# Cap Theorem- Consistent, Availability and Partition Tolerance

The theorem formalizes the tradeoff between consistency and availability when there’s a partition.

NoSQL databases are great for distributed networks. They allow for **horizontal scaling**, and they can quickly scale across multiple nodes.

CA databases

Some [**relational databases**,](https://www.educative.io/blog/relational-database-deep-dive) such as [PostgreSQL,](https://www.educative.io/blog/mongodb-versus-postgresql-databases) allow for consistency and availability

CP databases

[MongoDB](https://www.educative.io/blog/mongodb-with-docker) is an example of a CP database. The CP system is structured so that there’s only one primary node that receives all of the write requests in a given replica set. Secondary nodes replicate the data in the primary nodes, so if the primary node fails, a secondary node can stand-in.

AP databases

Apache Cassandra is an example of an AP database

## **Differences between final and immutability**

* final means that you can’t change the object’s reference to point to another reference or another object, but you can still mutate its state (using setter methods e.g). Whereas immutable means that the object’s actual value can’t be changed, but you can change its reference to another one.
* final modifier is applicable for variable but not for objects, Whereas immutability applicable for an object but not for variables. • By declaring a reference variable as final, we won’t get any immutability nature, Even though reference variable is final. We can perform any type of change in the corresponding Object. But we can’t perform reassignment for that variable.
* final ensures that the address of the object remains the same whereas the Immutable suggests that we can’t change the state of the object once created.

Rate Limiting-Introduce and enforce a [RATE LIMIT](https://microservice-api-patterns.org/patterns/quality/qualityManagementAndGovernance/RateLimit.html) to safeguard against API clients that overuse the API. A [RATE LIMIT](https://microservice-api-patterns.org/patterns/quality/qualityManagementAndGovernance/RateLimit.html) can also restrict the amount of concurrency allowed, i.e., the number of concurrent requests a client is allowed to make. A [RATE LIMIT](https://microservice-api-patterns.org/patterns/quality/qualityManagementAndGovernance/RateLimit) gives the provider control over the client’s API consumption, but deciding on the right limits is not easy.

* By implementing a [RATE LIMIT,](https://microservice-api-patterns.org/patterns/quality/qualityManagementAndGovernance/RateLimit.html) an API provider can protect its offering from malicious clients, such as unwelcome bots, and maintain the quality of its service.

# When does Hibernate throw a LazyInitializationException

Hibernate throws the *LazyInitializationException* when it needs to initialize a lazily fetched association to another entity without an active session context. That’s usually the case if you try to use an uninitialized association in your client application or web layer.

synchronized(myObject1){

// code

synchronized(myObject1){

// code

}

}

Deadlock occur in above scenerio

AOP- Advantage-: It is easy to configure 2)Spring Aop is implemented in pure java. So separation compilation unit or separate classloader is not used.3) It utilized spring IOc for dependency Injection 4)It integrate cross cutting concern into the classes.

Disadvantage-Debuging of AOP framework based application code is a little challenge.2)Only method with public visibility will be recommenede not private protected or default 3) Aspect can not be advised from other aspect.This is because when a class is marked as aspect then Spirng prevents it from autoproxied

Aspect-The modularization of concerns that act as cross cutting concern.

Join point-It is a point during the execution of the method.

Advice-The action taken by an advice at the join point.

Point cut

-

The predicate that matche

s

the

one or more

joins

point is called point cut.

Introduction

-

Define additional method fields for a type.

Target Object

-

Those object which are advised by aspect are called target object

Weaving

-

The process of

adding advised to a point cut.It is a process of connecting aspect to

other type of application. Weaving can be done at runtime,load time or compile time

Intermediate operation which is stateful

-

distinct(), sorted(),limit() skip().

All other

are stateless. Like map(),filter(),flatmap()

Give the name Of functional Interface In Java8

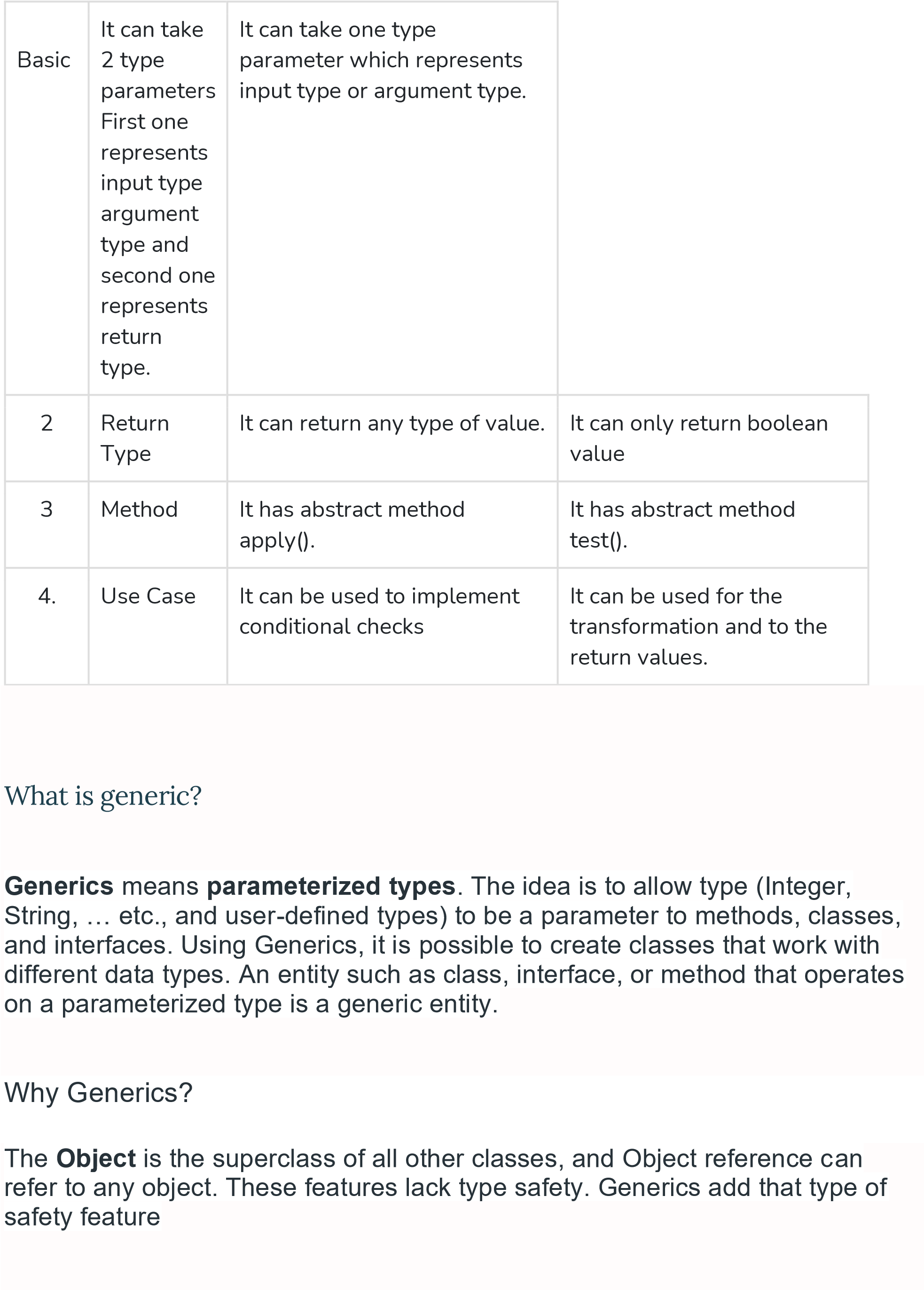
-

Function,Predicate,Producer,Consumer,Supplier,Runnable Callable

***What is the N+1 problem in Hibernate?***

The ***N+1 query problem*** is said to occur when an ORM, like hibernate, executes 1 query to retrieve the parent entity and N queries to retrieve the child entities. As the number of entities in the database increases, the queries being executed separately can easily affect the performance of the application.

Difference between Predicates and function in java 8?



Hystrix configuration is done in four major steps.

1. Add Hystrix starter and dashboard dependencies.

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-hystrix</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-hystrix-dashboard</artifactId> </dependency>

1. Add @EnableCircuitBreaker annotation
2. Add @EnableHystrixDashboard annotation
3. Add annotation @HystrixCommand(fallbackMethod =

"myFallbackMethod")

**What is Circular Dependency Problem?**

We can use empty check also with optional?

We can do with the help of optional.empty()

Circular dependency is a relation between two or more modules which either directly or indirectly depend on each other to function properly. Let me explain with an example.

Suppose there are two classes A and B. A is dependent on B and B is dependent on A.

**public** **class** **A** { **private** B b;

**public** **A**(){ **this**.b = **new** B();

}

}

**public** **class** **B** {

**private** A a;

**public** **B**(){ **this**.a = **new** A();

}

}

As you see the constructor of class ‘A’, it initialize object of class B, that means A is dependent on B.

As you see the constructor of class ‘B’, it initialize object of class A, that means B is dependent on A.

A is dependent on B and B is dependent on A. This is called circular dependency problem.

**What happen if you try to instantiate class A (or) B?**

Let us try to define object for class A, the constructor of class A calls the constructor of class B. The constructor of class B calls the constructor of class A. This is an infinite recursion call, finally application end up in StackOverflowError.

**public** **class** **CircularDependencyDemo** { **public** **static** **void** **main**(String args[]){

A obj = **new** A();

}

}

Run above application, you will end up in StackOverflowError.

Exception in thread "main" java.lang.StackOverflowError at B.<init>(B.java:5) at A.<init>(A.java:6) at B.<init>(B.java:6) at A.<init>(A.java:6) at B.<init>(B.java:6) at A.<init>(A.java:6)

**How to resolve circular dependency problem?** There are two ways to resolve this problem.

1. Removing the dependency from constructor definition
2. Point the circular dependency to current object

**Removing the dependency from constructor definition** Define the dependencies using setter methods like below.

**public** **class** **A** { **private** B b;

**public** **A**() {

}

**public** **void** **setB**(B b) { **this**.b = b;

}

}

**public** **class** **B** { **private** A a;

**public** **B**() {

} **public** **void** **setA**(A a) { **this**.a = a;

}

}

**Point the circular dependency to current object** Update classes A and B like below.

**public** **class** **A** { **private** B b;

**public** **A**() { b = **new** B(**this**);

}

}

**public** **class** **B** { **private** A a;

**public** **B**(A a) { **this**.a = a;

}

}

Find all the element which startwith 1 List<Integer> al=Arrays.asList(10,15,20,25,8,9); al.stream.map(s->s+””).filter(s->s.startwith(“1”).forEach(System.out::println) If there are 10 thread in thread pool. One thread has run time exception.Then what happen?

