1. **Difference between Stateless protocol and Stateful protocol**

| **Sr. No.** | **Key** | **Stateless** | **Stateful** |
| --- | --- | --- | --- |
| 1 | Definition | Stateless Protocol is a network protocol in which Client send request to the server and server response back as per the given state. | Stateful Protocol is a network protocol in which if client send a request to the server then it expects some kind of response, in case of no response then it resend the request. |
| 2 | Examples | HTTP (Hypertext Transfer Protocol), UDP (User Datagram Protocol), DNS (Domain Name System) | FTP (File Transfer Protocol), Telnet. |
| 3 | Server Restriction | In Stateless, server is not needed to keep the server information or session details to itself. | In stateful, a server is required to maintain the current state and session information. |
| 4 | Dependency | In stateless, server and client are loosely coupled and can act independently. | In stateful, server and client are tightly bound. |
| 5 | Design | Server design is simple to implement. | Server design is comparatively complex and difficult to implement. |
| 6 | Crash Proof | A failed server can be restarted easily after crash. | A server has to keep the information of session and other details. Crash management is difficult. |
| 7 | Transactions | Server handles transactions in a very quicker way. | Server is slow comparatively. |

1. **Difference between git fetch and git pull**

**git fetch** really only downloads new data from a remote repository - but it doesn't integrate any of this new data into your working files. Fetch is great for getting a fresh view on all the things that happened in a remote repository.  
Due to it's "harmless" nature, you can rest assured: fetch will never manipulate, destroy, or screw up anything. This means you can never fetch often enough. master

**git pull**, in contrast, is used with a different goal in mind: to update your current HEAD branch with the latest changes from the remote server. This means that pull not only downloads new data; it also directly **integrates** it into your current working copy files

1. **Difference between git fetch and git merge**

***Git fetch command helps the user download commits, refs, and files from the remote repository to the local repository***.  In other words, executing this command will help you see all the updates on the remote repository. You might be thinking, what if you don’t want to keep the changes? Well, this command does not hurt your working repository at all.

Git merge command is the positive conclusion of your decision to incorporate the changes you saw using Git fetch command. Let me straighten it out. Once the user is ready to accept the changes from the remote repository, they can merge these changes to the local repository. As the name suggests, you are confirming to “***merge***” these changes.

1. **Difference between log4j and slf4j**

We can’t compare log4j with slf4j as both are different components.  SLF4J is a simple logging façade for java. It is not a logging component, and even it does not do the actual logging. It is only an abstraction layer to an underlying logging component.

In the case of **Log4j**, it is a logging component, and it does the logging instructed to do.

## Why is SLF4J better than Log4J?

It is always difficult to prefer one between the SLF4J and Log4j. If you have a choice, I would suggest you; logging abstraction is always preferable than logging framework. If you use a logging abstraction, SLF4J in particular, we can migrate to any logging framework we require at the time of deployment without opting for single dependency.

Following are the reasons, which are good enough to choose SLF4J over Log4j:

* It is always better to use abstraction.
* SLF4J is an open-source library or internal library that makes it independent of any particular logging implementation, which means no need to manage multiple logging configurations for multiple libraries.

1. **How to monitor microservices?**

To monitor and profile microservices, distributed tracing technique is used. Distribute tracing provides a place where we can see that "what is happening with a specific request? we will use **Spring** **Cloud Sleuth** with **Zipkin.**

**Spring** **Cloud Sleuth**  provides the ability to track the progress of subsequent microservices by adding **trace** and **span Ids** on the appropriate HTTP request headers. With the help of the Zipkin server, we can put all the logs of all the components in the **MQ** (RabbitMQ). Then the requests fired by user can be traced using zipkin UI dashboard.

Under the hood **virtual DOM** also uses real **DOM** to render the page or content. So there is no way that **virtual DOM** is **faster** than real **dom**. ... By using **virtual DOM**, we can find out what is changed and with that, we can apply only those changes to real **DOM** instead of replacing entire **DOM**

An **event** is an action that could be triggered as a result of the user action or system generated **event**. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called **events**. With JSX, a function is passed as the **event** handler instead of a string.

**Rendering** is the most important procedure that a programmer has to manage in frontend development. In **React**, the **render**() **method** is the only required **method** in a class component, and is responsible for describing the view to be **rendered** to the browser window.

Businesses that use **ReactJS** are assured of **better** performance compared to those that use other frameworks. Because **ReactJS** helps to prevent updating of DOM, it means that the apps will be faster and deliver **better** UX. **ReactJS** was designed to help **improve** the total rendered pages from the website server.

