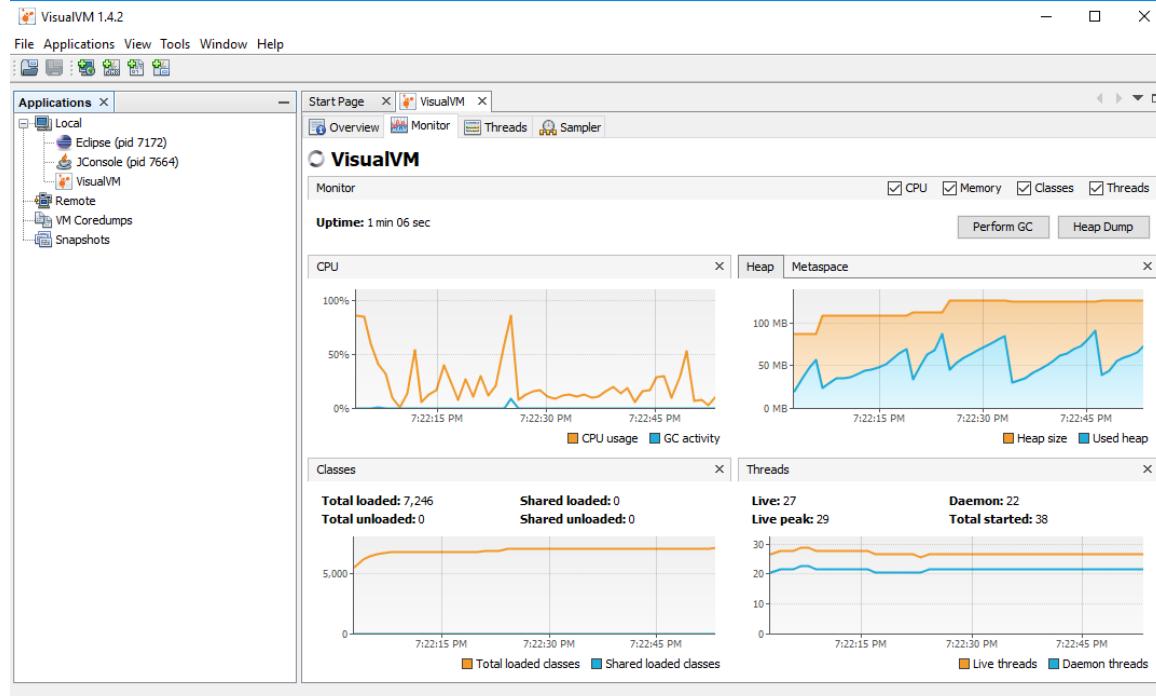
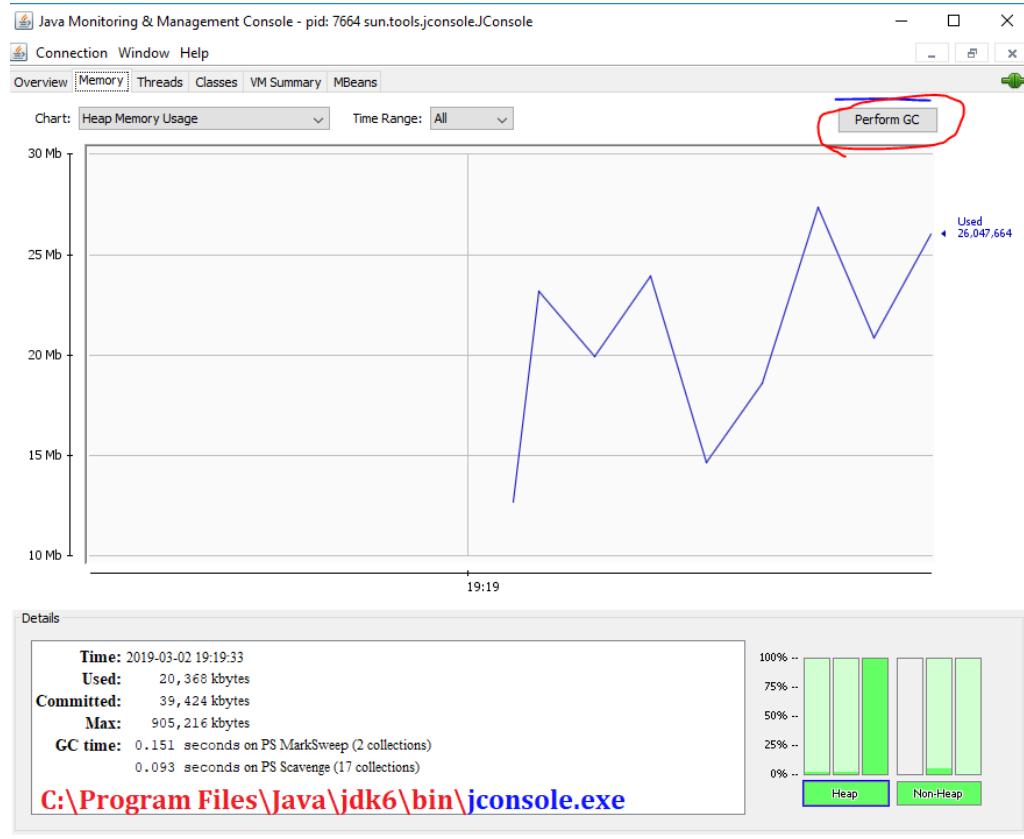
Strings created without using the new keyword are **NEVER garbage collected**. Even if there are no references to them. All such strings go into the String pool and just sit there till the whole program ends (ie. the JVM). **The String Const. pool cleaned up when the class is unloaded by the JVM.**

#### ## How to 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** (<https://visualvm.github.io/>) 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?

- Minor collection prints “**GC**” if garbage collection logging is enable using **-verbose:gc or -XX:PrintGCDetails**
- Major collection prints “**Full GC**”.

### How to Generate GC Log File?

In order to understand the GC log, you first need to generate one. Passing the following system properties to your JVM would generate GC logs

```
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:D://gc.log
```

Or add visual-gc plugin to visualVM

### What Security model used by Java?

**Sandbox.** The sandbox security model makes it easier to work with software that comes from sources you don't fully trust.

### What is "Phantom" memory

**A memory that doesn't exist in reality.**

A phantom reference lets you do final touch up closing on an object, even after it has been declared dead — no longer referenced by any live object.

During garbage collection, these reference objects are handled specially; that is, the referent field is not traced during the marking phase. When marking is complete, the references are processed in sequence for sweeping phase.

1. Weak
2. Soft
3. Phantom
4. Final

### How many JVMs can run on a single machine?

**Multiple, yes** You can run as many JVMs as you can fit on your disk and in memory :)

Whenever you start a Java application, you're first starting the JVM and then telling it which application to run. The answer to "which JVM" is simply: The JVM that you loaded the application with! for example, for execution of applets a separate JVM may exist and another JVM can be started by the User for execution of Java Byte Code, on a single machine.

### Difference between Object Oriented and Object Based language

#### Object Oriented Languages

- Object Oriented Languages supports all the features of OOPS Abstraction, Encapsulation, Polymorphism, Inheritance.
- C#, Java, VB. Net are the examples of object-oriented languages.

#### Object Based Languages

- Object based languages does not support **inheritance or, polymorphism or, both.**
- Object based languages does not support **built-in objects.**
- Javascript, VB are the examples of object based languages.

If I don't provide any arguments on commandline, then String array of main() is Empty or NULL?

- Complete array args is Empty
- Array element args[0] is NULL

```
public class ThreadDemo {  
    public static void main(String args[]) {  
        System.out.println("arry[] : " + args.length);  
        System.out.println("arry[0] : " + args[0]);  
    }  
}  
arry[] : 0  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0  
at com.root.ThreadDemo.main(ThreadDemo.java:6)
```

Is main() method compulsory in Java?

The answer to this question depends on version of java you are using. **Prior to JDK 5, main method was not mandatory in a java program.**

- You could write your full code under static block and it ran normally.
- The static block is first executed as soon as the class is loaded before the main();

String is Immtable. So, String class is Final?

Yes, String Class in Final in Java. Check String class code

```
public final class String implements Serializable, Comparable, CharSequence {  
}
```

Why String is Immutable or Final in Java?

In object-oriented programming, the **immutable string or objects** that cannot be modified once it is created. But we can only change the reference to the object. **String is immutable** in Java because of the security, synchronization and concurrency, caching, and class loading.

The reason of making string **final** is – no one can extend final class, and we wont change value of it.

- The String objects are cached in the **String pool**. The cached String literals are accessed by multiple clients. So, there is always a risk, where action performs by one client affects all other clients. For example, if one client performs an action and changes the string value from **Pressure** to **PRESSURE**, all remaining clients will also read that value.
- The String pool cannot be possible if String is not immutable in Java. A lot of heap space is saved by JRE. The same string variable can be referred to by more than one string variable in the pool. String interning can also not be possible if the String would not be immutable.

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

## 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");
```

```
byte b =100; // -127 to 128
System.out.println("Byte : "+b);

byte arr[] = {10, 30, 50, 70, 100, 120};
String s = new String(arr);
System.out.println("String : "+s);
```

```
Byte : 100
String :
_2Fdx
```

Data Type	Size	Range of values that can be stored	Default value
byte	1 byte	-128 to 127	0
short	2 bytes	-32768 to 32767	0
int	4 bytes	-2,147,483,648 to 2,147,483,647	0
long	8 bytes	-9,223,372,036,854,775,808 to 9223372036854750000	0
float	4 bytes	3.4e-038 to 3.4e+038	0.0f
double	8 bytes	1.7e-308 to 1.7e+038	0.0d
boolean	1 bit	true or false	false
char	2 bytes		\u0000

### 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.

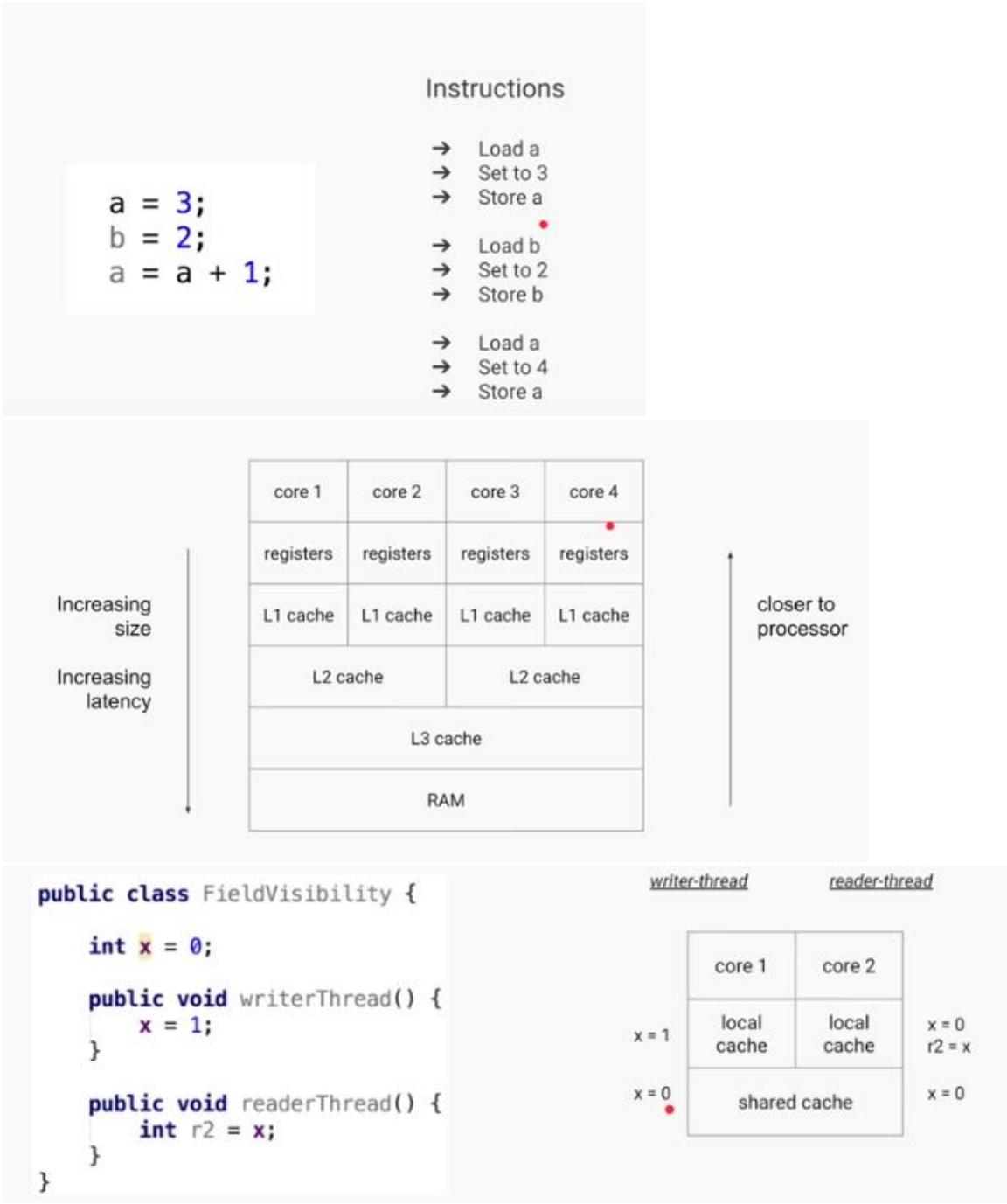
```
byte b1 = 10;           // 1 byte
long l1 = b1;          // one byte to 8 bytes, assigned implicitly
```

### Is ++ operator is thread-safe in Java?

No, it's not a thread safe operator because its involve multiple instructions like reading a value, incrementing it and storing it back into memory which can be overlapped between multiple threads.

### Volatile Vs Atomic variables?

#### Volatile Example



- If **writerThread()** is executed by one thread & **readerThread()** is executed by another thread
- the 'x' value is different for two threads because they are reading value from their LocalCache.
- Here the changes of X value is not **visible** globally (Field Visibility), because they are changing in **LocalCache**.

To avoid this, we need to use '**volatile**' keyword for fields.

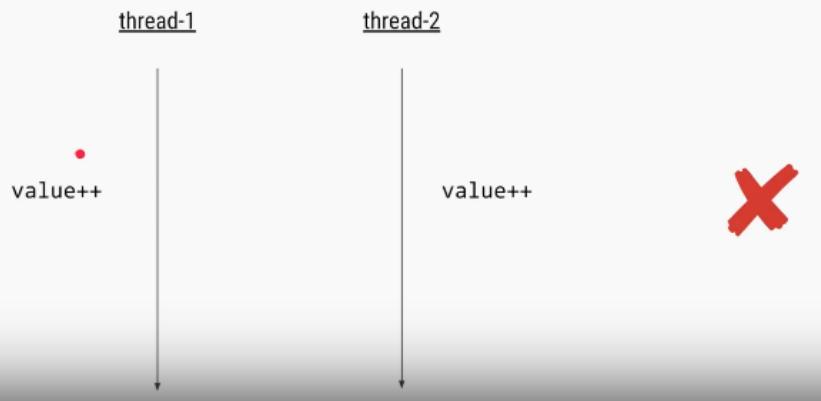
- The Java **volatile** keyword is used to mark a Java variable as "**being stored in main memory**".
- that means, every **read** of a volatile variable will **be read from the main memory(Shared Memory)**, and not from the CPU cache
- **every write** to a volatile variable will be **written to main memory**, and not just to the CPU cache.

## Amtomic Problem

**Atomic** - forming a single irreducible unit or component in a larger system.

Increment (++) is a **Compound Operation(multiple)**. **AtomicVariables** makes compound operations as **Atomic(Single)**

```
volatile int value = 1;
```



```
volatile int value = 1;
```

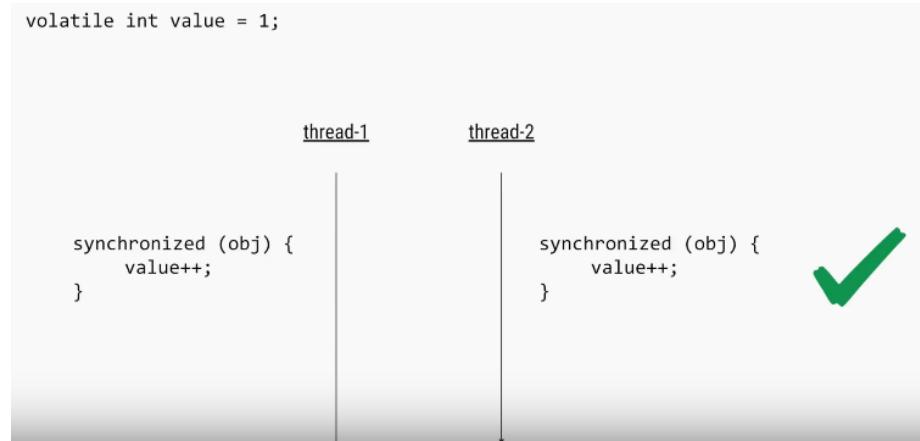
*Even with volatile*

#	Thread-1	Thread-2
1	Read value (=1)	
2		Read value (=1)
3	Add 1 and write (=2)	
4		Add 1 and write (=2)



We can solve this problem using Synchronization

```
volatile int value = 1;
```



## Another Way using Atomic Variables

```
AtomicInteger value = new AtomicInteger(1);
```



Type	Use Case
<code>volatile</code>	Flags
<code>AtomicInteger</code> <code>AtomicLong</code>	Counters
<code>AtomicReference</code>	Caches (building new cache in background and replacing atomically) Used by some internal classes Non-blocking algorithms

## Atomic Variables

The `java.util.concurrent.atomic` package defines classes that support atomic operations on single variables. All classes have **get and set methods** that work like reads and writes on volatile variables.

We have following Atomic classes

- `AtomicInteger`
- `AtomicLong`
- `AtomicBoolean`
- `AtomicReference`
- `AtomicIntegerArray`
- `AtomicLongArray`
- `AtomicReferenceArray`

## Common methods

- **incrementAndGet()**: Atomically increments by one the current value.

- **decrementAndGet()**: Atomically decrements by one the current value.
- **addAndGet(int delta)**: Atomically adds the given value to the current value.
- **compareAndSet(int expect, int update)**: Atomically sets the value to the given updated value if the current value == the expected value.
- **getAndAdd(int delta)**: Atomically adds the given value to the current value.
- **set(int newValue)**: Sets to the given value.

### 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.

- $f1 = (0.1+0.1+0.1\dots 11 \text{ times}) = 1.0999999999999999$
- $f2 = 0.1*11 = 1.1$

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 its. 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 an object and it stores meta data overhead about the object and int is primitive type so it takes less space.

### Autoboxing and Unboxing?

If a **method(remember only method – not direct)** requires Integer Object value, we can directly pass primitive value without issue. Autoboxing will take care about these.

We can also do direct initializations (1.8 V)

```
Integer i = 10; // it will create Integer value of 10 using Autoboxing
int j = i; // it will convert Integer to int using Autoboxing
```

Previously it shows

```
Integer i = 10; // it will create Integer value of 10 using Autoboxing
int j = i; // But we cant assign int to Integer Type mismatch: cannot convert from Integer to int
```

### 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();
```

### How does Autoboxing of Integer work in Java? (answer)

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

### 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 – including `public`.

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

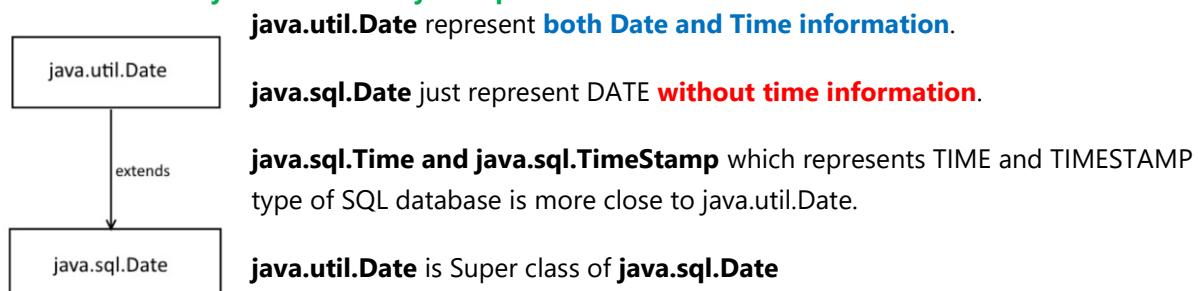
### What is blank final variable?

A **blank final** variable in Java is a `final` variable that is not initialized during declaration. Below is a simple example of blank final.

```
// A simple blank final example  
final int i;
```

- Value must be assigned in `constructor/ static /instance block` before using it.
- If we have more than one constructor or overloaded constructor in class, then blank final variable must be initialized in all of them.

### Difference between java.util.Date & java.sql.Date?



### Why Java does not support Operator Overloading?

The meaning of an operator is always same for variable of basic types like: int, float, double etc. For example: To add two integers, + operator is used.

However, for user-defined types (like: objects), you can redefine the way operator works. For example: If there are two objects of a class that contains string as its data members. You can redefine the meaning of + operator and use it to concatenate those strings.

To overload an operator, a special operator function is defined inside the class as

```
class className
{
    ...
public
    returnType operator symbol (arguments)
    {
        ...
    }
    ...
};

#include <iostream>
using namespace std;

class Test
{
private:
    int count;

public:
    Test(): count(5){}

    void operator ++()
    {
        count = count+1;
    }

    void Display() { cout<<"Count: "<<count; }

};

int main()
{
    Test t;
    // this calls "function void operator ++()" function
    ++t;
    t.Display();
    return 0;
}
```

- This function is called when ++ operator operates on the object of *Test* class (object *t* in this case).
- In the program, void operator ++ () operator function is defined (inside *Test* class).
- This function increments the value of *count* by 1 for *t* object.

## Why Java Doesn't Support it?

### 1.Java is Simple, No Confusions Please!!

Java does not support operator overloading: Java is relatively a very simple language to use compared C/C++ with the non-support of complex and confusing features like **pointers**, **multiple inheritance** and **operator overloading**. These features are rarely used in practice and at the same time poorly understood by the language beginners.

### 2.JVM Performance: How many things Should i do?

From JVM perspective supporting operator overloading is more difficult and if the **same thing can be achieved by using method overloading in more intuitive and clean way it does make sense to not support operator overloading in java**. a complex JVM will result in slower JVM

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

- 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 ArrayStoreException at runtime

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

Object[] names = new Integer[3];
names[0] = new String("a"); // ArrayStoreException at runtime
Exception in thread "main" java.lang.ArrayStoreException: java.lang.String
```

### What is difference between ArrayIndexOutOfBoundsException 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.
- **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.

### 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

```
public class Test {
    public static void main(String args[]) {
        String a[] = { "a", "b", "c" };
        System.out.println(a.toString());

        // 1. Using Arrays.toString(a)
        System.out.println(Arrays.toString(a));

        // 2. Using Arrays.asList(a)
        System.out.println(Arrays.asList(a).toString());
    }
}
[Ljava.lang.String;@15db9742
[a, b, c]
[a, b, c]
```

### Where does array stored in memory? [answer]

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

## What is an array?

Dimensions	Example	Terminology
1		Vector
2		Matrix
3		3D Array (3rd-order Tensor)
N		ND Array

## Reverse Array using Iterative and Recursive approaches

Steps to Solve this

1. **Initialize** array
2. Choose **start** index
3. Choose **end** index
4. Swap the elements using **temp** variable

```
public class ReverseArry {

    /* Recursive approach: In recursive approach the function calls itself until the
     * condition is met. And it is slower than iteration, which means it uses more
     * memory than iteration. recursion is like a selection structure, and which
     * makes code smaller and clean. And a function partially defined by itself.
     * Here tracing the code will be more difficult in the case large programs
     */

    public static int[] recursiveArry(int a[], int start, int end) {
        if (start <= end) {
            int temp;
            temp = a[start];
            a[start] = a[end];
            a[end] = temp;
            recursiveArry(a, start + 1, end - 1); // calling it again
        }
        return a;
    }

    /* Iterative approach: Iterative approach is a repetition process until the
     * condition fails. here loops are used such as for, while etc. Here code may be
     * longer but it is faster than recursive. And it consumes less memory compared
     * to recursive approach. If the loop condition is always true in such cases it
     * will be an infinite loop*/
    public static int[] iterativeArry(int a[], int start, int end) {
        while (start < end) {
            int temp;
            temp = a[start];
            a[start] = a[end];
            a[end] = temp;
            start++;
            end--;
        }
        return a;
    }

    static void printArray(int arr[], int size) {
        int i;
        for (i = 0; i < size; i++)
            System.out.print(arr[i] + " ");
        System.out.println("");
    }
}
```

```

public static void main(String[] args) {
    // 1. Initialize array
    int arr[] = { 1, 2, 3, 4, 5, 6 };
    System.out.println("Input array is ");
    printArray(arr, 6);

    // 2. Choose Starting & ending point
    int b[] = recursiveArry(arr, 0, 5);
    System.out.println("Recursive -Reversed array is ");
    printArray(b, 6);

    int c[] = iteravtiveArray(arr, 0, 5);
    System.out.println("IteravtiveArray -Reversed array is ");
    printArray(c, 6);
}
}

Input array is 1 2 3 4 5 6 Recursive-
Reversed array is 6 5 4 3 2
1

```

### How do you reverse an array in Java?

org.apache.commons.lang.ArrayUtils class to reverse Array in Java.`ArrayUtils.reverse(arry)`

```

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. `equals(Object)`:** Compares two Objects for equality. String class(not StringBuilder, StringBuffer) & All Wrapper classes equals() method is overridden for Content Comparison.

**3.hashCode()**: returns the integer representation of memory location. which is used by JVM while saving/adding Objects into Hashsets, Hashtables or Hashmap for calculating bucket location.

**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 `obj.getClass()`, 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.

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

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

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

### Relationship between `equals()` & `hashcode()`

`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**

```
public static void main(String[] args) {
    StringBuffer sb1 = new StringBuffer("Satya");
    StringBuffer sb2 = new StringBuffer("Satya");

    if (sb1.equals(sb2)) {
        System.out.println("EQUAL");
    } else {
        System.out.println("NOT EQUAL");
    }
}
```

NOT EQUAL

**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
        System.out.println(e1.hashCode()); //366712642
        System.out.println(e2.hashCode()); //1829164700 - here different
    }
}
```

So are we done? 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. **But here it is not!!**

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.Employee@15db9742, basic.Employee@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.

```
Demo.java:1: error: modifier private not allowed here
private class Demo {
    ^
```

### 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, you can make an array (both primitive and reference type array e.g. an [int array](#) and [String array](#)) volatile in Java, but only changes to reference pointing to an array will be visible to all threads, not the whole array.

This is all about **Individual Array Elements VS complete Array**

- Complete Array (`primes = new int[20];`) – Changes will visible, **volatile work properly**
- **Individual Array Elements** (`a[0]=10, a[1]=20`) – Changes May/May not visible, **volatile won't work properly**

#### **Indepth Complete Explanation**

What this means is that suppose you have a reference variable called `primes` as shown below:

```
protected volatile int[] primes = new int[10];
```

then if you assign a new array to `primes` variable, like below

```
primes = new int[20];
```

change will **be visible to all threads**, but changes to individual indices(`a[0], a[1]...a[n]`) will not be covered under volatile guarantee i.e.

It will follow the "**happens-before**" rule(*Happens-before relationship is a guarantee that action performed by one thread is visible to another action in different thread.*) and cause memory refresh, but following code will not do so

if multiple threads are changing individual array elements e.g. storing updates, there won't be any happens-before guarantee provided by the volatile modifier for such modification

```
primes[0] = 10;  
primes[1] = 20;  
primes[2] = 30;
```

So, if your use-case is to provide memory visibility guarantee for **individual array elements** than **volatile is not the right choice**. You must rely on other synchronization and a thread-safety mechanism to cover this case e.g. synchronized keyword, atomic variables, or ReentrantLock.

#### **Is it possible to make ArrayList, HashSet volatile in Java?**

On a similar note, sometimes instead of an array, Interviewer put the collection i.e. they will ask *can you make a collection variable volatile in Java or not* e.g. an ArrayList or HashMap. The answer is same, of course, you can make a reference variable pointing to a Collection volatile in Java, but the happens-before guarantee will only be provided if the value of that reference variable is changed e.g. you assign a new collection to it.

Any modification done on actual collection object e.g. adding or removing elements from ArrayList will not invoke happens-before guarantee or memory barrier refresh.

#### **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 `hashcode`

#### **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("OBJECT Created FIRST TIME");  
    }  
    public static Student getObject() {  
        if (st == null) {  
            st = new Student();  
        } else {  
            System.out.println("OBJECT ALREADY 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()); //7855445  
        System.out.println(s2.hashCode()); //7855445  
    }  
}
```

**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 the **process of saving the state of the object permanently** in the form of a file/byte stream. It's achieved by implementing Serializable interface which is a marker interface (an interface without any method). uses default implementation to handle the object serialization process.

```
class Student implements Serializable {
    // Exception in thread "main" java.io.NotSerializableException: io.Student if it won't implement
    Serializable
    private int sno;
    private String name;
    private String addr;
    //Setters & getters setName,SetAddr methods...
}

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);
}
}

//data saved in student.txt
-í sr
io.Student@{|^| _____ I _____snoL _____
addr Ljava/lang/String;L _____
nameq ~ xp et
VIJAYAWADAT
```

## 2.Deserialization

De-serialization is a **process of retrieve the data from the file in the form of object**.

```
public class Deserialization {
    public static void main(String[] args) throws Exception{
        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());
    }
}

101
Satya Kaveti
VIJAYAWADA
```

If we use above process to implement serialization, all the data members will participate in Serialization process. If you want to use selected data members for serialization, use **Transient** keyword.

**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)

- static binding is related to overloaded method
- 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 - Spring
- Factory pattern
- Adapter Design pattern
- Singleton
- Decorator design pattern is used to modify the functionality of an object at runtime.

## 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();
```

## How can we invoke any external process in java?

```
Runtime.getRuntime().exec(...)
```

## Static imports rules ?

The static import feature of Java 5 facilitates the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

```
import static java.lang.System.*;
import static java.lang.System.out;

class StaticImportExample{
    public static void main(String args[]){
        out.println("Hello");//Now no need of System.out
        out.println("Java");
    }
}
```

## Ambiguity in static import

```
// both have MAX_VALUE as static
import static java.lang.Integer.*;
import static java.lang.Byte.*;
class Geeks {
    public static void main(String[] args)
    {
        out.println(MAX_VALUE);
    }
}
Error:Reference to MAX_VALUE is ambiguous
```

If we import fully Qualified name `java.lang.Integer.MAX_VALUE`, `Integer` will get more priority than `Byte`.

### 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. But we can re-declare in sub class, it will be treated as a new method, bcoz parent class private method is not visible to subclass.

```
class A{  
    private void show() {  
        System.out.println("Parent");  
    }  
}  
public class Demo extends A{  
    private void show() {  
        System.out.println("Child");  
    }  
    public static void main(String[] args) {  
        A a = new Demo();  
        a.show();  
    }  
}  
  
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
  The method show() from type A is not visible  
  at Demo.main(Demo.java:12)
```

Though, you can override them inside the inner class as they are accessible there.

### Method overriding Rules

The compiler performs the following checks when you override a nonprivate method:

1. The method in the child class must have the **same signature** as the method in the parent class.
2. The method in the child class must be at least as accessible or more accessible than the method in the parent class. (if overridden method in parent class is **protected**, then overriding method in child class can not be **private**. It must be either **protected** (same access) or **public** (wider access))
3. The method in the child class may not throw a **New Exception** or Larger Exception than their parent class method. (parent class throws **FileNotFoundException**, the overriding method in child class can throw **FileNotFoundException**; but it is not allowed to throw **IOException** or **Exception**)
4. If the method returns a value, it must be the same or a subclass of the method in parent class, known as **covariant return types**
5. The method defined in the child class must be marked as **static** if it is marked as static in the parent class (method hiding). Likewise, the method must not be marked as static in the child class if it is not marked as static in the parent class (method overriding).

### 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 responsibility	
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 – Model class

**"One class should have one and only one responsibility"**

In other words, we should write, change and maintain a class for only one purpose. **Model class is best example to this. Model class should strictly represent only one actor/ entity.** This will give us the flexibility to make changes in future without worrying the impacts of changes for another entity.

## **2. Open Closed Principle – Spring Framework Classes**

**"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 cannot 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 Spring 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.**

`CurdRepository, SpringBoot Configuration application.properties examples of it.`

## **3. Liskov's Substitution Principle**

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

## **4. Interface Segregation/Separation Principle**

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

Take an example. Developer Alex created an interface `ReportUtils` and added two methods `generateExcel()` and `generatedPdf()`. Now client 'A' wants to use this interface, but he intends 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.

## Gang of Four Design Patterns

### Creational

- ✓ **Factory Method**
- **Abstract Factory**
- **Builder**
- **Prototype**
- ✓ **Singleton**

### Structural

- ✓ **Adapter**
- **Bridge**
- **Composite**
- **Decorator**
- **Façade**
- **Flyweight**
- **Proxy**

### Behavioral

- **Interpreter**
- **Template Method**
- **Chain of Responsibility**
- **Command**
- **Iterator**
- **Mediator**
- **Memento**
- **Observer**
- **State**
- ✓ **Strategy**
- **Visitor**

### 1. Creational Design Patterns (Object Creation)

Creational patterns often used in place of direct instantiation with constructors. They make the creation process more adaptable and dynamic. In particular, they can provide a great deal of flexibility about which objects are created, **how those objects are created, and how they are initialized.**

DESIGN PATTERN NAME	PURPOSE
<a href="#"><b>Builder</b></a>	Builder design pattern is an alternative way to construct complex objects and should be used only when we want to build different types of immutable objects using same object building process.
<a href="#"><b>Prototype</b></a>	Prototype design pattern is used in scenarios where application needs to create a large number of instances of a class, which have almost same state or differ very little.
<a href="#"><b>Factory</b></a>	Factory design pattern is most suitable when complex object creation steps are involved. To ensure that these steps are centralized and not exposed to composing classes.
<a href="#"><b>Abstract factory</b></a>	Abstract factory pattern is used whenever we need another level of abstraction over a group of factories created using factory pattern.
<a href="#"><b>Singleton</b></a>	Singleton enables an application to have one and only one instance of a class per JVM.

## 2. Structural Design Patterns

Structural design patterns show us how to glue different pieces of a system together in a flexible and extensible fashion. These patterns help us guarantee that **when one of the parts changes, the entire application structure does not need to change.**

DESIGN PATTERN NAME	PURPOSE
<a href="#">Adapter</a>	An adapter convert the interface of a class into another interface clients expect. It lets classes work together that couldn't otherwise because of incompatible interfaces.
<a href="#">Bridge</a>	Bridge design pattern is used to decouple a class into two parts – <i>abstraction</i> and its <i>implementation</i> – so that both can evolve in future without affecting each other. It increases the loose coupling between class abstraction and its implementation.
<a href="#">Composite</a>	Composite design pattern helps to compose the objects into tree structures to represent whole-part hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
<a href="#">Decorator</a>	Decorator design pattern is used to add additional features or behaviors to a particular instance of a class, while not modifying the other instances of same class.
<a href="#">Facade</a>	Facade design pattern provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
<a href="#">Flyweight</a>	Flyweight design pattern enables use sharing of objects to support large numbers of fine-grained objects efficiently. A flyweight is a shared object that can be used in multiple contexts simultaneously. The flyweight acts as an independent object in each context.
<a href="#">Proxy</a>	In proxy design pattern, a proxy object provide a surrogate or placeholder for another object to control access to it. Proxy is heavily used to implement lazy loading related usecases where we do not want to create full object until it is actually needed.

## 3. Behavioral Design Patterns

Behavioral patterns abstract an action **we want to take on the object or class that takes the action**. By changing the object or class, we can change the algorithm used, the objects affected, or the behavior, while still retaining the same basic interface for client classes.

DESIGN PATTERN NAME	PURPOSE
<a href="#">Chain of responsibility</a>	Chain of responsibility design pattern gives more than one object an opportunity to handle a request by linking receiving objects together in form of a chain.

<b><u>Command</u></b>	Command design pattern is useful to abstract the business logic into discrete actions which we call commands. These command objects help in loose coupling between two classes where one class (invoker) shall call a method on other class (receiver) to perform a business operation.
<b><u>Interpreter</u></b>	Interpreter pattern specifies how to evaluate sentences in a language, programmatically. It helps in building a grammar for a simple language, so that sentences in the language can be interpreted.
<b><u>Iterator</u></b>	Iterator pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
<b><u>Mediator</u></b>	Mediator pattern defines an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets us vary their interaction independently.
<b><u>Memento</u></b>	Memento pattern is used to restore state of an object to a previous state. It is also known as snapshot pattern.
<b><u>Observer</u></b>	Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. It is also referred to as the publish-subscribe pattern.
<b><u>State</u></b>	In state pattern allows an object to alter its behavior when its internal state changes. The object will appear to change its class. There shall be a separate concrete class per possible state of an object.
<b><u>Strategy</u></b>	Strategy pattern is used where we choose a specific implementation of algorithm or task in run time – out of multiple other implementations for same task.
<b><u>Template method</u></b>	Template method pattern defines the sequential steps to execute a multi-step algorithm and optionally can provide a default implementation as well (based on requirements).
<b><u>Visitor</u></b>	Visitor pattern is used when we want a hierarchy of objects to modify their behavior but without modifying their source code.

### 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?

**Class -Final, Private Datamembers -Final, Public params Constructor, Only getters**

Immutable classes are Java classes whose objects cannot 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 & its 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 {
    private final String name;
    private int regNo;

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

class Test {
    public static void main(String args[]) {
        Student s = new Student("ABC", 101);

        System.out.println(s.getName()); // ABC
        System.out.println(s.getRegNo()); // 101

        System.out.println(s.name);
        System.out.println(s.regNo);
    }
}
```

Exception in thread "main" java.lang.Error: Unresolved compilation problems:  
The field Student.name is not visible  
// Uncommenting below line causes error  
// s.regNo = 102;  
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
The final field Student.regNo cannot be assigned

## 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.

### Private Constructor, Private Object, Public Static Factory Method

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("OBJECT Created FIRST TIME");  
    }  
  
    public static Student getObject() {  
        if (st == null) {  
            st = new Student();  
        } else {  
            System.out.println("OBJECT ALREADY 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 out the application life cycle.