The answer to this depends on whether or not the code is using threadPool.execute(...) or threadPool.submit(...). If you are using execute(...) and the task throws an uncaught exception then the thread terminates and pool forgets about the thread and starts another one immediately if appropriate. The task and the thread can get garbage collected.

If you are using submit(...) then the task will finish, the exception will be caught, and the thread will continue to run and will dequeue the next job submitted to the thread-pool and execute it. You can see what exception is thrown by extending ThreadPoolExecutor and overriding:

If one thread is updating same record name and another thread updating sane record age. Then what will happen?How can we avoid?

1. We can also do a conditional update with where clause in the query and use executeUpdate() method. Ex: The Hibernate - Query - executeUpdate() method updates and return the number of entities updated. So if the executeUpdate() returns "zero" it means the row has been already updated by another thread. (No Exception)
2. Using @Version. (OptimisticLockException)
3. Using Row Level DB lock. (DB Exception)
4. Using Synchronization. (Java Synchronization Exception)

|  |
| --- |
| Cache based on which datastructure?-→HashMap Difference between byte and char?  The byte is used to store row binary data. While char is use to store character or text.  Unicode- Unicode is a character encoding standard and is capable of represent almost every character of well known language of the world. |

How to implement internationalization?

We

have one interceptor class. Inside that interceptor class we get the

value of User language from session. When that language is true. We can

split and set in Locale.

The file for the default locale will have the name

*messages.properties*

, and

files

h locale will be named

for eac

*messages\_XX.properties*

, where

*XX*

is the

locale code

.

Filters

are called by your

Server(tomcat)

. while Interceptors are called by

Spring

.

**If a key does not exist in a certain reque**

**sted locale, then the application**

**will**

**fall back to the default locale value.**

Through which tool we ca

n give mock up data to UI team. So

development will not hold

.

Ans Swagge

r

Is request scoped spring beans threadsafe?

Yes. Each request will get its own bean so thread safety is guaranteed.

Is spring prototype bean threadsafe?

No. Prototype beans are stateful however if it is managed by multiple threads, then steps must be taken to ensure it is thread safe.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Singleton Beans is thread safe or not depends on how the class whose scope is** | | | | | |  |
| **singleton is written** |  | | | | |
|  |
| **Static variable cannot be serialized because it is class level variable** | | |  | | |
|  | | |
| Below is a representation of the default Maven lifecycle and its 8 steps: **Validate** | | | | | **,** |
| **Compile, Test, Package, Integration test, Verify, Install and Deploy**. | |  | | |
|  | |
| Double checked locking should only be used when you have requirement for lazy | | | | |
| initialization otherwise [use Enum to implement singleton](http://javarevisited.blogspot.com/2012/07/why-enum-singleton-are-better-in-java.html) or simple static final variable. | | | | | |
|  | | | | | |
| In order to call *Instrumentation.getObjectSize(Object)* to get object's size, we | | | | | | |
| need to be able to access the instance of Instrumentation first. **We need to** | | | | | | |
| **use the instrumentation agent** and there are two ways to do it, as | | | |  | | |
| described in the documentation for the [*java.lang.instrument* package.](https://docs.oracle.com/en/java/javase/11/docs/api/java.instrument/java/lang/instrument/package-summary.html) | | | | |  | |
| Pro of lambda expression | | | | |

* Looking at each method’s call we know directly which interface is used
* Looking at each call we know which methods are implemented
* The code is directly readable without interpretation
* A developer that reads the code doesn’t need to know the code very well
* Cons of Lambdas
* Dependency Management
* HardToUse Without IDE

We can use try catch to handle exception in lambda expression **Difference between Callable and Runnable are following:**

* 1. Callable is introduced in JDK 5.0 but Runnable is introduced in JDK 1.0
  2. Callable has call() method but Runnable has run() method.
  3. Callable has call method which returns value but Runnable has run method which doesn't return any value.
  4. call method can throw checked exception but run method can't throw checked exception.
  5. Callable use submit() method to put in task queue but Runnable use execute() method to put in the task queue.

**Dependency Injection is more of a architectural pattern for loosely coupling software components.** **Factory pattern is just one way to separate the responsibility of creating objects of other classes to another entity**. Factory pattern can be called as a tool to implement DI.

|  |
| --- |
|  |

Stream is eager or lazy ?

Ans **Streams are lazy** because intermediate operations are not evaluated until terminal operation is invoked.

EntityManager em = emf.createEntityManager(); em.getTransaction().begin();

TypedQuery<Author> q = em.createQuery(

|  |
| --- |
| How to fix lazy Initialization exception?  The right way to fix  a *LazyInitializationException* is to fetch all required associations within your service layer. The best option for that is to load the entity with all required associations |

"SELECT a FROM Author a",

Author.class);

List<Author> authors = q.getResultList(); em.getTransaction().commit(); em.close();

for (Author author : authors) {

List<Book> books = author.getBooks();

log.info("... the next line will throw LazyInitialization

...");

books.size();

}

in one query. Or you can use a DTO projection, which doesn’t support lazy loading and needs to be fully initialized before you return it to the client.

Let’s take a closer look at the different options to initialize lazily fetched association and at the best way to use DTO projections.

What is class level Lock?

**Class Lock**: In java, each and every class has a unique lock usually referred to as a class level lock. These locks are achieved using the keyword ‘static synchronized’ and can be used to make static data thread-safe. It is generally used when one wants to prevent multiple threads from entering a synchronized block.

Example:

**public** **class** **ClassLevelLockExample** { **public** **void** **classLevelLockMethod**() { **synchronized**

(ClassLevelLockExample.class)

{ //DO your stuff here

}

} }

**Object Lock**: In java, each and every object has a unique lock usually referred to as an objectlevel lock. These locks are achieved using the keyword ‘synchronized’ and can be used to protect nonstatic data. It is generally used when one wants to synchronize a non-static method or block so that only the thread will be able to execute the code block on a given instance of the class.

Example:

|  |
| --- |
| **public** **class**  **ObjectLevelLockExample**  { **public** **void**  **objectLevelLockMethod**()  {  **synchronized** (**this**)  {  //DO your stuff here }  }  } |

Prototype creates a brand new instance everytime you call getBean on the

ApplicationContext. Whereas for Request, only one instance is created for an HttpRequest. So in a single HttpRequest, I can call getBean twice on Application and there will only ever be one bean instantiated, whereas that same bean scoped to Prototype in that same single HttpRequest would get 2 different instances.

## **HttpRequest scope**

Mark mark1 = context.getBean("mark"); Mark mark2 = context.getBean("mark"); mark1 == mark2; //This will return true **Prototype scope**

Mark mark1 = context.getBean("mark");

Mark mark2 = context.getBean("mark"); mark1 == mark2; //This will return false Transactional-

**Programmatic means you have transaction management code surrounding your**

**business code**. This gives extreme flexibility, but is difficult to maintain and, well, boilerplate. Declarative means you separate transaction management from the business code. You can use annotations or XML based configuration.

If we write transactional annotation the any exception will arise. Then data will not saved in any service database.

If we will not use transactional first service code there is no exception then it will save in database. Another service have some exception then it will not save in database. But in case transaction both table will not save any data.

• By default transaction propagation is required if we have not mentioned any transaction propagation only transactional annotation.

Spring Support 7 Propagation in transaction?

Ans\_:

REQUIRED: TransactionDefination.Propagation-REQUIRED

SUPPORTS: TransactionDefination.Propagation- SUPPORTS

MANDATORY: TransactionDefination.Propagation- MANDATORY

REQUIRES-NEW: TransactionDefination.Propagation-REQUIRES-NEW

NOT-SUPPORTED: TransactionDefination.Propagation- NOT-SUPPORTED

NEVER: TransactionDefination.Propagation-NEVER

NESTED: TransactionDefination.Propagation-NESTED

Two Service Uservice and Message Service scenerio

**REQUIRED: TransactionDefination.Propagation-REQUIRED-**If addmessage() is call directly it will create its own transaction

If addmessage is called from another service

\*If the calling MessageService Method has a transaction then method make use of the existing transaction \* ifthe calling Message Service doesnot have any transaction then the method will create new transaction.

**So in case of REQUIRED addmessage() method use of the calling UserService method transaction if it exists else create its own.**

**SUPPORTS: TransactionDefination.Propagation- SUPPORTS**-Inside message service we give transaction type **Supports** and directly hitting message service endpoint. Then message service works but there is no transaction created for message.

If message service is calling inside user service. And user service have transaction type required and message service have transaction type SUPPORT then only one transaction creates i.e from USER service. And message service use same transaction.

**MANDATORY: TransactionDefination.Propagation**- MANDATORY-Inside message service if we use **transaction propagation as MANDATORY**. We directly hit message service endpoint. Then at that time message service will give exception because there is no transaction.

If message service is calling inside userservice and userservice have **propagation level Required** then one transaction created by user service and same transaction is used by message service. And there is no exception.

**REQUIRES-NEW: TransactionDefination.Propagation-REQUIRES-NEW**- Inside message service propagation level is **REQUIREDNEW** then new traction created by message service. If we are calling message service from user service endpoint and user service propagation level is **required** message service propagation level is **REQUIRESNEW** then user service created new transaction and when message service is calling then message service suspend user service transaction and create new transaction for message service.

**NOT-SUPPORTED: TransactionDefination.Propagation- NOT-SUPPORTED**-if addmessage is called directly then it does not create new transaction

**If addmessage is called from another service i.e UserService**

If the calling message service method have a transaction then method does not use of existing transaction neither does it create its own transaction.It run without transaction

If the calling message service method does not have any transaction then the method will not create its own new transaction and run without transaction.

**So incase non-supported the add message will never run transaction/**

NEVER: TransactionDefination.Propagation-NEVER- if the add message called directly it will not create new transaction.

**If addmessage is called from another service i.e UserService**

If the calling message service method have transaction then method will throws exception.

If the calling message service method does not have any transaction then method will not create new transaction and will run without transaction.