# What are the differences between ClassNotFoundException and NoClassDefFoundError in Java?

Both **ClassNotFoundException**and **NoClassDefFoundError**are the errors when JVM or ClassLoader not able to find appropriate class while loading at run-time. **ClassNotFoundException**is a checked exception and **NoClassDefFoundError**is an Error which comes under unchecked.

**ClassNotFoundException**comes when we try to load a class at run-time. **NoClassDefFoundError**is thrown when a class has been compiled with a specific class from the classpath but if same class not available during run-time.

The searched-for class definition existed when the currently executing class was compiled, but the definition can no longer be found."

1. **What is Optional?**

Java introduced a new class Optional in jdk8. It is a public final class and used to deal with NullPointerException in Java application. You must import java.util package to use this class. It provides methods which are used to check the presence of value for particular variable. To avoid the abnormal termination, we use Optional class.

1. **What is the difference between HashTable and ConcurrentHashMap?**

| **Sr. No.** | **Key** | **ConcurrentHashMap** | **HashTable** |
| --- | --- | --- | --- |
| 1 | Basic | Concurrent Hashmap is a class which was introduced in jdk1.5 | HashTable is thread safe legacy class which introduced in the Jdk1.1 |
| 2 | Locking | It apply lock on entire collection | Concurrent hash map apply locks only at bucket level called fragment  while adding or updating the map |
| 3 | Performance | It is slower than  concurrentHashMap | It is better than HashTable |
| 4. | Null | It doesn't allow null key and value | It allows null key and value |

1. **What is the difference between @RequestParam and @PathVariable annotation?**

The @**RequestParam** is used to extract query parameters while @**PathVariable** is used to extract data right from the URI. ... Even though both are used to extract data from URL, @**RequestParam** is used to retrieve query parameters, anything after? **in the** URL, while @**PathVariable** is used to retrieve values from URI itself.

1. **Why iterator is used to remove element from a collection?**

Even though java.util.ArrayList provides the remove() methods, like remove (int index) and remove (Object element), you cannot use them to remove items while iterating over ArrayList in Java because they will throw **[ConcurrentModificationException](http://javarevisited.blogspot.sg/2018/01/concurrentmodificationexception-while-removing-elements-from-ArrayList-in-loop-Java-solution.html" \t "_blank)** if called during iteration. The right way to remove objects from ArrayList while iterating over it is by using the *Iterator's* remove() method.  
  
When you use iterator's remove() method, **ConcurrentModfiicationException** is not thrown. Because it also updates the counters and variables used by Iterator like modCount, which indicates that modification is done by the Iterator itself and not somewhere around.

1. **Why @springbootapplication,@repository and @component annotation is used?**

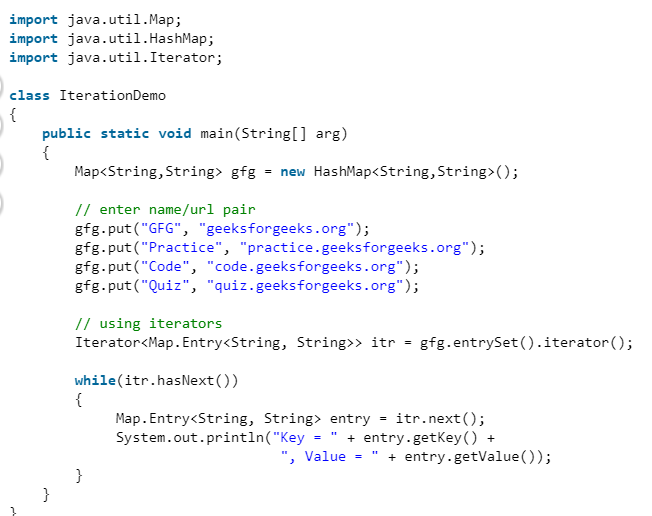
@springbootapplication **annotation** enables component-scanning so that the web controller classes and other components you create will be automatically discovered and registered as beans in Spring's **Application** Context. All the@Controller classes you write are discovered by this **annotation**.

@**Repository** is a **Spring** annotation that indicates that the decorated class is a **repository**. A **repository** is a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects

@**Component** is an **annotation** that allows Spring to automatically detect our custom beans.

1. **How to iterate a Map using iterator?**

we **cannot** iterate a Map directly using [iterators](https://www.geeksforgeeks.org/iterators-in-java/), because Map are not [Collection.](https://www.geeksforgeeks.org/collections-in-java-2/) *Map.entrySet()* method returns a collection-view(*Set<Map.Entry<K, V>>*) of the mappings contained in this map. So we can iterate over key-value pair using *getKey()* and *getValue()* methods of [Map.Entry<K, V>](https://docs.oracle.com/javase/7/docs/api/java/util/Map.Entry.html" \t "_blank).