This will bring us to **double checked locking pattern**, where only critical section of code is locked.

Programmers 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;  
}
```

- **For Cloning** – we need to Override **clone()** method & Should throw **CloneNotSupportedException**
- **For Reflection** - we need to throw **RuntimeException(unchecked)** in **private Constructor**

**How can we create Multiple Objects, even with class is Singleton – How to avoid do so?**

We have Two ways

- Cloning
- Reflection API

**Cloning** is a concept to create duplicate objects. **Using clone, we can create copy of object.**

Suppose we create clone of a singleton object, then it will create a copy that Singleton Object. that means we have two instances of a singleton class; **hence the class is no more singleton.**

```
public static void main(String args[]) throws CloneNotSupportedException {
    Student s1 = Student.getObject();
    Student s2 = Student.getObject();

    Student s3 = (Student) s1.clone();
    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s3);
}
Student@15db9742
Student@15db9742
Student@6d06d69c // Creates new Object, our singleton failed.
```

To overcome this, we should override **clone()** method of Singleton class, it should throw Exception, anyone tries to do clone

```
class Student implements Cloneable{
    .....
    @Override
    protected Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException();
    }
    public static void main(String args[]) throws CloneNotSupportedException {
        Student s1 = Student.getObject();
        Student s2 = Student.getObject();

        Student s3 = (Student) s1.clone();
        System.out.println(s1);
        System.out.println(s1);
        System.out.println(s3);
    }
}
Exception in thread "main" java.lang.CloneNotSupportedException //We are GOOD now
at Student.clone(Student.java:25)
at Student.main(Student.java:33)
```

**Reflection:** You can make the new instance of the Singleton class by changing the **constructor visibility as public** in run-time and create new instance using that constructor.

```
public class SingletonFailed {
public static void main(String args[]) throws Exception {

    Student s1 = Student.getObject();
    Student s2 = null;

//1. Making Constructor visible
    Constructor<Student>[] constructors = (Student.class).getDeclaredConstructors();
    for (Constructor constructor : constructors)
    {   // Below code will destroy the singleton pattern
        constructor.setAccessible(true);
        s2 = (Student) constructor.newInstance();
    }
    System.out.println(s1);
    System.out.println(s2);

//Using Class of newInstance()
}
```

```

        Class c = Student.class;
        Student s1 = Student.getObject();
        Student s2 = (Student) c.newInstance();

        System.out.println(s1); // Student@15db9742
        System.out.println(s2); // Student@6d06d69c -Failed again ☹
    }
Student@15db9742
Student@6d06d69c //Failed again ☹

```

To prevent Singleton failure while due to reflection you have to **throw a run-time exception in constructor**, if the constructor is already initialized.

```

class Student implements Cloneable{
    private static Student st;

    private Student() {
        if(st!=null)
            throw new RuntimeException("Go Fucker.....");
    }
Exception in thread "main" java.lang.RuntimeException: Go Fucker.....  

    at Student.<init>(Student.java:15)  

    at sun.reflect.NativeConstructorAccessor

```

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

The Java library provides **StringBuffer** 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);
    }
}
```

## Check 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) {
            s.o.p("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 d = Pattern.compile("\\d{6}");
        // Pattern digitPattern = Pattern.compile("\\d\\d\\d\\d\\d\\d\\d");

        for (String number : numbers) {
            s.o.p("does " + number + " is 6 digit number : " + d.matcher(number).matches());
        }
    }
}
```

## Java StringTokenizer With Multiple Delimiters?

The **java.util.StringTokenizer** class allows an application to break a string into tokens.

### StringTokenizer(String str)

This constructor creates a string tokenizer for the specified string.

### StringTokenizer(String str, String delim)

This constructor constructs a string tokenizer for the specified string.

The 6 useful methods of StringTokenizer class are as follows:

public method	Description
---------------	-------------

<code>boolean hasMoreTokens()</code>	checks if there is more tokens available.
<code>String nextToken()</code>	returns the next token from the StringTokenizer object.
<code>String nextToken(String delim)</code>	returns the next token based on the delimiter.
<code>boolean hasMoreElements()</code>	same as hasMoreTokens() method.
<code>Object nextElement()</code>	same as nextToken() but its return type is Object.
<code>int countTokens()</code>	returns the total number of tokens.

### Normal Example

```
public class Simple{
    public static void main(String args[]){
        StringTokenizer st = new StringTokenizer("my name is khan"," ");
        while (st.hasMoreTokens()) {
            System.out.println(st.nextToken());
        }
    }
=====
Output:my
      name
      is
      khan
```

### Java StringTokenizer with Multiple De-limiters

for this we have to specify the Delimiters, separated by //. for Example

```
StringTokenizer tokenizer = new StringTokenizer(s, "// //!/,//?//.//_//'//@");
```

```
public class Singleton {
    public static void main(String[] args) {
        String s = "He is a very very good boy, isn't he?";
        StringTokenizer tokenizer = new StringTokenizer(s, "// //!/,//?//.//_//'//@");
        System.out.println(tokenizer.countTokens());
        while (tokenizer.hasMoreTokens()) {
            System.out.println(tokenizer.nextToken());
        }
    }
=====
10
He
is
a
very
very
good
boy
isn
t
he
```

### 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);
    }
}

```

## Using StringTokenizer

```

public class Test {
    public static void main(String args[]) {
        System.out.println("Using Constructor 1 -By Space ");
        StringTokenizer st1 = new StringTokenizer("Hello Geeks How are you", " ");
        System.out.println("Get TokensCount in case of revers: "+st1.countTokens());
        while (st1.hasMoreTokens())
            System.out.println(st1.nextToken());

        System.out.println("Using Constructor 2 - By Given Symol");
        StringTokenizer st2 = new StringTokenizer("JAVA : Code : String", " :");
        while (st2.hasMoreTokens())
            System.out.println(st2.nextToken());

        System.out.println("Using Constructor 3 - Using Flag");
        /*
         * If the flag is false, delimiter characters serve to separate tokens. For
         * example, if string is "hello geeks" and delimiter is " ", then tokens are
         * "hello" and "geeks".
         *
         * If the flag is true, delimiter characters are considered to be tokens. For
         * example, if string is "hello geeks" and delimiter is "
         * ", then tokens are "hello", " " and "geeks".
         */
        StringTokenizer st3 = new StringTokenizer("JAVA : Code : String", " :", true);
        while (st3.hasMoreTokens())
            System.out.println(st3.nextToken());
    }
}

```

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

`s.intern()` – used to get the String literal which is created in StringPool while creating String Object. And also deal with the String duplication problem in Java.

By carefully using the `intern()` you can save a lot of heap memory consumed by duplicate String objects.

A String object is said to be duplicate if it contains the same content as another string but occupied different memory location e.g. `str1 != str2` but `str1.equals(str2)` is true.

For example, when you create a String literal like `"ABC"` then it's automatically stored in **String pool**, but when you create a new String object, e.g. `new String("ABC")`, even though it's the same String, a new object at a different memory location is created. **This is a duplicate String.**

`String s = new String("Satya");`

Here `"Satya"` literal is Created in String Pool & but not associated with any Variable. and you want to create a String literal with content `"Satya"`, you have two options

```

String intern = "Satya"      // New Literal
String intern = s.intern();  // Using Stringpool literal

public class Demo {
    public static void main(String[] args) {
        String s = new String("Satya");
        String intern = s.intern();
        System.out.println(intern);
    }
}

```

**Output**  
-----  
Satya

### ## 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 date = new SimpleDateFormat("MM/dd/yyyy").parse(string);

```

### Formatted output in Java

Sometimes we need to print the output in a given specified format. For doing that we have `printf()` method. `printf()` can take **multiple arguments**, but `System.out.print()` and `System.out.println()` take a single argument.

```

public class Demo {
    public static void main(String[] args) {
        int x = 100;
        System.out.printf("Printing simple integer: x = %d\n", x);

        System.out.printf("Formatted with precision: PI = %.2f\n", Math.PI);
        // this will print it upto 2 decimal places

        float n = 5.2f;
        System.out.printf("Formatted to specific width: n = %.4f\n", n);
        // automatically appends zero to the rightmost part of decimal

        n = 2324435.3f;
        System.out.printf("Formatted to right margin: n = %20.4f\n", n);
        // here number is formatted from right margin and occupies a width of 20 characters
    }
}

Printing simple integer: x = 100
Formatted with precision: PI = 3.14
Formatted to specific width: n = 5.2000
Formatted to right margin: n =          2324435.2500

```

We have The `java.lang.String.format(String format, Object... args)` method returns a formatted string using the specified format string and arguments.

```

public class Demo {
    public static void main(String[] args) {
        double pi = Math.PI;
    }
}

```

```

        // returns a formatted string using the specified format string, and arguments
        System.out.format("%f\n", pi);

        float f = 246.83278387f;
        String s = String.format("%.2f\n", f);
        System.out.println(s);
    }
}
3.141593
246.83

```

### 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

- format() method **returns a formatted String**
- 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.

```

String str = String.format("%04d", 9); // 0009
System.out.printf("original number %d, numeric string with padding : %s", 9, str);

original number 9, numeric string with padding : 0009

```

we can also use **DecimalFormat** class with passing format

```

DecimalFormat df = new DecimalFormat("0000");
String a = df.format(9); // 0009
String b = df.format(99); // 0099
String c = df.format(999); // 0999
System.out.println("\n"+a+"\n"+b+"\n"+c+"\n");

```

0009  
0099  
0999

## Enum

Enumerations serve the purpose of representing a group of named constants in a programming language. enums are **Compile time Constants, because they are public static final**

Enums are used when we know all possible values at **compile time**, such as choices on a menu, rounding modes, command line flags, etc. It is not necessary that the set of constants in an enum type stay **fixed** for all time

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 it 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.** Because 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. But we can write private/protected Constructor.**

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.

### What's difference between enums and final variables?

Technically one could indeed view `enums` as a `class` with a bunch of typed constants, and this is in fact how enum constants are implemented internally. Using an enum gives you useful methods such as `ordinal()`, `values()` or `valueOf()`.

We can group the constats in Single enum class for reusing. But final static need to be defined in a class, which may or moy not resuasble. Ref . <https://www.geeksforgeeks.org/enum-in-java/>

Example: Grouping multliple values in Single Property

```
public enum ProductDecommissionEnum {
    MI("ImageShare", "MyQueries", "MI Reports"),
    MTA("Analytics"),
    ECLINICALMETRICS("Reports Webfocus Report");
    // To get Values, Use : ProductDecommissionEnum.MI.values
}
```

```

private List<String> getAllActiveFeatureProductList(final Long trialId) throws ServiceException {
    try {
        Features features = featureAdminService.getFeatures(DECOMM_FEATURE_CONTEXT, trialId);
        List<FeatureVO> featuresList = features.getFeatures();
        for (FeatureVO featureVO : featuresList) {
            for (FeatureVO vo : featureVO.getChildFeatures()) {

                if (ProductDecommissionEnum.MI.getValues().contains(vo.getName()))
                    mytrialsFeaturesProductList.add(DECOMM_FEATURE_MI);

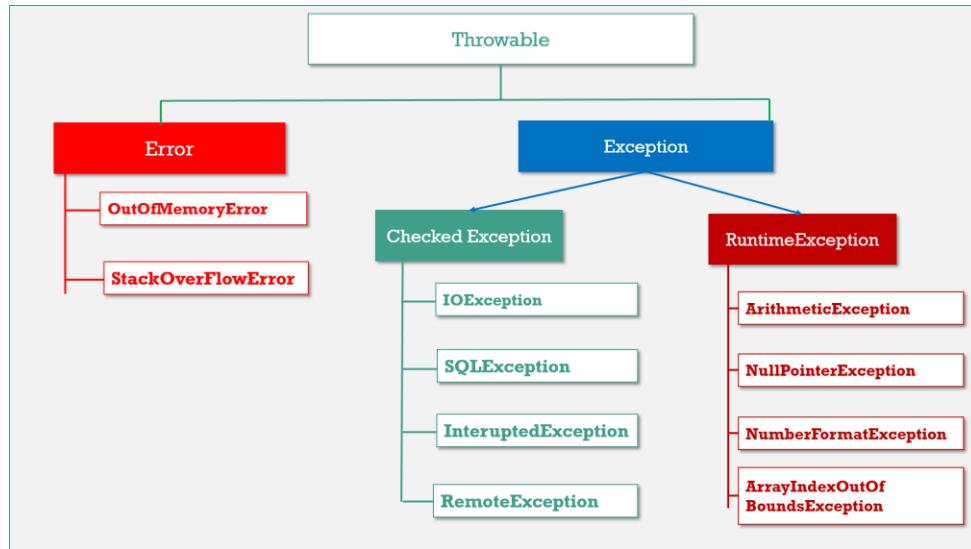
                if (ProductDecommissionEnum.MTA.getValues().contains(vo.getName()))
                    mytrialsFeaturesProductList.add(DECOMM_FEATURE_MTA);

                if (ProductDecommissionEnum.ECLINICALMETRICS.getValues().contains(vo.getName()))
                    mytrialsFeaturesProductList.add(DECOMM_FEATURE_METRICS);

            }
        }
    }
}

```

## Exception Handling



**Throwable** is the parent of entire java exception hierarchy. It has 2 child classes

- 1) **Exception.**
- 2) **Error.**

### 1.Exception

These are recoverable. Most of the cases exceptions are raised due to bad code.

- **Checked Exceptions: They Checked by Compiler** (yes, at the time of code only Eclipse shows error message – add try/catch), they will check that 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**
  - **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.
- RuntimeException** and its child classes, **Error** and it's child classes are considered as unchecked exceptions and all the remaining considered as checked.

## 2.Error

Errors are non-recoverable. Most of the cases errors are due to lack of system resources but not due to our programs.

**JVM + Memory+ OS level issues.** OutOfMemory, StatckOverFlow

### Partially Checked Vs Fully Checked

- **Fully Checked:** A checked exception is said to be **fully checked** iff all its child classes also checked. **Ex:** - **IOException**.
- **Partially Checked:** A checked exception is said to be partially checked if some of it's child classes are not checked. **Ex:** - **Exception, Throwable**.

### **When we get StackOverflow error? Can we handle that Exception?**

A stack overflow is usually called by nesting function calls too deeply (especially easy when using recursion, i.e. a function that calls itself) or allocating a large amount of memory on the stack where using the heap would be more appropriate.

```
public class ThreadDemo {  
    public static void main(String args[]) {  
        main(new String[1]);  
        System.out.println("Next line after main method");  
    }  
}  
  
Exception in thread "main" java.lang.StackOverflowError  
at com.root.ThreadDemo.main(ThreadDemo.java:5)  
at com.root.ThreadDemo.main(ThreadDemo.java:5)  
at com.root.ThreadDemo.main(ThreadDemo.java:5)
```

Like any other exception, we can handle **StackOverflowError** too. But we need use **Error** or **Throwable** while catching. The statement inside catch block is not execute in this case

```
public class ThreadDemo {  
    public static void main(String args[]) {  
        try {  
            main(new String[1]);  
        }catch (Throwable e) {  
            System.err.println("Exception Hadled, So Next stmt will execute");  
        }  
        System.out.println("Next line after main method");  
    }  
  
Next line after main method  
Next line after main method  
Next line after main method
```

### **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, calling **System.exit(0)** in try or catch block. In advanced case it will execute in following case.

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

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

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

```

        try {
            System.out.println("try");
            System.exit(0);
        }catch (Exception e) {
            System.out.println("catch");
        } finally {
            System.out.println("finally");
        }
    }
}

try      - (Security Exception not thrown)

```

- 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.

Remember, if a method returns any thing in try, we must place return in catch as well.

```

public class Test {

    public int number() {
        try {
            int c = 10 / 0;
            return 100;
        }
    }
}

```

```

        } catch (Exception e) {
            return 200;
        } finally {
            return 300;
        }
    }

    public static void main(String args[]) {
        System.out.println(new Test().number());
    }
}
0/p: 300

```

**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, then a robust program must be able to handle that situation gracefully. (one of the java feature is `robust (healthy and strong)`)

By declaring **IOException** as checked Exception, Java ensures that yours 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.

**if we got this Error - we Clear server cache & restart servers**

```
java -Xmx512m myprogram
-Xms512m -Xmx1152m -XX:MaxPermSize=256m -XX:MaxNewSize=256m
```

# I/O Streams

## 1. Byte Streams (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);
}
InputStream inputStream = new FileInputStream("c:\\a.txt");
int i;
while ((i = inputStream.read()) != -1) {
    System.out.println("I : " + i);
}
```

## 2. Character Streams (2 Bytes at a time) : Character 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 -Programming 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:** above we have only Char & Byte types. 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**

```

class Student implements Serializable {
    private int sno;
    private static String name;
}

public class Demo {
    public static void main(String[] args) throws Exception {
        Student student = new Student();
        student.setSno(101);
        student.setName("Satya Kaveti");

        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());
    }
}

101
Satya Kaveti      (Static serliaze ?)

```

Confused? Let's stop the program and remove writing part & re-run it!

```

public class Demo {
    public static void main(String[] args) throws Exception {
        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());
    }
}

101
null

```

This clearly indicates that static variable is not stored in the file, but the value of the static variable that is currently loaded into memory is being picked up, when you use `st.getName()`;

## 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.

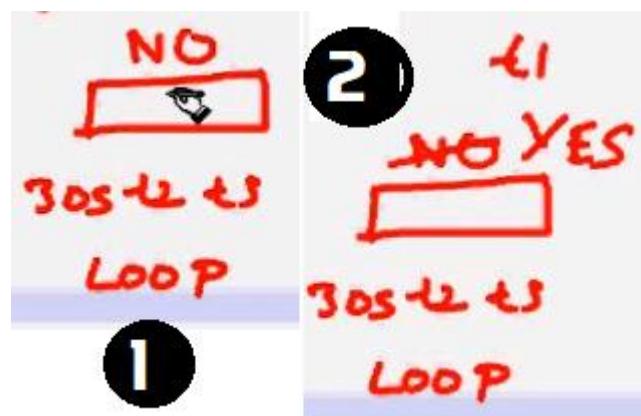
### What is busy spin?

We have T1, T2, T3 tasks which are executed by the threads t1,t2,t3. But T2, T3 Should execute after T1 completes its execution. for that they must communicate via signaling.



We can achieve this singling process in following ways

#### 1. Busy Wait



1.here we have FLAG variable which is initialized to NO. t2, t3 threads will check for FLAG = Yes or not for every 30 seconds via loop, until FLAG = yes.

This is called Busy Spin / Busy wait.

2.After some time, t1 changes status to YES, so that t2, t3 can proceed its execution.

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");  
}
```

```
class Producer implements Runnable {  
//Maintains Flag, this is BusySpinning condition, threads always checks this condition should be changed  
    boolean isProgress;  
    ArrayList<Integer> proList;  
    Producer() {  
        isProgress = true;  
        proList = new ArrayList<>();  
    }  
  
    @Override  
    public void run() {  
        for (int i = 1; i <= 8; i++) {  
            proList.add(i);  
            System.out.println("Producer is still Producing, Produced : " + i);  
            try {  
                Thread.sleep(500);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
        isProgress = false;  
    }  
}  
  
class Consumer implements Runnable {  
    Producer producer;  
  
    Consumer(Producer prod) {  
        this.producer = prod;  
    }  
    public void run() {  
        while (this.producer.isProgress) {  
            System.out.println("BUSY SPIN condition - Production Still going on");  
        }  
        System.out.println("Consumer starts consuming products.");  
        int size = this.producer.proList.size();  
        for (int i = 0; i < size; i++) {  
            System.out.println("Consumed : " + this.producer.proList.remove(0) + " ");  
        }  
    }  
}  
  
public class BusySpin {  
    public static void main(String args[]) throws InterruptedException {  
        Producer producer = new Producer();  
        Consumer consumer = new Consumer(producer);  
        Thread producerThread = new Thread(producer, "prodThread");  
        Thread consumerThread = new Thread(consumer, "consThread");  
        producerThread.start();  
        consumerThread.start();  
    }  
}  
BUSY SPIN condition - Production Still going on  
Producer is still Producing, Produced : 1  
BUSY SPIN condition - Production Still going on  
Producer is still Producing, Produced : 2  
BUSY SPIN condition - Production Still going on  
BUSY SPIN condition - Production Still going on
```

```

Producer is still Producing, Produced : 3
BUSY SPIN condition - Production Still going on
Producer is still Producing, Produced : 4
BUSY SPIN condition - Production Still going on
Producer is still Producing, Produced : 5
BUSY SPIN condition - Production Still going on
Producer is still Producing, Produced : 6
BUSY SPIN condition - Production Still
BUSY SPIN condition - Production Still going on
BUSY SPIN condition - Production Still going on
..... so on till 10
Consumer starts consuming products.
Consumed : 1
Consumed : 2
Consumed : 3
Consumed : 4
Consumed : 5
Consumed : 6
Consumed : 7
Consumed : 8

```

To avoid this BusySpin, we can use `wait()` & `notify()`

- Inside `run()` method of consumer we are using `wait()`
- producer use `notify()` method to notify Consumer that producer is done with the production

```

class Producer implements Runnable {
    ArrayList<Integer> proList;

    Producer() {
        proList = new ArrayList<>();
    }

    @Override
    public void run() {
        synchronized (this) {
            for (int i = 1; i <= 8; i++) {
                proList.add(i);
                System.out.println("Producer is still Producing, Produced : " + i);
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println("Production Complted .... Notify Counsumer");
            this.notify();
        }
    }
}

class Consumer extends Thread {
    Producer producer;

    Consumer(Producer prod) {
        producer = prod;
    }

    public void run() {

```

```

        synchronized (this.producer) {
            try {
                System.out.println("Consumer Waiting for Notify .... ");
                this.producer.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Consumer starts consuming products.");
        int size = this.producer.proList.size();
        for (int i = 0; i < size; i++) {
            System.out.println("Consumed : " + this.producer.proList.remove(0) + " ");
        }
    }

public class BusySpin {
    public static void main(String args[]) throws InterruptedException {
        Producer producer = new Producer();
        Consumer consumer = new Consumer(producer);
        Thread producerThread = new Thread(producer, "prodThread");
        Thread consumerThread = new Thread(consumer, "consThread");
        consumerThread.start();
        producerThread.start();
    }
}

Consumer Waiting for Notify ....
Producer is still Producing, Produced : 1
Producer is still Producing, Produced : 2
Producer is still Producing, Produced : 3
Producer is still Producing, Produced : 4
Producer is still Producing, Produced : 5
Producer is still Producing, Produced : 6
Producer is still Producing, Produced : 7
Producer is still Producing, Produced : 8
Production Complted .... Notify Counsumer
Consumer starts consuming products.
Consumed : 1
Consumed : 2
Consumed : 3
Consumed : 4
Consumed : 5
Consumed : 6
Consumed : 7
Consumed : 8

```

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

**"Race condition occurs when two or more threads try to read & write a shared variable at the same time"**

Because the thread scheduling algorithm can swap between threads at any time, **you don't know the order in which the threads will attempt to access the shared data**. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data.

```

class Counter implements Runnable {
    private int count;
    @Override
    public void run() {
        for (int i = 1; i <= 100; i++) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
            }
        }
    }
}

```

```

        e.printStackTrace();
    }
    count++;
}
}

public int getCount() {
    return this.count;
}

}

public class RaceCondition{
    public static void main(String[] args) throws InterruptedException {
        Counter c = new Counter();

        Thread t1 = new Thread(c);
        t1.start();

        Thread t2 = new Thread(c);
        t2.start();

        // wait for threads to finish processing
        t1.join();
        t2.join();
        System.out.println("Excepting =200, Actual is " + c.getCount());
    }
}
Excepting =200, Actual is =141 -1st Run
Excepting =200, Actual is =157 -2nd Run

```

Here T1, T2 are in Race to Increment values.

### 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**

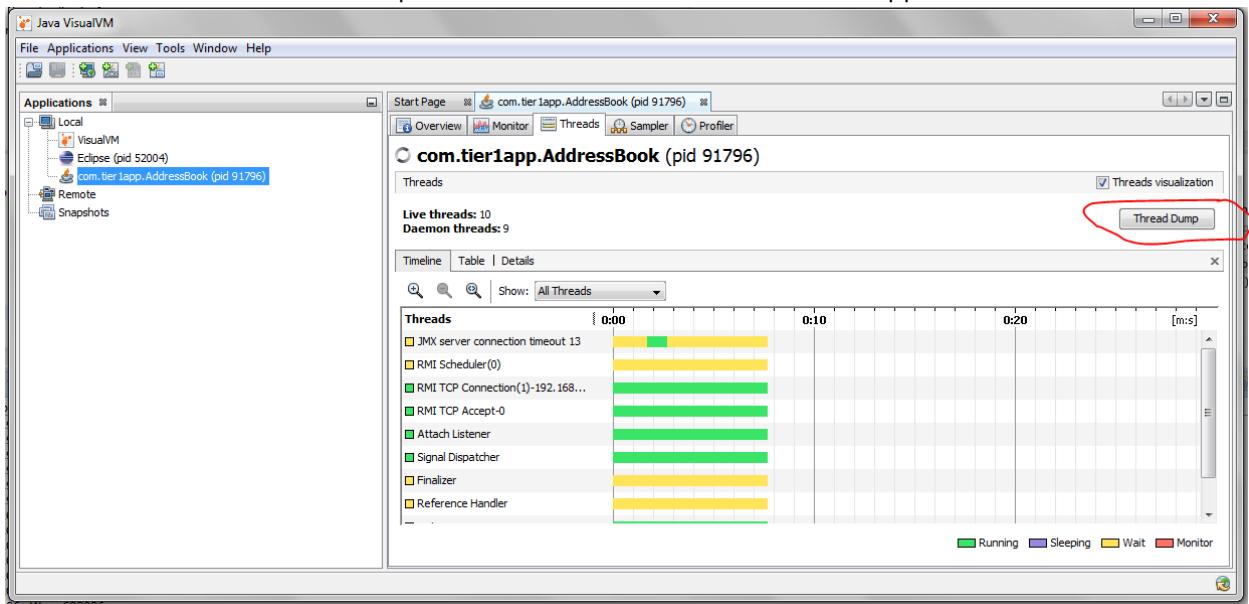
- 'jstack' is an effective command line tool to capture thread dumps.
- In IBM WAS server , DMGR> Troubleshooting> Java Dumps

The screenshot shows the IBM WebSphere Application Server Administration Console interface. On the left, there's a navigation tree with 'View: All tasks' at the top. Under 'Troubleshooting', 'Logs and trace' is expanded, showing 'Configuration problems', 'Class loader viewer', 'Java dumps and cores', 'Configuration Validation', 'Diagnostic Provider', and 'Runtime Messages'. The 'Java dumps and cores' link is also circled in red. The main panel is titled 'Java dumps and cores' and contains a sub-section 'Java dumps and cores' with the following text: 'Use this panel to generate heap dumps, Java cores or system dumps for a running process. The files resulting from these operations are placed on the local file system.' Below this is a 'Preferences' link. At the bottom of the panel, there are tabs for 'Heap dump', 'Java core', and 'System dump', with 'Heap dump' being the active tab. A table lists resources for generating dumps:

Select	Server	Node	Host Name	Version	Type	Status
<input type="checkbox"/>	apps_clone01	node01	bil-uxpwas-936v.pii.pxl.int	ND 9.0.5.10	servers	<span style="color:red;">X</span>
<input type="checkbox"/>	apps_clone02	node02	BIL-UXPWAS-937v.pii.pxl.int	ND 9.0.5.10	servers	<span style="color:red;">X</span>
<input type="checkbox"/>	dmgr	DmgrNode01	bil-uxpdgr-936v.pii.pxl.int	ND 9.0.5.10	servers	<span style="color:green;">+</span>
<input type="checkbox"/>	mdm_clone01	node01	bil-uxpwas-936v.pii.pxl.int	ND 9.0.5.10	servers	<span style="color:red;">X</span>
<input type="checkbox"/>	mdm_clone02	node01	bil-uxpwas-936v.pii.pxl.int	ND 9.0.5.10	servers	<span style="color:red;">X</span>
<input type="checkbox"/>	nodeagent	node01	bil-uxpwas-936v.pii.pxl.int	ND 9.0.5.10	servers	<span style="color:green;">+</span>
<input type="checkbox"/>	nodeagent	node02	BIL-UXPWAS-937v.pii.pxl.int	ND 9.0.5.10	servers	<span style="color:green;">+</span>
<input checked="" type="checkbox"/>	services_clone01	node01	bil-uxpwas-936v.pii.pxl.int	ND 9.0.5.10	servers	<span style="color:green;">+</span>
<input checked="" type="checkbox"/>	services_clone02	node02	BIL-UXPWAS-937v.pii.pxl.int	ND 9.0.5.10	servers	<span style="color:green;">+</span>

Total 9

- Java VisualVM is a GUI tool that provides detailed information about the applications



### Why Swing is not thread-safe in Java?

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 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 other's thread local variable.
- These variables 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
    }
}
```

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.

```
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;  
    }  
}  
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);  
    }  
}
```

## 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.

We can do it in Two ways

### 1. Using **interrupt()**

```
public class Demo {  
    public static void main(String[] args) throws Exception {  
        Runnable runnable = ()->{  
            System.out.println("Running...");  
            try {  
                Thread.sleep(5000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
            System.out.println("I have Completed ...");  
        };  
        Thread t1 = new Thread(runnable);  
        t1.start();  
        t1.interrupt();  
    }  
}  
Running...  
I have Completed ...  
java.lang.InterruptedIOException: sleep interrupted  
at java.lang.Thread.sleep(Native Method)  
at Demo.lambda$0(Demo.java:16)  
at java.lang.Thread.run(Thread.java:748)
```

In this case we can only stop Sleeping thread.

### 2. volatile boolean variable

In this case we need to use **volatile boolean variable**, & we need to change value to TRUE if we want to stop a thread.

I have a server; I need it to Stop

```
class Server implements Runnable {  
    private volatile boolean stop = false;  
  
    public void run() {  
        while (!stop) {  
            System.out.println("Server is running.....");  
        }  
        System.out.println("Server is stopped....");  
    }  
    public void stop() {  
        stop = true;  
    }  
}  
  
public class Demo {  
    public static void main(String[] args) throws Exception {  
  
        Server myServer = new Server();  
        Thread t1 = new Thread(myServer, "T1");  
        t1.start();  
        // Now, let's stop our Server thread  
        System.out.println(Thread.currentThread().getName() + " is stopping Server thread");  
        TimeUnit.MILLISECONDS.sleep(8);  
        myServer.stop();  
        // Let's wait to see server thread stopped  
  
        System.out.println(Thread.currentThread().getName() + " is finished now");  
    }  
}
```

Actually we are not stopping the Thread, we just coming out of the run().

## **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 Deadlock, Starvation, and Livelock?**

**Deadlock:** is a situation where two more threads are blocked because of waiting for each other forever.one of the possible situation is **nested Synchronized blocks.**

To create simple deadlock situation for a servlet, just place **doPost()** method inside **doGet()** and **doGet()** method inside **doPost()**.

```
public class Business {  
    private Object lock1 = new Object();  
    private Object lock2 = new Object();  
  
    public void foo() throws InterruptedException {  
        synchronized (lock1) {  
            Thread.sleep(5000);  
            synchronized (lock2) {  
                System.out.println("foo");  
            }  
        }  
    }  
  
    public void bar() throws InterruptedException {  
        synchronized (lock2) {  
            Thread.sleep(5000);  
            synchronized (lock1) {  
                System.out.println("bar");  
            }  
        }  
    }  
  
    public static void main(String[] args) {  
        Business business = new Business();  
  
        Thread t1 = new Thread(new Runnable() {  
            public void run() {  
                business.foo();  
            }  
        });  
        t1.start();  
  
        Thread t2 = new Thread(new Runnable() {  
            public void run() {  
                business.bar();  
            }  
        });  
        t2.start();  
    }  
}
```

## **how to avoid deadlock**

- Avoid acquiring more than one lock at a time.
- If not, make sure that you acquire multiple locks in consistent order.

In the above example, you can avoid deadlock by synchronize two locks in the same order in both methods.

```
public void foo() {  
    synchronized (lock1) {  
        synchronized (lock2) {  
            System.out.println("foo");  
        }  
    }  
}  
  
public void bar() {  
    synchronized (lock1) {  
        synchronized (lock2) {  
            System.out.println("bar");  
        }  
    }  
}
```

**2. LiveLock**: A real-world example of livelock occurs when two people meet in a narrow corridor(காலைமுடி), 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.