**So in case of NEVER the add message will never uses any transaction**

**NESTED transaction is not supported by HIBERNATE JPA**

**Transaction Anomaly-**

**DIRTY READ-**

**NON\_REPEATBLE READ**

**PHANTOMS READ**

**DirtyRead**- Transaction A updating Price information of Product NO-123. In the meantime before transaction

A commits transaction check the price info of Product No-123. Transaction has not committed the data yet.So Transaction B will get stale data.If tansaction A rollback transaction already used because Price is less. And rollback become more . So customer unhappy. This is called dirty read.

**NonRepeatable Read**- If transaction A read some Data like select \* from product where product No-123. Transaction B update some data like update product price=1000 where product No-123. And commit So when transaction A read the same data .again It will see different sets of data.Because Transaction updated new record. There is a difference When the transactionA read the data first time and when transaction A read the same data again

PHANTOM READ- It is similar to non-repateable read except there is a where clause. Trasaction reads all the product whose price is less that 1000 But Transaction B update one record who price is 900 will come inside same where clause. So when transaction Transaction A read the data it see new sets of record added

.

These anomaly can be overcome by ISOLATION .

There are 4 ISOLATION LEVEL-

TRANSACTION READ UNCOMMITED-If we use this Dirty Read, nonrepeatable read and phantom read occur

TRANSACTION READ COMMITTED\_It will ensure only committed read can happen. We can avoid dirty read but Non-Repeatable Read and Phantom READ still occur

TRANSACTION REPEATABLE READ-By default my database have transaction repeatable read. We can prevent dirty read and Non-Repeatable READ but still phantom READ occur

TRANSACTION SERIALIZABLE-No two transaction can use same set of data. So performance degraded. It has table level lock. Transaction Serializable is most strict and less performing

When we need to fetch the data which is save which is deleted on basis of flag?how we can do hibernate?Ans Cascade

What is cascade type?Cascading is a feature in Hibernate, which is used to manage the state of the mapped entity whenever the state of its relationship owner (superclass) affected. When the relationship owner (superclass) is saved/ deleted, then the mapped entity associated with it should also be saved/ deleted automatically.

What is completable Future?

A **CompltableFuture** is used for asynchronous programming. Asynchronous programming means writing non-blocking code. It runs a task on a separate thread than the main application thread and notifies the main thread about its progress, completion or failure.

Why wait ,notify,notifyall in object class not in thread class?

Q) Why wait, notify and notifyAll is defined in Object Class and not on Thread class in Java ?

Ans > In Java, you wait on a particular instance of an Object – a monitor assigned to that object to be precise.

If you want to send a signal to one thread that is waiting on that specific object instance then you call notify() on that object.

If you want to send a signal to all threads that are waiting on that object instance, you use notifyAll() on that object.

If wait() and notify() were on the Thread instead then each thread would have to know the status of every other thread.

How would thread1 know that thread2 was waiting for access to a particular resource? If thread1 needed to call thread2.notify() it would have to somehow find out that thread2 was waiting.

In simple terms, the reasons are as follows :- 1) Object has monitors.

1. Multiple threads can access one Object. Only one thread can hold object monitor at a time for synchronized methods/blocks.
2. wait(), notify() and notifyAll() method being in Object class allows all the threads created on that object to communicate with other
3. Locking (using synchronized or Lock API) and Communication (wait() and notify()) are two different concepts.

# If Thread class contains wait(), notify() and notifyAll() methods, then it will create below problems: a) Thread communication problem

b) Synchronization on object won’t be possible. If each thread will have monitor, we won’t have any

|  |  |
| --- | --- |
| way of achieving synchronization |  |
| c) Inconsistency in state of object. |

Why Rest is stateless?

A stateless protocol is one in which the receiver is not required to keep session state from previous requests. The sender sends relevant session state to the receiver in such a way that each request may be interpreted without reference to prior requests’ session state, which the receiver retains. HTTP (HyperText Transfer Protocol) is an example of Stateless Protocol because each request is executed independently of the requests that came before it. This implies that once a transaction is completed, the connection between the browser and the server is also terminated

When we send request from Web . In JSON WEB id will created in server.Incase of saop request will stored in cookie form very request taken by session everytime server create new jsonweb id created but in rest whether we login from web or mobile browser. There is no JSON WEBID created.

When we put same key and different value . HashMap will override the value. Suppose we hashMap hs =new HashMap()

hs.put(“1”,10) hs.put(“1”,11)

So it will override the value. We can’t override and hashcode and equal method in this scenario. HashCode and Equal method can be override only whnever there is a Pojo class. And That pojo is a key. In that pojo class we can override the hashcode and equal method.

Complexity of HashMap is o(1)

When hashCollision collision occur then only hashmap internally stored in linkedlist.

HashCollision occur when hashcode is same . There is no relation of key in hashcollision.. When the index created hashcode of key is same then only hashcollision occur.

Difference between Repository and Service?

In repository there is no need to handle transaction save persist , detached whether commit and rollback it is taken care by repository annotation. But in service we need to handle transaction code wise.

Dependency Injection is framework with respect to Spring.

The framework is something which we can’t modify. But Design pattern is something which we can modify as per our requirement. As per requirement we can create the object .

Java 8 Default Method 3Rd Rule\_

If Interface I have default method show() and Interface J have also default method with name show(). Class A have normal method with name show() and Class B extends ClassA and Implements Interface I and Interface J. Then which method called when we create object of Class A and called show methods. Class C normal method will called because Normal method have more priority then default method.

2)If you override any method present in Object class.Then we can’t define in interface as a default method

3) Inside interface only one abstract method and we can give any number of default method

Concurrent HashMap Internal Implementation?

Ans 1) Underlying data structure for concurrent HashMap is Hash Table

2)Concurrent HashMap allow concurrent read and thread safe update operation.

30 To perform Read operation thread will not require lock. But to perform update operation thread will require lock. But it is the lock of only a particular part of the map (i.e. Bucket level lock or segment level lock)

4)Instead of whole map concurrent update achieved by internally dividing map into smaller portion which is defined by concurrency level.

1. The default concurrency level is 16
2. That is concurrent HashMap allow simultaneous read operation and simultaneous 16 write operation
3. Null is not allowed for both keys and values.
4. While one thread is iterating over other thread can perform update operation and concurrent HashMap never through concurrent Modification exception.

|  |  |
| --- | --- |
| Anonymous Inner Class | Lambda Expression |
| It is a class without name. | It is a method without name. (anonymous function) |
| It can extend abstract and concrete class.  It can implement an interface that | It can’t extend abstract and concrete class. |
| contains any number of abstract methods. | It can implement an interface which contains a single abstract method. |
| Inside this we can declare instance variables. | It does not allow declaration of instance variables, whether the variables declared simply act as local variables. |
| Anonymous inner class can be instantiated.  Inside Anonymous inner class, “this” | Lambda Expression can’t be instantiated. |
| always refers to current anonymous inner class object but not to outer object. | Inside Lambda Expression, “this” always refers to current outer class object that is, enclosing class object. |
| Anonymous Inner Class | Lambda Expression | |
| It is the best choice if we want to handle multiple methods. | It is the best choice if we want to handle interface. | |
| At the time of compilation, a separate .class file will be generated. | At the time of compilation, no separate .class file will be generated. It simply convert it into private method outer class. | |
| Memory allocation is on demand, whenever we are creating an object. | It resides in a permanent memory of JVM. | |

Ques0-How much iteration required if there are 27 elements is present in sorted order

And O(log27)where base is 2

Complexity of Sorting operation-3

O(logn)

Complexity Of Binary search- O(logn) Lazy Initialization in spiring Boot?

By default in Spring, all the defined beans, and their dependencies, are created when the application context is created.

* + **Single Responsibility Principle (SRS)**: Do not place more than one responsibility into a single class or function, refactor into separate classes and functions.
  + **Dependency Injection**: Do not hardcode the dependencies, instead inject them.
  + **Interface segregation**: Do not create lengthy interfaces, instead split them into smaller interfaces based on the functionality. The interface should not contain any dependencies (parameters), which are not required for the expected functionality.
  + **Open Closed Principle**: While adding new functionality, existing code should not be modified. New functionality should be written in new classes and functions.
  + **Liskov substitutability principle**: The child class should not change the behavior (meaning) of the parent class. The child class can be used as a substitute for a base class.

**In contrast, when we configure a bean with lazy initialization, the bean will only be created, and its dependencies injected, once they're needed**

* 1. Lazy initialization may reduce the number of beans created when the application is starting – therefore, **we can improve the startup time** of the application
  2. As none of the beans are created until they are needed, **we could mask issues, getting them in run time instead of startup time**
  3. The issues can include out of memory errors, misconfigurations, or class-definition-found errors
  4. Also, when we're in a web context, **triggering bean creation on demand will increase the latency of HTTP requests** – the bean creation will affect only the first request, but **this may have a negative impact on load-balancing and auto-scaling**.

Select all employee with manager name?

CREATE VIEW AS

SELECT e1.emp\_Id EmployeeId, e1.emp\_name EmployeeName, e1.emp\_mgr\_id ManagerId, e2.emp\_name AS ManagerName

FROM tblEmployeeDetails e1

JOIN tblEmployeeDetails e2

ON e1.emp\_mgr\_id = e2.emp\_id

If there are 100 microservice? Which is failing.How we can trace the microservice is failing?

Ans Using Kibana logging we can trace. Every Microservice have trace id. With the help of each trace id we can check which microservice is failing .

Real Time where we use abstract class and interface?

We can use Audit section abstract class we can use. Export Functinality where Export class is abstract class Wherever common functionality is there . We can use abstract class.