1. **What is JPA? How many types of repositories are there?**

The Java Persistence API (JPA) is the standard way of persisting Java objects into relational databases.   JpaRepository extends [PagingAndSortingRepository](http://static.springsource.org/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html) and, in turn, the [CrudRepository](http://static.springsource.org/spring-data/data-commons/docs/current/api/org/springframework/data/repository/CrudRepository.html).

* [**CrudRepository**](http://static.springsource.org/spring-data/data-commons/docs/current/api/org/springframework/data/repository/CrudRepository.html) - provides CRUD functions
* [**PagingAndSortingRepository**](http://static.springsource.org/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html) - provides methods to do pagination and sort records
* [**JpaRepository**](http://static.springsource.org/spring-data/data-jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html) - provides JPA related methods such as flushing the persistence context and delete records in a batch

Because of this inheritance relationship, the **JpaRepository contains the full API of CrudRepository and PagingAndSortingRepository.**

* findAll() – get a List of all available entities in database
* findAll(…) – get a List of all available entities and sort them using the provided condition
* save(…) – save an Iterable of entities. Here, we can pass multiple objects to save them in a batch
* flush() – flush all pending task to the database
* saveAndFlush(…) – save the entity and flush changes immediately
* deleteInBatch(…) – delete an Iterable of entities. Here, we can pass multiple objects to delete them in a batch

1. **What is ExecutorService?**

The Java ExecutorService is the interface which allows us to execute tasks on threads asynchronously. The Java ExecutorService interface is present in the java.util.concurrent package. The ExecutorService helps in maintaining a pool of threads and assigns them tasks. It also provides the facility to queue up tasks until there is a free thread available if the number of tasks is more than the threads available.

We can use Java ExecutorService to create a single thread, a pool of threads, or a scheduled pool of threads. The Executors class provides factory methods to instantiate an ExecutorService as follows-

* ExecutorService executorService1 = Executors.newSingleThreadExecutor();

(Creates a ExecutorService object having a single thread. )

* ExecutorService executorService2 = Executors.newFixedThreadPool(10);

(Creates a ExecutorService object having a pool of 10 threads. )

* ExecutorService executorService3 = Executors.newScheduledThreadPool(10);

(Creates a scheduled  pool of 10 threads)

## How to shut down ExecutorService ?

Once we are done with our tasks given to ExecutorService, then we have to shut it down because ExecutorService performs the task on different threads. If we don't shut down the ExecutorService, the threads will keep running, and the JVM won?t shut down.

The process of shutting down can be done by the following three methods-

* shutdown() method
* shutdownNow() method
* awaitTermination() method

### How many types of locks are there?

### Locks

Instead of using implicit locking via the synchronized keyword the Concurrency API supports various explicit locks specified by the Lock interface. Locks support various methods for finer grained lock control thus are more expressive than implicit monitors.

Multiple lock implementations are available in the standard JDK which will be demonstrated in the following sections.

#### ReentrantLock

The class ReentrantLock is a mutual exclusion lock with the same basic behavior as the implicit monitors accessed via the synchronized keyword but with extended capabilities. As the name suggests this lock implements reentrant characteristics just as implicit monitors.

Let's see how the above sample looks like using ReentrantLock:

ReentrantLock lock = new ReentrantLock();

int count = 0;

void increment() {

lock.lock();

try {

count++;

} finally {

lock.unlock();

}

}

A lock is acquired via lock() and released via unlock(). It's important to wrap your code into a try/finally block to ensure unlocking in case of exceptions. This method is thread-safe just like the synchronized counterpart. If another thread has already acquired the lock subsequent calls to lock() pause the current thread until the lock has been unlocked. Only one thread can hold the lock at any given time.

#### ReadWriteLock#

The interface ReadWriteLock specifies another type of lock maintaining a pair of locks for read and write access. The idea behind read-write locks is that it's usually safe to read mutable variables concurrently as long as nobody is writing to this variable. So the read-lock can be held simultaneously by multiple threads as long as no threads hold the write-lock. This can improve performance and throughput in case that reads are more frequent than writes.

1. **What is the role for a ClassLoader in Java?**

A Java program is made up of a number of custom classes (written by programmers like us) and core

classes (which come pre-packaged with Java). When a program is executed, JVM needs to load the

content of all the needed class. JVM uses a ClassLoader to find the classes.