Livelock occurs when two or more processes continually repeat the same interaction in response to changes in the other processes without doing any useful work. These processes are not in the waiting state, and they are running concurrently. This is different from a deadlock because in a deadlock all processes are in the waiting state.

```
public class Test {  
    boolean selected=false;  
    public boolean isSelected() {  
        return selected;  
    }  
  
    public void setSelected(boolean selected) {  
        this.selected = selected;  
    }  
  
    public static void main(String args[]) {  
        Test itemOne = new Test();  
        Test itemTwo = new Test();  
  
        Thread tOne = new Thread(new Runnable() {  
            @Override  
            public void run() {  
                while(!itemOne.isSelected()) {  
                    try {  
                        Thread.sleep(10);  
                    } catch (InterruptedException e) {  
                        e.printStackTrace();  
                    }  
                }  
                itemTwo.setSelected(true);  
            }  
        });  
  
        Thread tTwo = new Thread(new Runnable() {  
            @Override  
            public void run() {  
                while(!itemTwo.isSelected()) {  
                    try {  
                        Thread.sleep(10);  
                    } catch (InterruptedException e) {  
                        e.printStackTrace();  
                    }  
                }  
                itemOne.setSelected(true);  
            }  
        });  
    }  
}
```

```

        tOne.start();
        tTwo.start();
    }
}

```

**3. Starvation:** describes a situation where a thread holds a resource for a long time so other threads are blocked forever. The blocked threads are waiting to acquire the resource, but they never get a chance. Thus they starve to death. BTW, Starvation **means suffering or death caused by lack of food.**

Starvation can occur due to the following reasons:

- Threads are blocked infinitely because a **thread takes long time to execute** some synchronized code (e.g. heavy I/O operations or infinite loop).
- A **thread doesn't get CPU's time for execution because it has low priority as compared to other threads which have higher priority.**
- Threads are waiting on a resource forever, but they remain waiting forever because other threads are constantly notified instead of the hungry ones.
- When a starvation situation occurs, the program is still running but doesn't run to completion because some threads are not executed.

```

public class Worker {

    public synchronized void work() {
        String name = Thread.currentThread().getName();
        String fileName = name + ".txt";

        try ( BufferedWriter writer = new BufferedWriter(new FileWriter(fileName)); ) {
            writer.write("Thread " + name + " wrote this mesage");
        } catch (IOException ex) {
            ex.printStackTrace();
        }

        while (true) {
            System.out.println(name + " is working");
        }
    }

    public static void main(String[] args) {
        Worker worker = new Worker();

        for (int i = 0; i < 10; i++) {
            new Thread(new Runnable() {
                public void run() {
                    worker.work();
                }
            }).start();
        }
    }
}

//we created 10 threads but, 1st threads run continuously in infinite loop, remaining will die
//because they won't get any chance to execute.

```

Finally

- **Deadlock:** All threads are blocked; the program hangs forever.
- **Livelock:** No threads blocked but they run into infinite loops. The program is still running but unable to make further progress.
- **Starvation:** Only one thread is running, and other threads are waiting forever

<https://www.codejava.net/java-core/concurrency/understanding-deadlock-livelock-and-starvation-with-code-examples-in-java>

### 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)); //true or false -checked
```

### 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'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**.

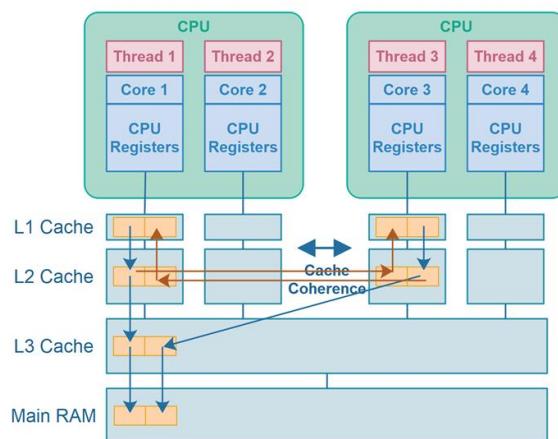
The *Callable* interface is a generic interface containing a single `call()` method – which returns a generic value `V`:

```
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* in Java occurs when two threads running on two different CPUs write to two different variables which happen to be stored within the same CPU cache line



<https://alidg.me/blog/2020/5/1/false-sharing>

## 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 on instance methods**, 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();
    }
}

t1
in block t1
in block t1 end

t2
in block t2
in block t2 end
t3
in block t3
in block t3 end
//only one thread is running
```

**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 waiting 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 waiting COnsumer Thread
28055 : Produced, notify waiting COnsumer Thread
28056 : Produced, notify waiting COnsumer Thread
28057 : Produced, notify waiting COnsumer Thread
28058 : Produced, notify waiting COnsumer Thread
28059 : Produced, notify waiting COnsumer Thread
28060 : Produced, notify waiting 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

```

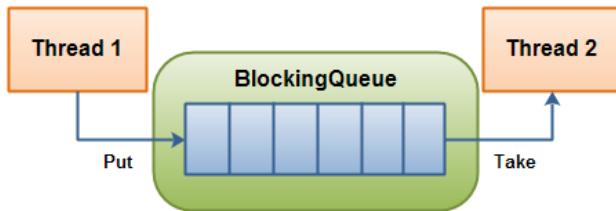
```

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 until produce

```

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

A **BlockingQueue** is typically used when one thread will produce objects, another thread consumes those Objects.



	Throws Exception	Special Value	Blocks	Times Out
<b>Insert</b>	add(o)	offer(o)	put(o)	offer(o, timeout, timeunit)
<b>Remove</b>	remove(o)	poll()	take()	poll(timeout, timeunit)
<b>Examine</b>	element()	peek()		

- o **BlockingQueue in Java doesn't allow null elements**, various implementations like ArrayBlockingQueue, LinkedBlockingQueue throws NullPointerException when you try to add null on queue
- o **two types of BlockingQueue:**
  - a. **Bounded queue** – with maximal capacity defined

```
BlockingQueue<String> blockingQueue = new LinkedBlockingDeque<>(10);
```

b. **UnBounded queue** –no maximum capacity, can grow almost indefinitely

```
BlockingQueue<String> blockingQueue = new LinkedBlockingDeque<>();
```

### Producer-Consumer Example

BlockingQueue provides a **put()** method to store the element and **take()** method to retrieve the element. Both are blocking method, which means **put()** will block if the queue has reached its capacity and there is no place to add a new element.

Similarly, **take()** method will block if blocking queue is empty. So, you can see that critical requirement of the producer-consumer pattern is met right there, you don't need to put any thread synchronization code.

```

class Producer extends Thread {
    private BlockingQueue<Integer> sharedQueue;
    public Producer(BlockingQueue<Integer> aQueue) {
        super("PRODUCER");
        this.sharedQueue = aQueue;
    }
    public void run() { // no synchronization needed
        for (int i = 0; i < 10; i++) {
            try {
                System.out.println(getName() + " produced " + i);
                sharedQueue.put(i);
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

        // if we remove sleep, put will execute 10 times, then take will execute
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}

class Consumer extends Thread {
    private BlockingQueue<Integer> sharedQueue;

    public Consumer(BlockingQueue<Integer> aQueue) {
        super("CONSUMER");
        this.sharedQueue = aQueue;
    }

    public void run() {
        try {
            while (true) {
                Integer item = sharedQueue.take();
                System.out.println(getName() + " consumed " + item);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class BlockingQueueDemo {
    public static void main(String[] args) {
        BlockingQueue<Integer> sharedQ = new LinkedBlockingQueue<Integer>();
        Producer p = new Producer(sharedQ);
        Consumer c = new Consumer(sharedQ);
        p.start();
        c.start();
    }
}

```

- **ArrayBlockingQueue** – a blocking queue class based on bounded Java Array. Once instantiated, cannot be resized.
- **PriorityBlockingQueue** – a priority queue based blocking queue. It is an unbounded concurrent collection.
- **LinkedBlockingQueue** – an optionally bounded Java concurrent collection. Orders elements based on FIFO order.

### 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.

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, **thread scheduler schedules the threads according to their priority** (known as pre-emptive scheduling). **But it is not guaranteed because it depends on JVM specification** that which scheduling it chooses.

- 1) public static final int **MIN\_PRIORITY** (1);
- 2) public static final int **NORM\_PRIORITY** (5);
- 3) public static final int **MAX\_PRIORITY** (10);

### Example

```
public class ThreadPriority extends Thread{
    @Override
    public void run() {
        Thread th= Thread.currentThread();
        System.out.println("Name :" +th.getName() +"\t Priority:" +th.getPriority());
    }
    public static void main(String[] args) {
        ThreadPriority t1 = new ThreadPriority();
        ThreadPriority t2 = new ThreadPriority();
        ThreadPriority t3 = new ThreadPriority();

        t1.setPriority(MIN_PRIORITY);
        t2.setPriority(NORM_PRIORITY);
        t3.setPriority(MAX_PRIORITY);

        t1.start();
        t2.start();
        t3.start();
    }
}
```

Name :Thread-2 Priority:10  
Name :Thread-1 Priority:5  
Name :Thread-0 Priority:1

Even though t1 starts first, it has **MIN\_PRIORITY** so, it executes last that to depends on JVM Specification

## 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 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)**

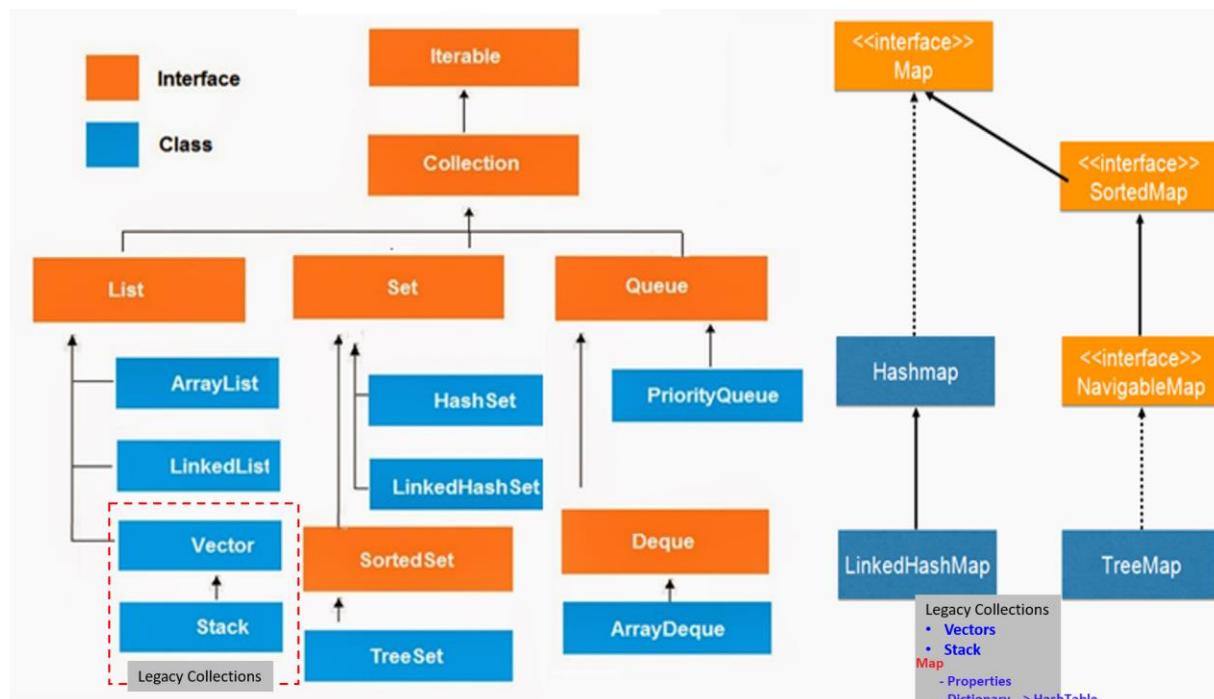
ConcurrentHashMap achieves its scalability and thread-safety by partitioning actual map into a number of sections. This partitioning is achieved using concurrency level.

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



List Interface	Set	Queue interface
Duplicable		
ArrayList	random access, add/remove expensive(shift),not ordered	FIFO first in first out
LinkedList	sequence access,add/remove cheap(no shift), ordered	Ordered list of item to be processed
Vector	like ArrayList,but synchronized,more methods	Priori-tyQueue no null item, ordered by priority
Stack	extends Vector, LIFO, more methods boolean push(),boolean peek(),boolean push(obj)	Deque interface, doubled ended queue
<b>HashMap,TreeMap and Hashable</b>		
	HashMap: unique key, dup values;allow null values	ArrayList and Stack
	TreeMap ordered object	
	HashTable synchronized, no nulls	
<b>Comparable and Comparator interfaces</b>		
	Comparator equals(), Compare()	
	Comparable compareTo()	

## Java Collections class

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

**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 a Comparator, if we want some custom ordering.

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

**int binarySearch (list,"element") :** This method searches the key using binary search in the specified list. before calling this method,list should be sorted by natural ordering, 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

```
public class TreeSetStringBuffer {
public static void main(String[] args) {
    TreeSet t = new TreeSet();
    t.add(new StringBuffer("A"));
    t.add(new StringBuffer("X"));
    t.add(new StringBuffer("O"));
    t.add(new StringBuffer("L"));
    System.out.println(t);
}
}

Exception in thread "main" java.lang.ClassCastException: java.lang.StringBuffer cannot be cast to
java.lang.Comparable
```

- If we are depending on Def. Natural Sorting Order objects should be **Homogeneous (same type objects)** & **Comparable**. Otherwise we will get Runtime Exception `java.lang.ClassCastException`

```
public static void main(String[] args) {
    // List list = new ArrayList(); // [1, 2, 3, 4] - No Error
    // Set list = new HashSet(); // [1, 2, 3, 4] - No Error
```

```

Set list = new TreeSet(); // [1, 2, 3, 4] - Error

list.add("1");
list.add("2");
list.add("3");
list.add(4);
System.out.println(list);
}

Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to
java.lang.Integer
at java.lang.Integer.compareTo(Integer.java:52)

```

- An object is said to be comparable if and only if corresponding class implements **Comparable** interface.**java.lang.String & all wrapper classes (Int, Float, Byte) already implements Comparable interface**

```
public final class java.lang.String implements java.io.Serializable, java.lang.Comparable
```

- java.lang.StringBuffer doesn't implements comparable interface**

```
public final class java.lang.StringBuffer extends java.lang.AbstractStringBuilder implements
java.io.Serializable,java.lang.CharSequence
```

So it throws **Exception in thread "main" java.lang.ClassCastException: java.lang.StringBuffer cannot be cast to java.lang.Comparable**

- If we Take **EmpBo**, if we pass employee list Objects to **Collection.sort(EmpBo)** method, it will throws Error, because it only accepts objects of Comparable types only.

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

We have Two ways to provide Sorting order for StringBuffer & Other classes which are not implementing Comparable Interface

- Implement **java.lang.Comparable** interface & override **int compareTo(Object)**
- Implement **java.util.Comparator** interface & override **int compare(Object, Object)**

## 1.Comparable interface

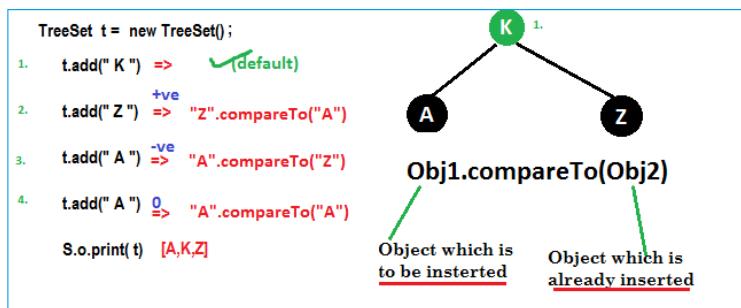
- It provides single sorting sequence only i.e. you can sort the elements on based on single data member only. For example, it may be rollno, name, age or any one of them, not all else.
- Comparable is an interface defining a strategy of comparing an object with other objects of the same type. This is called the class's "natural ordering".so we need to define CompareTo() method
- We use **public int compareTo(Object obj)** to compare the current object with the specified object.

```

public class ComparableDemo {
    public static void main(String[] args) {
        System.out.println("A".compareTo("Z")); // 1-26 = -25
        System.out.println("Z".compareTo("C")); // 26-3 = 23
        System.out.println("A".compareTo("A")); // 1-1 = 0
        // System.out.println("A".compareTo(null)); //R.E NPE
    }
}

```

While adding Objects into TreeSet JVM will call `compareTo()` method. It will compare inserting value with exiting values one by one using `compareTo()` method.



```

public class Employee implements Comparable<Employee> {
    int id;
    String name;
    double salary;
    //Setters & Getters

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

    @Override
    public int compareTo(Employee o) {
//Here add(105) object will be the CurrentID, Object.ID is existing element ID's. Which are not sorted.
        System.out.println("Current ID :" + this.id + "\t Obj.ID : " + o.id);

        if (this.id < o.id) {
            return -1;
        } else if (this.id > o.id) {
            return 1;
        } else {
            return 0;
        }
    }

    public static void main(String[] args) {
        Set<Employee> employees = new TreeSet<Employee>();
        employees.add(new Employee(105, "Satya", 3000));
        System.out.println("After 105 : -----> " + employees + "\n");

        employees.add(new Employee(102, "RAJ", 2000));
        System.out.println("After 102 : -----> " + employees + "\n");

        employees.add(new Employee(104, "Madhu", 5000));
        System.out.println("After 104 : -----> " + employees + "\n");

        employees.add(new Employee(101, "Srinivasa", 1000));
        System.out.println("After 101 : -----> " + employees + "\n");

        employees.add(new Employee(103, "Vinod", 4000));
        System.out.println("After 103 : -----> " + employees + "\n");

        //See here we are adding 100, which is less than all existing elements.
        //So, it will compare with almost all elements using compareTo() method
        employees.add(new Employee(100, "Vinod", 1000));
        System.out.println("After 100 : -----> " + employees + "\n");
        System.out.println("After : " + employees + "\n");
    }
}
    
```

```

Current ID :105 Obj.ID : 105
After 105 : -----> [105]

Current ID :102 Obj.ID : 105
After 102 : -----> [102, 105]

Current ID :104 Obj.ID : 105
Current ID :104 Obj.ID : 102
After 104 : -----> [102, 104, 105]

Current ID :101 Obj.ID : 104
Current ID :101 Obj.ID : 102
After 101 : -----> [101, 102, 104, 105]

Current ID :103 Obj.ID : 104
Current ID :103 Obj.ID : 102
After 103 : -----> [101, 102, 103, 104, 105]

Current ID :100 Obj.ID : 104
Current ID :100 Obj.ID : 102
Current ID :100 Obj.ID : 101
After 100 : -----> [100, 101, 102, 103, 104, 105]

After : [100, 101, 102, 103, 104, 105]

```

Remember, here we can't pass comparable Object to TreeSet(), like comparable.

```

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, "Srinivasa", 1000));
        employees.add(new Employee(103, "Vinod", 4000));

        System.out.println("Before : " + employees);
        //Until here no sorting will be performed

        Collections.sort(employees);
        //Collection.sort(Comparable) method will intern call CompareTo,
        & it will compare each element with other & Sort the elements

        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]]
```

In above we sorted Employees only on their ID type, but if we want to sort by Name & Salary at a time it won't be possible. It accepts only one variable comparison at a time.

If we want to sort by Id, Name & Salary at a time, we can use Comparator interface.

### Comparator Interface

Comparator present in `java.util package` & it defines two methods `compare(ob1, ob2)` & `equals(ob1)`

```
public int compare(Object ob1, Object ob2);  
public boolean equals(Object ob)
```

Whenever we are implementing comparator interface we should provide implementation only for `compare()` method & we are not required implementation for `equals()` method, because it is already available to our class from Object class through inheritance.

#### Example: To insert integer Objects into TreeSet where Sorting Order is Descending order

TreeSet t = new TreeSet( new MyComparator()); (Line 1)

1. t.add( 10 ) => 10 ✓
2. t.add( 0 ) => compare(10,0)
3. t.add( 15 ) => compare(15,0)
4. t.add( 5 ) => compare(5,15)
5. t.add( 20 ) => compare(20,5)
6. t.add( 20 ) => ✓

[0,5,10,15,20] S.o.print( t ) [20,15,10,5,0]

```
class MyComparator implements Comparator {  
    public int compare(Object o1, Object o2) {  
        Integer i1 = (Integer) o1;  
        Integer i2 = (Integer) o2;  
        if (i1 < i2) {  
            return +1;  
        } else if (i1 > i2) {  
            return -1;  
        } else {  
            return 0;  
        }  
    }  
}
```

```
public class MyComparator implements Comparator {  
    @Override  
    public int compare(Object oldObj, Object newObj) {  
        System.out.println("newObj: " + newObj + ", oldObj: " + oldObj);  
  
        Integer i1 = (Integer) oldObj;  
        Integer i2 = (Integer) newObj;  
  
        if (i1 < i2) {  
            return +1;  
        } else if (i1 > i2) {  
            return -1;  
        } else {  
            return 0;  
        }  
    }  
}
```

```

public static void main(String[] args) {
    //TreeSet t = new TreeSet(); // Line-1
    TreeSet t = new TreeSet(new MyComparator()); // Line-2

    t.add(50);
    System.out.println("After 50: -----> " + t + "\n");

    t.add(40);
    System.out.println("After 40: -----> " + t + "\n");

    t.add(10);
    System.out.println("After 10: -----> " + t + "\n");

    t.add(30);
    System.out.println("After 30: -----> " + t + "\n");

    t.add(20);
    System.out.println("After 20: -----> " + t + "\n");

    //See here we are adding 1, which is less than all existing elements.
    //So, it will compare with almost all elements using compare() method
    t.add(1);
    System.out.println("After 1: -----> " + t + "\n");
    System.out.println(t);
}

newObj: 50, oldObj: 50
After 50: -----> [50]

newObj: 50, oldObj: 40
After 40: -----> [50, 40]

newObj: 50, oldObj: 10
newObj: 40, oldObj: 10
After 10: -----> [50, 40, 10]
newObj: 40, oldObj: 30
newObj: 10, oldObj: 30
After 30: -----> [50, 40, 30, 10]

newObj: 40, oldObj: 20
newObj: 10, oldObj: 20
newObj: 30, oldObj: 20
After 20: -----> [50, 40, 30, 20, 10]

newObj: 40, oldObj: 1
newObj: 20, oldObj: 1
newObj: 10, oldObj: 1
After 1: -----> [50, 40, 30, 20, 10, 1]

[50, 40, 30, 20, 10, 1]

```

[50, 40, 30, 20, 10, 1] At **Line1**, if we **not** passing **MyComparator** object then internally JVM will call **compareTo()** method which is for default Natural Sorting order.in this case output is [0,5,10,15,20].

At **Line2**, if we passing **MyComparator** object then JVM will call **compare()** method which is for customize Sorting order.in this case output is [20,15,10,5,0].

```

class MyComparator implements Comparator
{
    public int compare(Object o1, Object o2)
    {
        Integer i1 = (Integer) o1;
        Integer i2 = (Integer) o2;

        1.     return i1.compareTo(i2)          // [0,5,10...]
        2.     -return i1.compareTo(i2)         // [20,15,10..]
        3.     return i2.compareTo(i1)          // [20,15,10..]
        4.     -return i2.compareTo(i1)         // [0,5,10]
        5.     return +1                      // [Insertion Order]
        6.     return -1;                   // Reverse of Insertion order
        7.     return 0;                    // Only 1st element will insert
                                         // remaining are Duplicates
    }
}

```

### various possible implementations of compare() method

As the same way if we want to change String order we do as follows

```

public class TreesetStringComp {
    public static void main(String[] args) {
        TreeSet t = new TreeSet(new MyComparators());
        t.add("HYDERABAD");
        t.add("VIJAYAWADA");
        t.add("BANGLORE");
        t.add("VIZAG");
        System.out.println(t);
    }
}

class MyComparators implements Comparator {
    public int compare(Object newObj, Object oldObj) {
        String s1 = (String) newObj;
        String s2 = (String) oldObj;
        int i1 = s1.length();
        int i2 = s2.length();
        if (i1 < i2) {
            return +1;
        } else if (i1 > i2) {
            return -1;
        } else {
            return 0;
        }
    }
}
[VIJAYAWADA, HYDERABAD, BANGLORE, VIZAG]

```

- `EmpName` implements Comparator for NAME Sorting
- `EmpSalary` implements Comparator for SALARY Sorting
- `Comparable` for ID Sorting for Employe Class

### 1.EmpName implements Comparator for NAME Sorting

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

### 2.EmpSalary implements Comparator for SALARY Sorting

```
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;
    }
}
```

### 3.Comparable for ID Sorting for Employee Class

```
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;
    }
    public int compareTo(Employee o) {
        if (this.id < o.id) {
            return -1;
        } else if (this.id > o.id) {
            return 1;
        } else {
            return 0;
        }
    }
    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 Employees based on Multiple Sorting(EmpName, EmpSaltry)
        //sort method accepts Comparator: Collections.sort(<list>, Comparator)
        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]]

```

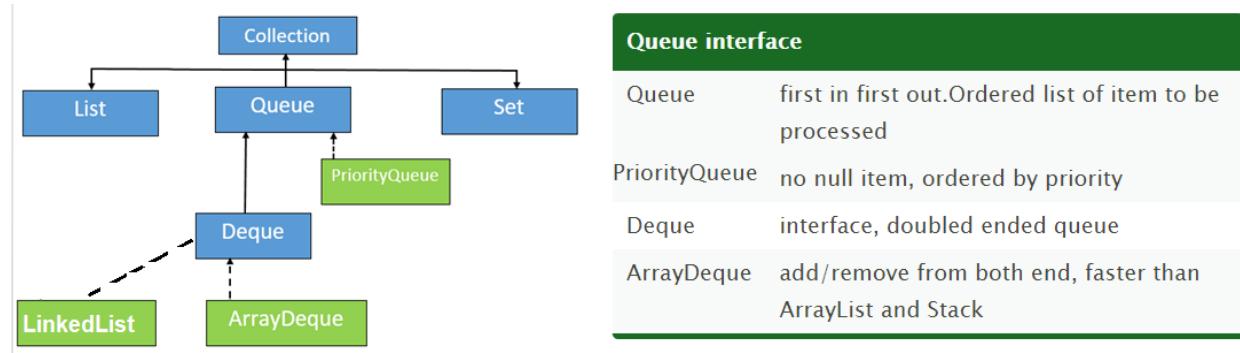
- **Comparable** interface can be used to provide **single way of sorting** whereas **Comparator** interface is used to provide **different ways of sorting**.
- For using Comparable, Class needs to implement it whereas for using Comparator we don't need to make any change in the class, **we can implement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

<b>Comparable</b>	<b>Comparator</b>
1.it is meant for <b>Defalut Natural Sorting order</b> 2.present in <code>java.lang</code> package 3.it defines only one method <code>compareTo()</code> 4. <b>String &amp; all Wrapper classes</b> implements Comparable interface	1.it is meant for <b>Customized Natural Sorting order</b> 2.present in <code>java.util</code> package 3.it defines 2 methods <code>compare()</code> & <code>equals()</code> 4.only 2 GUI classes Collator&RulebasedCollator implements Comparator

<b>Comparable</b>	<b>Comparator</b>
Default Natural Sorting Order	Customized Sorting order
Def. implementation provided in String, Wrapper classes	Def. Implementation not provided.
Comparison possible on same type of Object Ex. <code>this.compareTo(Obj)</code>	Comparison possible using different type of Objects. Ex . <code>ob1.compare(Obj2)</code>
We can't pass comparable object to TreeSet for customized Sorting order. <b>TreeSet(Comparable c) X not available</b>	We can pass Comparator object to TreeSet for customized Sorting order. <b>TreeSet(Comparator c) ✓ available</b>
Available in <code>java.lang</code> package	Available in <code>java.util</code> package

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

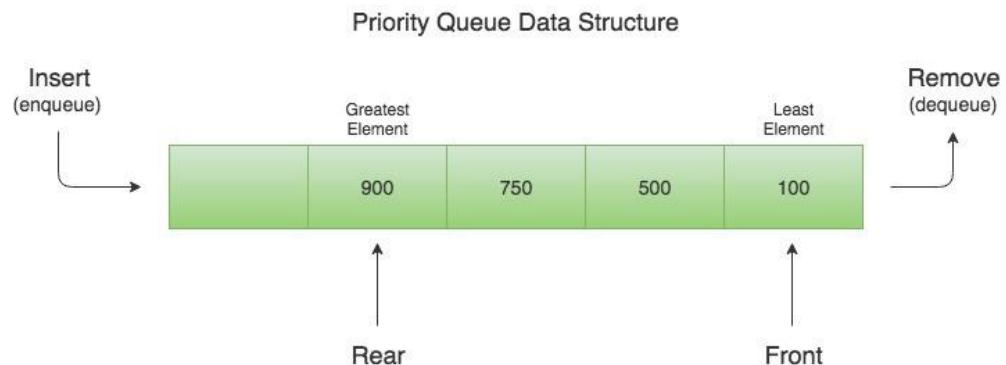
LinkedList class implements the Queue interface and therefore it can be used as a Queue. The process of adding an element at the end of the Queue is called **Enqueue**, and the process of removing an element from the front of the Queue is called **Dequeue**.



A priority queue in Java is a special type of queue wherein all the elements are **ordered**. We define Priority of elements by ordering.

- 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

```
public class Test {  
    public static void main(String args[]) {  
  
        String a[] = { "a", "b", "c" };  
        System.out.println(a.toString());  
  
        // 1. Using Arrays.toString(a)  
        System.out.println(Arrays.toString(a));  
  
        // 2. Using Arrays.asList(a)  
        System.out.println(Arrays.asList(a).toString());  
  
    }  
}  
[Ljava.lang.String@15db9742  
[a, b, c]  
[a, b, c]
```

#### 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

Vector **Doubles size** when its reach max capacity. ArrayList increases its size by **(curr.capacity\*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?

Assume **Hashtable** and **ConcurrentHashMap** are two types of Homes.

- **Hashtable** locks home's main door.
- **ConcurrentHashMap** locks specific room door instead of main door.
- **Hashtable** is belongs to the Collection framework; **ConcurrentHashMap** belongs to the Executor framework.
- **Hashtable** uses **single lock** for whole data. **ConcurrentHashMap** uses **multiple locks** on segment level (16 by default) instead of object level i.e. whole **Map**.
- **ConcurrentHashMap** locking is applied only for updates. In case of retrievals, it allows full concurrency, retrievals reflect the results of the most recently completed update operations. So reads can happen very fast while writes are done with a lock.
- **ConcurrentHashMap** doesn't throw a **ConcurrentModificationException** if one thread tries to modify it while another is iterating over it and does not allow null values.
- **ConcurrentHashMap** returns **Iterator**, which fails-safe (i.e. iterator will make a copy of the internal data structure) on concurrent modification.
- **ConcurrentHashMap** uses a database shards logic (**Segment<K, V>[] segments**) is known as **Concurrency-Level**, i.e. divides the data into shards(segments) then puts locks on each shard (segment) instead of putting a single lock for whole data (**Map**). The default value is 16.

Ref. 1 <https://stackoverflow.com/questions/12646404/concurrenthashmap-and-hashtable-in-java/31579480>

Read more: <http://www.java67.com/2014/07/21-frequently-asked-java-interview-questions-answers.html#lxzz5fofpCIHh>

## 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**.

HashMap	Hashtable
HashMap is <b>non synchronized</b> . It is <b>not-thread safe</b> and can't be shared between many threads without proper synchronization	Hashtable is <b>synchronized</b> . It is <b>thread-safe</b> and can be shared with many threads
HashMap <b>allows one null key and multiple null values</b> .	Hashtable <b>doesn't allow any null key or value</b> .
HashMap is a <b>new class introduced in JDK 1.2</b> .	Hashtable is a <b>legacy class</b>
HashMap is <b>traversed by Iterator</b> .	Hashtable is <b>traversed by Enumerator and Iterator</b> .
Iterator in HashMap is <b>fail-fast</b> .	Enumerator in Hashtable is <b>not fail-fast</b> .
HashMap inherits <b>AbstractMap</b> class.	Hashtable inherits <b>Dictionary</b> class.

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

In order to use any object as Key in HashMap, it must implement `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?

Enumeration (vector/stack)	Iterator(ArrayList)	ListIterator (LinkedList)
Can iterate over a Collection  <b>Remove operation not allowed</b> <b>Add operation not allowed</b> <b>Backward direction not allowed</b>  <b>1. Enumeration elements()</b> Ex. Enumeration = v.elements()  <b>2. boolean hasMoreElements()</b> <b>3. E nextElement()</b>	Can iterate over a Collection  Remove operation allowed  <b>Add operation not allowed</b>  <b>Backward not allowed</b>  <b>1. Iterator iterator()</b> Ex. Iterator = l.iterator()  <b>2. boolean hasNext()</b>  <b>3. E next() void remove()</b>	Can iterate over a Collection  Remove operation allowed  Add operation allowed  Backward direction allowed   <b>1. ListIterator listIterator()</b> Ex. ListIterator = l.listIterator()  <b>2. boolean hasNext() boolean hasPrevious()</b>  <b>add(E e) nextIndex() previous() previousIndex() remove() next() set(E e)</b>

### 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` specific `remove()` method throws Exception, but not `Iterator` based `remove()`

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 methods 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](#))

Two things,

- CopyOnWriteArrayList performs operation on **creating cloned copy of ArrayList**.
- CopyOnWriteArrayList **doesn't throw any ConcurrentModification**, because its acts on cloned copy of Object

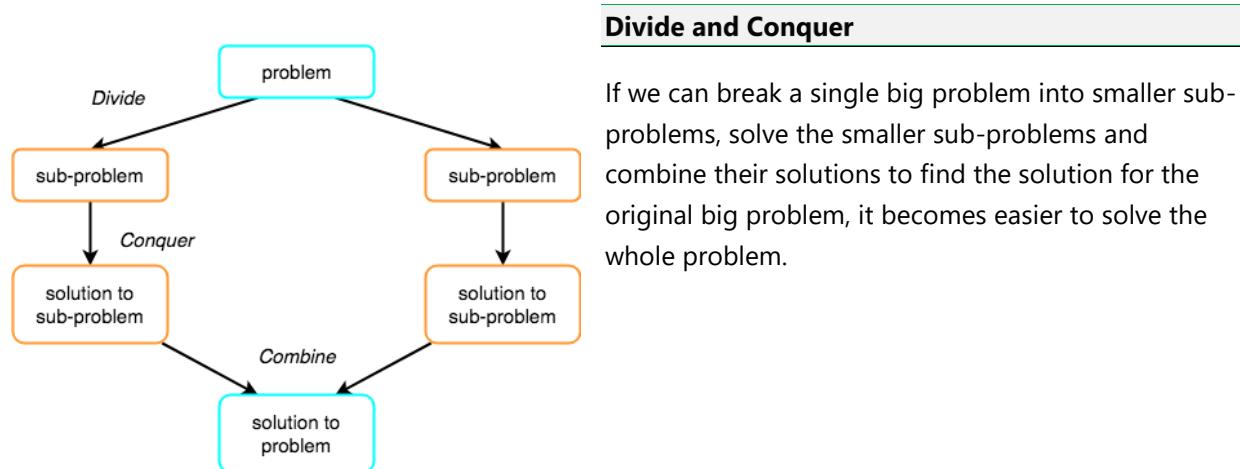
CopyOnWriteArrayList is new List implementation introduced in Java 1.5 which provides better concurrent access than Synchronized List. better concurrency is achieved by Copying ArrayList over each write and replace with original instead of locking.