If you want to create customized vector library where all the feature of vector we can use? How

Ans Through Inheritance

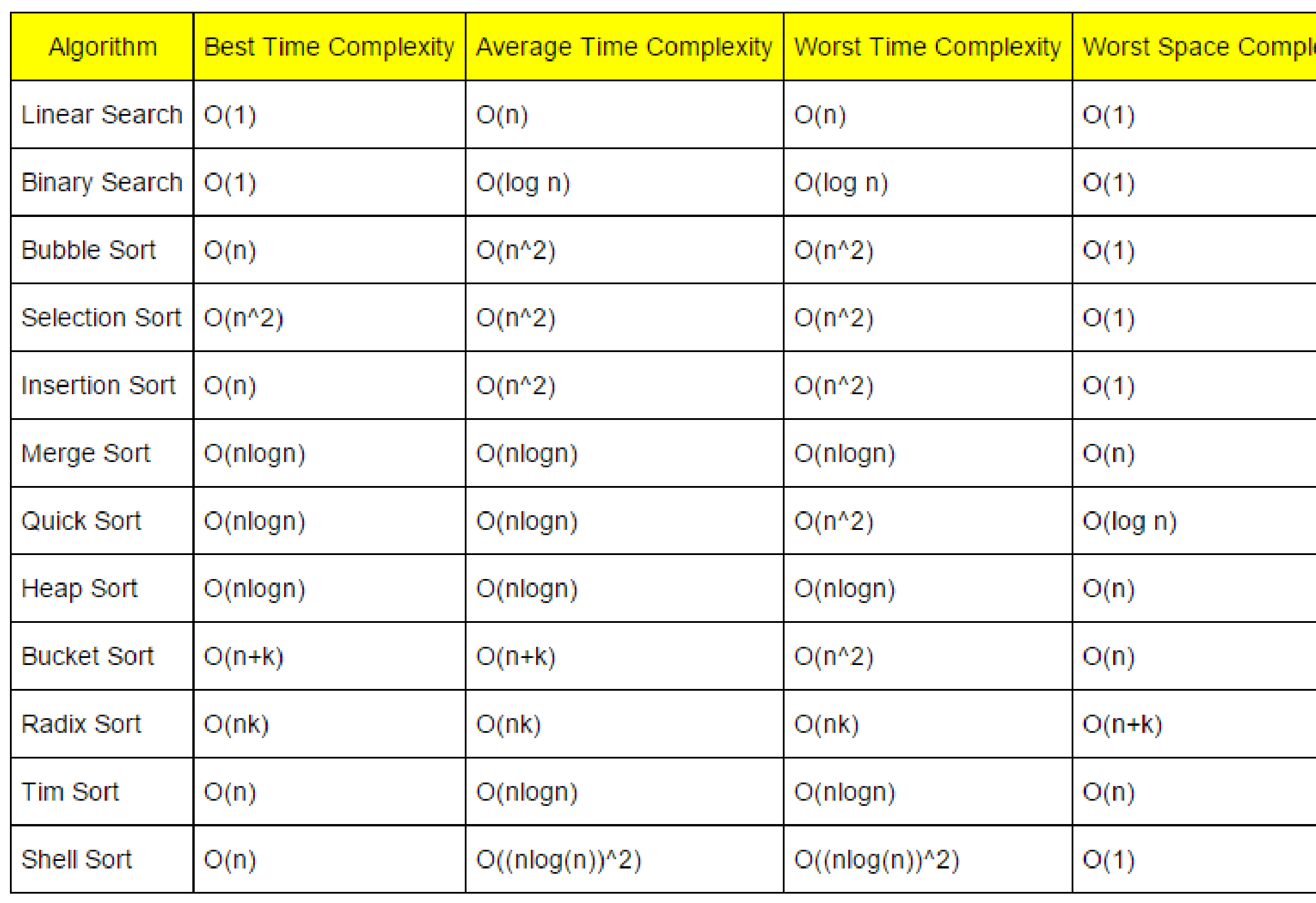
In customized vector inbuilt method user will not see. only customized method they can see and used? How

Ans Composition

How to read JSON file from resource folder?

Ans -:@Value("classpath:data/data.json") Resource resourceFile; How to read propertyfile?

Ans @PropertySource(“classpath:message.properties”)

 How we can same method interface in two implementation class Ans- We can do with the help of qualifier.

@Autowired

@Qualifier("asposePdfConverter") private PdfConverter pdfConverter;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

public interface MyService {

void doWork();

}

@Service

@Qualifier("firstService")

public static class FirstServiceImpl implements MyService {

@Override public void doWork() {

System.out.println("firstService work");

}

}

@Service

@Qualifier("secondService")

public static class SecondServiceImpl implements MyService {

@Override public void doWork() {

System.out.println("secondService work");

}

}

@Component

public static class FirstManager {

private final MyService myService;

@Autowired // inject FirstServiceImpl

public FirstManager(@Qualifier("firstService") MyService myService) {

this.myService = myService;

}

@PostConstruct public void startWork() {

System.out.println("firstManager start work"); myService.doWork();

}

}

@Component

public static class SecondManager {

private final List<MyService> myServices;

@Autowired // inject MyService all implementations public SecondManager(List<MyService> myServices) { this.myServices = myServices;

}

@PostConstruct public void startWork() {

System.out.println("secondManager start work"); myServices.forEach(MyService::doWork);

}

}

}

When Singleton is ready for garbage collection?

|  |  |
| --- | --- |
| Ans-No, a Singleton will not be garbage collected. Back in the old [Java](http://www.javaranch.com/) 1.1 days they could be, but | |
| now a Singleton will only be garbage collected when there are no references to its ClassLoader. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Ques If we want to the server will accept only two or any number of request. AND we need all | | | |
| those response which is successful |  | | |
| And Using executor framework we can restrict the number of requests. | | |  |
| Callable we can get response of all request. | |  |

|  |  |
| --- | --- |
| Why We use streams? |  |
| Ans To process data from collection. If you want to process object from collection | |

Ques- for Loop and While Loop when we will use?

you should use a for loop when you know how many times the loop should run. If you want the loop to break based on a condition other than the number of times it runs, you should use a while loop.

# Predicates-

Predicate is **a functional interface, which accepts an argument and returns a boolean**. Usually, it used to apply in a filter for a collection of objects

What kind of object we can use for garbage collection?

An **object is eligible for garbage collection** when there are no more references to that **object**.

References that are held in a variable are usually dropped when the variable goes out of scope. Or, you can explicitly drop an **object** reference by setting the variable to the special value null.

Why String is immutable?

## Ans

1. [String pool](https://www.journaldev.com/797/what-is-java-string-pool) is possible only because String is immutable in java, this way Java Runtime saves a lot of java heap space because different String variables can refer to same String variable in the pool. If String would not have been immutable, then String

interning would not have been possible because if any variable would have changed the value, it would have been reflected to other variables also.

1. Since String is immutable, it is safe for [multithreading](https://www.journaldev.com/1079/multithreading-in-java) and a single String instance can be shared across different threads. This avoid the usage of synchronization for thread safety, Strings are implicitly thread safe.
2. If String is not immutable then it would cause severe security threat to the application. For example, database username, password are passed as String to get database connection and in [socket programming](https://www.journaldev.com/741/java-socket-programming-server-client) host and port details passed as String. Since String is immutable it’s value can’t be changed otherwise any hacker could change the referenced value to cause security issues in the application.
3. Strings are used in [java classloader](https://www.journaldev.com/349/java-classloader) and immutability provides security that correct class is getting loaded by Classloader. For example, think of an instance where you are trying to load java.sql.Connection class but the referenced value is changed to myhacked.Connection class that can do unwanted things to your database.
4. **Caching Hashcode**

The hashcode of the string is frequently used in Java. For example, in a HashMap. Being immutable guarantees that hashcode will always the same so that it can be cashed without worrying the changes.That means, there is no need to calculate hashcode every time it is used. This is more efficient.

### 6. Security

String is widely used as a parameter for many java classes, e.g. network connection, opening files, etc. Were String not immutable, a connection or file would be changed and lead to a serious security threat. The method thought it was connecting to one machine, but was not.

Mutable strings could cause a security problem in Reflection too, as the parameters are strings.

## Is-A and has-A

An Is-A relationship is also know as inheritance and Has-A relation is also knowm as composition

How can we access private variable from a class.?

Ans- We can private variable of a class through getter and setter method

Interface-We cannot create the object of interface but we can create refernce of interface

Difference between Abstract Factory and Factory Design Pattern

AbstractFactory pattern uses composition to delegate responsibility of creating object to another class while Factory design pattern uses [inheritance](http://javarevisited.blogspot.sg/2012/10/what-is-inheritance-in-java-and-oops-programming.html) and relies on derived class or sub class to create [object.](http://javarevisited.blogspot.sg/2012/12/what-is-object-in-java-or-oops-example.html)

Abstract Factory may use Factory design pattern for creating objects but they are not just limited to that they can also use [Builder design pattern](http://javarevisited.blogspot.ro/2012/06/builder-design-pattern-in-java-example.html) to build object by doing series of steps or Prototype pattern to build object by copying or customizing prototype of that object. It completely depends upon your implementation whether to use Factory pattern or Builder pattern for creating products

The main difference between the two is that with the Abstract Factory pattern, a class delegates the responsibility of object instantiation to another object via composition whereas the Factory Method pattern uses inheritance and relies on a subclass to handle the desired object instantiation.

To overcome the problem of creating the object of singleton class during deserilizaton. By this code singleton we are not able to create new instance of singleton class during deserialization

protected Object readResolve()

{

return instance; }

Overcome Cloning issue:- To overcome this issue, override clone() method and throw an exception from clone method that is CloneNotSupportedException. Now whenever user will try to create clone of singleton object, it will throw exception and hence our class remains singleton.

Overcome reflection issue: To overcome issue raised by reflection, [enums](https://www.geeksforgeeks.org/enum-in-java/) are used because java ensures internally that enum value is instantiated only once. Since java Enums are globally accessible, they can be used for singletons. Its only drawback is that it is not flexible i.e it does not allow lazy initialization.

**The try-with-resources Statement**

The try-with-resources statement is a try statement that declares one or more resources. A *resource* is an object that must be closed after the program is finished with it. The try-withresources statement ensures that each resource is closed at the end of the statement. Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.

The following example reads the first line from a file. It uses an instance of BufferedReader to read data from the file. BufferedReader is a resource that must be closed after the program is finished with it:

static String readFirstLineFromFile(String path) throws IOException { **try (BufferedReader br =**

**new BufferedReader(new FileReader(path)))** { return br.readLine();

}

}

For Each Method

Java provides a new method forEach() to iterate the elements. It is defined in Iterable and Stream interface. It is a default method defined in the Iterable interface. Collection classes which extends Iterable interface can use forEach loop to iterate elements.

This method takes a single parameter which is a functional interface. So, you can pass lambda expression as an argument.

Lambda expression return any value or not?

If **lambda** body is a code block, you must always **return** a **value** explicitly. But, if**lambda** body is just an **expression**, **return** statement is not required.