Three Class Loaders are shown in the picture

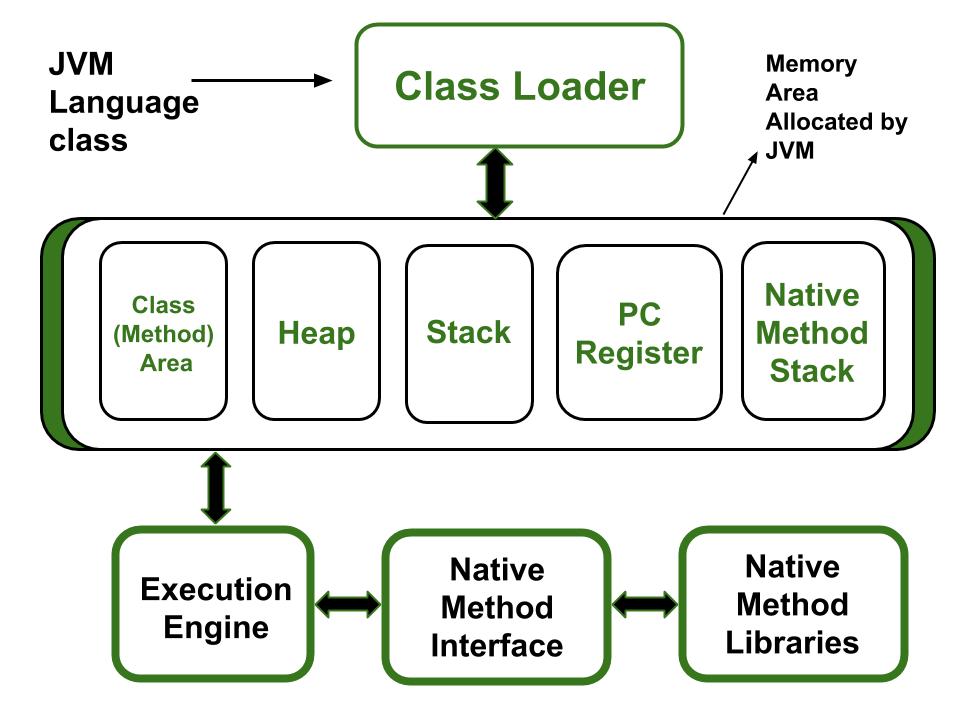
* System Class Loader - Loads all classes from CLASSPATH
* Extension Class Loader - Loads all classes from extension directory
* Bootstrap Class Loader - Loads all the Java core files

When JVM needs to find a class, it starts with System Class Loader. If it is not found, it checks with

Extension Class Loader. If it not found, it goes to the Bootstrap Class Loader. If a class is still not found, a ClassNotFoundException is thrown.

1. **How many types of memory are there in java?**

The **memory in the JVM divided into 5 different parts**:

[](https://media.geeksforgeeks.org/wp-content/uploads/20190614230114/JVM-Architecture-diagram.jpg)

1. Class(Method) Area
2. Heap
3. Stack
4. Program Counter Register
5. Native Method Stack

Let’s see about them in brief:

1. **Class Loader:**It is a subsystem of JVM which is used to load class files.It is mainly responsible for three activities.
   * Loading
   * Linking
   * Initialization
2. **Class(Method) Area:** It stores class level data of every class such as the runtime constant pool, field and method data, the code for methods.
3. **Heap:** It is used to allocate memory to objects at run time
4. **Stack:**
   * Each thread has a private JVM stack, created at the same time as thread. It is used to store data and partial results which will be needed while returning value for method and performing dynamic linking.
   * Java Stack stores frames and a new frame is created each time at every invocation of the method.  
     A frame is destroyed when its method invocation completes
5. **Program Counter Register:** Each JVM thread which carries out the task of a specific method has a program counter register associated with it. The non-native method has a PC which stores the address of the available JVM instruction whereas, in a native method, the value of the program counter is undefined. PC register is capable of storing the return address or a native pointer on some specific platform.
6. **Native method Stacks:** Also called as C stacks, native method stacks are not written in Java language. This memory is allocated for each thread when its created And it can be of a fixed or dynamic nature.
7. **when an object becomes eligible for garbage collection?**"

Every Java program has more than one thread. Each thread has its execution stack. There is a thread to run in Java program that is a main() method. Now we can say that an object is eligible for garbage collection when no live thread can access it. The garbage collector considers that object as eligible for deletion. The garbage collection system attempts to objects from the memory when they are not in use.

Transaction management in SQL

Transaction could a single or a set of multiple SQL Statements. When the database state changes, it means a transaction has happened.

For example, at a given time, say Database (T1) is present with few tables and data already stored in it. Then a new row gets inserted in a table. This “INSERT” is a transaction, because it will change the database state say to T2.