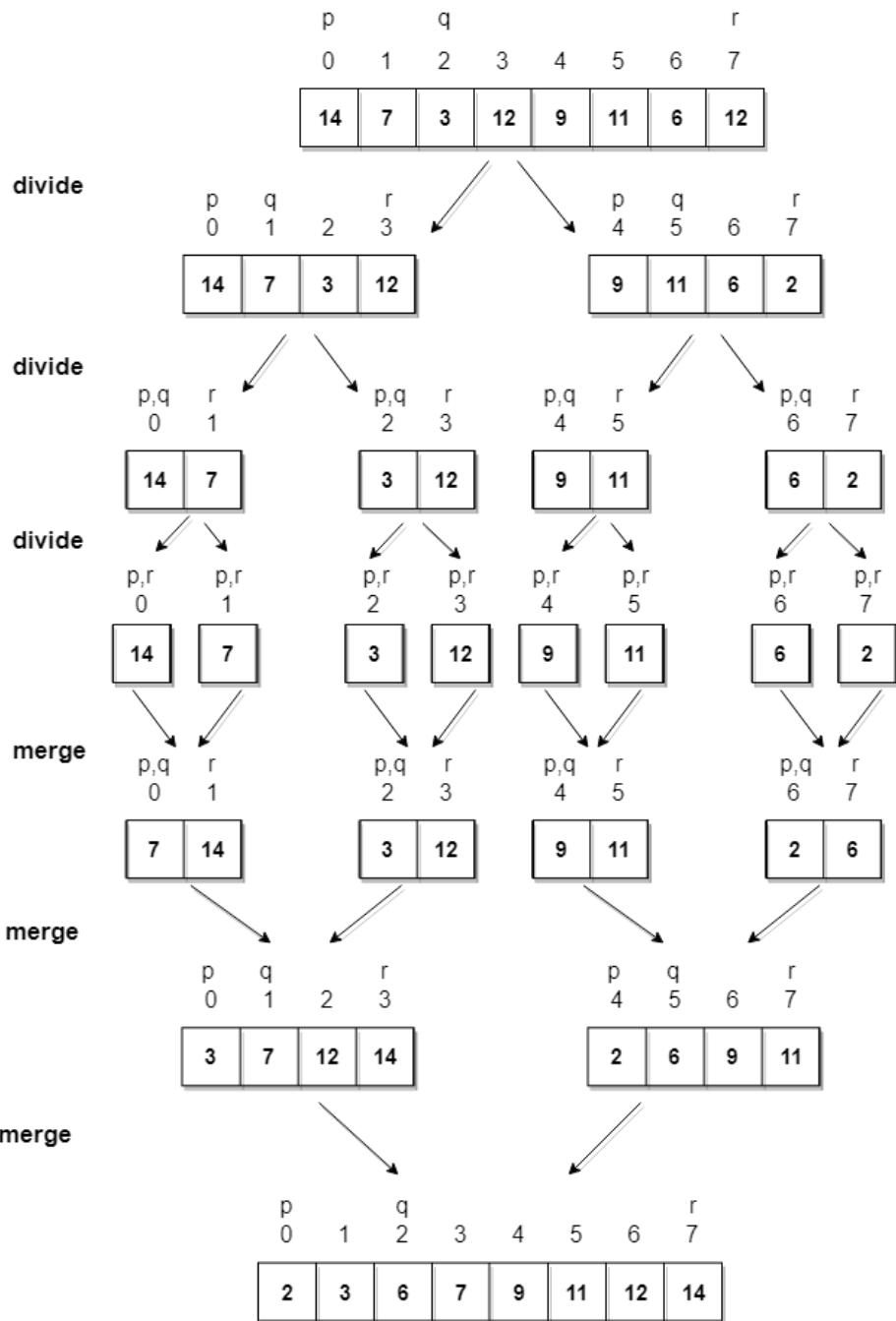
Also, CopyOnWriteArrayList doesn't throw any **ConcurrentModification** Exception. Its different than ArrayList because its thread-safe and ArrayList is not thread-safe and it's different than Vector in terms of Concurrency.

## Difference between Stable and Unstable Sorting Algorithm - MergeSort vs QuickSort

### Merge Sort Algorithm

Merge Sort follows the rule of **Divide and Conquer** to sort a given set of numbers/elements, recursively, hence consuming less time.





### Algorithm

Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too.

- Step 1** – if it is only one element in the list it is already sorted, return.
- Step 2** – divide the list recursively into two halves until it can no more be divided.
- Step 3** – merge the smaller lists into new list in sorted order.

## Quick Sort

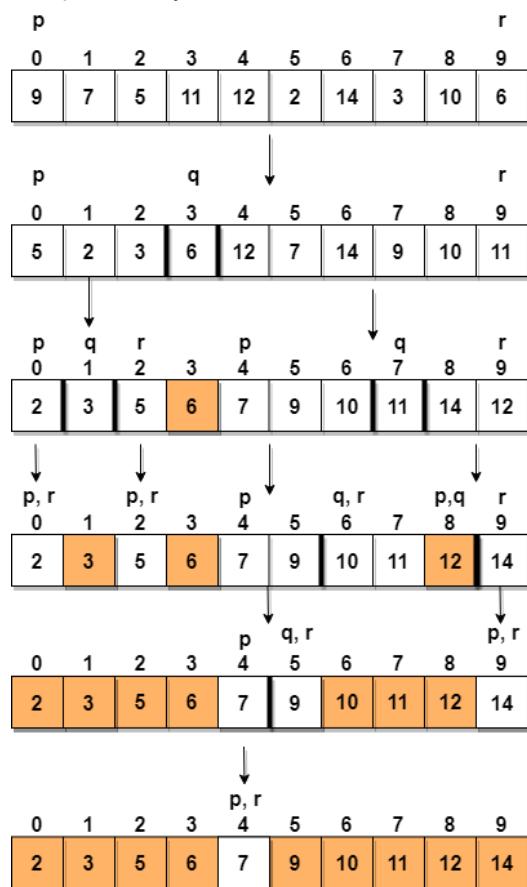
Quick sort is based on the divide-and-conquer approach based on the idea of **choosing one element as a pivot element(normally height index value)** and partitioning the array around it such that:

- Left side of pivot contains all the elements that are less than the pivot element
- Right side contains all elements greater than the pivot

**For example:** In the array {52, 37, 63, 14, 17, 8, 6, 25}, we take 25 as **pivot**. So after the first pass, the list will be changed like this.

{6 8 17 14 25 63 37 52}

Hence after the first pass, pivot will be set at its position, with all the elements **smaller** to it on its left and all the elements **larger** than to its right. Now 6 8 17 14 and 63 37 52 are considered as two separate subarrays, and same recursive logic will be applied on them, and we will keep doing this until the complete array is sorted.



**Step 1** – Choose the highest index value has pivot

**Step 2** – Take two variables to point left and right of the list excluding pivot

**Step 3** – left points to the low index

**Step 4** – right points to the high

**Step 5** – while value at left is less than pivot move right

**Step 6** – while value at right is greater than pivot move left

**Step 7** – if both step 5 and step 6 does not match swap left and right

**Step 8** – if left  $\geq$  right, the point where they met is new pivot

## QUICKSORT

An efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order

Sorts the elements by comparing each element with the pivot

Suitable for small arrays

Works faster for small data sets

Requires minimum space

Not efficient for large arrays

## unstable Sorting

## MERGE SORT

An efficient, general purpose, comparison-based sorting algorithms

Divides the array into two subarrays ( $n/2$ ) again and again until one element is left

Works for any type of array

Works in consistent speed for all datasets

Requires more space

More efficient

## Stable Sorting

### Stable vs Unstable Algorithm

Suppose you need to sort following key-value pairs in the increasing order of keys:

INPUT: (4, 5), (3, 2) (4, 3) (5, 4) (6, 4)

Now, there is two possible solution for the two pairs where the key is the same i.e. (4,5) and (4,3) as shown below:

OUTPUT1: (3, 2), (4, 5), (4, 3), (5, 4), (6, 4)

OUTPUT2: (3, 2), (4, 3), (4, 5), (5, 4), (6, 4)

The sorting algorithm which will produce the first output will be known as stable sorting algorithm because the original order of equal keys are maintained, you can see that (4, 5) comes before (4,3) in the sorted order, which was the original order i.e. in the given input, (4, 5) comes before (4,3) .

On the other hand, the algorithm which produces second output will know as an unstable sorting algorithm because the order of objects with the same key is not maintained in the sorted order. You can see that in the second output, the (4,3) comes before (4,5) which was not the case in the original input.

Some examples of

- **stable algorithms** are Merge Sort, Insertion Sort, Bubble Sort, and Binary Tree Sort.
- **unstable algorithms** are QuickSort, Heap Sort, and Selection sort

If you remember, `Collections.sort()` method from Java Collection framework uses iterative merge sort which is a stable algorithm.

### How much time does it take to retrieve an element if stored in HashMap, Binary tree, and a Linked list? how it change if you have millions of records?

- **HashMap** it takes **O(1)** time,because it uses hashing to get element location.
- **Binary tree** it takes **O(logN)** where N is a number of nodes in the tree
- **LinkedList** it takes **O(n)** time where n is a number of element in the list.

Millions of records don't affect the performance if the data structure is working as expected e.g. HashMap has no or relatively less number of collision or binary tree is balanced. If that's not the case then their performance degrades as a number of records grows.

### ### can we insert elements in middle of LinkedList?

- **You can insert the elements at both the ends and also in the middle of the LinkedList.**
- To insert element in the middle, we need to use `set(int index, E element)` method
- It maintains the **index** also. But here index is LinkedList node index number.

<code>void addFirst(Object o)</code>	<code>Object getFirst()</code>	<code>Object removeFirst()</code>
<code>void addLast(Object o)</code>	<code>Object getLast()</code>	<code>Object removeLast()</code>
<code>Object set(int index, E element)</code>	<code>Object get(int index)</code>	

### ListIterator add, remove is possible?

Using ListIterator we can Add, Remove elemets, and also retrieve elements in Backward direction.

<b>Public ListIterator listIterator()</b> Ex. ListIterator = l.listIterator ()	<code>add(E e)</code> <code>nextIndex()</code> <code>previous()</code> <code>previousIndex()</code> <code>remove()</code> <code>next()</code> <code>set(E e)</code>
<b>boolean hasPrevious()</b> <b>boolean hasNext()</b>	

```

public class Test {
    public static void main(String[] args) throws InterruptedException {
        LinkedList l = new LinkedList<>();
        for (int i = 0; i < 10; i++) {
            l.add(i);
        }

        System.out.println(l);
        ListIterator iterator = l.listIterator();
        while(iterator.hasNext())
        {
            int next = (int) iterator.next();
            System.out.println(next);
            if(next>5)
                l.add(10);
        }
        System.out.println(l);
    }
}
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
0
1
2
3
4
5
6
Exception in thread "main" java.util.ConcurrentModificationException
at java.util.LinkedList$ListItr.checkForComodification(LinkedList.java:966)
at java.util.LinkedList$ListItr.next(LinkedList.java:888)

```

## Coding

### How to Remove Duplicates from Array?

1. Convert Array to List

2. Add List to Set(HashSet/TreeSet) allows Unique values only

```

public class ArrayDuplicates {
    public static void main(String[] args) {
        String arr[] = {"B", "C", "D", "A", "B", "C", "D", "A", "E", "E"};
        List list = Arrays.asList(arr);

        HashSet h = new HashSet(list);
        System.out.println("HashSet : "+h);

        TreeSet t = new TreeSet(list);
        System.out.println("TreeSet : "+t);
    }
}
HashSet: [A, B, C, D, E]
TreeSet: [A, B, C, D, E]

```

### How do you get the last digit of an integer?

By using **modulus operator(%)**, **number % 10** returns the last digit of the number, for example,

- 2345%10 will return 5
- 567%10 will return 7.

Similarly, **division operator(/)** can be used to get rid of the last digit of a number e.g.

- 2345/10 will give 234
- 567/10 will return 56.

This is an important technique to know and useful to solve problems like **number palindrome or reversing numbers**

```
public static int reverse(int number){ //say 12345
    int reverse = 0;
    int remainder = 0;
    int i=1;
    do{
        remainder = number%10; //To Get last Number
        reverse = reverse*10 + remainder; //To add places, 10, 100, 1000
        number = number/10; // To remove Last Number
        /*reverse = reverse*10 + remainder;
        *      5      = 0*10+5
        *      54     = 5*10 = 50+4 = 54,
        *      543    = 54*10 = 540+3 = 543
        * */
    }

    System.out.println(i+"---> remainder: "+remainder+", reverse: "+reverse+", number: "+number);
    i++;
}while(number > 0);

return reverse;
}
```

### How to Find Missing Number on Integer Array of 1 to 100

#### METHOD 1(Use sum formula)

Algorithm:

1. Get the sum of numbers  
$$\text{total} = n*(n+1)/2$$
2. Subtract all the numbers from sum and you will get the missing number.

```
class Main
{
    // Function to find missing number
    static int getMissingNo (int a[], int n)
    {
        int i, total;
        total = (n+1)*(n+2)/2;
        for ( i = 0; i < n; i++)
            total -= a[i];
        return total;
    }

    /* program to test above function */
    public static void main(String args[])
    {
        int a[] = {1,2,4,5,6};
        int miss = getMissingNo(a,5);
        System.out.println(miss);
    }
}
```

### Write code to check a String is palindrome or not? ([solution](#))

A palindrome is those String whose reverse is equal to the original. This can be done by using either **StringBuffer reverse()** method or by technique demonstrated in the solution here.

### Write a method which will remove any given character from a String? ([solution](#))

you can remove a given character from String by converting it into a **char[]** array and compare give char with each char of array, remove that & append remaining.

```
public class Test {
```

```

public static String remove(String word, char unwanted) {
    StringBuilder sb = new StringBuilder();
    char[] letters = word.toCharArray();

    for (char c : letters) {
        if (c != unwanted) {
            sb.append(c);
        }
    }
    return sb.toString();
}

public static void main(String[] args) throws InterruptedException {
    System.out.println(remove("satya kaveti", 'a'));
}
}
sty kveti

```

### Print all permutation of String? ([solution](#))

for a String of 3 characters like "xyz" has 6 possible permutations, xyz, xzy, yxz, yzx, zxy, zyx

```

public class Permutation {
    public static void permutation(String str) {
        permutation("", str);
    }
    private static void permutation(String prefix, String str) {
        int n = str.length();
        if (n == 0) System.out.println(prefix);
        else {
            for (int i = 0; i < n; i++)
                permutation(prefix + str.charAt(i), str.substring(0, i) + str.substring(i+1, n));
        }
    }
    public static void main(String args[]) {
        permutation("XYZ");
    }
}
XYZ
XZY
YXZ
YZX
ZXY
ZYX

```

### How to check if two String Are Anagram? ([solution](#))

two String are called anagram, if they contain same characters but on different order

e.g. **army** and **mary**, **stop** and **pots** etc

- Convert Two Strings into toCharArray
- Sort the Two Arrays
- Check both arrays equal or not

```

public class Anagram {
    public static boolean Check(String word1, String word2) {
        char[] charFromword1 = word1.toCharArray();
        char[] charFromword2 = word2.toCharArray();
        Arrays.sort(charFromword1);
        Arrays.sort(charFromword2);
        return Arrays.equals(charFromword1, charFromword2);
    }
    public static void main(String args[]) {
        System.out.println(Check("stop", "pots"));
        System.out.println(Check("army", "mary"));
    }
}
true
true

```

## Java Program to print Fibonacci Series

Fibonacci number is sum of previous two Fibonacci numbers  $fn = fn-1 + fn-2$ . first 10 Fibonacci numbers are 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

```
public class Permutation {  
  
    public static void main(String args[]) {  
  
        // input to print Fibonacci series upto how many numbers  
        int number = 10;  
  
        System.out.println("Fibonacci series upto " + number + " numbers : ");  
        // printing Fibonacci series upto number  
        for (int i = 1; i <= number; i++) {  
            System.out.print(fibonacci(i) + " ");  
        }  
    }  
  
    public static int fibonacci(int number) {  
        if (number == 1 || number == 2) {  
            return 1;  
        }  
  
        return fibonacci(number - 1) + fibonacci(number - 2); // tail recursion  
    }  
}  
Fibonacci series upto 10 numbers :  
1 1 2 3 5 8 13 21 34 55
```

## How to find the factorial of a number in Java

the factorial of a number is calculated by formula **number\*(number -1)** till zero and since the **value of factorial zero is 1**.

```
public class Permutation {  
    public static int fact(int number){  
        int result = 1;  
        while(number != 0){  
            result = result*number;  
            number--;  
        }  
  
        return result;  
    }  
  
    public static void main(String args[]) {  
        int res = fact(5);  
        System.out.println("Fianl Factoril is : "+res);  
    }  
}  
Fianl Factoril is : 120
```

## Java program Armstrong numbers in the range of 0 and 9999.

An Armstrong number is a number such that the sum of its digits raised to the third power is equal to the number itself. For example, 153 is an Armstrong number, since  $1^{**}3 + 5^{**}3 + 3^{**}3 = 153$ .

```
public class Permutation {  
  
    public static void main(String args[]) {  
  
        int count = 999;  
        int index = 0;  
        for (int i = 0; i < count; i++) {  
            if (isArmstrongNumber(i)) {  
                System.out.printf("Armstrong number %d: %d %n", index, i);  
            }  
        }  
    }  
}
```

```

        index++;
    }

}

/** 
 * Java Method to check if given number is Armstrong Number or not
 *
 * @param number
 * @return true, if Armstrong number, false otherwise.
 */
public static boolean isArmstrongNumber(int number) {
    int sum = 0;
    int copyOfInput = number;
    while (copyOfInput != 0) {
        int lastDigit = copyOfInput % 10;
        sum += (lastDigit * lastDigit * lastDigit);
        copyOfInput /= 10;
    }

    if (sum == number) {
        return true;
    }
    return false;
}
}

Armstrong number 0: 0
Armstrong number 1: 1
Armstrong number 2: 153
Armstrong number 3: 370
Armstrong number 4: 371
Armstrong number 5: 407

```

### Java Program to print 1 to 100 without using loop

```

public class Permutation {

    public static void usingRecursion(int number){
        if(number > 1){
            usingRecursion(number-1);
        }
        System.out.println(number);
    }

    public static void main(String args[]) {
        usingRecursion(20);
    }
}

```

### Links

<https://javarevisited.blogspot.com/2011/06/top-programming-interview-questions.html>

<https://javarevisited.blogspot.com/2017/07/top-50-java-programs-from-coding-Interviews.html>

<https://javarevisited.blogspot.com/search/label/Coding%20Interview%20Question?max-results=100>

# Architecture

The objective of Escheatment is to identify accounts with funds and no customer-initiated activity for some specified period.

## Technologies

The Escheatment application is developed using the J2EE architecture. In order to evaluate customer activity, Escheatment must check many sources. Most of these sources exist in IDW. Three do not. These are loaded and maintained in Escheatment proprietary database (ESH2\_DATA & ESH2\_DATA) on IDW.

## Who uses it

This application is used by Escheatment **Specialists in the Taxation Department**.

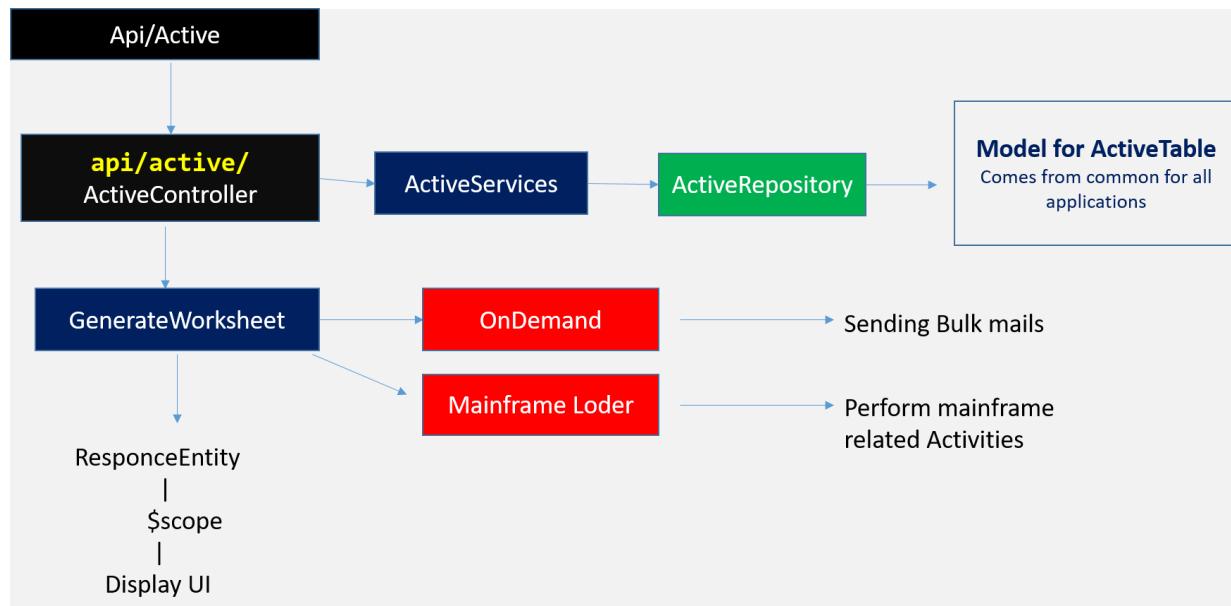
## How it Works

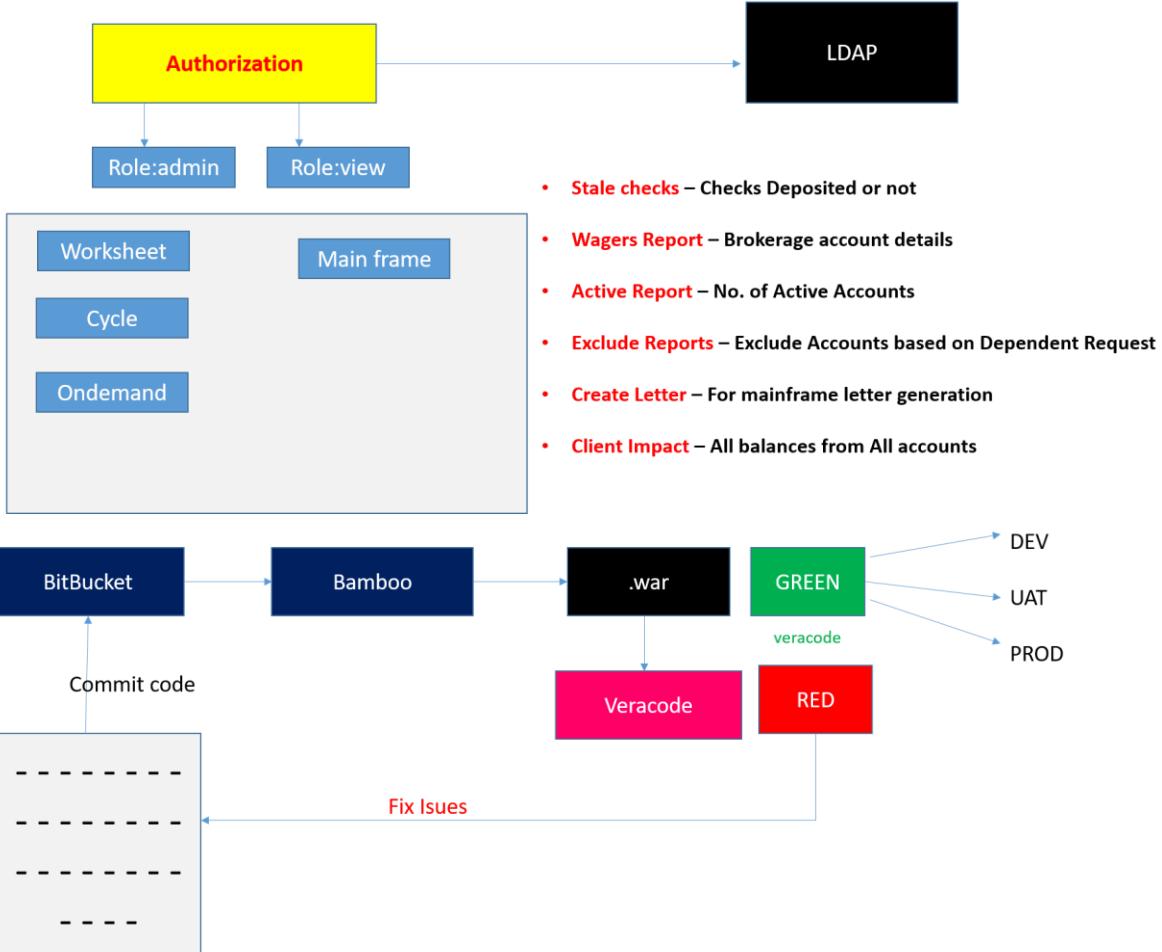
The objective of Escheatment is to identify accounts with funds and no customer-initiated activity for some specified period.

- **Individual state laws** require that assets in dormant accounts be turned over to the state.
- **The system will scan all US Schwab customer accounts** for customer-originated activity. Records that have been **dormant for a period greater than or equal to the State dormancy period**
- Those Accounts will be handed over to the respective state.

## What we Achieve

- **Elimination of extensive manual processing to confirm and/or disprove the status of accounts**, i.e., active or inactive, identified by the current escheat system
- Presentation of account data in a format that facilitates report preparation and processing





### How windows credentials loads to browser in organization

Active Directory is an official Microsoft technology which makes use of LDAP and others:

### Compare Http Headers for LDAP authorization

nope

### Repository for manull Quiries

### AngularJS Response formation

Object, json, produces JSON

# SQL - Interview Questions

## Q1. What is the difference between SQL and MySQL?

### SQL vs MySQL

SQL	MySQL
SQL is a standard language which stands for Structured Query Language based on the English language	MySQL is a database management system.
SQL is the core of the relational database which is used for accessing and managing database	MySQL is an RDMS (Relational Database Management System) such as SQL Server, Informix etc.

## Q2. What are the different subsets of SQL?

- Data Definition Language (DDL) – It allows you to perform various operations on the database such as CREATE, ALTER, and DELETE objects.
- Data Manipulation Language(DML) – It allows you to access and manipulate data. It helps you to insert, update, delete and retrieve data from the database.
- Data Control Language(DCL) – It allows you to control access to the database. Example – Grant, Revoke access permissions.

## Q3. What do you mean by DBMS? What are its different types?

A Database Management System (**DBMS**) is a software application that interacts with the user, applications, and the database itself to capture and analyze data. A database is a structured collection of data.

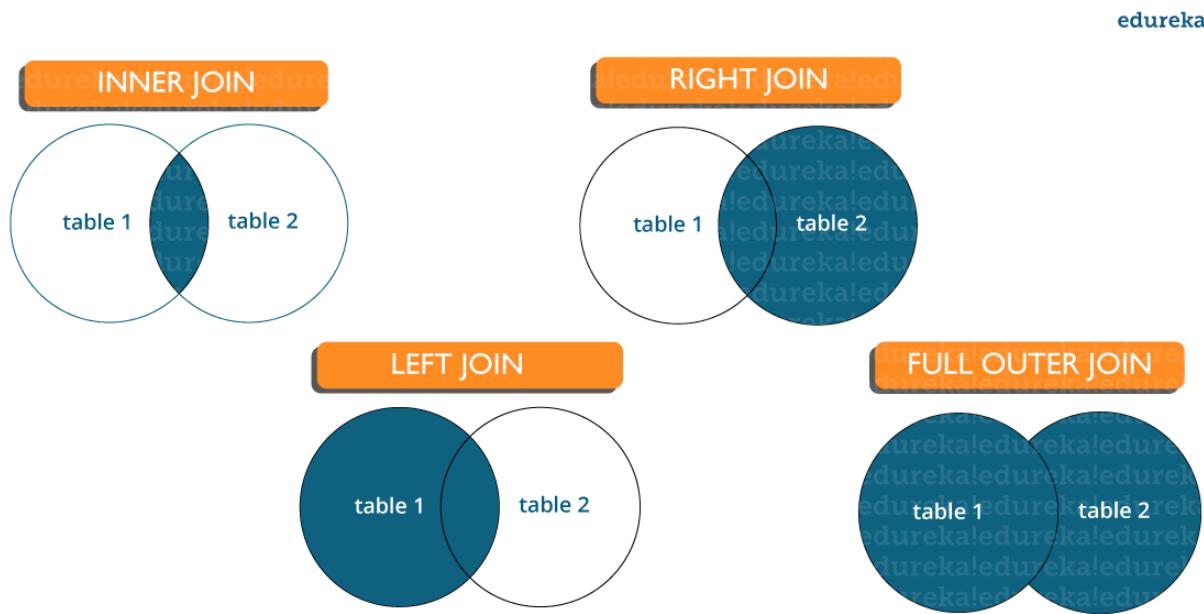
A DBMS allows a user to interact with the database. The data stored in the database can be modified, retrieved, and deleted and can be of any type like strings, numbers, images, etc.

There are two types of DBMS:

- *Relational Database Management System*: The data is stored in relations (tables). Example – MySQL.
- *Non-Relational Database Management System*: There is no concept of relations, tuples and attributes. Example – MongoDB

## Q5. What are joins in SQL?

A JOIN clause is used to combine rows from two or more tables, based on a related column between them. It is used to merge two tables or retrieve data from there. There are 4 types of joins, as you can refer to below:



- **Inner join:** [Inner Join in SQL](#) is the most common type of join. It is used to return all the rows from multiple tables where the join condition is satisfied.
- **Left Join:** Left Join in SQL is used to return all the rows from the left table but only the matching rows from the right table where the join condition is fulfilled.
- **Right Join:** Right Join in SQL is used to return all the rows from the right table but only the matching rows from the left table where the join condition is fulfilled.
- **Full Join:** Full join returns all the records when there is a match in any of the tables. Therefore, it returns all the rows from the left-hand side table and all the rows from the right-hand side table.

## Q6. What is the difference between CHAR and VARCHAR2 datatype in SQL?

Both **Char** and **Varchar2(VaribaleCHARacter)** are used for characters datatype but varchar2 is used for character strings of variable length whereas Char is used for strings of fixed length.

For example, `char(10)` can only store 10 characters and will not be able to store a string of any other length – cat, rat – it only store String length exactly 10.

whereas `varchar2(10)` can store any length i.e 6,8,2 in this variable.

### What is a Primary key?

- A Primary key in SQL is a column (or collection of columns) or a set of columns that uniquely identifies each row in the table.
- Uniquely identifies a single row in the table
- Null values not allowed
- Example- In the Student table, Stu\_ID is the primary key.

### What are Constraints?

Constraints in SQL are used to specify the limit on the data type of the table. It can be specified while creating or altering the table statement. The sample of constraints are:

- NOT NULL
- CHECK
- DEFAULT
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY

```
CREATE TABLE `patent` (
    `patent_id` INT(11) NOT NULL AUTO_INCREMENT,
    `name` VARCHAR(50) NULL DEFAULT NULL COLLATE 'utf8mb4_general_ci',
    `age` VARCHAR(50) NULL DEFAULT NULL COLLATE 'utf8mb4_general_ci',
    `address` VARCHAR(50) NULL DEFAULT NULL COLLATE 'utf8mb4_general_ci',
    `trial_id` INT(11) NULL DEFAULT NULL,
    PRIMARY KEY (`patent_id`) USING BTREE,
    INDEX `FK_patent_trial` (`trial_id`) USING BTREE,
    CONSTRAINT `FK_patent_trial` FOREIGN KEY (`trial_id`) REFERENCES `trialapp`.`trial` (`trial_id`)
    ON UPDATE CASCADE ON DELETE CASCADE
)
COLLATE='utf8mb4_general_ci'
ENGINE
```



CREATE\_TABLES.SQL

### What is the difference between DELETE and TRUNCATE statements?

DELETE	TRUNCATE
Delete is used to delete SINGLE/Multiple Rows in a table.	Truncate is used to delete all the rows from a table.
You can rollback data after using delete statement.	You cannot rollback data.
It is a DML command.	It is a DDL command.
It is slower than truncate statement.	It is faster.

### What is a Unique key?

- Uniquely identifies a single row in the table.
- Multiple values allowed per table.
- Null values allowed.

### **Q11. What is a Foreign key in SQL?**

- Foreign key maintains referential integrity by enforcing a link between the data in two tables.
- The foreign key in the child table references the primary key in the parent table.
- The foreign key constraint prevents actions that would destroy links between the child and parent tables.

### **Q12. What do you mean by data integrity?**

Data Integrity defines the accuracy as well as the consistency of the data stored in a database. It also defines integrity constraints to enforce business rules on the data when it is entered into an application or a database.

### **What is an Index? Explain different types of index in SQL?**

An index refers to a performance tuning method of allowing faster retrieval of records from the table. An index creates an entry for each value and hence it will be faster to retrieve data. There are three types of index in SQL namely:

#### **Unique Index:**

- This index does not allow the field to have duplicate values if the column is unique indexed. If a primary key is defined, a unique index can be applied automatically.

#### **Clustered Index:**

- This index reorders the physical order of the table and searches based on the basis of key values. Each table can only have one clustered index.

#### **Non-Clustered Index:**

- Non-Clustered Index does not alter the physical order of the table and maintains a logical order of the data. Each table can have many nonclustered indexes.

### **Difference between Clustered and Non-clustered index**

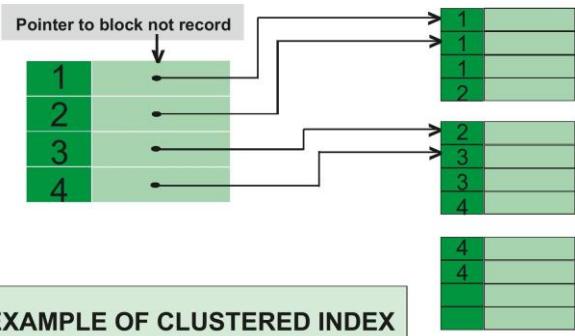
#### **1. Clustered Index :**

Clustered index is created only when both the following conditions satisfy –

- The data or file, that you are moving into secondary memory should be in sequential or sorted order.
- There should be non key value, meaning it can have repeated values.

Whenever you apply clustered indexing in a table, it will perform sorting in that table only. You can create only one clustered index in a table like primary key. Clustered index is as same as dictionary where the data is arranged by alphabetical order.

In clustered index, index contains pointer to block but not direct data.



### Example of Clustered Index –

If you apply primary key to any column, then automatically it will become clustered index.

```
create table Student
( Roll_No int primary key,
Name varchar(50),
Gender varchar(30),
Mob_No bigint );

insert into Student
values (4, 'ankita', 'female', 9876543210 );

insert into Student
values (3, 'anita', 'female', 9675432890 );

insert into Student
values (5, 'mahima', 'female', 8976453201 );
```

In this example, Roll no is a primary key, it will automatically act as a clustered index.

The output of this code will produce in increasing order of roll no.

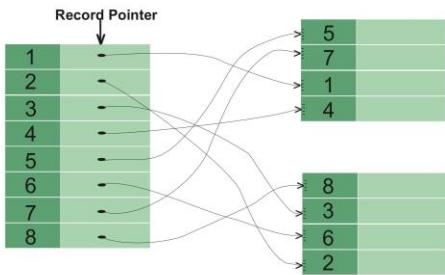
Roll_No	Name	Gender	Mob_No
3	anita	female	9675432890
4	ankita	female	9876543210
5	mahima	female	8976453201

You can have only one clustered index in one table, but you can have one clustered index on multiple columns, and that type of index is called composite index.