Underline Data Structure

HashMap and HashSet- Hash

TreeSet and Tree map-Binary Tree Arraylist-Array

Reentrant Lock-The ReentrantLock class implements the Lock interface and provides synchronization to methods while accessing shared resources. The code which manipulates the shared resource is surrounded by calls to lock and unlock method. This gives a lock to the current working thread and blocks all other threads which are trying to take a lock on the shared resource.

What are the ways to create thread? Which one preferable and Why?

* By extending Thread class
* By implementing Runnable interface

In the first approach, Our class always extends Thread class. There is no chance of extending any other class. Hence we are missing Inheritance benefits. In the second approach, while implementing Runnable interface we can extends any other class. Hence we are able to use the benefits of Inheritance.

Because of the above reasons, implementing Runnable interface approach is recommended than extending Thread class.

What is Volatile keyword? Where we can use volatile keyword?

Volatile keyword is used to modify the value of a variable by different threads. It is also used to make classes thread safe. It means that multiple threads can use a method and instance of the classes at the same time without any problem. The volatile keyword can be used either with primitive type or objects.

volatile keyword in Java guarantees that value of the volatile variable will always be read from main memory and not from Thread's locale cache.

How to check memory utilization in java?

Many commands can check the memory utilization of JAVA processes, for example, pmap, ps, jmap, jstat.

Int i=10 where it stored in memory?

I stored in stack because reference stored in stack and value stored in heap memory.

Parameter and Local variable are allocated in Stack. Object is stored in heap

As enums don’t have any constructor so it is not possible for Reflection to utilize it. Enums have their by-default constructor, we can’t invoke them by ourself. JVM handles the creation and invocation of enum constructors internally. As enums don’t give their constructor definition to the program, it is not possible for us to access them by Reflection also. Hence, reflection can’t break singleton property in case of enums.

**Key Differences Between Stack and Heap**

1. In a stack, the allocation and deallocation is done by CPU and is automatic whereas, in heap, it needs to be done by the programmer manually.
2. Heap frame handling is costlier than stack frame handling.
3. Implementation of a stack is complex. As against, implementation of a heap is simple.
4. A function call in stack takes O(N) time. In contrast, it takes O(1) time in a heap.
5. Stack implementation mainly suffers from the memory shortage problem. On the contrary, the main issue in a heap is fragmentation.
6. Access to a stack frame is easier than the heap as the stack is confined to the small region of memory and it always hit the cache, but heap frames are dispersed throughout the memory so the memory accessing can cause more cache misses.
7. Stack is not flexible, the memory size allotted cannot be changed. On the other hand, a heap is flexible, and the allotted memory can be altered.
8. A heap takes more accessing time than a stack.

You should use static methods whenever,

* The code in the method is not dependent on instance creation and is not using any instance variable.
* A particular piece of code is to be shared by all the instance methods.
* The definition of the method should not be changed or overridden.
* you are writing utility classes which should not be changed.

1) When any method have try catch finally block then finally will called always

2)When two interface have same default method in it with same parameter and method name. A class implement both the interface. Then through the object of that class which interface will default method will called.

Ans As both the method have same name and parameter then either we need override that method . Otherwise it will give ambiguity error.

SELECT D.D\_ID, SUM(E.SALARY)

FROM DEPARTMENT D

LEFT JOIN EMPLOYEE E ON D.D\_ID=E.D\_ID

GROUP BY D.D\_ID

UNION

SELECT 0 D\_ID, SUM(SALARY) SALARY

FROM EMPLOYEE

//Query 2

SELECT DeptId, SUM(Salary) AS "Department wise Total

Salary" FROM Employee GROUP BY DeptId;

## Serialization

1. If a parent class has implemented Serializable interface then child class doesn’t need to implement it but vice-versa is not true.
2. Only non-static data members are saved via Serialization process.
3. Static data members and transient data members are not saved via Serialization process.So, if you don’t want to save value of a non-static data member then make it transient.
4. Constructor of object is never called when an object is deserialized.
5. Associated objects must be implementing Serializable interface.

## Memory In Java

1. Heap memory-Class instances and arrays are allocated in heap memory.
2. Non-heap memory-It comprise method are and other memory required for internal processng**.** It stores per-class structures, code for methods and constructors. Per-class structure means runtime constants,fields,method data,code of methods ,constructor and [static fields**.**](http://javapapers.com/core-java/explain-the-java-static-modifier/)
3. **Object is stored in heap. But stack memory contain the reference to it**. For eg-int i=12 I

is reference so it is stored in stack and 12 is value stored in heap memory.

1. Stack memory-It is used by only one thread. Of execution.

### What are Class loaders? Explain the types of class loader

Class loaders are the part of the Java Runtime Environment that dynamically loads Java classes into the Java virtual machine. It is responsible for locating libraries, reading there content and loading the classes contained within the libraries. When JVM is started three class loaders are used Bootstrap class loader

1. Extensions class loader

1. System class loader

Bootstrap class loader loads the core java libraries. It is written in native code. The bootstrap class loader is responsible for loading key java classes like java.lang.Object and other runtime code into memory. The runtime classes are packaged inside jre/lib/rt.jar file.

Extensions class loader loads the code in the extension directories. It is implemented by ExtClassLoader class.

Application ClassLoader searches for the class in the Application Classpath. If the class is available then it is loaded, if not then a **ClassNotFoundException** exception is generated.

System class loader the code found on the java.class.path which map to the system class path variables. It is implemented by AppClassLoader class. All user classes by default are load by the system class loader.

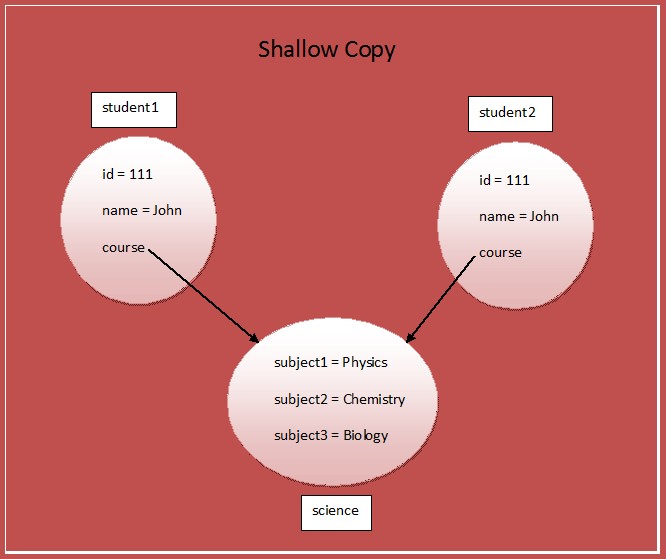
[**Dynamic loading**](http://www.careerride.com/java-dynamic-loading.aspx)

Dynamic loading - Classloader and its method LoadClass() is used for dynamic loading of a class...

|  |  |
| --- | --- |
| **Shallow Copy** | **Deep Copy** |
| Cloned Object and original object are not 100% disjoint. | Cloned Object and original object are 100% disjoint. |
| Any changes made to cloned object will be reflected in original object or vice versa. | Any changes made to cloned object will not be reflected in original object or vice versa. |
| Default version of clone method creates the shallow copy of an object. | To create the deep copy of an object, you have to override clone method. |
| Shallow copy is preferred if an object has only primitive fields. | Deep copy is preferred if an object has references to other objects as fields. |
| Shallow copy is fast and also less expensive. | Deep copy is slow and very expensive |

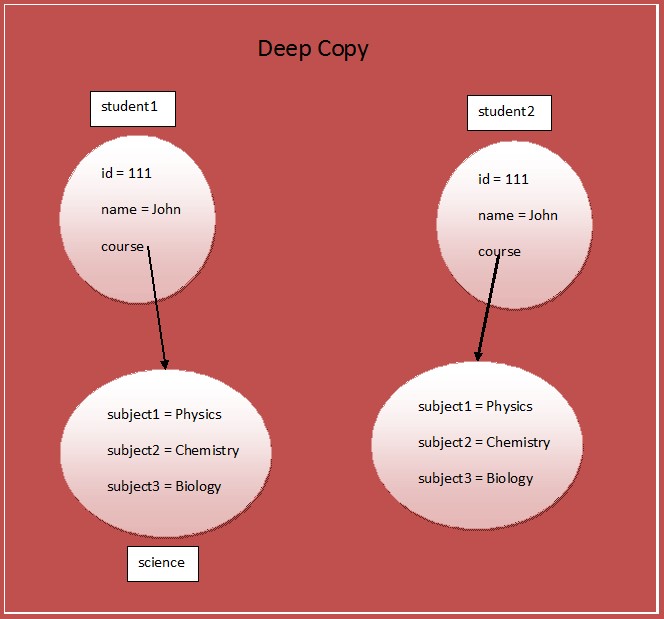
The shallow copy of an object will have exact copy of all the fields of original object. If original object has any references to other objects as fields, then only references of those objects are copied into clone object, copy of those objects are not created. For eg- For eg- If student and Course class are there. if we override clone method

in only in student class. That is shallow clone



Deep copy of an object will have exact copy of all the fields of original object just like shallow copy. But in additional, if original object has any references to other objects as fields, then copy of those objects are also created by calling clone() method on them.

For eg- If student and Course class are there. if we override clone method in both the classes. That is deep cloning



**How HashMap works?**

Ans Lets note down the steps one by one:

1. First of all, key object is checked for null. If key is null, value is stored in table[0] position.

Because hash code for null is always 0.

1. Then on next step, a hash value is calculated using key’s hash code by calling its hashCode() method. This hash value is used to calculate index in array for storing Entry object. JDK designers well assumed that there might be some poorly written hashCode() functions that can return very high or low hash code value. To solve this issue, they introduced another **hash()** function, and passed the object’s hash code to this hash() function to bring hash value in range of array index size.
2. Now *indexFor(hash, table.length)* function is called to calculate exact index position for storing the Entry object.
3. Here comes the main part. Now, as we know that two unequal objects can have same hash code value, how two different objects will be stored in same array location [**called bucket**]. Answer is LinkedList. If you remember, Entry class had an attribute **“next”**. This attribute always points to next object in chain. This is exactly the behavior of LinkedList.

So, in case of collision, Entry objects are stored in LinkedList form. When an Entry object needs to be stored in particular index, HashMap checks whether there is already an entry?? If there is no entry already present, Entry object is stored in this location.