Similarly, Delete and Update are transactions which changes the data.

Transaction Management involves the process to ensure that Transaction was executed properly and there is no corruption of data.

Transactions have the following four standard properties, usually referred to by the acronym **ACID**.

* **Atomicity** − ensures that all operations within the work unit are completed successfully. Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state. For example, you are making changes in a multiple tables in a single stored procedures, and for one table, some error ocurred, using try catch and putting **ROLLBACK** in catch will rollback all the changes made in other tables as well.
* **Consistency** − ensures that the database properly changes states upon a successfully committed transaction.
* **Isolation** − enables transactions to operate independently of and transparent to each other like when you are Inserting a data SQL puts lock on table and if at the same time, other user is inserting, SQL will keep this other query in a que. Who may see all such incidence using sp\_who2.
* **Durability** − ensures that the result or effect of a committed transaction persists in case of a system failure.

Mastering the Transaction Management in SQL comes with experience. To simplify, a developer needs to ensure the consistency and sanctity of data and database while modifying it through Insert, Update or Delete commands.

@Override

**public** **boolean** **equals**(Object o){

**if**(o==**null**)

**return** **false**;

**if**(**this**.getClass()!=o.getClass())

**return** **false**;

          Employee e=(Employee)o;

**return** e.id.equals(**this**.id) && e.name.equals(**this**.name);

   }

   @Override

**public** **int** **hashCode**(){

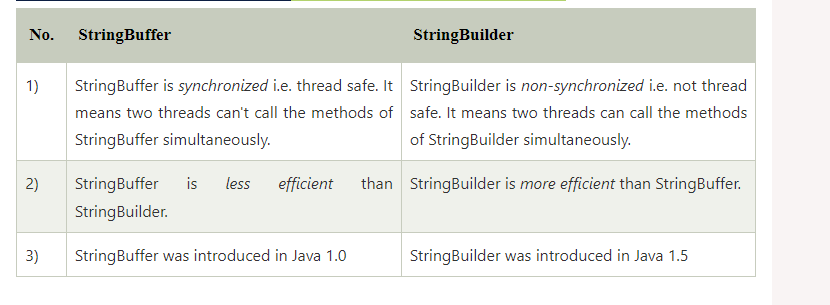
**return** id.hashCode() + name.hashCode();

   }

| **Algorithm** | **Time Complexity** | | |  |
| --- | --- | --- | --- | --- |
|  | **Best** | **Average** | **Worst** |  |
| [Selection Sort](http://geeksquiz.com/selection-sort/) | Ω(n^2) | θ(n^2) | O(n^2) |  |
| [Bubble Sort](http://geeksquiz.com/bubble-sort/) | Ω(n) | θ(n^2) | O(n^2) |  |
| [Insertion Sort](http://geeksquiz.com/insertion-sort/) | Ω(n) | θ(n^2) | O(n^2) |  |
| [Heap Sort](http://geeksquiz.com/heap-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) |  |
| [Quick Sort](http://geeksquiz.com/quick-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n^2) |  |
| [Merge Sort](http://geeksquiz.com/merge-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) |  |
| [Bucket Sort](https://www.geeksforgeeks.org/bucket-sort-2/) | Ω(n+k) | θ(n+k) | O(n^2) |  |
| [Radix Sort](https://www.geeksforgeeks.org/radix-sort/) | Ω(nk) | θ(nk) | O(nk) |  |
| [Count Sort](https://www.geeksforgeeks.org/counting-sort/) | Ω(n+k) | θ(n+k) | O(n+k) |  |

Which sorting technique Arrays.sort() uses?

In Java, **Arrays** is the class defined in the java.util package that provides **sort()** method to sort an array in ascending order. It uses **Dual-Pivot Quicksort algorithm** for sorting. Its complexity is **O(n log(n))**. It is a **static** method that parses an **array** as a parameter and does not return anything.



Java is not a "PURE" Object Oriented Language , because it uses primitive data types such as (int,float,char...etc). The developers of java could have made these primitive data types as objects(like String... etc), but the primitive data types such as int float... are more faster than objects! So, in order to achieve faster execution of java program they kept Primitive data type as it is!!  
  
So Java is said to be 98% Object oriented language but not PURE Object Oriented language

How to run SQL transaction?

1. Every SQL trans should start with BEGIN TRANSACTION, BEGIN TRAN, or BEGIN TRANSACTION Transaction\_Name.
2. Every Transaction in SQL Server must end with either COMMIT or ROLLBACK statements.
3. COMMIT TRANSACTION: This statement tells the SQL to save the changes made between the BEGIN and COMMIT.