### 2. Non-clustered Index :

The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.

In non-clustered index, index contains the pointer to data.



### EXAMPLE OF NON-CLUSTERED INDEX

#### Example of Non-clustered Index –

```
create table Student
( Roll_No int primary key,
Name varchar(50),
Gender varchar(30),
Mob_No bigint );

insert into Student
values (4, 'afzal', 'male', 9876543210 );

insert into Student
values (3, 'sudhir', 'male', 9675432890 );

insert into Student
values (5, 'zoya', 'female', 8976453201 );

create nonclustered index NIX_FTE_Name
on Student (Name ASC);
```

Here, roll no is a primary key, hence there is automatically a clustered index.

If we want to apply non-clustered index in NAME column (in ascending order), then the new table will be created for that column.

Output before applying non-clustered index:

Roll_No	Name	Gender	Mob_No
3	sudhir	male	9675432890
4	afzal	male	9876543210
5	zoya	female	8976453201

Output after applying non-clustered index:

Name	Row address
Afzal	3452
Sudhir	5643
zoya	9876

Row address is used because, if someone wants to search the data for sudhir, then by using the row address he/she will directly go to that row address and can fetch the data directly.

### Difference between Clustered and Non-clustered index :

CLUSTERED INDEX	NON-CLUSTERED INDEX
Clustered index is faster.	Non-clustered index is slower.
Clustered index requires less memory for operations.	Non-Clustered index requires more memory for operations.
In clustered index, index is the main data.	In Non-Clustered index, index is the copy of data.
A table can have only one clustered index.	A table can have multiple non-clustered index.
Clustered index has inherent ability of storing data on the disk.	Non-Clustered index does not have inherent ability of storing data on the disk.
Clustered index store pointers to block not data.	Non-Clustered index store both value and a pointer to actual row that holds data.
In Clustered index leaf nodes are actual data itself.	In Non-Clustered index leaf nodes are not the actual data itself rather they only contains included columns.
In Clustered index, Clustered key defines order of data within table.	In Non-Clustered index, index key defines order of data within index.
A Clustered index is a type of index in which table records are physically reordered to match the index.	A Non-Clustered index is a special type of index in which logical order of index does not match physical stored order of the rows on disk.

### What do you understand by query optimization?

- The phase that identifies a plan for evaluation query which has the least estimated cost is known as query optimization.
- The advantages of query optimization are as follows:
- The output is provided faster
- A larger number of queries can be executed in less time
- Reduces time and space complexity

### What is Normalization and what are the advantages of it?

Normalization in SQL is the process of organizing data to avoid duplication and redundancy. Some of the advantages are:

- Better Database organization
- More Tables with smaller rows
- Efficient data access
- Greater Flexibility for Queries
- Quickly find the information
- Easier to implement Security
- Allows easy modification
- Reduction of redundant and duplicate data
- More Compact Database
- Ensure Consistent data after modification

**First Normal Form (1NF) – No Multiple values** A relation will be 1NF if it contains an atomic value. It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute. First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**Example:** Relation EMPLOYEE is not in 1NF because of multi-valued attribute **EMP\_PHONE**.

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

### Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

**Example:** Let's assume, a school can store the data of **teachers** and the **subjects** they teach. In a school, a teacher can teach more than one subject.

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute **TEACHER\_AGE** is dependent on **TEACHER\_ID** which is a proper. But Subject is no way related to Teacher\_ID, which is same functional group. So we need to separate Subject data to another table to achieve 2NF, we decompose it into two tables:

TEACHER_ID	TEACHER AGE
25	30
47	35
83	38

**TEACHER SUBJECT table:**

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

### Third Normal Form

When a table is in the Second Normal Form and has no transitive dependency, then it is in the Third Normal Form. **Dependent solely on the primary key and no other non-key (supporting) column value.**

Suppose a company wants to store the complete address of each employee, they create a table named `employee_details` that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

**Super keys:** {emp\_id}, {emp\_id, emp\_name}, {emp\_id, emp\_name, emp\_zip}...so on

**Candidate Keys:** {emp\_id}

**Non-prime attributes:** all attributes except emp\_id are non-prime as they are not part of any candidate keys.

Here, emp\_state, emp\_city & emp\_district dependent on emp\_zip. And, emp\_zip is dependent on emp\_id that makes non-prime attributes (emp\_state, emp\_city & emp\_district) transitively dependent on super key (emp\_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

**employee table:**

emp_id	emp_name	emp_zip
--------	----------	---------

1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

#### **employee\_zip table:**

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

#### **What is the difference between DROP and TRUNCATE commands?**

DROP command removes a table and it cannot be rolled back from the database whereas TRUNCATE command removes all the rows from the table. Table Still exist.

#### **What do you mean by "Trigger" in SQL?**

Trigger in SQL is are a special type of stored procedures that are defined to execute automatically in place or after data modifications. It allows you to execute a batch of code when an insert, update or any other query is executed against a specific table. We have implemented version history using Triggers.

Example: If a Sponsor added to Our System, we need to maintain Audit log like – who added, when,etc

```

CREATE TRIGGER TR_SPONSOR_AUDIT_INSERT
    AFTER INSERT ON TL_SPONSOR
    REFERENCING NEW AS NEW
    FOR EACH ROW

BEGIN
    INSERT INTO TL_SPONSOR_AUDIT(AUDIT_SPONSOR_ID,SPONSOR_ID, NAME, CODE,ALIAS, PROVISIONING_REF,
ECLINICAL_SPONSOR_ID,
    USER_NAME, USER_ACTION, MODIFIED_DATE ) VALUES
    (SPONSOR_AUDIT_SEQ.NEXTVAL, NEW.SPONSOR_ID, NEW.NAME, NEW.CODE, NEW.ALIAS, NEW.PROVISIONING_REF,
NEW.ECLINICAL_SPONSOR_ID,
    NEW.USER_NAME, 'I', NEW.MODIFIED_DATE);
END
;

```

#### **What is the difference between cross join and natural join?**

The cross join produces the cross product or Cartesian product of two tables whereas the natural join is based on all the columns having the same name and data types in both the tables.

### **Q23. What is the ACID property in a database?**

ACID stands for Atomicity, Consistency, Isolation, Durability. It is used to ensure that the data transactions are processed reliably in a database system.

- **Atomicity:** Atomicity refers to the transactions that are completely done or failed where transaction refers to a single logical operation of a data. It means if one part of any transaction fails, the entire transaction fails and the database state is left unchanged.
- **Consistency:** Consistency ensures that the data must meet all the validation rules. In simple words, you can say that your transaction never leaves the database without completing its state.
- **Isolation:** The main goal of isolation is concurrency control.
- **Durability:** Durability means that if a transaction has been committed, it will occur whatever may come in between such as power loss, crash or any sort of error.

### **Write a SQL query to get the third-highest salary of an employee from employee\_table?**

```
SELECT TOP 1 salary  
FROM(  
SELECT TOP 3 salary  
FROM employee_table  
ORDER BY salary DESC) AS emp  
ORDER BY salary ASC;
```

### **What is the main difference between 'BETWEEN' and 'IN' condition operators?**

**BETWEEN** operator is used to display rows based on a range of values in a row whereas the **IN** condition operator is used to check for values contained in a specific set of values.

#### **Example of BETWEEN:**

```
SELECT * FROM Students where ROLL_NO BETWEEN 10 AND 50;
```

#### **Example of IN:**

```
SELECT * FROM students where ROLL_NO IN (8,15,25);
```

### **What is the difference between 'HAVING' CLAUSE and a 'WHERE' CLAUSE?**

HAVING clause can be used only with **SELECT** statement. It is usually used in a GROUP BY clause and whenever GROUP BY is not used, HAVING behaves like a WHERE clause.

Having Clause is only used with the GROUP BY function in a query whereas WHERE Clause is applied to each row before they are a part of the GROUP BY function in a query.

### **List the ways in which Dynamic SQL can be executed?**

Following are the ways in which dynamic SQL can be executed:

- Write a query with parameters.
- Using EXEC.
- Using sp\_executesql.

### **How can you fetch common records from two tables?**

You can fetch common records from two tables using INTERSECT. For example:

### **What is an ALIAS command?**

ALIAS command in SQL is the name that can be given to any table or a column. This alias name can be referred in WHERE clause to identify a particular table or a column.

```
Select emp.empID, dept.Result from employee emp, department as dept where emp.empID=dept.empID
```

### **How can you fetch alternate records from a table?**

You can fetch alternate records i.e both odd and even row numbers. For example- To display even numbers, use the following command:

```
Select studentId from (Select rowno, studentId from student) where mod(rowno,2)=1
```

### **How can you fetch first 5 characters of the string?**

There are a lot of ways to fetch characters from a string. For example:

```
Select SUBSTRING(StudentName,1,5) as studentname from student
```

### **What is a View? What are Views used for?**

A view is a virtual table which consists of a subset of data contained in a table. Since views are not present, it takes less space to store. View can have data of one or more tables combined and it depends on the relationship. A view refers to a logical snapshot based on a table or another view. It is used for the following reasons:

- Restricting access to data.
- Making complex queries simple.
- Ensuring data independence.
- Providing different views of same data.

### **What is a Stored Procedure?**

A Stored Procedure is a function which consists of many SQL statements to access the database system. Several SQL statements are consolidated into a stored procedure and execute them whenever and wherever required which saves time and avoid writing code again and again.

Mostly used for DB backup & SQL scripts for getting data from multiple databases. Mostly use for Re-Use purpose.

**Advantages:** A Stored Procedure can be used as a modular programming which means create once, store and call for several times whenever it is required. This supports faster execution. It also reduces network traffic and provides better security to the data.

**Disadvantage:** The only disadvantage of Stored Procedure is that it can be executed only in the database and utilizes more memory in the database server.

## What is Auto Increment in SQL?

Autoincrement keyword allows the user to create a unique number to get generated whenever a new record is inserted into the table.

This keyword is usually required whenever PRIMARY KEY in SQL is used.

AUTO INCREMENT keyword can be used in Oracle and IDENTITY keyword can be used in SQL SERVER.

```
CREATE TABLE `patent` (
    `patent_id` INT(11) NOT NULL AUTO_INCREMENT,
    `name` VARCHAR(50) NULL,
    `age` VARCHAR(50) NULL,
    `trial_id` INT(11) NULL DEFAULT NULL,
    PRIMARY KEY (`patent_id`),
    INDEX `FK_patent_trial`(`trial_id`),
    CONSTRAINT `FK_patent_trial` FOREIGN KEY (`trial_id`) REFERENCES `trialapp`.`trial`(`trial_id`)
    ON UPDATE CASCADE ON DELETE CASCADE
)
```

## What is a Datawarehouse?

Datawarehouse refers to a central repository of data where the data is assembled from multiple sources of information. Those data are consolidated, transformed, and made available for the mining as well as online processing. Warehouse data also have a subset of data called Data Marts.

## Delete Duplicates From a Table in SQL Server

To delete the duplicate rows from the table in SQL Server, you follow these steps:

- Find duplicate rows using GROUP BY clause or ROW NUMBER() function.
- Use DELETE statement to remove the duplicate rows.

## MongoDB – Interview Questions

<https://www.interviewbit.com/mongodb-interview-questions/>

<https://github.com/smlcodes/jtutorials/blob/main/MongoDB%20Interview%20Questions.md>

# 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 username or password for connecting to that database.

For Doing that,

1. **JNDI Url is configured in Server side, we need to just place that Url in Context.xml with some Resource name=""**
2. **Configure Resource name in web.xml**

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 data source 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

```
public Connection getConnection() {
    try {
        InitialContext context = new InitialContext();
        DataSource ds = (DataSource) context.lookup("jdbc:mysql://localhost:3306/mkyongdb");
        Connection conn = ds.getConnection();
    } catch (SQLException ex) {
    }
    return conn;
}
```

## 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 types of 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 - takes 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 execute(), executeQuery, executeUpdate in JDBC?

- **ResultSet executeQuery()** : it can be used for SELECT query.
- **int executeUpdate()** : it can be used to change/update table.  

Returns 1 if → success  
Returns 0 if → Failure
- **boolean execute()** : it can be used for SELECT / NON- SELECT / any kind of SQL Query.
  - It returns **TRUE** on SELECT queries. we can getResultSet by calling below method  

`ResultSet rs = statement.getResultSet();`
  - It returns **FALSE** on NON-SELECT queries. we can get Int value by calling below method  

`int i = statement.getUpdateCount();`

## 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 – because it creates Connection for each time we request , which can give slow response 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>

# Hibernate

## EmployeeBo.java

```
public class EmployeeBo {  
    private int eid;  
    private String name;  
    private String address;  
    //setters / getters  
}
```

## Example: EmployeeBo.hbm.xml

```
<hibernate-mapping>  
    <class name="bo.EmployeeBo" table="employee">  
        <id name="eid" column="eid">  
            <generator class="assigned" />  
        </id>  
        <property name="name" column="name" />  
        <property name="address" column="address" />  
    </class>  
</hibernate-mapping>
```

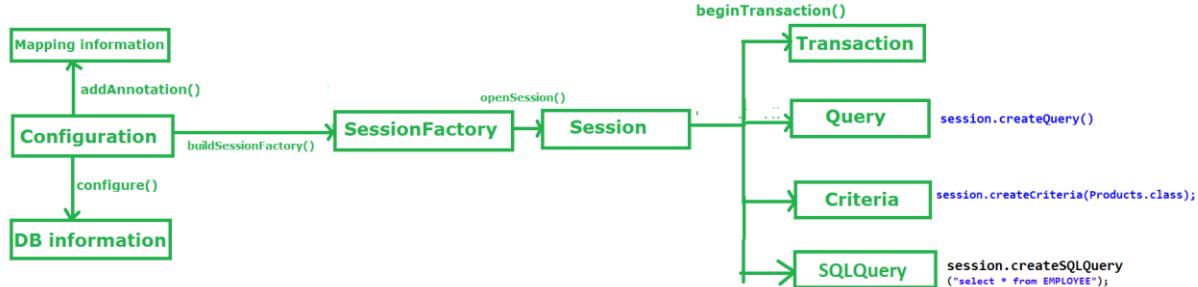
```
<hibernate-configuration>  
    <session-factory>  
        <! -- Database connection settings -->  
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>  
        <property name="connection.url">jdbc:mysql://localhost:3306/mydb</property>  
        <property name="connection.username">root</property>  
        <property name="connection.password">root</property>  
        <property name="connection.pool_size">1</property>  
  
        <!-- Hibernate Properties -->  
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>  
        <property name="show_sql">true</property>  
        <property name="hbm2ddl.auto">validate</property>  
  
        <!-- Mapping file name(s)-->  
        <mapping resource="res/employee.hbm.xml"/>  
    </session-factory>  
</hibernate-configuration>
```

```
public class EmployeeSave {  
    public static void main(String[] args) {  
        Configuration cfg = new Configuration();  
        cfg.configure("hibernate.cfg.xml");  
  
        SessionFactory factory = cfg.buildSessionFactory();  
        Session session = factory.openSession();  
  
        EmployeeBo bo = new EmployeeBo();  
        bo.setEid(5);  
        bo.setName("DILEEP");  
        bo.setAddress("BANGLORE");  
  
        Transaction tx = session.beginTransaction();  
        session.save(bo);  
    }  
}
```

```

        System.out.println("Employee Data saved successfully.....!!!");
        tx.commit();
        session.close();
        factory.close();
    }
}

```



### Session class methods

```

Select
Object get(Class , Serializable id)
Object load(Class, Serializable id)

Insert
Serializable save(Object object)
void persist(Object object)
void saveOrUpdate(Object object)

Update
Object merge(Object object)
void update(Object object)

Delete
void delete(Object object)

Clear
void evict(Object object) : Remove this instance from the session cache.
void clear() : Completely clear the session.

Other
boolean isDirty() : Does this session contain any changes which must be synchronized with the database? In other words, would any DML operations be executed if we flushed this session?

void refresh(Object object)
Re-read the state of the given instance from the underlying database.

```

### 1.Table per subclass hierarchy

#### Vehicle.hbm.xml

```

<hibernate-mapping>
    <class name="inheritance.Vehicle" table="vehicle">
        <id name="vid" column="vid"></id>
        <discriminator column="DISC" type="string"/>
        <property name="price" column="price"></property>

        <subclass name="inheritance.Bike" discriminator-value="BIKE_DISC">
            <property name="biketype" column="biketype"></property>
        </subclass>

        <subclass name="inheritance.Car" discriminator-value="CAR_DISC">
            <property name="cartype" column="cartype"></property>
        </subclass>
    </class>
</hibernate-mapping>

```

```
C:\Windows\system32\cmd.exe
mysql> select * from vehicle;
+---+---+---+---+
| vid | price | DISC | biketype | cartype |
+---+---+---+---+
| 101 | 50000 | BIKE_DISC | HONDA | NULL |
| 102 | 60000 | CAR_DISC | NULL | MARUTHI |
+---+---+---+---+
2 rows in set <0.00 sec>

mysql> select * from bike;
Empty set <0.00 sec>

mysql> select * from car;
Empty set <0.00 sec>
```

## 2.Table per joined-subclass hierarchy

### Vehicle.hbm.xml

```
<hibernate-mapping>
    <class name="inheritance.Vehicle" table="vehicle">
        <id name="vid" column="vid"></id>
        <property name="price" column="price"></property>

        <joined-subclass name="inheritance.Bike" table="bike">
            <key column="BIKE_KEY" />
            <property name="biketype" column="type"></property>
        </joined-subclass>

        <joined-subclass name="inheritance.Car" table="car">
            <key column="CAR_KEY" />
            <property name="cartype" column="type"></property>
        </joined-subclass>
    </class>
</hibernate-mapping>
```

```
C:\Windows\system32\cmd.exe
mysql> select * from vehicle;
+---+---+---+---+
| vid | price | DISC | biketype | cartype |
+---+---+---+---+
| 101 | 50000 | BIKE_DISC | HONDA | NULL |
| 102 | 60000 | CAR_DISC | NULL | MARUTHI |
+---+---+---+---+
2 rows in set <0.00 sec>

mysql> select * from bike;
+---+---+
| type | BIKE_KEY |
+---+---+
| HONDA | 101 |
+---+---+
1 row in set <0.00 sec>

mysql> select * from car;
+---+---+
| type | CAR_KEY |
+---+---+
| MARUTHI | 102 |
+---+---+
1 row in set <0.00 sec>
```

## 3.Table per union-subclass class hierarchy

### Vehicle.hbm.xml

```
<hibernate-mapping>
    <class name="inheritance.Vehicle" table="vehicle">
        <id name="vid" column="vid"></id>
        <property name="price" column="price"></property>

        <union-subclass name="inheritance.Bike" table="bike">
            <property name="biketype" column="type"></property>
        </union-subclass>

        <union-subclass name="inheritance.Car" table="car">
            <property name="cartype" column="type"></property>
        </union-subclass>
    </class>
</hibernate-mapping>
```

```

mysql> select * from vehicle;
Empty set (0.00 sec)

mysql> select * from bike;
+-----+-----+
| type | vid  | price |
+-----+-----+
| HONDA | 101 | 50000 |
+-----+-----+
1 row in set (0.00 sec)

mysql> select * from car;
+-----+-----+
| type | vid  | price |
+-----+-----+
| MARUTHI | 102 | 600000 |
+-----+-----+
1 row in set (0.00 sec)

```

Data Saved in **Parent** Class Table

```

<class name="Vehicle" table="vehicle">
  <subclass></subclass>
  <subclass></subclass>
</class>

```

Data Saved in **BOTH Parent & Child** Class Tables

```

<class name="Vehicle" table="vehicle">
  <joined-subclass></joined-subclass>
  <joined-subclass></joined-subclass>
</class>

```

Data Saved in **Child** Class Table

```

<class name="Vehicle" table="vehicle">
  <union-subclass></union-subclass>
  <union-subclass></union-subclass>
</class>

```

If we want to select a **Complete Object** from the database, we use POJO class reference in place of `*` while constructing the query

```

// In SQL
sql> select * from Employee
Note: Employee is the table name.

// In HQL
hql> select s from EmployeeBo s
[ or ]
from EmployeeBo s
Note: here s is the reference of EmployeeBo

```

If we want to load the **Partial Object** from the database that is only selective properties of an objects, then we need to replace column names with POJO class variable names

```

// In SQL
sql> select eid,name,address from Employee
Note: eid,name,address are the columns of Employee the table.

// In HQL
hql> select s.eid,s.name,s.address from EmployeeBo s

```

Query Interface - `session.createQuery()`

- `public int executeUpdate()` -is used to execute **the update or delete query**.
- `public List list()` -returns the result of the relation as a list.

```

Query qry = session.createQuery("--- HQL command ---");
List l = qry.list();
Iterator it = l.iterator();
while(it.hasNext())
{
  Object o = it.next();
  EmployeeBo s = (EmployeeBo)o;
  -----
}

```

```

Criteria Interface - session.createCriteria(Products.class);

Criteria crit = session.createCriteria(Products.class);
    Criterion c1=Restrictions.gt("price", new Integer(12000));
    //price is our pojo class variable
    crit.add(c1); // adding criterion object to criteria class object
List l = crit.list(); // executing criteria query

```

Projections – select partial object using on Criteria

```

Criteria crit = session.createCriteria(Products.class);
crit.setProjection(Projections.property("proName"));
List l=crit.list();
Iterator it=l.iterator();
while(it.hasNext()){
{
String s = (String)it.next();
// ---- print -----
}

```

SQLQuery – session.createSQLQuery(String queryString)

```

//We are letting hibernate to know our pojo class too
SQLQuery q=session.createSQLQuery("select *from EMPLOYEE").addEntity(EmployeeBo.class);
List l = q.list();
Iterator it = l.iterator();
while(it.hasNext()){
    EmployeeBo s = (EmployeeBo)it.next();
}

```

Named Queries

- In HQL, we need to use <query name= "query\_name"> to configure query

```

<query name="bankHQLQuery">
<![CDATA[from BankBo b where b.balance>:bal ]]>
</query>

```

- In NativeSQL, we need to use <sql-queryname= "query\_name"> to configure query

```

<sql-query name="bankNativeQuery">
    select * from Employee
</sql-query>

```

- In our main program, we need to use getNamedQuery() given by session interface, for getting the **Query** reference and we need to execute that query by calling **list()**

```

Query qry = session.getNamedQuery("Name given in hib-mapping-xml");
qry.setParameter("bal",new Integer(3000));
List l = qry.list();

```

### What are advantages of Hibernate?

- Lazy Loading – not every time hits the database, will get Proxy instead.
- 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 is caching?

Anything you can do to minimize traffic between a database and an application server is probably a good thing. In theory, an application ought to be able to maintain a cache containing data already loaded from the database, and only hit the database when information has to be updated. When the database is hit, the changes may invalidate the cache

## First Level Cache & Second Level Cache?

Every fresh session having its own **cache memory**, **Caching is a mechanism for storing the loaded objects into a cache memory.**

The advantage of cache mechanism is, whenever again we want to load the same object from the database then instead of hitting the database once again, it loads from the local cache memory only, **so that the no. of round trips between an application and a database server got decreased**. It means caching mechanism increases the performance of the application.

In hibernate we have two levels of caching

1. **First Level Cache** (Session Cache)
2. **Second Level Cache** (Session Factory Cache/ JVM Level Cache)

### 1. First Level Cache

- By default, for each hibernate application, **the first level cache is automatically enabled. We can't Enable/Disable first level cache.**
- the first level cache is associated with the **session** object and **scope** of the cache is limited **to one session only**
- When we load an object for the first time from the database then the object will be loaded from the database and the loaded object will be stored in the cache memory maintained by that session object
- If we load the same object once again, with in the same session, then the object will be loaded from the local cache memory not from the database
- If we load the same object by opening other session, then again, the object will load from the database and the loaded object will be stored in the cache memory maintained by this new session

```
Session session = factory.openSession();
Object ob1 = session.get(Actor.class, new Integer(101)); //1

Object ob2 = session.get(Actor.class, new Integer(101)); //2
Object ob3 = session.get(Actor.class, new Integer(101)); //3
session.close(); //4

Session ses2 = factory.openSession();
Object ob5 = ses2.get(Actor.class, new Integer(101)); //5
```

**1,** We are loaded object with id 101, now it will load the object from the database. Because it is the first time, and it keeps this object in the session cache

**2,3** I tried to load the same object 2 times, but here the object will be loaded from the stored cache only not from the database, as we are in the same session

**4,** we close the first session, so the cache memory related this session also will be destroyed

**5,** again I created one new session and loaded the same object with id 101, but this time hibernate will loads the object from the database

if we want to remove the objects that are stored in the cache memory, then we need to call either **evict(Object ob)** or **clear()** methods

```
Void evict(Object ob): Remove that particular object from the session cache.  
Void clear(): Completely clear the session.  
Void close(): Close the session
```

## 2.Second Level Cache

Whenever we are loading any object from the database, then hibernate verify whether that object is available in the local cache(**first level cache**) memory of that particular session, if not available then hibernate verify whether the object is available in global cache(**second level cache**), if not available then hibernate will hit the database and loads the object from there, and then **first stores in the local cache of the session , then in the global cache**

**SessionFactory** holds the second level cache data. It is global for all the session objects and not enabled by default.

Different vendors have provided the implementation of Second Level Cache

1. **EH Cache**
2. **OS Cache**
3. **Swarm Cache**
4. **JBoss Cache**

To enable second level cache in the hibernate, then the following **3** changes are required

1. **Add provider class** in hibernate configuration file

```
<property name="hibernate.cache.provider_class">  
    org.hibernate.cache.EhCacheProvider  
</property>
```

2. **Configure cache element** for a class in hibernate mapping file

```
<cache usage="read-only" />
```

- o **read-only:** caching will work for read only operation.
- o **nonstrict-read-write:** caching will work for read and write but one at a time.
- o **read-write:** caching will work for read and write, can be used simultaneously.
- o **transactional:** caching will work for transaction.

3. create xml file called **ehcache.xml** and place where you have mapping and configuration xml's

### Example:

```
public class Employee {
    private int eid;
    private String name;
    private String address;
//Setters & Getters
}
```

### Employee.hbm.xml

```
<hibernate-mapping package="cache">
    <class name="Employee" table="employee">
        <cache usage="read-only" />
        <id name="eid" column="eid">
            <generator class="native"></generator>
        </id>
        <property name="name"></property>
        <property name="address"></property>
    </class>
</hibernate-mapping>
```

### ehcache.xml

```
<?xml version="1.0"?>
<ehcache>
    <defaultCache maxElementsInMemory="100" eternal="false"
                  timeToIdleSeconds="120" timeToLiveSeconds="200" />
    <cache name="cache.Employee" maxElementsInMemory="100"
           eternal="false" timeToIdleSeconds="5" timeToLiveSeconds="200" />
</ehcache>
```

### hibernate.cfg.xml

```
<hibernate-configuration>
    <session-factory>
        <property> Driver Class, URL, Username, password, etc </property>
        <property name="cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
        <property name="hibernate.cache.use_second_level_cache">true</property>
        <mapping resource="cache/employee.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

### **What are some core interfaces of hibernate?**

- Configuration
- SessionFactory
- Session
- Transaction
- Query
- Criteria

### **4 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, null in database	If it does get the object with id, it throws ObjectNotFound in Cache
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.

No	Save()	Persist()
1.	The return type of save() is java.io.Serializable and it return the generate id value.	Return type of persist() is void.
2.	Save() can be used inside of outside the transaction boundaries.	persist() can be used only with in the boundaries of a transaction.
3.	save() take compare more time to execute.	persist() takes less time to execute.
4.	save() is only supported by Hibernate.	persist() is supported by JPA also.

### What is the difference between and merge and update?

```

Employee emp1 = new Employee();
emp1.setEmpId(100);
emp1.setEmpName("Dinesh");
Session session1 = createNewHibernateSession();
session1.saveOrUpdate(emp1);
session1.close();
//emp1 object in detached state now

emp1.setEmpName("Dinesh Rajput");//Updated Name

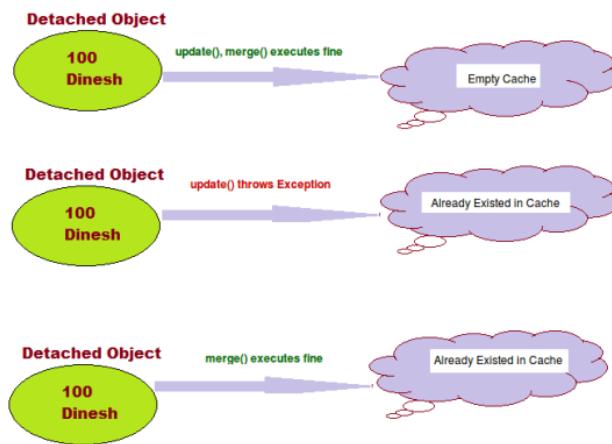
//Create session again
Session session2 = createNewHibernateSession();
Employee emp2 =(Employee)session2.get(Employee.class, 100);
//emp2 object in persistent state with id 100

//below we try to make on detached object with id 100 to persistent state by using update method of hibernate
        session2.update(emp1);//It occurs the exception NonUniqueObjectException because emp2 object is having employee with same empid as 100 in Cache.Here cache is not Empty. See diageram.

//to avoid this exception we are using merge like given below instead of session.update(emp1);

        session2.merge(emp1); //it merge the object state with emp2
session2.update(emp1); //Now it will work with exception

```



#### Update:

- Suppose we are dealing with any employee object in the same session then we should use update() or saveOrUpdate() method.
- if you are sure that the session does not contains an already persistent instance with the same identifier, then use update to save the data in hibernate

#### Merge:

- Suppose we are creating a session and load an employee object. Now object in session cache. If we close the session at this point and we edit state of object and tried to save using update() it will throw exception. To make object persistent we need to open another session. Now we load same object again in current session. So if we want to update present object with previous object changes we have to use merge() method. Merge method will merge changes of both states of object and will save in database.
- if you want to save your modifications at any time without knowing about the state of an session, then use merge() in hibernate.

#### Different between cascade and inverse

Many Hibernate developers are confusing about the cascade option and inverse keyword. In some ways. They really look quite similar at the beginning; both are related with relationship.

However, there is no relationship between **cascade** and **inverse**, both are totally different notions.

#### 1. inverse

This is used to decide which side is the relationship owner to manage the relationship (insert or update of the foreign key column).

In this example, the relationship owner is belonging to stockDailyRecords (inverse=true).

```
<!-- Stock.hbm.xml -->
<hibernate-mapping>
  <class name="Stock" table="stock">
    ...
    <set name="stockDailyRecords" table="stock_daily_record" inverse="true">
      <key>
        <column name="STOCK_ID" not-null="true" />
      </key>
      <one-to-many class="StockDailyRecord" />
    </set>
    ...
  </class>
</hibernate-mapping>
```

When you save or update the stock object

```
session.save(stock);
session.update(stock);
```

Hibernate will only insert or update the STOCK table, no update on the foreign key column. [More detail example here...](#)

## 2. cascade

In cascade, after one operation (save, update and delete) is done, it decide whether it need to call other operations (save, update and delete) on another entities which has relationship with each other.

In this example, the cascade="save-update" is declare on stockDailyRecords.

```
<!-- Stock.hbm.xml -->
<hibernate-mapping>
    <class name="com.mkyong.common.Stock" table="stock" ...>
        ...
        <set name="stockDailyRecords" table="stock_daily_record"
            cascade="save-update" inverse="true">
            <key>
                <column name="STOCK_ID" not-null="true" />
            </key>
            <one-to-many class="com.mkyong.common.StockDailyRecord" />
        </set>
        ...
    </class>
</hibernate-mapping>
```

Copy

When you save or update the stock object

```
session.save(stock);
session.update(stock);
```

It will inserted or updated the record into STOCK table and call another insert or update statement (cascade="save-update") on StockDailyRecord. [More detail example here...](#)

## Conclusion

In short, the "inverse" is decide which side will update the foreign key, while "cascade" is decide what's the follow by operation should execute. Both are look quite similar in relationship, but it's totally two different things. Hibernate developers are worth to spend time to research on it, because misunderstand the concept or misuse it will bring serious performance or data integrity issue in your application.

### 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 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 initialization**,

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 (Bean) 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 NamedSQLQuery in Hibernate?

Named queries are SQL queries which are defined in mapping document using `<query>` & `<sql-query>` tags 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. **@NamedQuery** 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");

```

## Explain Criteria API

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like “search” screens where there is a variable number of conditions to be placed upon the result set.

<b>Title</b> <input type="text"/> <i>e.g. The Godfather</i>	<b>User Rating</b> <input type="text"/> to <input type="text"/>
<b>Title Type</b>	
<input checked="" type="checkbox"/> Feature Film <input type="checkbox"/> TV Movie <input type="checkbox"/> TV Series <input type="checkbox"/> TV Episode <input type="checkbox"/> TV Special <input type="checkbox"/> Mini-Series <input type="checkbox"/> Documentary <input type="checkbox"/> Video Game <input type="checkbox"/> Short Film <input type="checkbox"/> Video <input type="checkbox"/> Unknown Work	
<b>Release Date</b> <input type="text"/> to <input type="text"/> <i>Format: YYYY-MM-DD, YYYY-MM, or YYYY</i>	
<b>Number of Votes</b> <input type="text"/> to <input type="text"/>	
<b>Genres</b>	
<input type="checkbox"/> Action <input type="checkbox"/> Adventure <input type="checkbox"/> Animation <input type="checkbox"/> Biography <input checked="" type="checkbox"/> Comedy <input type="checkbox"/> Crime <input type="checkbox"/> Documentary <input type="checkbox"/> Drama <input type="checkbox"/> Family <input type="checkbox"/> Fantasy <input type="checkbox"/> Film-Noir <input type="checkbox"/> Game-Show <input type="checkbox"/> History <input type="checkbox"/> Horror <input type="checkbox"/> Music <input type="checkbox"/> Musical <input type="checkbox"/> Mystery <input type="checkbox"/> News <input type="checkbox"/> Reality-TV <input type="checkbox"/> Romance	

```

List employees = session.createCriteria(Employee.class)
.add(Restrictions.like("name", "%"))
.add(Restrictions.like("address", "Boston"))
.addOrder(Order.asc("name"))
.list();

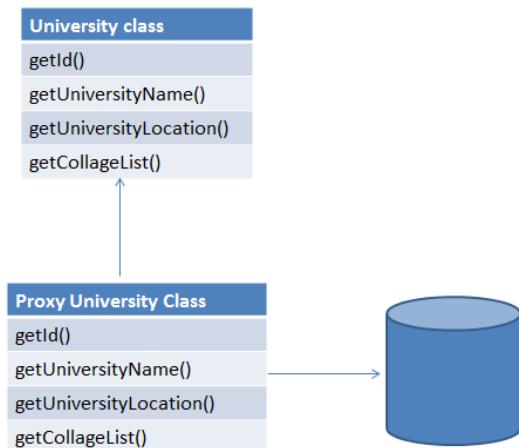
```

## How do you switch between relational databases without code changes?

Using Hibernate SQL Dialects, we can switch databases. Hibernate will generate appropriate hql queries based on the dialect defined.

## What is Hibernate proxy?

The proxy attribute enables lazy initialization of persistent instances of the class. Hibernate will initially return CGLIB proxies which implement the named interface. The actual persistent object will be loaded when a method of the proxy is invoked.