If there is already an object sitting on calculated index, its next attribute is checked. If it is null, and current Entry object becomes next node in LinkedList. If next variable is not null, procedure is followed until next is evaluated as null.

What if we add the another value object with same key as entered before. Logically, it should replace the old value. How it is done? Well, after determining the index position of Entry object, while iterating over LinkedList on calculated index, HashMap calls equals method on key object for each Entry object. All these Entry objects in LinkedList will have similar hash code but equals() method will test for true equality. If **key.equals(k)** will be true then both keys are treated as same key object. This will csause the replacing of value object inside Entry object only.

In this way, HashMap ensure the uniqueness of keys.

**How get() methods works internally**

Now we have got the idea, how key-value pairs are stored in HashMap. Next big question is : what happens when an object is passed in get method of HashMap? How the value object is determined?

Answer we already should know that the way key uniqueness is determined in put() method , same logic is applied in get() method also. The moment HashMap identify exact match for the key object passed as argument, it simply returns the value object stored in current Entry object.

If no match is found, get() method returns null.

***31) What do you understand by iterator fail-fast property?***

**Fail-fast Iterators fail as soon as they realized that structure of Collection has been changed since iteration has begun**. Structural changes means adding, removing or updating any element from collection while one thread is Iterating over that collection.Fail-fast behavior is implemented by keeping a modification count and if iteration thread realizes the change in modification count it throws ConcurrentModificationException.

***What is difference between fail-fast and fail-safe?***

You have understood fail-fast in previous question. **Fail-safe iterators** are just opposite to failfast. **They never fail if you modify the underlying collection on which they are iterating**, because they work on clone of Collection instead of original collection and that’s why they are called as fail-safe iterator.Iterator of CopyOnWriteArrayList is an example of fail-safe Iterator also iterator written by ConcurrentHashMap keySet is also fail-safe iterator and never throw ConcurrentModificationException.

***Ques-33) How to avoid ConcurrentModificationException while iterating a collection?***

You should first try to **find another alternative iterator which are fail-safe**. For example if you are using List and you can use List Iterator. If it is legacy collection, you can use enumeration.

If above options are not possible then you can use one of three changes:

* If you are using JDK1.5 or higher then you can use ConcurrentHashMap and CopyOnWriteArrayList classes. It is the recommended approach.
* You can convert the list to an array and then iterate on the array.
* You can lock the list while iterating by putting it in a synchronized block. **Finally is not called in following scenerio-:** The only times finally won't be called are:

1. if you call System.exit() *or*
2. if the JVM crashes first
3. if there is an infinite loop in the try block
4. if the power turns off

**Transient Keyword**-

Transient is a Modifier applicable only for variables

At the time of serialisation if we dont want to save the value of a particular variable to meet security constraints then we should go for transient keyword

At the time of serialisation JVM ignores original value of transient variable and save default value to a file.

Hence transient means not to serialize

Static vs Transient?

Staic variable is not a part of Object state and hence they won’t participate in serialization. Due to declaring static variable as transient there is no use.

**how does garbage collection work in java?**

All objects are allocated on the heap area managed by the JVM. ... As long as an object is being referenced, the JVM considers it alive. Once an object is no longer referenced and therefore is not reachable by the application code, the **garbage collector** removes it and reclaims the unused memory.

**Thread contention** is a status in which one thread is waiting for a lock, held by another thread, to be lifted. Different threads frequently access shared resources on a web application. For example, to record a log, the thread trying to record the log must obtain a lock and access the shared resources.

**Deadlock** is a special type of thread contention, in which two or more threads are waiting for the other threads to complete their tasks in order to complete their own tasks.

Different issues can arise from thread contention. To analyze such issues, you need to use the **thread dump**. A thread dump will give you the information on the exact status of each thread.

HashCollision-When two different key have same hashCode then hash collision occur.

List the method which is final in Object Class?

1)getClass()-:Returns the runtime class of this Object. The returned Class object is the object that is locked by static synchronized methods of the represented class.

2)notify()

3)notifyAll()

4)wait()

What does following denote

System.exit(0); ---> OK

System.exit(-1); ---> analogues to Exception

System.exit(1); ---> analogues to Error

|  |  |  |
| --- | --- | --- |
| **o.** | **Throw** | **throws** |
| 1) | Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to |
| 2) | Checked exception cannot be propagated using throw only. | Checked exception can be propa |
| 3) | Throw is followed by an instance. | Throws is followed by class. |
| 4) | Throw is used within the method. | Throws is used with the method |
| 5) | You cannot throw multiple exceptions. | You can declare multiple  public void method()throws IOE |

**void** m(){

**throw** **new** ArithmeticException("sorry");

} **void** m()**throws** ArithmeticException{

//method code

}

Weak reference and Soft reference?

Garbage collector can collect an object if only weak references are pointing towards it and they are eagerly collected, on the other hand Objects with **SoftReference** are collected when JVM absolutely needs memory.

SoftReference looks perfect for implementing caches, so when JVM needs memory it removes object which have only SoftReference pointing towards them.

On the other hand WeakReference is great for storing meta data e.g. storing ClassLoader reference. If no class is loaded then no point in keeping reference of ClassLoader, a WeakReference makes [ClassLoader e](http://javarevisited.blogspot.sg/2012/12/how-classloader-works-in-java.html)ligible for Garbage collection as soon as last strong reference removed.

Strong Reference-These type of references we use daily while writing the code. Any

object in the memory which has active **strong reference** is not eligible for garbage collection

**Synchronised Method and synchronised Block-**Synchronised block generally reduce the scope of lock.Lock is inversely proportional to performance. Its always better to lock in criticcal section

1. Synchronized block provide **granular control over lock**, as you can use arbitrary any lock to provide mutual exclusion to critical section code. On the other hand synchronized method always lock either on current object represented by [this keyword](http://javarevisited.blogspot.com/2012/01/this-keyword-java-example-tutorial.html) or class level lock, if its static synchronized method.
2. ) Synchronized block can throw throw [java.lang.NullPointerException](http://java67.blogspot.sg/2012/09/what-is-nullpointerexception-in-java.html) if expression provided to block as parameter evaluates to null, which is not the case with synchronized methods. 4) ) In case of synchronized method, lock is acquired by thread when it enter method and released when it leaves method, either normally or by throwing Exception. On the other hand in case of synchronized block, thread acquires lock when they enter synchronized block and release when they leave synchronized block.

|  |  |
| --- | --- |
| Errors | Exceptions |
| Errors in java are of type java.lang.Error. | Exceptions in java are of type java.lang.Exception. |
| All errors in java are unchecked type. | Exceptions include both checked as well as unchecked type. |

|  |  |
| --- | --- |
| It is impossible to recover from errors. | You can recover from exceptions by handling the through try-catch blocks. |
| Errors are mostly caused by the environment in which application is running. | Exceptions are mainly caused by the application itself. |

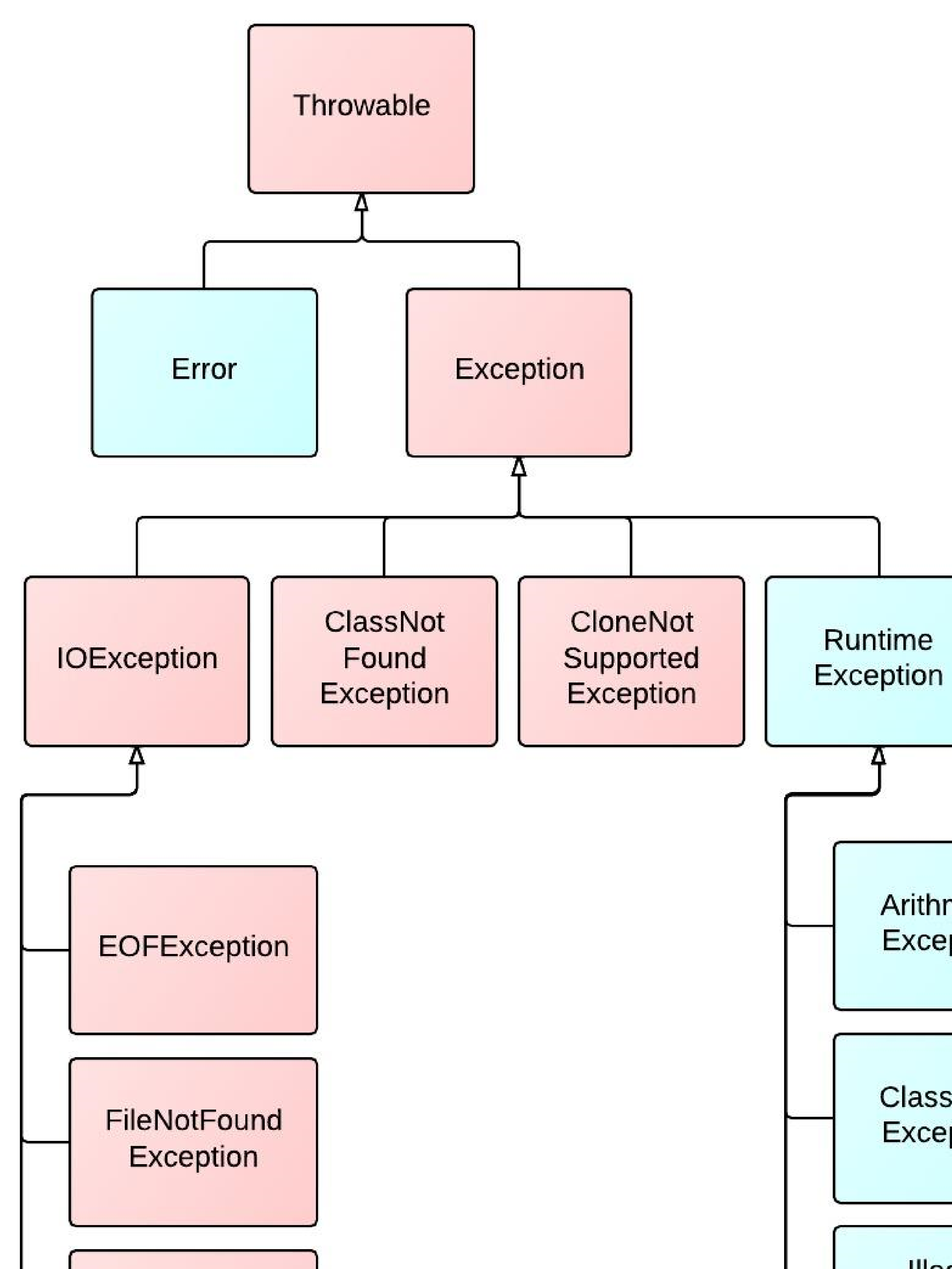
|  |  |
| --- | --- |
| Examples :  java.lang.StackOverflowError, java.lang.OutOfMemoryError | Examples :  Checked Exceptions : SQLException, IOException Unchecked Exceptions : |

### Error and Exception

Checked exceptions are known to compiler where Errors happen at run time. They will not be known to unchecked exceptions are not known to compiler compiler. because they occur at run time.