## What is automatic dirty checking?

Automatic dirty checking is a feature that saves us the effort of explicitly asking Hibernate to update the database when we modify the state of an object inside a transaction.

If Dirty-checking is enabled, if we forget to call save() before the commit, dirty-checking automatically saves the data into the database.

Consider the below code which loads a simple Entity from the database and updates it.

```
public static void testUpdate() {  
    Session session = sessionFactory.openSession();  
    Transaction transaction = session.beginTransaction();  
    Employee e = (Employee) session.load(Employee.class, 1);  
    e.setName("Satya");  
    transaction.commit();  
    session.close();  
}
```

Although we haven't made any session.update(Employee) call, the logs indicate that the database record was updated successful

## 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;  
}
```

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.

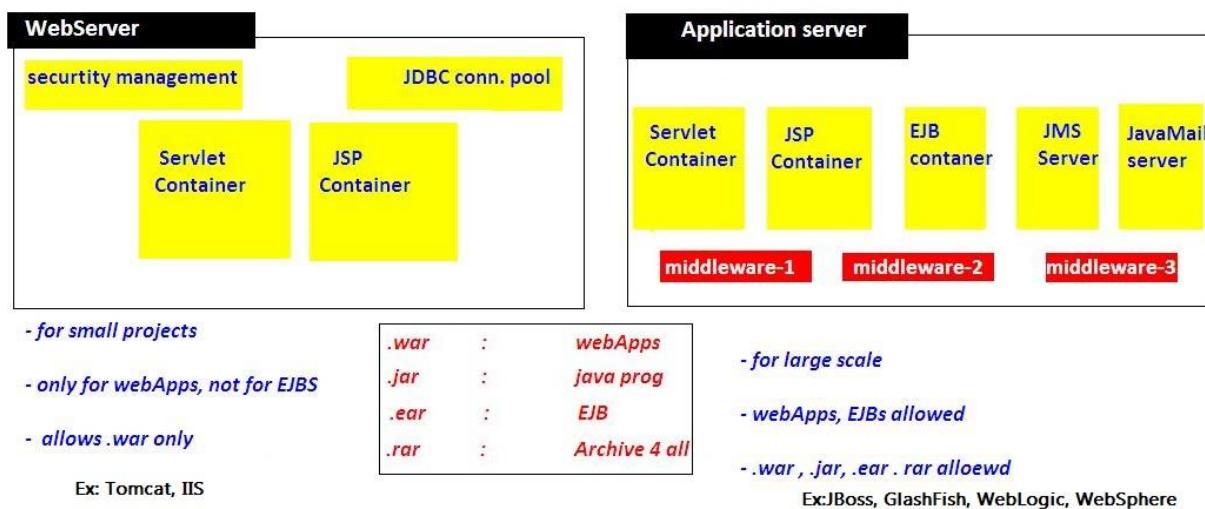
**FetchType.EAGER**: it fetches the child entities along with parent.

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"`

## Servlets

---

### Web Server VS Application Server?



### What are the core components of the HTTP request and HTTP response?

Answer: HTTP request has following 5 major components:

HTTP Requests	Meaning/work
<b>Verb</b>	Indicate HTTP methods like GET, PUT, POST, etc
<b>URI</b>	Identifies the resource on server
<b>HTTP Version</b>	Indicates version.
<b>Request Header</b>	Contains metadata like client type, cache settings, message body format, etc for HTTP request message.
<b>Request Body</b>	Represents content of the message.

HTTP response has following 4 major components:

HTTP Response	Meaning/work
<b>Status code</b>	Indicates the status of the server for requested resource.
<b>HTTP version</b>	Represents HTTP version.
<b>Response Header</b>	Consists of metadata like content length, content type, server length, etc for HTTP response message.
<b>Response Body</b>	Represents response message content.

### Difference between IBM WAS server vs IBM Portal server?

**IBM WebSphere Portal** is a set of software tools that enables companies to build and manage **web portals**. It provides a single access point to web content and applications, while delivering differentiated, personalized experiences for each user.

**WebSphere Application Server (WAS)** is a software product that performs the role of a web application server. More specifically, it is a software framework and **middleware** that hosts Java based web applications. It is the flagship product within IBM's WebSphere software suite.

NOTE: Portal 'runs on top of' WAS using many of the services provided by WAS.

Example Portal MVC Appl'n

<https://www.youtube.com/watch?v=DGMXQwzyKDQ&list=PLDF56A9385D44FB63&index=23>

### Example of IBM Portlet Spring MVC ?

1.web.xml – FrontController Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd"
version="2.5" id="${project.artifactId}" metadata-complete="true">

<display-name>${project.artifactId}</display-name>
<description> ${project.version}</description>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<servlet>
    <servlet-name>view-servlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.ViewRendererServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>view-servlet</servlet-name>
    <url-pattern>/WEB-INF/_servlet/view</url-pattern>
</servlet-mapping>
</web-app>
```

## 2.spring-servlet.xml – Spring View Configuration

```
<beans>
    <bean id="viewResolver" class="org.springframework.InternalResourceViewResolver">
        <property name="viewClass" value="org.springframework.InternalResourceView" />
        <property name="prefix" value="/pages/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```



portlet.xml

- All Portlets configure here.



productdecommission-portlet.xml



trialuserlist-portlet.xml

## Individual Portlet Configuration

### 3. Controller Classes



ProductDecommissionController.java



TrialUserListViewController.java

### 4. View Pages

```
✓ pages
    > bulkupload
    > changeproductverticalroles
    > common
    > decomexport
    > managecountrysite
    > passwordexpirynotify
    ✓ productdecommission
        roles.jsp
        revisionHistory.jsp
        view.jsp
```

### 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 create **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.

- **SAX** is an event-based parser and can be used to **parse a large file**,
- **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>

### What is JSESSIONID in J2EE Web application - JSP Servlet?

HTTP protocol and Web Servers are stateless. it means for web server every request is a new request to process and they can't identify if it's coming from client that has been sending request previously.

**Session** is a conversational state between client and server, and it can consist of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session (session id) is passed between server and client in every request and response.

When we use `HttpServletRequest getSession()` method and it creates a new request, it creates the new HttpSession object and also add a Cookie to the response object with name `JSESSIONID` and value as session id. This cookie is used to identify the HttpSession object in further requests from client. If the cookies are disabled at client side and we are using URL rewriting, then this method uses the jsessionid value from the request URL to find the corresponding session. JSESSIONID cookie is used for session tracking, so we should not use it for our application purposes to avoid any session related issues.

### How servlet session will work, if Cookies disabled?

In a web app, when user logs in, a HttpSession is created using `HttpSession s = request.getSession(true);` This creates a cookie with jsessionid on the browser. But if cookies are disabled on browser, How can i proceed with login?

If Cookies are disabled. You should be using URL Rewriting mechanism for Session tracking. In this process we are not use SessionID – we just pass information in link.

```
String n = request.getParameter("uname");
out.print("<a href='SecondServlet?uname=" + n + "'>visit</a>");
```

We can use any other methods like below , for Session tracking instead of url rewring

Hidden Form Field  
Cookies  
HttpSession  
URL Rewriting

### Can you describe the difference between valid and well-formed XML?

A well-formed XML is the one which has **root element, and all tags are closed properly**, attributes are defined properly, their value is also quoted properly.

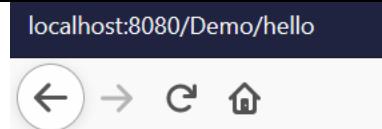
On another hand, a valid XML is the one which **can be validated against an XSD file or schema**. So it's possible for an XML to be well-formed but not valid because they contain tags **which may not be allowed** by their schema.

### What happens if wont set contentType in Servlet response

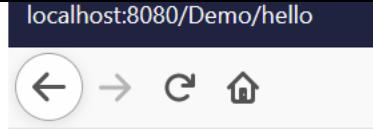
If you won't set Content type, by default Servlet prints the content as it is, without encoding or formating.

For example, output for below code will be

```
PrintWriter pw = res.getWriter();
//res.setContentType("text/html");
pw.write("<br> Hello, world <br>");
```



<br> Hello, world <br>  
**without setContentType**



Hello, world  
**with setContentType**

Here HTML tags prints as it is, as a String. But we uncomment above `res.setContentType("text/html");`

## Maven& Ant build for different environments using properties

### Ant

Ant's build file, called **build.xml** should reside in the base directory of the project. However, **there is no restriction on the file name or its location**. You are free to use other file names or save the build file in some other location

```
<?xml version = "1.0"?>
<project name = "Hello World Project" default = "info">
    <target name = "info">
        <echo>Hello World - Welcome to Apache Ant!</echo>
    </target>
</project>
```

```
<project name="Sample Build Script" default="init" basedir=".">
    <property environment="env" />
    <!-- ***** COMMAND LINE ARGUMENTS DEMOED HERE -->
    <property name="build_type" value= "${env.build_type}" />
    <property name="version" value="${env.version}" />
    <!-- ***** END OF COMMAND LINE ARG -->

    <property name="src.dir" value="${basedir}/source"/>
    <property name="build.classes.dir" value="${basedir}/classes"/>
    <property name="project.name" value="myproject"/>

    <target name="make-war" depends="compile-servlet">
        <delete file="${build.classes.dir}/war/${project.name}.war"/>
        <war destfile="${build.classes.dir}/war/${project.name}.war" webxml="${src.dir}/WEB-INF/web.xml">
            <webinf dir="${src.dir}/WEB-INF" />
            <fileset dir="${src.dir}/html">
                <include name="*.html" />
            </fileset>
            <classes dir="${build.classes.dir}">
                <include name="/my/package/*.*"/>
            </classes>
            <lib dir="/some/lib/loc">
                <include name="some-lib.jar"/>
            </lib>
        </war>
    </target>
</project>
```

```

        </war>
    </target>

    <target name="init" >
        <echo message="Using Source directory=${src.dir}" />
        <echo message="Using Build-Classes directory=${build.classes.dir}" />
        <!-- ***** VERIFY COMMAND LINE ARGS HERE ***** -->
        <echo message="Build Type=${build_type}" />
        <echo message="Build Version =${version}" />
        <!-- *** END VERIFY COMMAND LINE ARGUMENTS -->
    </target>
</project>

```

## Maven – pom.xml

I have a web app in Maven, with the default directory structure. No problem there. The default directory structure has some property files that point to my localhost database.

Currently I create an Ant script to create different war files - one for production and one for development, using these commands:

```

ant deploy-dev
ant deploy-prod
ant deploy-sit
ant deploy-uat

```

I prefer use maven profiles for this situation. For example we have directory structure:

```

src/main/resources
|
+- local
|   |
|   `-- specific.properties
+- dev
|   |
|   `-- specific.properties

```

In pom.xml define two profiles:

```

<profiles>
    <profile>
        <id>local</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <build>
            <resources>
                <resource>
                    <directory>src/main/resources/local</directory>
                </resource>
            </resources>
        </build>
    </profile>
    <profile>
        <id>dev</id>
        <build>
            <resources>
                <resource>
                    <directory>src/main/resources/dev</directory>
                </resource>
            </resources>
        </build>
    </profile>
</profiles>

```

To activate this you would type this on the command line:

```
mvn groupId:artifactId:goal -Denvironment=local
```

### How PrintWriter is different from ServletOutputStream?

**Ans:** PrintWriter is basically a character-stream class. On the other hand, ServletOutputStream is a byte-stream class. The PrintWriter class can be used to write **only character-based information** whereas ServletOutputStream class can be used to write **primitive values as well as character-based** information.

### Explain is servlet mapping?

**Ans:** Servlet mapping is a process of defining an association between a URL pattern and a servlet. The mapping is used to map requests to Servlets.

### What are the annotations used in Servlet 3?

**Ans:** The important 3 annotations used in the servlets are.

- **@WebServlet** : for servlet class.
- **@WebListener** : for listener class.
- **@WebFilter** : for filter class.

### What is the use of RequestDispatcher Interface?

**Ans:** The RequestDispatcher interface defines the object that receives the request from the client and dispatches it to the resources such as a servlet, JSP, HTML file. The RequestDispatcher interface has the following two methods:

**Forwards** request from one servlet to another resource like servlet, JSP, HTML etc.

```
public void forward(ServletRequest request, ServletResponse response)
```

**Includes** the **content** of the resource such as a servlet, JSP, and HTML in the response.

```
public void include(ServletRequest request, ServletResponse response)
```

### Can a JSP be called using a Servlet?

**Ans:** Yes, Servlet can call a JSP using RequestDispatcher interface.

```
RequestDispatcher reqdis=request.getRequestDispatcher("log.jsp");
reqdis.forward(request,response);
```

### Explain the Servlet Filter?

A **Filter** is defined as a **pluggable** object that is invoked either at the pre-processing or post-processing of a **request**. We need **Servlet Filters** for the following reasons:

- **Logging** the request parameters to log files.
- **Authentication** and **Authorization** of the request for the needed resources.
- **Formatting** of the request body/header before sending it to the servlet.
- **Compressing** response data sent to the client.
- Change the response by adding some **cookies** and **header** information.

### Why is init() method is used in Servlets?

**Ans:** The `init()` method is used to create or load some data that will be used throughout the life of the servlet.

### What is load-on-startup in Servlet?

**Ans:** The `load-on-startup` element of servlet in `web.xml` is used to load the servlet at the time of deploying the **project** or the **server** to start. This saves time for the **response** of the first request.

### What are the different methods involved in the process of session management in servlets?

**Ans:** The different methods involved in the session management in servlets are as follows:

- **User Authentication:** A **user** tries to access a protected resource, such as a JSP page. If the **user** has been **authenticated**, the **Servlet** container makes the resource available; otherwise, the **user** is asked for a username and password
- **HTML Hidden Field:** The `<input type="hidden">` defines a **hidden input field**. A **hidden field** let web developers include data that cannot be seen or modified by users when a **form** is submitted. A **hidden field** often stores what database record that needs to be updated when the **form** is submitted
- **Cookies :** A small text file created by a website that is stored in the user's computer either temporarily for that session only or permanently on the hard disk. **Cookies** provide a way for the website to recognize you and keep track of your preferences
- **URL Rewriting:** URL rewriting is an automatic process of altering a program written for manipulating the parameters in a **URL** (Uniform Resource Locator). **URL** manipulation is employed as a convenience by a Web server administrator, or for nefarious purposes by a hacker.
- **Session Management API:** is built on top of the **Request-Response** methods for session tracking. Session Tracking is a way to maintain state/data of a user. It is also known as session management in servlet.

### How do you get the IP address of the client in servlet?

We can use `request.getRemoteAddr()` code to get the **client IP address** in servlet.

### Can you send an Authentication error from a Servlet?

**Ans:** Yes, we can use `setStatus(statuscode)` method of **HttpServletResponse** to send an authentication error. All we have to do is to set an error code and a valid reason along with the error code.

```
response.sendError(404, "Page not Found!!!" );
```

### What are different HTTP status codes?

For setting HTTP status code other than 200, we have to use **HttpServletResponse** class for response. Below are some of the sample return statements showing it's usage.

```
res.setStatus(HttpServletResponse.SC_FORBIDDEN);
```

HTTP Status Codes		
Level 200 (Success) 200 : OK 201 : Created 203 : Non-Authoritative Information 204 : No Content	Level 400 400 : Bad Request 401 : Unauthorized 403 : Forbidden 404 : Not Found 409 : Conflict	Level 500 500 : Internal Server Error 503 : Service Unavailable 501 : Not Implemented 504 : Gateway Timeout 599 : Network timeout 502 : Bad Gateway

Forbidden - not allowed; banned.

## JSP

---

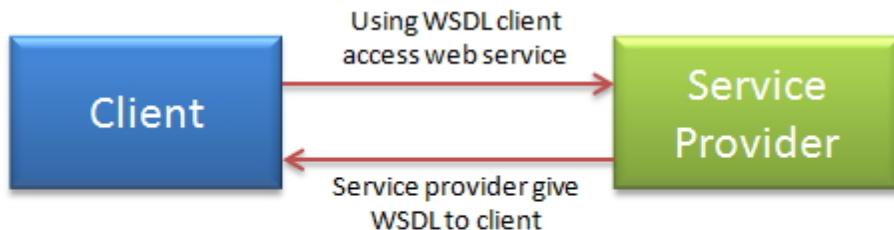
<https://www.edureka.co/blog/interview-questions/jsp-interview-questions/>

# Web services

## How to access SOAP web service?

There are two ways to access web service

1. If Service provider knows client then it will provide its wsdl to client and client will be able to access web service.

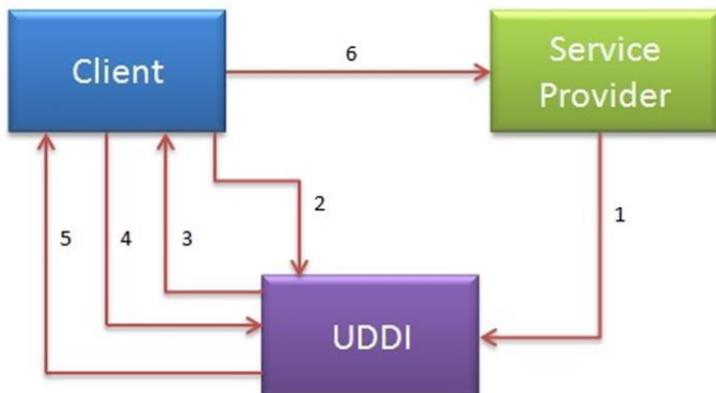


(If Service provider knows client)

2. If Service provider register its WSDL to UDDI and client can access it from UDDI

**UDDI:** UDDI stands for Universal Description, Discovery and Integration. It is a directory service. Web services can register with a UDDI and make themselves available through it for discovery. So following steps are involved.

1. Service provider registers with UDDI.
2. Client searches for service in UDDI.
3. UDDI returns all service providers offering that service.
4. Client chooses service provider
5. UDDI returns WSDL of chosen service provider.
6. Using WSDL of service provider, client accesses web service

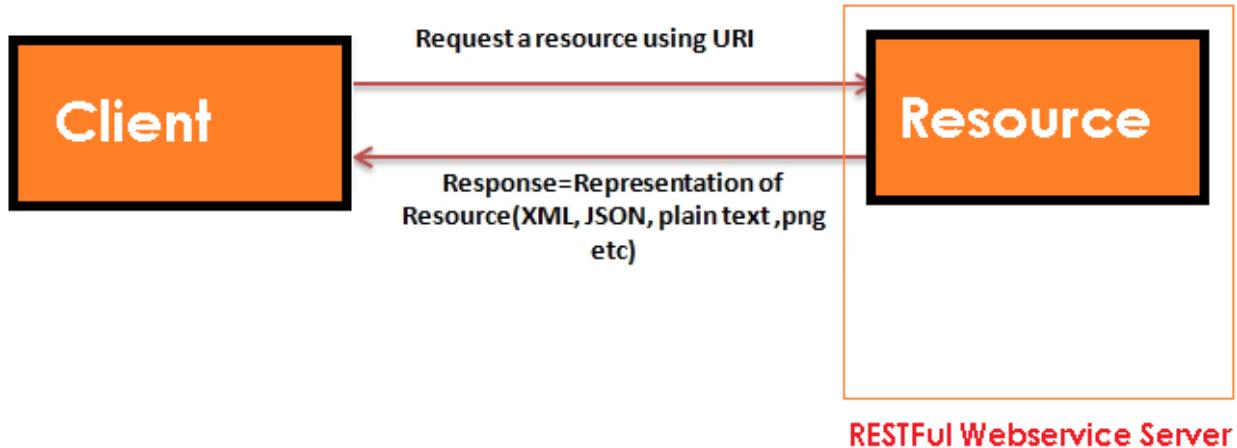


## What are Rest components

It consists of two components

1. **REST server:** which provides access to the resources

2. **REST client:** which accesses and modify the REST resources.

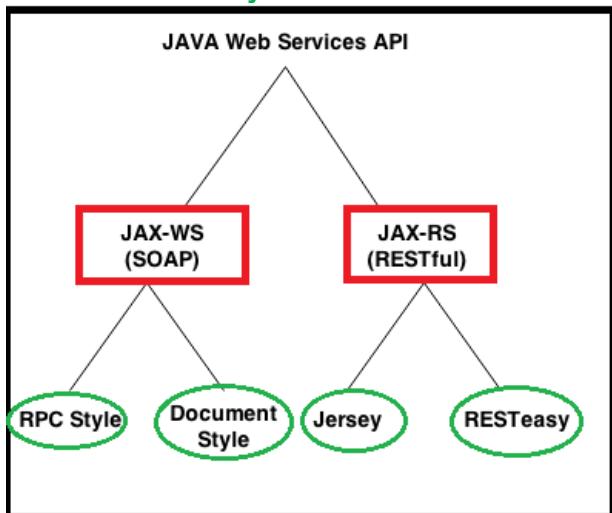


## What is Idempotent?

Idempotent means result of multiple successful requests will not change state of resource after initial application

- **GET is idempotent.** If Delete() is idempotent method because when you first time use delete, it will delete the resource (initial application) but after that, all other request will have no result because resource is already deleted.
- **Post is not idempotent** method because when you use post to create resource, it will keep creating resource for each new request, so result of multiple successful requests will not be same.

## Webservices API in java?



## Difference between RPC-Style and Document Style

The way of generating SOAP message format is main difference between them.

### 1. RPC Style:

SOAP Body must conform to a structure that indicates the **method name & Parameter's name**

```
<soap:envelope>
<soap:body>
<myMethod>
<x xsi:type="xsd:int">5</x>
<y xsi:type="xsd:float">5.0</y>
</myMethod>
</soap:body>
</soap:envelope>
```

### 2. Document Style

SOAP Body can be structured in any way you like. There is no TYPE attribute here

```
<soap:envelope>
<soap:body>
<xElement>5</xElement>
<yElement>5.0</yElement>
</soap:body>
</soap:envelope>
```

## JAX-WS Encoding Styles?

There are two encoding use models that are used to translate a WSDL binding to a SOAP message. They are **literal** and **encoded**.

- **literal** – You can read, So we can validate by comparing XSD.
- **encoded** – You can't Read, So you can't validate its structure by comparing XSD Schema.

The combination of the different style and use models give us four different ways to translate a WSDL binding to a SOAP message.

```
Document/literal
Document/encoded
RPC/literal
RPC/encoded
```

When using a **literal use model**, the body contents should conform to a user-defined **XML-schema (XSD) structure**. The advantage is two-fold.

- one, you can validate the message body with the user-defined XML-schema.
- Two, you can also transform the message using a transformation language like XSLT.

With a **(SOAP) encoded use model**, the message has to use XSD datatypes, but the structure of the message need not conform to any user-defined XML schema. This makes it difficult to validate the message body or use XSLT based transformations on the message body.

## Steps to create JAX-WS Webservice

### 1. JAX-WS Web Service End Point files

1. Create a Web Service Endpoint Interface with `@SOAPBinding(style = Style.RPC)`
2. Create a Web Service Endpoint Implementation
3. Create an Endpoint Publisher
4. Test generated WSDL. Ex: <http://localhost:8080/ws/hello?wsdl>

### 2. Web Service Client files

1. Java Web Service Client

## 1. JAX-WS Web Service End Point files

### 1. Create a Web Service Endpoint Interface

```
package endpoint;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld{
    @WebMethod
    String getHelloWorldMsg(String msg);
}
```

### 2. Create a Web Service Endpoint Implementation

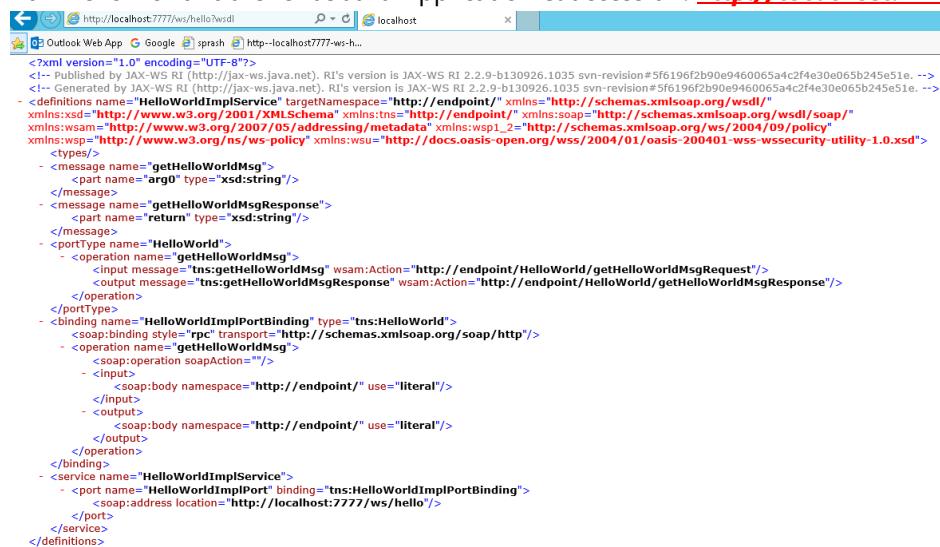
```
package endpoint;
import javax.jws.WebService;
//Service Implementation
@WebService(endpointInterface = "endpoint.HelloWorld")
public class HelloWorldImpl implements HelloWorld{
    @Override
    public String getHelloWorldMsg(String msg) {
        // TODO Auto-generated method stub
        return "Your Message from WebService is : "+msg;
    }
}
```

### 3. Create an Endpoint Publisher

```
package endpoint;
import javax.xml.ws.Endpoint;
//Endpoint publisher
public class HelloWorldPublisher{
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:7777/ws/hello", new HelloWorldImpl());
        System.out.println("WSDL Published !!");
    }
}
```

### 4. Test generated WSDL

Run HelloWorldPublisher as Java Application & access url: <http://localhost:7777/ws/hello?wsdl>



<http://endpoint/> uses package name of Service endpoint publisher

## wimport tool VS wsgen

1.**wimport -(WSDL Import)** tool is will import WSDL file and generates JAX-WS Web Service End Point files.

```
>wimport -keep http://localhost:7777/ws/hello?wsdl
```

## 2.wsGen -(WSDL Generator)

It will read the JAX-WS Web Service End Point files & Generates WSDL Document & Webservice client for Testing . This **wsgen** tool is available in **\$JDK/bin** folder

```
>wsgen -verbose -keep -cp . endpoint.RandomNumber
```

## Difference between JAX-RS & RESTful

- RESTful is a Generalized Web service Standard given by W3.ORG.
- JAX-RS is a specification for RESTful Web Services with Java and it is given by Sun.
- **Jersey** from Oracle, **Resteasy** from Jboss are the implementations of JAX-RS

## majorly used annotations in RESTful webservices

- **@Path('Path')**
- **@GET**
- **@POST**
- **@PUT**
- **@DELETE**
- **@Produces(MediaType.TEXT\_PLAIN [, more-types]) – Only for @GET**
- **@Consumes(type[, more-types]) - Only for @POST**
- **@PathParam()**
- **@QueryParam()**
- **@MatrixParam()**
- **@FormParam()**

## Steps to creates to Restful web-service in java?

- 1 Add Jersey jar files in pom.xml
2. Create RESTful webservice at Server End.

```
@Path("/hellojersey")
public class HelloWorldWebService {
    // This method is called if HTML and XML is not requested
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello() {
        return "Hello Jersey Plain";
    }

    // This method is called if HTML is requested
    @GET
    @Produces(MediaType.TEXT_HTML)
    public String sayHtmlHello() {
        return "<h1>" + "Hello Jersey HTML" + "</h1>";
    }
}
```

### 3.Configure web.xml

In `web.xml`, register “`com.sun.jersey.spi.container.servlet.ServletContainer`”, and puts your Jersey service folder under “`init-param`”, “`com.sun.jersey.config.property.packages`”

```
<web-app>
    <servlet>
        <servlet-name>jersey-servlet</servlet-name>
        <servlet-class>
            com.sun.jersey.spi.container.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>service</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>jersey-servlet</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

### 4.Test Service

<http://localhost:8080/JAXRS-Jersey-HelloWorld/rest/hellojersey>

or

```
System.out.println(target.path("rest").path("hellojersey").request().accept(MediaType.TEXT_PLAIN).get(String.class));

System.out.println(target.path("rest").path("hellojersey").request().accept(MediaType.TEXT_HTML).get(String.class));
}
```

## Examples

### 1.@Path Annotation

```
@Path("/country")
public class PathMethodLevelService {

    @GET
    @Produces("text/html")
    public Response selectCountry() {
        String output = " Default Country : <h1>INDIA</h1>";
        return Response.status(200).entity(output).build();
    }

    @GET
    @Path("/usa")
    @Produces("text/html")
    public Response selectUSA() {
        String output = "Selected Country : <h1>United States of America(USA)</h1>";
        return Response.status(200).entity(output).build();
    }

    @GET
    @Path("/uk")
    @Produces("text/html")
    public Response selectUK() {
        String output = "Selected Country : <h1>UNITED KINGDOM(UK)</h1>";
        return Response.status(200).entity(output).build();
    }
}
```

## Response Class in JAX-RS

`javax.ws.rs.core.Response` contains static methods to create a Response instance using a `ResponseBuilder`.

<http://localhost:8080/App/rest/students/101/Satya/Vijayawada>

```
@Path("/students")
public class PathParamService {
    @GET
    @Path("{rollno}/{name}/{address}")
    @Produces("text/html")
    public Response getResultByPassingValue(
        @PathParam("rollno") String rollno,
        @PathParam("name") String name,
        @PathParam("address") String address) {
        String output = "<h1>PathParamService Example</h1>";
        output = output+"<br>Roll No : "+rollno;
        output = output+"<br>Name : "+name;
        output = output+"<br>Address : "+address;
        return Response.status(200).entity(output).build();
    }
}
```

## How to set different status code in HTTP response?

For setting HTTP status code other than 200, we have to use `javax.ws.rs.core.Response` class for response. Below are some of the sample return statements showing it's usage.

```
return Response.status(422).entity(exception).build();
return Response.ok(response).build(); //200
```

HTTP Status Codes		
Level 200 (Success) 200 : OK 201 : Created 203 : Non-Authoritative Information 204 : No Content	Level 400 400 : Bad Request 401 : Unauthorized 403 : Forbidden 404 : Not Found 409 : Conflict	Level 500 500 : Internal Server Error 503 : Service Unavailable 501 : Not Implemented 504 : Gateway Timeout 599 : Network timeout 502 : Bad Gateway

<http://localhost:8080/App/rest/students?rollno=1218&name=SATYA &address=VIJAYAWADA>

```
@Path("/students")
public class QueryParamwithDefaultvalueService {
    @GET
    @Produces("text/html")
    public Response getResultByPassingValue(
        @DefaultValue("1000") @QueryParam("rollno") String rollno,
        @DefaultValue("XXXX") @QueryParam("name") String name,
        @DefaultValue("XXXX") @QueryParam("address") String address) {
        String output = "<h1>QueryParamwithDefaultvalueService Example</h1>";
        output = output + "<br>Roll No : " + rollno;
        output = output + "<br>Name : " + name;
        output = output + "<br>Address : " + address;
        return Response.status(200).entity(output).build();
    }
}
```

<http://localhost:8080/App/rest/students;rollno=1118;name=SATYA;address=VIJAYAWADA>

```
@Path("/students")
public class MatrixParamService{
    @GET
    @Produces("text/html")
    public Response getResultByPassingValue(
        @MatrixParam("rollno") String rollno,
        @MatrixParam("name") String name,
        @MatrixParam("address") String address) {

        String output = "<h1>@MatrixParam Example</h1>";
        output = output+"<br>Roll No : "+rollno;
        output = output+"<br>Name : "+name;
        output = output+"<br>Address : "+address;
        return Response.status(200).entity(output).build();
    }
}
```

The image contains two side-by-side screenshots of a web browser window. Both screenshots show the URL <http://localhost:8080/JAXRS-FormParam-Example/>.  
The left screenshot shows a 'Student Registration' page with three input fields and a submit button:

- Roll No.:
- Full Name:
- Address:

A 'Register' button is also present below the inputs.  
The right screenshot shows the result of the registration:

**@FormParam Example - REGISTRATION COMPLETED!!!**

Roll No : 100  
Name : Satya Kaveti  
Address : VIJAYAWADA

```
@Path("/students")
public class FormParamService {

    @POST
    @Path("/registerStudent")
    @Produces("text/html")
    public Response getResultByPassingValue(
        @FormParam("rollno") String rollno,
        @FormParam("name") String name,
        @FormParam("address") String address) {

        String output = "<h1>@FormParam Example - REGISTRATION COMPLETED!!!</h1>";
        output = output+"<br>Roll No : "+rollno;
        output = output+"<br>Name : "+name;
        output = output+"<br>Address : "+address;
        return Response.status(200).entity(output).build();
    }
}
```

#### JAX-RS Download files (text/image/pdf/execel) Example

We can download any type of files from the RESTful web services, **@produces** annotation

We should annotate our method with

- **@Produces("text/plain")** If you are expecting Text file as response
- **@Produces("image/your image type[jpg/.png/.gif]")** for downloading any Image files
- **@Produces("application/pdf")** for downloading PDF files
- **@Produces(MediaType.APPLICATION\_JSON)** -- JSON
- **@Produces(MediaType.APPLICATION\_XML).** -XML

## How to Test (JAX-RS) RESTful Web Services

in real time projects we will use different tools to test RESTful web services

### 1.Browser Addons

- Postman [ Chrome Extension ]
- REST Client [ Chrome Extension ]
- Advanced REST Client [ Chrome Extension ]
- Rest Client [ Firefox Add-On ]

### 2.JAX-RS Local System Tools

- [RESTClient UI](#)
- [SoupUi](#)

## What are advantages of SOAP Web Services?

SOAP web services have all the advantages that web services has, some of the additional advantages are:

- WSDL document provides contract and technical details of the web services for client applications without exposing the underlying implementation technologies.
- SOAP uses XML data for payload as well as contract, so it can be easily read by any technology.
- SOAP protocol is universally accepted, so it's an industry standard approach with many easily available open-source implementations.

## What are different components of WSDL?

Some of the different tags in WSDL xml are:

- **xsd:import** namespace and schemaLocation: provides WSDL URL and unique namespace for web service.
- **message:** for method arguments
- **part:** for method argument name and type
- **portType:** service name, there can be multiple services in a wsdl document.
- **operation:** contains method name
- **soap:address** for endpoint URL.

## What is difference between Top Down and Bottom Up approach in SOAP Web Services?

**In Top-Down approach first WSDL document is created to establish the contract between web service** and client and then code is written, it's also termed as contract first approach. This is hard to implement because classes need to be written to confirm the contract established in WSDL. Benefit of this approach is that both client and server code can be written in parallel.

**In Bottom-Up approach, first web service code is written and then WSDL is generated.** It's also termed as contract last approach. This approach is easy to implement because WSDL is generated based on code. In this approach client code have to wait for WSDL from server side to start their work.

### Can we maintain user session in web services?

By default SOAP & Rest Web services are stateless so we can't maintain user sessions in web services. But we can make SOAP as statefull by manually using HttpSession & using below steps.

The steps required on the server:

- Add the `@Resource` to the top of your Web service.
- Add a variable of type `WebServiceContext` that will have the context injected into it.
- Using the `WebServiceContext`, get the `HttpSession` object.
- Save objects in the `HttpSession` using the `setAttribute` method and retrieve saved object using `getAttribute`. Objects are identified by a string value you assign.

### What is difference between SOA and Web Services?

Service Oriented Architecture (SOA) is an architectural pattern where applications are designed in terms of services that can be accessed through communication protocol over network. SOA is a design pattern and doesn't go into implementation.