ArrayIndexOutOfBoundException,

ClassCastException, NullPointerException



Serialisation-The flow of data over the network is called serialization

Serialisation And Externalisation-In case of Serializable Java Virtual machine has full control for serializing object while in case of Externalizable, application gets control for persisting objects. writeExternal() and readExternal() method provides complete **control on format and content of Serialization process** to application which can be leverage to increase performance and speed of serialization process.

1. In case of Serializable, **default serialization process** is used. while in case of Externalizable custom Serialization process is used which is implemented by application.

2 JVM gives call back to readExternel() and writeExternal() of java.io.Externalizalbe interface for restoring and writing objects into persistence.

1. **Externalizable** interface provides complete control of serialization process to application.

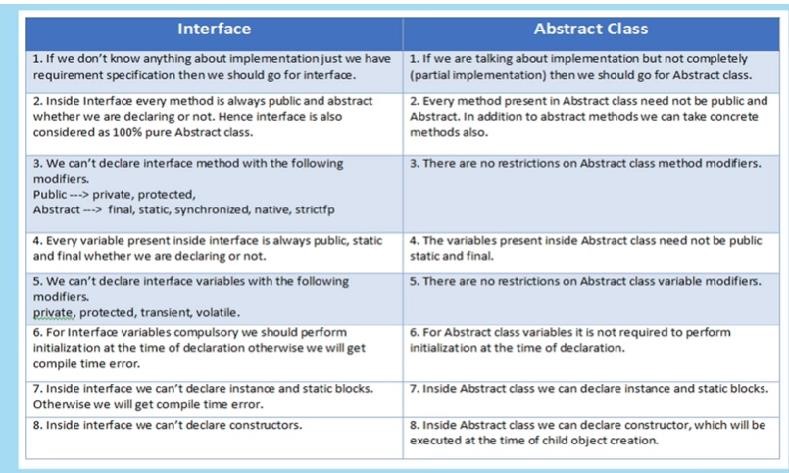
1. readExternal() and writeExternal() supersede any specific implementation of writeObject and readObject methods.

Scrum-

**Scrum** is an efficient framework within which you can develop software with teamwork. It is based on agile principles

* Scrum projects deliver software application features iteratively.
* Each iteration is called a sprint.
* Scrum projects have 4 stages:
* Planning
* Staging
* Development
* Release

|  |  |  |
| --- | --- | --- |
| **Abstract class** | **Interface** | |
| 1) Abstract class can **have abstract and nonabstract** methods. | Interface can have **only abstract** methods | |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance** | |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variab** | |
| 4) Abstract class **can have static methods, main method and constructor**. | Interface **can't have static methods, constructor**. | |
| 5) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implemen class**. | |
| 6) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare | |
| 7) **Example:**  public abstract class Shape{ | **Example:**  public | interface |



.

Abstract Class and Interface

When any non abstract class extend abstract class. Then it is mandatory that abstract method of abstract class should also present in non abstract class which extend abstract class.Otherwise it will give compile time error if abstract method is not implement in nonabtract class

For Ex-**package** BasicQuestion; **abstract** **class** A{ **public** A(){

System.*out*.println("A");

}

**public** **abstract** **void** method1(); **public** **void** method2(){

System.*out*.println("A class");

}

}

**class** B **extends** A{

|  |  |  |  |
| --- | --- | --- | --- |
| public  } | abstract void draw(); | void  } | |
| If we are talking about implementation but not completely partial implementation | | **If we don’t know anything about im about requirement specification** | |
| Every method present in abstract class need not be public and abstract.In addition to abstract method we ca have concrete methods | | **Inside interface every method is abstract whether we are declaring or n is considered 100% abstract class** | |
|  | |  | |
|  | |  |  |

**public** B(){

System.*out*.println("B");

}

@Override

**public** **void** method1() {

// **TODO** Auto-generated method stub

System.*out*.println("method outside abstarct");

}

/\*This method2 is mandatory\*/  **public void method2(){**

**System.*out*.println("B Class");**

**}**

}

**public** **class** InheritanceTest **extends** B{ **public** InheritanceTest(){

System.*out*.println("Inheritance");

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

//new InheritanceTest();

//B b=new B();

//b.method1();

A a=**new** B(); //a.method1(); a.method2();

//new B();

}

}

Association , Composition and Aggregation

Association-On Object-oriented programming, one object is related to other to use functionality and service provided by that object. This relationship between two objects is known as the *association.*

Both Composition and Aggregation are the form of association between two objects

Another example of **Composition** is Car and it's part e.g. engines, wheels etc. Individual parts of the car can not function when a car is destroyed. While in the case of Aggregation, including object can exists without being part of the main object e.g. a Player which is part of a Team, can exist without a team and can become part of other teams as well.

The composition is stronger than Aggregation. In Short, a relationship between two objects is referred as an association, and an association is known as composition when one object *owns* other while an association is known as aggregation when one object uses another object

[Composition,](http://javarevisited.blogspot.sg/2013/06/why-favor-composition-over-inheritance-java-oops-design.html) One object is OWNER of another object, while in the case of aggregation, one object is just a USER or another object.

Covert List to set

Set set = new HashSet(list);

Convert Set to List

List list = new ArrayList(set);

Final Finally and Finalize

No.final finalize

1)Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.

Finally:

1)Finally is used to place important code, it will be executed whether exception is handled or not.

Finalize:

1)Finalize is used to perform clean up processing just before object is garbage collected.

2)Final is a keyword.

Finally is a block. Finalize is a method.

**Static Block and static Initializer**

The **static block** is only loaded when the class object is created by the JVM for the 1st time whereas init {} **block** is loaded every time class object is created. Also first the **static block** is loaded then the init **block**

There are many rules if we talk about methodoverriding with exception handling.

The Rules are as follows:

* **If the superclass method does not declare an exception** o If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
* **If the superclass method declares an exception** o If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.
* QUEUE-
* Differnece b/w Poll and remove-The remove()and poll() methods differ only in their behavior when the queue is empty: the remove() method throws an exception, while

the poll() method returns null.

* Difference b/w Offer and add-The [offer](https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html#offer(E)) method inserts an element if possible, otherwise returning false. This differs from the [Collection.add](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#add(E)) method, which can fail to add an element only by throwing an unchecked exception. The offer method is designed for use when failure is a normal, rather than exceptional occurrence, for

example, in fixed-capacity (or "bounded") queues

* Difference b/w element and peek-The [element()](https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html#element()) and [peek()](https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html#peek()) methods return, but do

not remove, the head of the queue.It means give the first value of the queue. o **Create your own Immutable class :**

o

* **public final class CreateImmutable{**

o

* /\*
* \* There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc.
* \* In short, all the wrapper classes and String class is immutable. We can also create immutable class by
* \* creating final class that have final data members as the example given below:
* \*/ o /\*
* \* o \* In this example, we have created a final class named CreateImmutable.
* \* It have one final datamember, a parameterized constructor and getter method.
* \*/
* /\*In case of Date attribute, you can use constructor to set your date with every new object and import org.joda.time.DateTime class. This is a better version than the java.util.Date because it is immutable. Using a java.util.Date would be dangerous as it is a mutable class and we can’t control the calling thread\*//\*
* \*
* \* Don't provide "setter" methods — methods that modify fields or objects referred to by fields. o \* Don't allow subclasses to override methods. The simplest way to do this is to declare the class as final .
* \* A more sophisticated approach is to make the constructor private and construct instances in factory methods.
* \*/

o

* **final** String pancardNumber;

o

* **public** CreateImmutable(String pancardNumber){ o **this**.pancardNumber=pancardNumber;
* }

o

* **public** String getPancardNumber() { o **return** pancardNumber; o }
* o
* /\* o \* o \* o The above class is immutable because:
* The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
* The class is final so we cannot create the subclass. o There is no setter methods i.e. we have no option to change the value of the instance variable.
* These points makes this class as immutable.
* \*/

o

* } o

|  |  |
| --- | --- |
| o | public final class ImmutableReminder{ |
| o o | private final Date remindingDate; |
| o | public ImmutableReminder (Date remindingDate) { |
| o | if(remindingDate.getTime() < System.currentTimeMillis()){ |
| o | throw new IllegalArgumentException("Can not set reminder” + |
| o | “ for past time: " + remindingDate); |
| o | } |
| o | this.remindingDate = new Date(remindingDate.getTime()); |
| o o | } |

* **Why you have to override hashcode method in java if you override equals in java ??** o
* **see first below : How Hash Map Works In Java Or How Get() Method Works Internally???**

o

* It is not always necessary to override hashcode and equals.But if you think you need to override one, then you need to override both of them. Let's analyze what whould happen if we override one but not the other and we attempt to use a Map **.** o
* public class MyClass {

o

* private final String importantField;
* private final String anotherField;

o

|  |  |  |
| --- | --- | --- |
| public MyClass(final String equalField, final String anotherField) { | | |
| this.importantField = equalField; this.anotherField = anotherField; | |  |
| } |  |

* o o o

o

|  |  |  |
| --- | --- | --- |
| public String getEqualField() { | | |
| return importantField; | |  |
| } |  |

* o o

o

|  |  |  |
| --- | --- | --- |
| public String getAnotherField() { | | |
| return anotherField; | |  |
| } |  |