Web Services can be thought of as Services in SOA architecture and providing means to implement SOA pattern.

### Name some frameworks in Java to implement SOAP web services?

We can create SOAP web services using JAX-WS API, however some of the other frameworks that can be used are **Apache Axis** and **Apache CXF**.

### What is use of javax.xml.ws.Endpoint class?

Endpoint class provides useful methods to create endpoint and publish existing implementation as web service. This comes handy in testing web services before making further changes to deploy it on actual server.

### What is sun-jaxws.xml file?

This file is used to provide endpoints details when JAX-WS web services are deployed in servlet container such as Tomcat. This file is present in WEB-INF directory and contains endpoint name, implementation class and URL pattern. For example;

#### **sun-jaxws.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
  <endpoint
    name="PersonServiceImpl"
    implementation="com.journaldev.jaxws.service.PersonServiceImpl"
    url-pattern="/personWS"/>
</endpoints>
```

## Name important annotations used in JAX-RS API?

Some of the important JAX-RS annotations are:

- **@Path**: used to specify the relative path of class and methods. We can get the URI of a webservice by scanning the Path annotation value.
- **@GET, @PUT, @POST, @DELETE and @HEAD**: used to specify the HTTP request type for a method.
- **@Produces, @Consumes**: used to specify the request and response types.
- **@PathParam**: used to bind the method parameter to path value by parsing it.

## What is purpose of different HTTP Request Types in RESTful Web Service?

- **GET** request on /employee/101, you can retrieve details of that user.
- **POST** on employee/102 would create a new user with employee id 102,
- **PUT** request type on /employee/101 can be used to update details of employee with id 101.
- **DELETE** method on /employee/101 can be used to remove data for that id.

By the way, in the case of PUT and POST method representation would be in the **request body**

## Is SOAP a stateless or a stateful protocol?

Soap as data can be used in both variants in **service - stateful or stateless**. By default, soap service is stateless.

There are Web service use cases where a client may want to save data on the service during one invocation and then use that data during a subsequent invocation. For example, a shopping cart object may be added to by repeated calls to the **addToCart** web method and then fetched by the **getCart** web method. In a stateless Web service, the shopping cart object would always be empty, no matter how many **addToCart** methods were called. But by using HTTP Sessions to maintain state across Web service invocations, the cart may be built up incrementally, and then returned to the client.

Enabling stateful support in a JAX-WS Web service requires a minimal amount of coding on both the client and server.

The steps required on the server:

- Add the **@Resource** to the top of your Web service.
- Add a variable of type **WebServiceContext** that will have the context injected into it.
- Using the **WebServiceContext**, get the **HttpSession** object.
- Save objects in the **HttpSession** using the **setAttribute** method and retrieve saved object using **getAttribute**. Objects are identified by a string value you assign.

### Accessing HTTP Session on the Server

```
@WebService
public class ShoppingCart {
    @Resource // Step 1
    private WebServiceContext wsContext; // Step 2

    public int addToCart(Item item) {
        MessageContext mc = wsContext.getMessageContext(); // Step 3
```

```

HttpSession session = mc.get(MessageContext.SERVLET_REQUEST).getSession();

if (session == null)
    throw new WebServiceException("No HTTP Session found");
// Get the cart object from the HttpSession (or create a new one)

List < Item > cart = (List < Item > ) session.getAttribute("myCart"); // Step 4
if (cart == null)
    cart = new ArrayList < Item > ();
cart.add(item);

// Save the updated cart in the HttpSession
session.setAttribute("myCart", cart);
return cart.size();
}
}

```

#### **Enabling HTTP Session on the Client**

The client-side code is quite simple. All you need to do is set the SESSION\_MAINTAIN\_PROPERTY on the request context. This tells the client to pass back the HTTP Cookies that it receives from the Web service. The cookie contains a session ID that allows the server to match the Web service invocation with the correct HttpSession, providing access to any saved stateful objects.

```

ShoppingCart proxy = new CartService().getCartPort();
((BindingProvider)proxy).getRequestContext().put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

Item item = new Item('123456', 4);
System.out.println(proxy.addToCart(item));
System.out.println(proxy.addToCart(item));

```

Ref. [https://docs.oracle.com/cd/E17904\\_01/web.1111/e13734/stateful.htm#WSADV236](https://docs.oracle.com/cd/E17904_01/web.1111/e13734/stateful.htm#WSADV236)

## **when to use SOAP and when to use Restful web services**

<https://www.infoq.com/articles/rest-soap-when-to-use-each/>

### **Case 1: Developing a Public API**

REST focuses on resource-based (or data-based) operations and inherits its operations (GET, PUT, POST, DELETE) from HTTP. This makes it easy for both developers and web-browsers to consume it, which is beneficial for public APIs where you don't have control over what's going on with the consumer. Simplicity is one of the strongest reasons that major companies like Amazon and Google are moving their APIs from SOAP to REST.

### **Case 2: Extensive Back-and-Forth Object Information**

APIs used by apps that require a lot of back-and-forth messaging should always use REST. For example, mobile applications. If a user attempts to upload something to a mobile app (say, an image to Instagram) and loses reception, REST allows the process to be retried without major interruption, once the user regains cell service.

However, with SOAP stateful operations, the same type of service would require more initialization and state code. Because REST is stateless, the client context is not stored on the server between requests, giving REST services the ability to be retried independently of one another.

### **Case 3: Your API Requires Quick Developer Response**

REST allows easy, quick calls to a URL for fast return responses. The difference between SOAP and REST, in this case, is complexity—SOAP services require maintaining an open stateful connection with a complex

client. REST, in contrast, enables requests that are completely independent of each other. The result is that testing with REST is much simpler.

Helpfully, REST services are now well-supported by tooling. The available tools and browser extensions make testing REST services continually easier and faster.

### **Explain the term Synchronicity.**

Synchronicity generally refers to the binding of the client to the function's execution and it can be done in two ways i.e., synchronous and asynchronous. In Synchronous invocations, the client blocks and waits until the service complete its operation before continuing its work. In Asynchronous invocations, clients are allowed to invoke a service and execute other functions

### **Name three primary security issues of Web Services?**

The three primary security issues of web services include:

- Confidentiality
- Authentication
- Network Security

### **What do you mean by DISCO?**

DISCO (Discovery), as the name suggests, is a Microsoft technology that is being used to discover web services. It is the process of locating and interrogating web service descriptions which is a preliminary step for having access to web services over the Internet. The organization that provides web services generally provides a DISCO file on its server that includes the links of all the available web services so that it can be used within the local network.

### **What's the difference between Web services and CORBA or DCOM?**

<b>Web service</b>	<b>CORBA and DCOM</b>
Web services basically transfer messages to applications or receive messages from applications using the HTTP protocol. To encode data, a web service uses XML.	They basically transfer messages to applications or receive messages from applications using non-standard protocols like RPC, IIOP (Inter Internet Object Protocol), etc.
WSDL is used to define web services.	CORBA Interface Description Language is used to define CORBA components and Microsoft Interface Definition Language is used to define DCOM components.
UDDI is used to discover web services.	CORBA registry is used to define CORBA components and DCOM registry is used to define DCOM components.
They are firewalls friendly.	CORBA uses IIOP protocol i.e., non-internet friendly.

## Explain BEEP?

**Answer:** BEEP stands for Blocks Extensible Exchange Protocol. BEEP is determined as building new protocols for the variety of applications such as instant messaging, network management, file transfer, etc. It is termed as new Internet Engineering Task Force (IETF) which is layered directly over TCP. It has some built-in features like

- Authentication
- Security
- Error handling
- Handshake Protocol

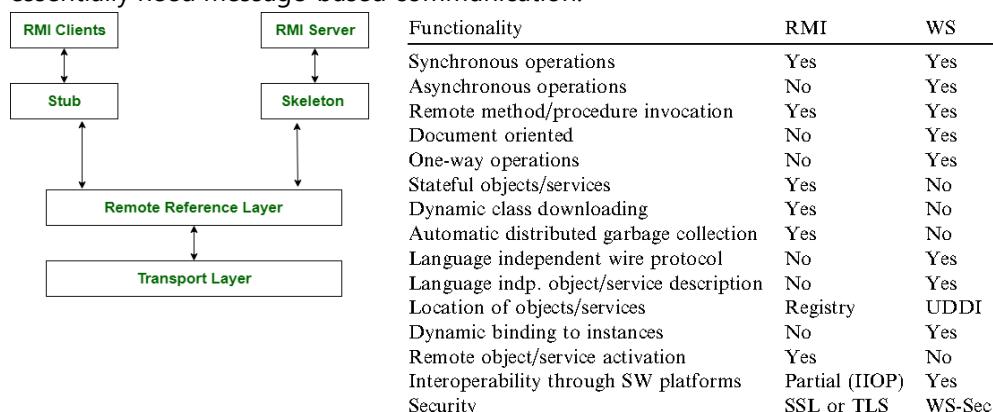
## What is gRPC? What benefits it offers over other Web service Alternatives

gRPC stands for **Google Remote Procedure Call** and is a variant based on the RCP architecture. Overall, gRPC aims to make data transmissions between microservices **faster**.

FEATURE	gRPC	REST
Protocol	HTTP/2 (fast)	HTTP/1.1 (slow)
Payload	Protobuf (binary, small)	JSON (text, large)
API contract	Strict, required (.proto)	Loose, optional (OpenAPI)
Code generation	Built-in (protoc)	Third-party tools (Swagger)
Security	TLS/SSL	TLS/SSL
Streaming	Bidirectional streaming	Client → server request only
Browser support	Limited (require gRPC-web)	Yes

## What is the difference between RMI and Web Services?

**RPC** stands for **Remote Procedure Call** which supports procedural programming. it's almost like IPC mechanism wherever the software permits the processes to manage shared information Associated with an environment wherever completely different processes area unit death penalty on separate systems and essentially need message-based communication.



### **What is WebServiceTemplate?**

The WebServiceTemplate is **the core class for client-side Web service access in Spring-WS**. It contains methods for sending Source objects, and receiving response messages as either Source or Result

### **How do you handle errors in Web Service call?**

We need to define our own exceptions based on our requirements to provide exact information to client.

### List out the new features available in Spring 4.0 and Spring 5.0?

- Spring 2.5 **Annotations/Autowire**
- Spring 3.0 **Java Configuration**
- Spring 4.0 is the first to support **Java 8 features**.
- Spring 5.0 has the support for **Reactive Programming**

### What types of Bean Scopes available in Spring ?

In Spring, scope can be defined using spring bean **@Scope** annotation. Let's quickly list down all six inbuilt bean scopes available to use in spring application context. These same scope apply to *spring boot bean scope* as well.

<b>singleton (default)</b>	Single bean object instance per spring IoC container
<b>prototype</b>	Opposite to singleton, it produces a new instance each and every time a bean is requested.
<b>request</b>	<b>A single instance</b> is created and available during complete lifecycle of an <a href="#">HTTPRequest</a> . Only valid in web-aware Spring ApplicationContext.
<b>session</b>	<b>A single instance</b> will be created and available during complete lifecycle of an <a href="#">HTTPSession</a> . Only valid in web-aware Spring ApplicationContext.
<b>application</b>	<b>A single instance</b> will be created and available during complete lifecycle of <a href="#">ServletContext</a> . Only valid in web-aware Spring ApplicationContext.

```
@Component  
@Scope("singleton / prototype/ request / ")  
public class User {  
}  
  
//or  
  
@Component  
@RequestScope / @SessionScope / @ApplicationScope  
public class User {  
}
```

### What is the default scope of bean in Spring framework? ([answer](#))

The default scope of a **Spring bean** is the **Singleton** scope and in the **WebApplication** 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.

### Does Spring Bean provide thread safety?

The default scope of Spring bean is *singleton*, so there will be *only one instance per context*. If two threads calls **factory.getBean("s")**, it returns same object. if any threads changes bean property by `setName()`, then other thread may get inconsistency results.

That means that all the having a class level variable that any thread can update will lead to inconsistent data. **Hence in default mode spring beans are not thread-safe.**

However, we can change spring bean scope to `request`, `prototype` or `session` to achieve thread-safety at the cost of performance. It's a design decision and based on the project requirements.

## What is Inversion of Control concept, how does Spring support IOC? (answer)

**Removing bean creation things from developer End.** The simple meaning of inversion of the control means that now the framework, Spring is responsible for creating objects, wiring dependencies and managing their life-cycle instead of a developer, which was the case before. That's where control is inverted from developer to framework.

### What is the difference between `@Autowired` and `@Inject` annotation in Spring?

The `@Inject` annotation also serves the same purpose as `@Autowired`, but the main difference between them is that

- `@Inject` is a **standard annotation(JSR -330)** for dependency injection
- `@Autowired` is **spring specific**.

### How to create ApplicationContext in a Java Program?

```
public class SpringDemo {  
    public static void main(String[] args) {  
        BeanFactory factory = new XmlBeanFactory(new ClassPathResource("sp.xml"));  
        Student student = (Student) factory.getBean("s");  
        System.out.println(student);  
  
        ApplicationContext context = new ClassPathXmlApplicationContext("sp.xml");  
        Student ob = (Student) context.getBean("s");  
        System.out.println(ob);  
  
        StudentDAO dao = (StudentDAO) context.getBean("dao");  
  
        dao.saveStudent(ob);  
    }  
}
```

- **AnnotationConfigApplicationContext:** If we are using Spring in standalone java applications and *using annotations for Configuration*, then we can use this to initialize the container and get the bean objects.
- **ClassPathXmlApplicationContext:** If we use `SpringConfig.xml` file in standalone application, then we can use this class to load the file and get the container object.
- **FileSystemXmlApplicationContext:** This is similar to ClassPathXmlApplicationContext except that the *xml configuration file can be loaded from anywhere in the file system*.

## **Name some of the design patterns used in Spring Framework?**

Spring Framework is using a lot of design patterns, some of the common ones are:

1. Singleton Pattern: Creating beans with default scope.
  2. Factory Pattern: Bean Factory classes
  3. Prototype Pattern: Bean scopes
  4. Adapter Pattern: Spring Web and Spring MVC
  5. Proxy Pattern: Spring Aspect Oriented Programming support
  6. Template Method Pattern: JdbcTemplate, HibernateTemplate etc
  7. Front Controller: Spring MVC DispatcherServlet
  8. Data Access Object: Spring DAO support
  9. Dependency Injection
- 10. Aspect Oriented Programming**

## **How to inject a java.util.Properties into a Spring Bean?**

In Spring reading properties file and setting property values can be done using-

- XML configuration
- Using **@PropertySource** Annotation

### **XML configuration**

You can configure property placeholders using **<context:property-placeholder>** in XML. The values to replace are specified as placeholders of the form  `${property-name}`

```
public class MySqlConfig { - this class properties will map in xml
    private String dbHost;
    private String dbPort;
    private String dbService;
    private String dbUrl;
    private String dbPassword;
}
```

```
db.host.url=db.java2novice.com
db.port.number=1521
db.service.name=test_db
db.user=java2novice
db.password=passw0rd@123
```

Xml based configuration file:

```
<beans>
    <context:property-placeholder location="classpath:db.properties" /> //location of prop file

    <bean id="dbConfig" class="com.java2novice.beans.MyDbConfig"> //class name to assign prop values
        <property name="dbHost" value="${db.host.url}" />
        <property name="dbPort" value="${db.port.number}" />
        <property name="dbService" value="${db.service.name}" />
        <property name="dbUrl" value="${db.user}" />
        <property name="dbPassword" value="${db.password}" />
    </bean>
</beans>
```

## **2.Using **@PropertySource** annotation**

```

@Configuration
@PropertySource("classpath:config/db.properties")
public class DBConfig {
    @Value("${db.driverClassName}")
    private String dbDriverClass;
    @Value("${db.url}")
    private String dbUrl;
    @Value("${db.username}")
    private String dbUser;
    @Value("${db.password}")
    private String dbPwd;
}

```

### 3. SpringBoot ConfigurationProperties with prefix

```

@ConfigurationProperties(prefix = "database")
public class Database {
    String url;
    String username;
    String password;
    // standard getters and setters
}

```

Spring Boot applies its convention over configuration approach again, automatically mapping between property names and their corresponding fields. All that we need to supply is the property prefix.

### How do you turn on annotation based autowiring?

- Include <context:annotation-config> in bean configuration file.
- Use **AnnotationConfigApplicationContext** to get Context Object.

### Differentiate between BeanFactory and ApplicationContext.

BeanFactory	ApplicationContext
It uses <b>Lazy initialization</b>	It uses <b>Eager/ Aggressive initialization</b>
It explicitly provides a resource object using the syntax	It creates and manages resource objects on its own
It <b>doesn't supports internationalization</b>	It supports internationalization
It <b>doesn't supports annotation based dependency</b>	It supports annotation-based dependency

### Can we have multiple Spring configuration files in one project?

You can load multiple Java-based configuration files:

```

@Configuration
@Import({MainConfig.class, SchedulerConfig.class})
public class AppConfig {
}

```

Or load one XML file that will contain all other configs:

```
ApplicationContext context = new ClassPathXmlApplicationContext("spring-all.xml");
```

And inside **this** XML file you'll have:

```

<import resource="main.xml"/>
<import resource="scheduler.xml"/>

```

### What's the difference between @Component, @Controller, @Repository & @Service annotations in Spring?

@Component or any Stereotype annotations **allows Spring to automatically detect our custom beans**.

In other words, without having to write any explicit code, Spring will: Scan our application for classes annotated with @Component. Instantiate them and inject any specified dependencies into them.

ANNOTATION	USE	DESCRIPTION
<b>@Component</b>	Type	Generic stereotype annotation for any Spring-managed component.
<b>@Controller</b>	Type	Stereotypes a component as a Spring MVC controller.
<b>@Repository</b>	Type	Stereotypes a component as a repository. Also indicates that SQLExceptions thrown from the component's methods should be translated into Spring DataAccessExceptions.
<b>@Service</b>	Type	Stereotypes a component as a service.

### What is ViewResolver in Spring?

ViewResolver implementations are used to resolve the view pages by name. Usually, we configure it in the spring bean configuration file. For example:

```
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
</beans:bean>
```

### What is View Resolver pattern? how it works in Spring MVC

View Resolver pattern is a J2EE pattern which allows a web application to dynamically choose its view technology e.g. HTML, JSP, Tapestry, JSF, XSLT or any other view technology.

In this pattern, View resolver holds mapping of different views, controller return name of the view, which is then passed to View Resolver for selecting an appropriate view.

### 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. That means Restful service doesn't require View page, it will directly show Response data.

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).

It clearly indicates that, this response doesn't have any view page. It just response in the form of text, String or JSON. By default, it returns in JSON formate. We can change response content type by using **produces="text/html"** etc

```
@RequestMapping(path = "/hello", method = RequestMethod.PUT , produces = "application/json;")
@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?

The **org.springframework.jdbc.core.simple.SimpleJdbcCall** class is mainly used to call a stored procedure with IN and OUT parameters

```
public Employee fetchEmployeeById(Integer employeeId) {
    SimpleJdbcCall simpleJdbcCall = getSimpleJdbcCall().withProcedureName("getEmployeeDetailsById");
    //Create INPUT Parameters & pass them to execute() method
    SqlParameterSource inputParameters = new MapSqlParameterSource("emp_id", employeeId);
    Map<String, Object> outputMap = simpleJdbcCall.execute(inputParameters);
```

```

Employee employee = new Employee();
if(outputMap != null) {
    employee.setEmployeeId(employeeId);
    employee.setEmail((String) outputMap.get("emp_email"));
    employee.setEmployeeName((String) outputMap.get("emp_name"));
    employee.setGender((String) outputMap.get("emp_gender"));
    employee.setSalary((Double) outputMap.get("emp_sal"));
}
return employee;
}

```

## How to get ServletContext and ServletConfig object in a Spring Bean?

There are two ways to get Container specific objects in the spring bean.

**Using @Autowired** annotation with bean variable of type ServletContext and ServletConfig. They will work only in servlet container specific environment only though. They already comes with Server jar

```

@.Autowired
ServletContext servletContext;

```

## How to upload file in Spring MVC Application?

Spring provides built-in support for uploading files through **MultipartResolver** interface implementations.

## Spring Data

### How to use Tomcat JNDI DataSource in Spring Web Application?

For using servlet container configured JNDI DataSource, we need to configure it in the spring bean configuration file and then inject it to spring beans as dependencies. Then we can use it with **JdbcTemplate** to perform database operations

```

<beans:bean id="dbDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <beans:property name="jndiName" value="java:comp/env/jdbc/MyLocalDB"/>
</beans:bean>

```

## 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);
    }
}
```

## Spring MVC

---

### Do you need spring-mvc.jar in your classpath or is it part of spring-core? (answer)

The `spring-mvc.jar` is not part of spring-core, which means if you want to use Spring MVC framework in your Java project, you must include `spring-mvc.jar` in your application's classpath. In Java web application, `spring-mvc.jar` is usually placed inside `/WEB-INF/lib` folder.

### What is the DispatcherServlet and what is it used for? (answer)

The DispatcherServlet is an implementation of Front Controller design pattern which handles all incoming web request to a Spring MVC application. A Front Controller pattern is a common pattern in web applications whose job is to receive all request and route it to different components of application for actual processing.

In case of Spring MVC, DispatcherServlet route web requests to Spring MVC controllers.

In Spring MVC, DispatcherServlet is used for finding the correct Controller to process a request, which it does with the help of handler mapping e.g. `@RequestMapping` annotation.

It is also responsible for delegating logical view name to ViewResolver and then sending the rendered response to the client.

### **Is the DispatcherServlet instantiated via an application context? ([answer](#))**

No, DispatcherServlet is instantiated by Servlet containers like Tomcat or Jetty. You must define DispatcherServlet into the web.xml file

### **What is the root applicationContext in Spring MVC? How is it loaded? ([answer](#))**

In Spring MVC, the context loaded using ContextLoaderListener is called the "root" application context which belongs to the whole application while the one initialized using DispatcherServlet is actually specific to that servlet.

Technically, Spring MVC allows multiple DispatcherServlet in a Spring MVC web application and so multiple such contexts each specific for respective servlet but having same root context may exist.

The ContextLoaderListener is configured in web.xml as listener and you put that inside a tag as shown below:

```
<listener>
<listener-class>
org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
```

When the Spring MVC web application is deployed, Servlet container creates an instance of ContextLoaderListener class which loads the Spring's WebApplicationContext

### **What is the @Controller annotation used for? How can you create a controller without an annotation? ([answer](#))**

The @Controller is a Spring MVC annotation to define Controller but in reality, it's just a stereotype annotation. You can even create a controller without @Controller by annotating the Spring MVC Controller classes using @Component annotation. **The real job is done by @RequestMapping - it will maps request to particular matched method.**

### **How is an incoming request mapped to a controller and mapped to a method? ([answer](#))**

Sometimes this question is also asked How does DispatcherServlet knows which Controller should process the request? Well, the answer lies in something called handler mappings.

Spring uses HandlerMapping to associate controllers with requests, commonly used handler mappings are BeanNameUrlHandlerMapping and SimpleUrlHandlerMapping. (**Bean → Class → URL**)

In BeanNameUrlHandlerMapping, when the request url matches the name of the bean, the class in the bean definition is the controller that will handle the request.

On the other hand, In SimpleUrlHandlerMapping, the mapping is more explicit. You can specify the number of URLs and each URL can be explicitly associated with a controller.

Btw, if you are using annotations to configure Spring MVC, which you should then @RequestMapping annotations is used to map an incoming request to a controller and a handler method.

### What are some of the valid return types of a controller method? (answer)

There are many return types available for a controller method in Spring MVC which is annotated by @RequestMapping inside the controller. Some of the popular ones are:

String	only view name
Void	
View	only view name
ModelAndView (Class)	addObject(K,V), setViewName()
Model (Interface)	Only Model Object
Map	JSON data
HttpEntity<?> or ResponseEntity<?>	Body, HttpStatus
HttpHeaders	

### What is the Model? (answer)

Model is a reference to encapsulate data or output for rendering. Model is always created and passed to the view in Spring MVC. If a mapped controller method has Model as a method parameter, then a model instance is automatically injected by Spring framework to that method.

Any attributes set on the injected model are preserved and passed to the View. Here is an example of using Model in Spring MVC:

```
public String personDetail(Model model) {  
...  
model.addAttribute("name", "Joe");  
...  
}
```

### What is the purpose of the session scope? (answer)

The purpose of the session scope is to create an instance of the bean for an HTTP Session. This means the same bean can serve multiple requests if it is scoped in session. You can define the scope of a Spring bean using scope attribute or `@Scope` annotation in Spring MVC application.

### What is the default scope in the web context? (answer)

Default scope of bean in spring is `singleton` and it is **no different in Web Application context**.

The other three Web context-aware scopes are a **request, session, and application**, which are only available in a web application aware ApplicationContext object.

### What are safe REST operations? (answer)

REST API uses HTTP methods to perform operations. Some of the HTTP operations which doesn't modify the resource at the server is known as safe operations e.g. **GET and HEAD**.

On the other hand, **PUT, POST, and DELETE** are **unsafe** because they modify the resource on the server

### What is Spring RestTemplate?

We use RestTemplate for accessing the REST API's inside a Spring application. Given that the RestTemplate class is a **synchronous client and designed to call REST services**. It has following methods.

- `getForObject(url, classType)` – retrieve a representation by doing a GET on the URL. The response (if any) is unmarshalled to given class type and returned.
- `getForEntity(url, responseType)` – retrieve a representation as *ResponseEntity* by doing a GET on the URL.
- `postForObject(url, request, classType)` – POSTs the given object to the URL, and returns the representation found in the response as given class type.
- `postForEntity(url, request, responseType)` – POSTs the given object to the URL, and returns the response as *ResponseEntity*.
- `exchange(url, httpMethod, requestEntity, responseType)` – execute the specified RequestEntity and return the response as *ResponseEntity*.

### **What are the advantages of the RestTemplate? (answer)**

The **RestTemplate** class is an implementation of Template method pattern in Spring framework. Similar to other popular template classes e.g. **JdbcTemplate** or **JmsTempalte**, it also simplifies the interaction with RESTful Web Services on the client side. i mean we can create client class to test webservice manulay using java.

You can use it to consume a RESTful Web Servicer very easily as shown in this example.

```
public class App implements CommandLineRunner {
    private static final Logger log = LoggerFactory.getLogger(App.class);

    public static void main(String args[]) {
        SpringApplication.run(App.class);
    }

    public void run(String... args) throws Exception {
        RestTemplate restTemplate = new RestTemplate();
        Response response = restTemplate.getForObject("localhost:9090/student/getall",
            Response.class);
        log.info("==== RESTful API Response using Spring RESTTemplate START =====");
        log.info(response.toString());
        log.info("==== RESTful API Response using Spring RESTTemplate END =====");
    }
}
```

### **What is an HttpMessageConverter in Spring REST? (answer)**

An **HttpMessageConverter** is a Strategy interface that specifies a converter that can convert from and to HTTP requests and responses. Spring REST uses this interface to convert HTTP response to various formats e.g. **JSON or XML**.

Each **HttpMessageConverter** implementation has one or several MIME Types associated with it. Spring uses the "**Accept**" header to determine the content type client is expecting.

It will then try to find a registered **HTTPMessageConverter** that is capable of handling that specific content-type and use it to convert the response into that format before sending to the client.

**to create a custom implementation of `HttpMessageConverter` to support a new type of request/responses,**

You just need to create an implementation of **AbstractHttpMessageConverter** and register it using the `WebMvcConfigurerAdapter#extendMessageConverters()` method with the classes which generate a new type of request/response.

#### **Is `@Controller` a stereotype? Is `@RestController` a stereotype? ([answer](#))**

Yes, both `@Controller` and `@RestController` are stereotypes. The `@Controller` is actually a specialization of Spring's `@Component` stereotype annotation. This means that class annotated with `@Controller` will also automatically be detected by Spring container as part of container's component scanning process.

And, `@RestController` is a specialization of `@Controller` for RESTful web service. It not only combines `@ResponseBody` and `@Controller` annotation but also gives more meaning to your controller class to clearly indicate **that it deals with RESTful requests.**

Spring Framework may also use this annotation to provide some more useful features related to REST API development in future.

#### **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.

It enables support for `@Controller`-annotated classes that use `@RequestMapping` to map incoming requests to handler methods.

#### **When do you need `@ResponseStatus` annotation in Spring MVC? ([answer](#))**

A good question for 3 to 5 years experienced spring developers. The `@ResponseStatus` annotation is required during error handling in Spring MVC and REST. Normally when an error or exception is thrown at server side, web server return a blanket HTTP status code 500 - Internal server error.

This may work for a human user but not for REST clients. You need to send them proper status code e.g. 404 if the resource is not found. That's where you can use `@ResponseStatus` annotation, which allows you to send custom HTTP status code along with proper error message in case of Exception.

For example, if you are writing a RESTful Web Service for a library which provides book information then you can use `@ResponseStatus` to create Exception which returns HTTP response code 404 when a book is not found instead of Internal Server Error (500), as shown below:

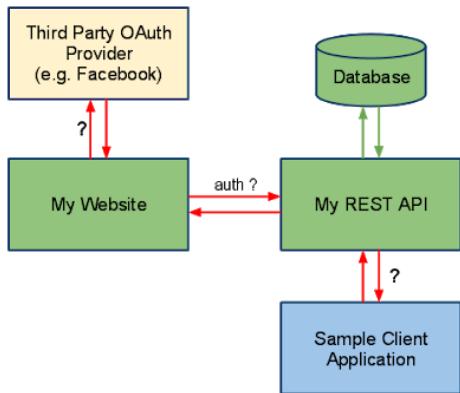
```
@ResponseStatus(value=HttpStatus.NOT_FOUND, reason="No such Book") // 404
public class BookNotFoundException extends RuntimeException {
    // ...
}
```

If this Exception is thrown from any handler method then HTTP error code 404 with reason "No such Book" will be returned to the client.

### Is REST secure? What can you do to secure it?

REST is normally not secure, but you can secure it by using **SpringSecurity**.

At the very least you can enable HTTP basic authentication by using HTTP in your Spring security configuration file. Similarly, you can expose your REST API using HTTPS if the underlying server supports HTTPS.



## SpringBoot IQ's

---

### What is the latest version of spring boot and its System requirement?

As per July 2018 Spring boot latest version is **2.1**.

Spring boot needs **Java 8+ version** and Spring 5 framework as minimum version.

As per Feb, 2022 Spring boot latest version is **2.6**.

Spring boot provides Java version options **Java 8, Java 11, Java 17**

### How does Spring enable create production ready applications in quick time?

Spring Boot aims to enable production ready applications in quick time. Spring Boot provides a few non functional features out of the box like caching, logging, monitoring and embedded servers.

- **spring-boot-starter-actuator** - To use advanced features like monitoring & tracing to your application out of the box
- **spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat** - To pick your specific choice of Embedded Servlet Container
- **spring-boot-starter-logging** - For Logging using logback
- **spring-boot-starter-cache** - Enabling Spring Framework's caching support

## What is the minimum baseline Java Version for Spring Boot 2 and Spring 5?

Spring Boot 2.0 requires Java 8 or later. Java 6 and 7 are no longer supported.

## What do Dev Tools in Spring boot mean?

By using devtools, You don't have to redeploy your application each time you influence the changes. The developer can reload the progressions without restart of the server. It maintains a strategic distance from the agony of redeploying application each time when you roll out any improvement. This module will can't be utilized in a production environment.

## Spring DevTools & uses

Applications that use spring-boot-devtools will **automatically restart whenever files on the classpath change**. This can be a useful feature when working in an IDE as it gives a very fast feedback loop for code changes.

## Would we be able to Use Spring Boot with Applications Which Are Not Using Spring?

No, it isn't conceivable starting at now. Spring boot is restricted to Spring applications only.

## How to connect to an external database like MySQL or oracle with Spring boot?

**Step 1** -The first step to connect the database like Oracle or MySql is adding the dependency for your database connector to pom.xml.

**Step 2** -The next step is the elimination of H2 Dependency from pom.xml

**Step 3** -Step 3 includes the schema and table to establish your database.

**Step 4** -The next step is configuring of the database by using Configure *application.properties* to connect to your database.

**Step 5**-And the last step is to restart your device and your connection is ready to use.

## How to disable a specific auto-configuration?

If we want to disable a specific auto-configuration, we can indicate it using the **exclude attribute** of the **@EnableAutoConfiguration** annotation. For instance, this code snippet neutralizes DataSourceAutoConfiguration:

```
// other annotations  
@EnableAutoConfiguration(exclude = DataSourceAutoConfiguration.class)  
public class MyConfiguration { }
```

If we enabled auto-configuration with the **@SpringBootApplication** annotation — which has **@EnableAutoConfiguration** as a meta-annotation — we could disable auto-configuration with an attribute of the same name:

```
@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
```

```
public class MyConfiguration { }
```

We can also disable an auto-configuration with the `spring.autoconfigure.exclude` environment property in the `application.properties`

```
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration
```

## How to deploy Spring Boot web applications to WebSphere as a WAR?

spring-boot-maven-plugin handles the Packing things.

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

If we mentioned in our POM.xml packaging as jar it will generate Jar by including Tomcat server as well.

To Build WAR with out Tomact we need to below 2 Changes

### 1.In POM.xml – package type as ‘war’

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.3.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.example</groupId>
<artifactId>demo</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>demo</name>
<description>Demo project for Spring Boot</description>
```

### 2.Remove tomcat starter Dependency

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
```

## What is Spring Boot DevTools used for?

Spring Boot Developer Tools, or DevTools, is a set of tools making the development process easier.

The `spring-boot-devtools` module is automatically disabled if the application runs in production

By default, DevTools applies properties suitable to a development environment. These properties disable template caching, enable debug logging for the web group, and so on. As a result, we have this sensible development-time configuration without setting any properties.

Applications using DevTools restart whenever a file on the classpath changes. This is a very helpful feature in development, as it gives quick feedback for modifications.

By default, static resources, including view templates, don't set off a restart. Instead, a resource change triggers a browser refresh. Notice this can only happen if the LiveReload extension is installed in the browser to interact with the embedded LiveReload server that DevTools contains.

### What is LiveReload?

The spring-boot-devtools module includes an **embedded LiveReload server that can be used to trigger a browser refresh when a resource is changed**. LiveReload browser extensions are freely available for Chrome, Firefox and Safari from livereload.com.

### How to exclude auto restart for static files?

By default changing resources in **/META-INF/maven, /META-INF/resources, /resources, /static, /public or /templates** will not trigger a restart.

But If you want to customize these exclusions you can use the **spring.devtools.restart.exclude** property.

If you want to keep those defaults and add additional exclusions, use the **spring.devtools.restart.additional-exclude** property instead.

### What is Hot swapping in spring boot?

**Reloading the changes without restarting the server is called hot swapping**, Modern IDEs (Eclipse, IDEA, etc.) all support hot swapping of bytecode, so if you make a change that doesn't affect the class or method signatures it should reload cleanly with no side effects.

### How to write integration tests?

When you create Project using Spring.io, by default test class for Application class also will created. It is annotated with **@SpringBootTest**

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class DemoApplicationTests {

    @Test
    public void contextLoads() {
    }
}
```

### What is Spring Boot Actuator?

Spring Boot provides actuator to monitor and manage our application. **Actuator is a tool which has HTTP endpoints**. when application is pushed to production, you can choose to manage and monitor your application using HTTP endpoints.

**The actuator provides features like auditing, health, metrics, environment information, thread dump etc.** using HTTP or JMX endpoints. Here are some of the most common built-in actuator endpoints:

- **beans – Displays a complete list of all the Spring beans in your application.**
- **auditevents – Exposes audit events information for the current application.**

- *caches* – Exposes available caches.
- *configprops* – Displays a collated list of all @ConfigurationProperties.
- *health* – **Shows application health information.**
- *info* – Displays arbitrary application info.
- *metrics* – Shows ‘metrics’ information for the current application.
- *mappings* – **Displays a collated list of all @RequestMapping paths.**
- *sessions* – Allows retrieval and deletion of user sessions from a Spring Session-backed session store.
- *threaddump* – Performs a thread dump.

### How do you Change tomcat HTTP port & Context URL?

You can change the Tomcat HTTP port by changing default HTTP property in the **application.properties**

```
Server.port = 8080
Server.context.path = /
```

### Can you control logging with Spring Boot? How?

Yes, we can control logging with Spring Boot by specifying log levels on **application.properties** file.

Spring Boot loads this file when it exists in the classpath and it can be used to configure both Spring Boot and application code.

Spring Boot uses Commons Logging for all internal logging and you can change log levels by adding following lines in the **application.properties** file:

```
logging.level.org.springframework=DEBUG
logging.level.com.demo=INFO
```

### What is YAML ?

YAML is a human-readable data serialization language. It is commonly used for configuration files.

Compared to properties file, YAML file is much more structured and less confusing in case we want to add complex properties in the configuration file. As can be seen YAML has hierarchical configuration data

### How to set the active profile in Spring Boot?

There are two ways to set the active profile in Spring Boot.

Pass in the active profile as an argument while launching the application.

```
java -jar -Dspring.profiles.active=production application-1.0.0-RELEASE.jar //pass as command line argument
```

Use the `application.properties` file to set the active profile.

```
spring.profiles.active=production
```

### How to return Different Data Formats (JSON, XML) from Spring REST API

We will return XML and JSON format depending on your HTTP Headers. By default, the REST will send you the JSON response.

In HTTP Headers You need to add key `Accept`'s value in to `text/xml` or `application/xml` to get the response in XML format

```
@GetMapping(value = "/ex/allcontent",
    produces = {
        MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE,
        MediaType.TEXT_XML_VALUE
    })
public List<TestReport> getEmployees() {
    return repository.findAll();
}
```

## SpringCloud – Microservices

### Summary of MicroServices

Order of Start Services

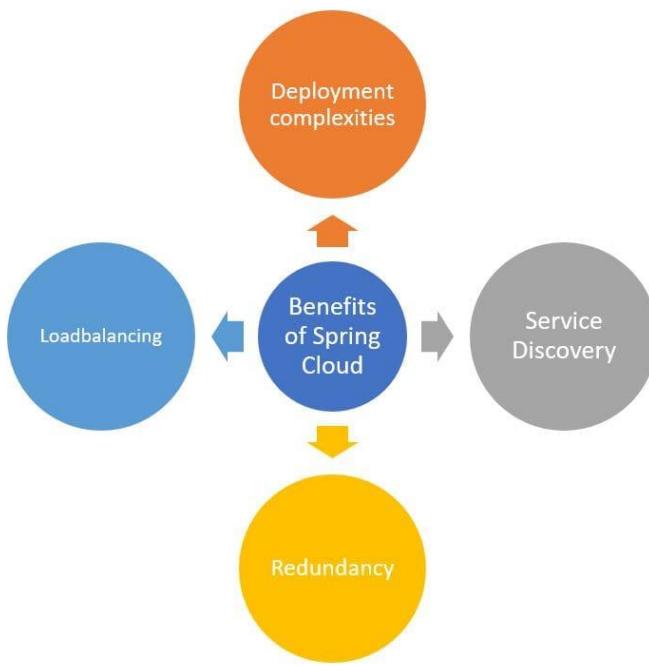
service-registry	Eureka Service Discovery Server	<a href="http://localhost:8761/">http://localhost:8761/</a>
Hystrix- Dashboard(Optional)	CirCuit Breaker/ Falut tolrence Dashboard only	<a href="http://localhost:8788/">http://localhost:8788/</a>
IDamApiGateway	Zuul API Gateway	<a href="http://localhost:8099">http://localhost:8099</a>
Sleuth and Zipkin	Distributed Tracing. <ul style="list-style-type: none"><li>Just adding dependency logs will generated with TracelID</li><li>Run Server cd C:\kafka <code>java -jar zipkin.jar</code></li></ul>	<a href="http://127.0.0.1:9411/">http://127.0.0.1:9411/</a>
UserServiceWithKafka	UserProisionKafkaser	<a href="http://localhost:8051/">http://localhost:8051/</a>
EDCMicroService		<a href="http://localhost:8031/edc/">http://localhost:8031/edc/</a>
MIMicroService		<a href="http://localhost:8041/mi/">http://localhost:8041/mi/</a>

Eureka Service Discovery Server	<ul style="list-style-type: none"> <li>Separate Server</li> <li>Add spring-cloud-starter-netflix-eureka-server dependency</li> <li>Add <b>@EnableEurekaServer</b> to main class</li> <li>Update <b>application.properties</b> with port &amp; other details</li> </ul> <p><b><u>EDC, MI Microservices End</u></b></p> <ul style="list-style-type: none"> <li>Add the <b>Eureka client</b> dependency <b>spring-cloud-eureka-client</b>.</li> <li>Add Eureka Server URL in <b>application.properties</b></li> <li><b>eureka.client.service-url.defaultZone=http://localhost:8761/eureka/</b></li> <li>By adding this, at start up they will register to eureka servre</li> </ul> <p>If EDC has 4 Nodes, MI has 4 nodes Eureka Server will take care about loadbalance.</p> <ul style="list-style-type: none"> <li>Add <b>@LoadBalanced</b> to SpringBoot Main Class</li> <li><b>RestTemplate</b> I used to call other MicroService call.</li> <li>The RestTemplate with <b>@LoadBalanced</b> annotation will internally use <b>Ribbon LoadBalancer</b> to resolve the <b>ServiceID</b> and invoke REST endpoint using one of the available servers</li> </ul>
Hystrix- Circuit Breaker/ Fault tolerance	<p>In above we are calling other microservice using <b>RestTemplate</b>. If microservice down, it will have fallbackmethod to handle those situations.</p> <ul style="list-style-type: none"> <li>Add <b>Hystrix</b> Dependency : <b>spring-cloud-starter-netflix-hystrix</b></li> <li>Enable Hystrix functionality in our EDC-Microservice, by adding <b>@EnableHystrix</b> on our main SpringBoot main Class</li> <li>Use <b>@HystrixCommand(fallbackMethod = "miReportFallBackMethod")</b> to define fallback method if called microservice is down or unreachable</li> </ul> <pre> ResponseEntity r= restTemplate.getForEntity("http://MI-MICROSERVICE/mi/anthology/all", Object.class);</pre>
Zuul API Gateway	<ul style="list-style-type: none"> <li>Separate Server</li> <li>Add spring-cloud-starter-netflix-zuul dependency</li> <li>Add <b>@EnableZuulProxy</b> to main class</li> <li>Update <b>application.yaml</b> file with Route Details</li> </ul> <pre># WITH Eureka Server zuul:   routes:     edcservice:       path: /edc/**       serviceId: EDC-MICROSERVICE      miservice:       path: /mi/**       serviceId: MI-MICROSERVICE</pre>

Sleuth and Zipkin	<p>Distributed Tracing.</p> <ul style="list-style-type: none"> <li>• Just adding dependency logs will be generated with TraceID</li> <li>• Run Server cd C:\kafka <code>java -jar zipkin.jar</code></li> </ul> <p>Zipkin is UI for log Tracing. We need to follow the steps in our microservices to push Distributed Trace logs to ZipKin</p> <ul style="list-style-type: none"> <li>• Add <code>spring-cloud-starter-sleuth</code> Dependency in (EDC, MI) to enable Sleuth</li> <li>• Add <code>spring-cloud-starter-zipkin</code> to push logs to ZipKin</li> <li>• Update application.props with zipkin server details</li> </ul> <pre><code>spring.zipkin.base-url=http://localhost:9411/ spring.sleuth.sampler.probability=1</code></pre>
UserServiceWithKafka	<p>ProducerService :</p> <ul style="list-style-type: none"> <li>• Add kafka dependencies</li> <li>• <code>@EnableKafka</code> in SpringBoot main class.</li> <li>• Update application.props with Server Details &amp; Topic name <code>topic.name.producer=user.provision.topic</code></li> <li>• Use <code>kafkaTemplate.send(kafkaTopic, data);</code> to send msg to Topic</li> </ul> <p>ConsumerService:</p> <ul style="list-style-type: none"> <li>• Add kafka dependencies</li> <li>• <code>@EnableKafka</code> in SpringBoot main class.</li> <li>• Update application.props with Server Details &amp; Topic name <code>topic.name.consumer=user.provision.topic</code></li> <li>• Use <code>@KafkaListener</code> to listen topic &amp; Consume messages</li> </ul> <pre><code>@KafkaListener(topics="\${topic.name.consumer}", groupId = "group_id") public void consume(ConsumerRecord&lt;String, String&gt; msg) {     System.out.println("Tópico: {}" + msg); }</code></pre>
EDCMicroService	
MIMicroService	

### Explain Spring cloud? or, What is Spring Cloud?

Spring cloud is a set of tools that can be used by developers to quickly build some common patterns in distributed systems such as service discovery, configuration management, intelligent routing, etc.



### What are the advantages of using Spring Cloud?

A: When developing distributed microservices with Spring Boot we face the following issues-

- **Complexity associated with distributed systems-**  
This overhead includes network issues, Latency overhead, Bandwidth issues, security issues.
- **Service Discovery-** Service discovery tools manage how processes and services in a cluster can find and talk to one another. It involves a directory of services, registering services in that directory, and then being able to lookup and connect to services in that directory.
- **Loadbalancing-**  
Load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives.
- **Performance issues-**  
Performance issues due to various operational overheads.

### What does one mean by Service Registration and Discovery? How is it implemented in Spring Cloud ?

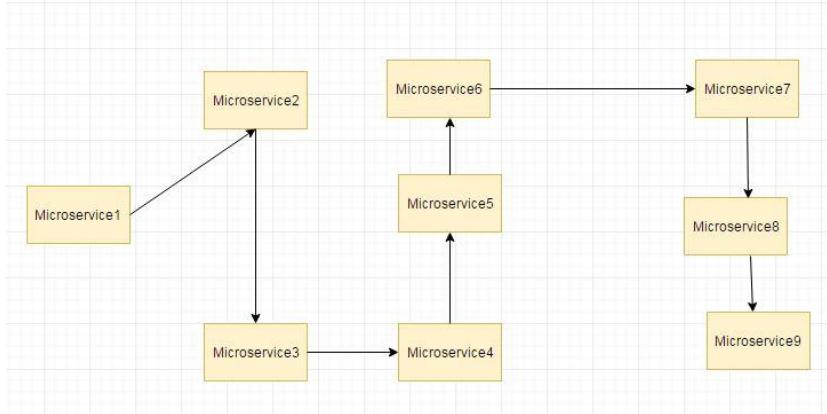
- When we start a project, we usually have all the configurations in the **properties** file. We will configure all required microservices URLs in properties file. When more and more services are developed and deployed then adding and modifying these properties become more complex. Some services might go down, while some the location might change. This manual changing of properties may create problems.

- Eureka Service Registration and Discovery helps in such cases. As all services are registered to the Eureka server and lookup done by calling the Eureka Server, any change in service locations need not be handled and is taken care of Microservice Registration and Discovery with Spring cloud using Netflix Eureka.

### What is Hystrix? How does it implement Fault Tolerance?

Usually for systems developed using Microservices architecture, there are many microservices involved. These microservices collaborate with each other.

Consider the following microservices-



Suppose if the microservice 9 in the above diagram failed, then using the traditional approach we will propagate an exception. But this will still cause the whole system to crash anyways.

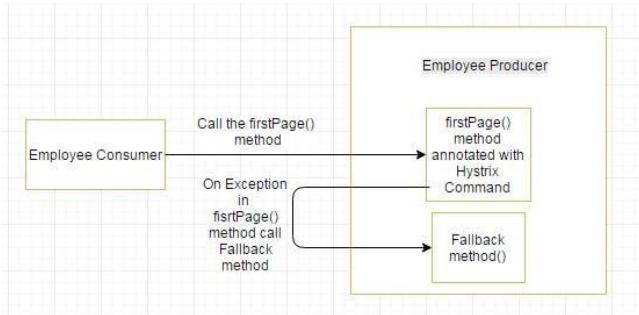
There is a Fallback method feature of Hystrix for this scenario. We have two services **employee-consumer** consuming the service exposed by **the employee-producer**.



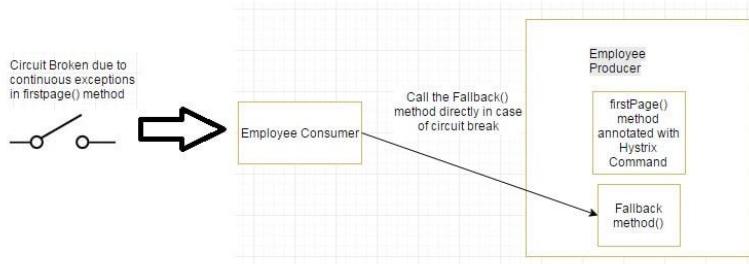
Now suppose due to some reason the employee-producer exposed service throws an exception. In this case using **Hystrix we define a fallback method**. This fallback method **should have the same return type** as the exposed service. In case of exception in the exposed service the fallback method will return some value

### What is Hystrix Circuit Breaker? Need for it?

Due to some reason the **employee-producer** exposed service throws an exception. In this case using Hystrix we defined a fallback method. In case of exception in the exposed service the fallback method returned some default value.



If the exceptions keep on occurring in the firstPage method() then the Hystrix circuit will break and the employee consumer will skip the firstPage method all together and directly call the fallback method.



### What are some common Spring cloud annotations? ([answer](#))

Answer: here is a list of some of the most essential Spring cloud annotations for Java developers

Spring Cloud has mainly following 5 main Annotations:

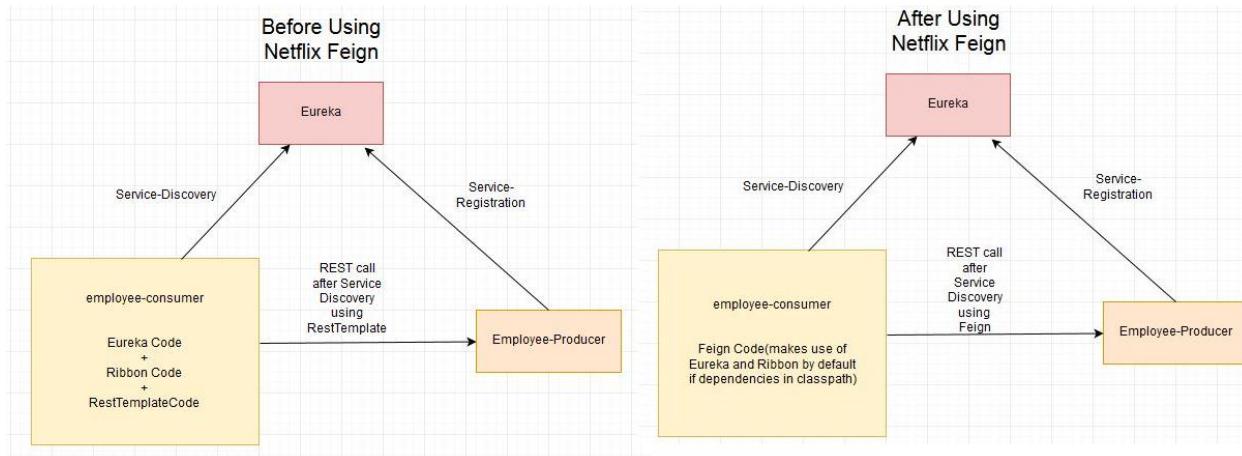
- **@EnableConfigServer**: This annotation converts the application into the server which is more applications use to get their configuration.
- **@EnableEurekaServer**: This annotation used for Eureka Discovery Services for other applications can use to locate services using it.
- **@EnableDiscoveryClient**: Help of this annotation application register in the service discovery, it discovers other services using it.
- **@EnableCircuitBreaker**: Use the Circuit Breaker pattern to continue operating when related services fail and prevent cascading failure. This Annotation is mainly used for Hystrix Circuit Breaker.
- **@HystrixCommand(fallbackMethod="fallbackMethodName")**: it is used to mark the method for fall back to another method if they cannot success normally.

## What is Zuul?

Zuul is an L7 application gateway that provides capabilities for dynamic routing, monitoring, resiliency, security, and more. Zuul is a JVM-based router and server-side load balancer developed by Netflix and available in the Netflix OSS package.

## What is the Netflix Feign Client? Need for it?

REST call using Netflix Feign Client. Previous examples in the employee-consumer we consumed the REST services exposed by the employee-producer using **RESTTemplate**.



We next define a Feign Client by creating an interface with `@FeignClient` annotation. We also specify the name value as "`employee-producer`". This value is the name of the service registered using Eureka for discovery. We define the method call to be made to consume the REST service exposed by the **employee-producer** module.

```
@FeignClient(name="employee-producer")
public interface RemoteCallService {
    @RequestMapping(method=RequestMethod.GET, value="/employee")
    public Employee getData();
}
```

Next, we autowire the `RemoteCallService` in the `ConsumerController` class. Load Balancing is automatically taken care by Feign Client.

```
@Controller
public class ConsumerControllerClient {

    @Autowired
    private RemoteCallService loadBalancer;

    public void getEmployee() throws RestClientException, IOException {
        try {
            Employee emp = loadBalancer.getData();
            System.out.println(emp.getEmpId());
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
}
```

Finally, we annotate the Spring Boot Main class with `@EnableFeignClients`.

# SQL

**Write SQL query to find second highest salary in employee table?**

```
SELECT MAX(Salary) FROM Employee WHERE Salary NOT IN (SELECT MAX(Salary) FROM Employee)
```

This query first finds maximum salary and then exclude that from the list and again finds maximum salary. Obviously second time, it would be second highest salary.

## Difference between WHERE vs HAVING clause in SQL - GROUP BY Comparison with Example

main difference between WHERE and HAVING clause in SQL is that, condition specified in WHERE clause is used while fetching data (rows) from table, on the other hand HAVING clause is later used to filter summarized data or grouped data.

```
SELECT * FROM Employee;
```

EMP_ID	EMP_NAME	EMP_AGE	EMP_SALARY	DEPT_ID
1	Virat	23	10000	1
2	Rohit	24	7000	2
3	Suresh	25	8000	3
4	Shikhar	27	6000	1
5	Vijay	28	5000	2

```
SELECT * FROM Department;
```

DEPT_ID	DEPT_NAME
1	Accounting
2	Marketing
3	Sales

```
SELECT d.DEPT_NAME, count(e.EMP_NAME) as NUM_EMPLOYEE, avg(e.EMP_SALARY) as AVG_SALARY
FROM Employee e,Department d
WHERE e.DEPT_ID=d.DEPT_ID
AND EMP_SALARY > 5000
GROUP BY d.DEPT_NAME;
```

DEPT_NAME	NUM_EMPLOYEE	AVG_SALARY
Accounting	1	8000
Marketing	1	7000
Sales	2	8000

From the number of employee (NUM\_EMPLOYEE) column you can see that only Vijay who work for Marketing department is not included in result set because his earning 5000. This example shows that, condition in WHERE clause is used to filter rows before you aggregate them.

```
SELECT d.DEPT_NAME, count(e.EMP_NAME) as NUM_EMPLOYEE, avg(e.EMP_SALARY) as AVG_SALARY
FROM Employee e,Department d
WHERE e.DEPT_ID=d.DEPT_ID
AND EMP_SALARY > 5000
GROUP BY d.DEPT_NAME
HAVING AVG_SALARY > 7000;
```

DEPT_NAME	NUM_EMPLOYEE	AVG_SALARY
Accounting	1	8000
Sales	2	8000

then HAVING clause comes in picture for final filtering, which is clear from following query, now Marketing department is excluded because it doesn't pass condition in HAVING clause i.e AVG\_SALARY > 7000

## ReactJS

---

<https://www.simplilearn.com/tutorials/reactjs-tutorial/reactjs-interview-questions>

## Angular

---

# AngularJs

Note : we must write module, if we used ng-app

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<div ng-app="myApp" ng-controller="myCtrl">

First Name: <input type="text" ng-model="firstName"><br> //binds input with model
Last Name: <input type="text" ng-model="lastName"><br> ///binds input with model
Full Name: {{firstName + " " + lastName}}           //Prints the model values

//same as {{expression}}

<p ng-bind="firstName"></p>
<p ng-bind="lastName "></p>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.firstName= "John";
    $scope.lastName= "Doe";

    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    };
});
</script>
```

## Internal Working

- The **AngularJS JavaScript** file is loaded, and the Angular global object `$scope` is created. The JavaScript file that registers the controller functions is executed.
- AngularJS scans the HTML to look for **AngularJS apps and views** and finds **a controller function corresponding to the view**.
- AngularJS **executes the controller functions and updates the views** with data from the model populated by the controller.
- AngularJS listens for browser events, such as button clicked, mouse moved, input field being changed, and so on. If any of these events happen, then AngularJS will update the view accordingly

## More

displays only the names containing the letter "i".

```
<li ng-repeat="x in names | filter : 'i'">
  {{ x }}
</li>

<li ng-repeat="x in names | orderBy:'country'">
  {{ x.name + ', ' + x.country }}
</li>
```

## AngularJS Services

- In AngularJS, a service is a **function, or object**, that is available for your AngularJS application.
- AngularJS has about 30 built-in services. One of them is the **\$location** service.

### 1. \$location

returns information about the location of the current web page:

```
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $location) {
  $scope.myUrl = $location.absUrl();
});

{{myUrl}} //prints
https://www.w3schools.com/angular/tryit.asp?filename=try_ng_services
```

### \$2. \$http

**\$http** is an AngularJS service for reading data from remote servers. The AngularJS **\$http** service makes a request to the server, and returns a response.

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm").then(function (response) {
    $scope.myWelcome = response.data;
  });
});
```

## More on Http

The example above uses the **.get** method of the **\$http** service.

The `.get` method is a shortcut method of the `$http` service. There are several shortcut methods:

- `.get()`
- `.post()`
- `.put()`
- `.delete()`
- `.head()`
- `.jsonp()`
- `.patch()`

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http({
    method : "GET",
    url : "welcome.htm"
  }).then(function mySuccess(response) {
    $scope.myWelcome = response.data;
  }, function myError(response) {
    $scope.myWelcome = response.statusText;
  });
});
```

## AngularJS Events

You can add AngularJS event listeners to your HTML elements by using one or more of these directives:

- `ng-blur`
- `ng-change`
- `ng-click`
- `ng-copy`
- `ng-cut`
- `ng-dblclick`
- `ng-focus`
- `ng-keydown`
- `ng-keypress`
- `ng-keyup`
- `ng-mousedown`
- `ng-mouseenter`

```
<div ng-app="myApp" ng-controller="myCtrl">
  <button ng-click="myFunction()">Click me!</button>
  <p>{{ count }}</p>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.count = 0;
```

```
$scope.myFunction = function() {
  $scope.count++;
}
});
```

## AngularJS Routes

If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the `ngRoute` module.

```
var app = angular.module('userregistrationsystem', [ 'ngRoute', 'ngResource' ]);

app.config(function($routeProvider) {
  $routeProvider.when('/list-all-users', {
    templateUrl : '/template/listuser.html',
    controller : 'listUserController'
  }).when('/register-new-user',{
    templateUrl : '/template/userregistration.html',
    controller : 'registerUserController'
  }).when('/update-user/:id',{
    templateUrl : '/template/userupdation.html' ,
    controller : 'usersDetailsController'
  }).otherwise({
    redirectTo : '/home',
    templateUrl : '/template/home.html',
  });
});
```

### Name the key features of AngularJS?

The key features of AngularJS are:

- Scope
- Controller
- Model
- View

- Services
- Data Binding
- Directives
- Filters
- Testable

### Can AngularJS have multiple ng-app directives in a single page?

No. Only one AngularJS application can be auto-bootstrapped per HTML document. The first ngApp found in the document will be used to define the root element to auto-bootstrap as an application. If another ng-app directive has been placed then it will not be processed by AngularJS and we will need to manually bootstrap the second app, instead of using second ng-app directive.

### Explain the architecture of AngularJS?

AngularJS is architecture on 3 components. They are

- The Template (View)
- The Scope (Model)
- The Controller (Controller)

### Explain Directives in AngularJs?

AngularJS directives are extended HTML attributes with the prefix ng-

The 3 main directives of angular js are

- **ng-app:-** directive is used to flag the HTML element that Angular should consider to be the root element of our application. Angular uses spinal-case for its custom attributes and camelCase for the corresponding directives which implement them.
- **ng-model:-** directive allows us to bind values of HTML controls (input, select, textarea) to application data. When using ngModel, not only change in the scope reflected in the view, but changes in the view are reflected back into the scope.
- **ng-bind:-** directive binds application modal data to the HTML view.
- **ng-controller**
- **ng-view**

### **Explain AngularJS digest cycle?**

AngularJS digest cycle is the process behind Angular JS data binding.

In each digest cycle, Angular compares the old and the new version of the scope model values. The digest cycle is triggered automatically. We can also use `$apply()` if we want to trigger the digest cycle manually.

### **What is data binding in AngularJS and What is the difference between one-way and two-way binding?**

Data binding is the automatic attunement of data between the view and model components. AngularJS uses two-way data binding. In one-way binding, the scope variable in the html is set to the first value that its model is assigned to.

In two-way binding, the scope variable changes its value every time its model binds to a different value.

### **Explain what a digest cycle is in AngularJS?**

During every digest cycle, all new scope model values are compared against the previous values. This is called dirty checking. If change is detected, watches set on the new model are fired and another digest cycle executes. This goes on until all models are stable.

The digest cycle is triggered automatically but it can be called manually using `".$apply()"`.

### **What is Single Page Application (SPA)?**

SPA is the concept whereby pages are loaded from the server not by doing post backs, rather by creating a single shell page or master page and loading the web pages into the master page.

### **How can SPA be implemented in AngularJS?**

SPA can be implemented with Angular by using Angular routes

## **JUnit**

---

### **How to create Parameterized tests?**

Answer:

There are five steps to create Parameterized tests–

- First, test class is annotated with `@RunWith` which is a `Parameterized.class`.

- Then create a public static method which is annotated with @Parameters. it returns a Collection of Objects as test data set.
- Next, create a public constructor which takes in one row of test data.
- Create an instance variable that is for each column of the test data row.
- Create tests case(s) using the instance variables as a source of the test data.
- The test case invokes once per each row of data.

### **What are JUnit classes? List some of them?**

JUnit classes are important classes which are used in writing and testing JUnits. Some of the important classes are:

- Assert – A set of assert methods.
- Test Case – It defines the fixture to run multiple tests.
- Test Result – It collects the results of executing a test case.
- Test Suite – It is a Composite of Tests.

## MongoDB vs SQL

SQL Terms/Concepts	MongoDB Terms/Concepts
database	<u>database</u>
table	<u>collection</u>
row	<u>document</u> or <u>BSON</u> document
column	<u>field</u>
index	<u>index</u>

<b>SQL Terms/Concepts</b>	<b>MongoDB Terms/Concepts</b>
table joins	<u>\$lookup</u> , embedded documents
primary key  Specify any unique column or column combination as primary key.	<u>primary key</u>  In MongoDB, the primary key is automatically set to the <u>_id</u> field.

<https://docs.mongodb.com/manual/reference/sql-comparison/>

## Read this, Don't forget important

# Real

Is Hibernate is slow? Did you face any memory issues?

Better use `FetchType.LAZY` instead.

The N+1 problem

Supplier with a one-to-many relationship with Product. One Supplier has (supplies) many Products.

***** Table: Supplier *****	
ID	NAME
1	Supplier Name 1
2	Supplier Name 2
3	Supplier Name 3
4	Supplier Name 4

***** Table: Product *****				
ID	NAME	DESCRIPTION	PRICE	SUPPLIERID
1	Product 1	Name for Product 1	2.0	1
2	Product 2	Name for Product 2	22.0	1
3	Product 3	Name for Product 3	30.0	2
4	Product 4	Name for Product 4	7.0	3

Factors:

- Lazy mode for Supplier set to "true" (default)
- Fetch mode used for querying on Product is Select
- Fetch mode (default): Supplier information is accessed
- Caching does not play a role for the first time the

Supplier is accessed

Fetch mode is Select `Fetch.Eager` (default)

```
// It takes Select fetch mode as a default
Query query = session.createQuery( "from Product p");
List list = query.list();
// Supplier is being accessed
displayProductsListWithSupplierName(results);
```

```
select ... various field names ... from PRODUCT
select ... various field names ... from SUPPLIER where SUPPLIER.id=?
select ... various field names ... from SUPPLIER where SUPPLIER.id=?
select ... various field names ... from SUPPLIER where SUPPLIER.id=?
```

Result:

- 1 select statement for Product
- N select statements for Supplier

This is N+1 select problem!

## The Solution

- Avoiding Eager Fetching
- Only fetching the data that are actually needed

```
jpaQuery = entityManager.createQuery("SELECT P FROM PurchaseOrder P WHERE P.customerId = :customerId", PurchaseOrder.class);
jpaQuery.setParameter("customerId", "Sayem")
```
- We can use JOIN FETCH in our queries whenever we need to fetch an entity with all of its children at the same time. This results in a much less database traffic resulting in an improved performance.

### Clone() will create new Object of Singleton Class?

Cloning is a concept to create duplicate objects. **Using clone we can create copy of object.** Suppose, we create clone of a singleton object, then it will create a copy that is there are two instances of a singleton class, **hence the class is no more singleton.**

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

    Student s1 = Student.getObject();
    Student s2 = Student.getObject();

    Student s3 = (Student) s1.clone();
    System.out.println(s1);
    System.out.println(s1);
    System.out.println(s3);

}

Student@15db9742
Student@15db9742
Student@6d06d69c // Creates new Object, our singleton failed.
```

To overcome this, we should override clone() method, it should throw Exception, anyone tries to do clone

```
class Student implements Cloneable{
    .....
    @Override
    protected Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException();
    }

    public static void main(String args[]) throws CloneNotSupportedException {
        Student s1 = Student.getObject();
        Student s2 = Student.getObject();

        Student s3 = (Student) s1.clone();
        System.out.println(s1);
        System.out.println(s1);
        System.out.println(s3);

    }
}
Exception in thread "main" java.lang.CloneNotSupportedException //We are GOOD now
at Student.clone(Student.java:25)
at Student.main(Student.java:33)
```

**Reflection:** You can make the new instance of the Singleton class by changing the **constructor visibility as public** in run-time and create new instance using that constructor .

```

public static void main(String args[]) throws InstantiationException, IllegalAccessException, InvocationTargetException {
    Student s1 = Student.getObject();
    Student s2 = null;

    //1.Making Constructor visible
    Constructor<Student>[] constructors = (Constructor<Student>[])
    Student.class.getDeclaredConstructors();
    for (Constructor constructor : constructors)
    {
        // Below code will destroy the singleton pattern
        constructor.setAccessible(true);
        s2 = (Student) constructor.newInstance();

    }
    System.out.println(s1);
    System.out.println(s2);

    //Using Class of newInstance()
    Class c = Student.class;
    Student s1 = Student.getObject();
    Student s2 = (Student) c.newInstance();

    System.out.println(s1); // Student@15db9742
    System.out.println(s2); // Student@6d06d69c -Failed again ⚡
}
Student@15db9742
Student@6d06d69c //Failed again ⚡

```

To prevent Singleton failure while due to reflection you have to **throw a run-time exception in constructor**, if the constructor is already initialized.

```

class Student implements Cloneable{
    private static Student st;

    private Student() {
        if(st!=null)
            throw new RuntimeException("Go Fucker.....");
    }
Exception in thread "main" java.lang.RuntimeException: Go Fucker.....  

    at Student.<init>(Student.java:15)  

    at sun.reflect.NativeConstructorAccessor

```

### Sum of multiple combinations of Integer array?

```

public class findSubsetsThatSumToATarget {
    private static HashSet<String> allSubsets = new HashSet<>();
    private static final String token = " ";

    /
    public static void findTargetSumSubsets(int[] input, int target, String ramp, int index) {

        if (index > (input.length - 1)) {
            if (getSum(ramp) == target) {
                allSubsets.add(ramp);
            }
            return;
        }

        // First recursive call going ahead selecting the int at the current index
        // value
    }
}

```

```

        findTargetSumSubsets(input, target, ramp + input[index] + token, index + 1);
        // Second recursive call going ahead WITHOUT selecting the int at the current
        // index value
        findTargetSumSubsets(input, target, ramp, index + 1);
    }

    private static int getSum(String intString) {
        int sum = 0;
        StringTokenizer sTokens = new StringTokenizer(intString, token);
        while (sTokens.hasMoreElements()) {
            sum += Integer.parseInt((String) sTokens.nextElement());
        }
        return sum;
    }

    public static void main(String[] args) {
        int[] n = { 24, 1, 15, 3, 4, 15, 3 };
        int counter = 1;
        FindSubsetsThatSumToATarget.findTargetSumSubsets(n, 25, "", 0);
        for (String str : allSubsets) {
            System.out.println(counter + ") " + str);
            counter++;
        }
    }
}

```

## Security attacks & Web Application Security ?

Cross site Scripting

## Log4J vanularability?

## What is CSRF vulnerability? How to solve it?

## JNDI Database configuration

## LDAP Server configuration in IBM WAS

### LDAP URL Entries Task

WAS login

ldap.url	ldap://BER-UXITDS-301v:389
ldap.username	cn=jitp,dc=perceptive,dc=com
ldap.user.password	p3rcprep

Select	Name	Value	Description	Required
<input type="checkbox"/>	studvreg.webservice.prefix.url	https://services-lhs-proda-vip.mytrials-prod.pii.pxi.int/study-registry-services/v2		false
<input type="checkbox"/>	service.registry.prefix.url	https://services-lhs-proda-vip.mytrials-prod.pii.pxi.int/study-registry-services	SR Web service.	false
<input type="checkbox"/>	service.availability.schema	SERVICE_AVAILABILITY		false
<input type="checkbox"/>	ldap.username	cn=jitp,dc=perceptive,dc=com		false
<input type="checkbox"/>	ldap.user.password	*****		false
<input type="checkbox"/>	ldap.url	ldap://bil-uxsds-013v.pii.pxi.int:389		false
<input type="checkbox"/>	isim.service.username	itim manager		false
<input type="checkbox"/>	isim.service.url	https://isim-lhs-vip.mytrials-prod.pii.pxi.int		false
<input type="checkbox"/>	isim.service.password	*****	isim service account password	false
<input type="checkbox"/>	isim.remove.users.retry.count	20		false
<input type="checkbox"/>	isim.remove.users.page.size	1000		false

Will use JNDI Lookup to get Values

## H R Mapping

### Reason for Job Change"

I am looking for better opportunities! Both Technically & Financially. I want to be part of Product based things, innovating / implementing something new instead of working with existing code.

I think this job would be a great opportunity in my career.

### Keep in Mind:

- Tell the truth, yes! because big shots like google or microsoft sees honesty in employees.
- Be positive. Tell them what's your work there and how it's affecting you blah blah
- Tell them, I'm good at 'x' but I'm doing 'y' there.

- Put your points in polite manner.

## 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/>