* o o

o

* @Override o public int hashCode() { o final int prime = 31; o int result = 1; o result = prime \* result
* + ((importantField == null) ? 0 : importantField.hashCode()); o return result;
* }

o

* @Override
* public boolean equals(final Object obj) { o if (this == obj) o return true; o if (obj == null) o return false;
* if (getClass() != obj.getClass()) o return false; o final MyClass other = (MyClass) obj; o if (importantField == null) {
* if (other.importantField != null) o return false;
* } else if (!importantField.equals(other.importantField)) o return false; o return true; o }

|  |  |
| --- | --- |
| o |  |
| o | } |
| o |  |
| o |  |

|  |  |
| --- | --- |
| MyClass first = new MyClass("a","first"); |  |
| MyClass second = new MyClass("a","second"); | |

o

o

* o o o
* As we know that Map internally works on the priciple of Hashing. i.e hashcode,equals and bucket. o
* **Override only hashCode**

o

* If you only override hashCode then when you call myMap.put(first,someValue) it takes first, calculates its hashCode and stores it in a given bucket. Then when you call myMap.put(second,someOtherValue) it should replace first with second as per the [Map Documentation](http://java.sun.com/j2se/1.4.2/docs/api/java/util/Map.html#put(java.lang.Object, java.lang.Object)) because they are equal (according to our definition). o
* Because we are not overiding equals method in our class to check whether they are equal or not.It will go inside Map Api and replaces the first one with second.(mine)
* 'or'
* But the problem is that equals was not redefined, so when the map hashes second and iterates through the bucket looking if there is an object k such that second.equals(k) is true it won't find any as second.equals(first) will be false. o o
* **Override only equals**

o

* If only equals is overriden, then when you call myMap.put(first,someValue) first will hash to some bucket and when you call myMap.put(second,someOtherValue) it will hash to some other bucket (as they have a different hashCode). So, although they are equal, as they don't hash to the same bucket, the map can't realize it and both of them stay in the map.
* Hope it was clearo **Q. Difference between a string buffer and string builder ?** o o

|  |  |
| --- | --- |
| **No. StringBuffer** | **StringBuilder** |

|  |
| --- |
| StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. |

|  |
| --- |
| 1 |

StringBuilder is non-synchronized i.e. not

thread safe. It means two threads can call the

methods of StringBuilder simultaneously.

How to convert the array of strings into the list ?

Arrays class of java.util package contains the method asList() which accepts the array as parameter.

Comparable and Comparator

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc. | Comparator provides **multiple sorting sequence**. In oth we can sort the collection on the basis of multiple elemen id, name and price etc. |
| 2) Comparable **affects the original class** i.e. actual class is modified. | Comparator **doesn't affect the original class** i.e. actu not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort eleme |
| 4) Comparable is found in **java.lang** package. | Comparator is found in **java.util** package. |
| 5) We can sort the list elements of Comparable type by**Collections.sort(List)** method. | We can sort the list elements of Compara by**Collections.sort(List,Comparator)** method. |

Now let’s see **difference between HashSet, LinkedHashSet and TreeSet in Java** :

**Performance** and **Speed** : First difference between them comes in terms of speed. HashSet is fastest, LinkedHashSet is second on performance or almost similar to HashSet but TreeSet is bit slower because of sorting operation it needs to perform on each insertion. TreeSet provides guaranteed O(log(n)) time for common operations like add, remove and contains, while HashSet and LinkedHashSet offer constant time performance e.g. O(1) for add, contains and remove given hash function uniformly distribute elements in bucket.

**Ordering :** HashSet does not maintain any order while LinkedHashSet maintains insertion order of elements much like List interface and TreeSet maintains sorting order or elements.

**Internal Implementation :** HashSet is backed by an HashMap instance, LinkedHashSet is implemented using HashSet and LinkedList while TreeSet is backed up by NavigableMap in Java and by default it uses TreeMap.

**null** : Both HashSet and LinkedHashSet allows null but TreeSet doesn't allow null but TreeSet doesn't allow null and throw [java.lang.NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html) when you will insert null into TreeSet. Since TreeSet uses [compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) of respective elements to compare them which throws NullPointerException while comparing with null, here is an example:

**Important question : How Set Ensures Uniqueness??**

**Ans :**

**Now , what happens internally when you pass duplicate elements in the add()**

method of the Set object , It will return false and do not add to the HashSet , as the element is already present .So far so good .

**But the main problem arises that how it returns false** . So here is the answer:

**public** **class** HashSet<E> **extends** AbstractSet<E> **implements** Set<E>,

Cloneable, java.io.Serializable

{

**private** **transient** HashMap<E,Object> map;

**public** HashSet() {

map = **new** HashMap<>();

}

// SOME CODE ,i.e Other methods in Hash Set

//internally using map. **public** **boolean** add(E e) { **return** map.put(e, PRESENT)==**null**;

}

So , we are achieving uniqueness in Set,internally in java through HashMap .

Whenever you create an object of HashSet it will create an object of HashMap. As we know in HashMap each key is unique . So what we do in the set is we pass the argument in the add(Elemene E) that is E as a key in the HashMap .

The main point to notice in above code is that put (key,value) will return :

1. null , if key is unique and added to the map
2. 2. Old Value of the key , if key is duplicate

\* Q48 How will you make Collections readOnly ?

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

General : Collections.unmodifiableCollection(Collection c)

Collections.unmodifiableMap(Map m)

Collections.unmodifiableList(List l)

Collections.unmodifiableSet(Set s)

Q49 What is UnsupportedOperationException?

This exception is thrown to indicate that the requested operation is not supported.

Example of UnsupportedOperationException:

In other words, if you call add() or remove() method on the readOnly collection .

We know readOnly collection can not be modified . Hence , UnsupportedOperationException will be thrown.

**Q.** [What is memory leak?](http://searchwindowsserver.techtarget.com/definition/memory-leak)

A **memory leak** is the gradual loss of available computer **memory** when a program (an application or part of the operating system) repeatedly fails to return **memory** that it has obtained for temporary use.

**Memory leak** occurs when programmers create a memory in heap and forget to delete it. **Memory leaks** are particularly serious issues for programs like daemonsand servers which by definition never terminate.

To avoid memory leaks, memory allocated on heap should always be freed when no longer needed.

Deamon Thread-**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

In Java, when a developer wants to create and use a new object using, e.g. *new Integer(5)*, he doesn’t have to allocate memory – this is being taken care of by the Java Virtual Machine (JVM). During the life of the application JVM periodically checks which objects in memory are still being used and which are not. Unused objects can be discarded and memory reclaimed and reused again. This process is called *garbage collection* and the corresponding piece of JVM is called a *Garbage Collector* or GC.

Java’s automatic memory management relies on [GC](https://plumbr.eu/handbook/what-is-garbage-collection) which periodically looks for unused objects and removes them. And here hides the dragon. Simplifying a bit, we can say that a **memory leak in Java is a situation where some objects are not used by the application any more, but GC fails to recognize them as unused**. As a result, these objects remain in memory indefinitely, reducing the amount of memory available to the application.

**OOPS :**

*Object*

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

*Class*

**Collection of objects** is called class. It is a logical entity.

*Inheritance*

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

***Polymorphism***

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

***Abstraction***

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

*Encapsulation*

**Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

**Can we override static method?**

No, static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

**Why we cannot override static method?** because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

**Can we override java main method?**

No, because main is a static method.

**Polymorphism in java** is a concept by which we can perform a *single action by different ways*. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in java: **Runtime polymorhism( Dynamic polymorphism)** and **Compile time polymorphism (static polymorphism)**. . We can perform polymorphism in java by method overloading and method overriding.

[**Method overriding**](http://beginnersbook.com/2014/01/method-overriding-in-java-with-example/#_blank) is a perfect example of runtime polymorphism. In this kind of polymorphism, reference of class X can hold object ofclass X or an object of any sub classes of class X. For e.g. if class Yextends class X then both of the following statements are valid:

Since in method overriding both the classes(base class and child class) have same method, compile doesn’t figure out which method to call at compile-time. In this case JVM(java virtual machine) decides which method to call at runtime that’s why it is known as runtime or dynamic polymorphism.

Compile time Polymorhism( or Static polymorphism)

Compile time polymorphism is nothing but the **method overloading** in java. In simple terms we can say that a class can have more than one methods with same name but with different number of arguments or different types of arguments or both.

**Method overloading** is also known as **Static Polymorphism** .

[**Static Polymorphism**](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/#_blank) is also known as compile time binding or early binding.

If you overload static method in java, it is the example of compile time polymorphism.

Here, we will focus on runtime polymorphism in java.

**class** X {

**void** methodA(**int** num)

{

System.***out***.println ("methodA:" + num);

}

**void** methodA(**int** num1, **int** num2)

{

System.***out***.println ("methodA:" + num1 + "," + num2);

}

**double** methodA(**double** num) {

System.***out***.println("methodA:" + num); **return** num;

}

}

**class** Y {

**public** **static** **void** main (String args [])

{

X Obj = **new** X(); **double** result; Obj.methodA(20); Obj.methodA(20, 30); result = Obj.methodA(5.5);

System.***out***.println("Answer is:" + result);

} }

output : methodA:20 methodA:20,30 methodA:5.5 Answer is:5.5

In such scenario, compiler is able to figure out the method call at compile-time that’s the reason it is known as compile time polymorphism.

**Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass.

### Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:

**class** Bike{

**void** run(){System.***out***.println("running");}

}

**class** Splender **extends** Bike{

**void** run(){System.***out***.println("running safely with 60km");}

**public** **static** **void** main(String args[]){ Bike b = **new** Splender();//upcasting

b.run();

} }

Output:running safely with 60km.

Splendar class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, **subclass method is invoked at runtime**. Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

Method is overridden not the datamembers, so runtime polymorphism can't be achieved by data members.

***Rule: Runtime polymorphism can't be achieved by data members.***

**class** Bike{ **int** speedlimit=90;

}

**class** Honda3 **extends** Bike{ **int** speedlimit=150;

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

Bike obj=**new** Honda3();

System.out.println(obj.speedlimit);//output : 90

}

#### Java Runtime Polymorphism with Multilevel Inheritance

**class** Animal{

**void** eat(){System.***out***.println("eating");}

}

**class** Dog **extends** Animal{

**void** eat(){System.***out***.println("eating fruits");}

}

**class** BabyDog **extends** Dog{

**void** eat(){System.***out***.println("drinking milk");}

**public** **static** **void** main(String args[]){ Animal a1,a2,a3; a1=**new** Animal(); a2=**new** Dog(); a3=**new** BabyDog();

a1.eat(); a2.eat(); a3.eat();

}

}

Output: eating eating fruits drinking Milk

**class** Animal{

**void** eat(){System.***out***.println("animal is eating...");}

}

**class** Dog **extends** Animal{

**void** eat(){System.***out***.println("dog is eating...");}

}

**class** BabyDog1 **extends** Dog{

**public** **static** **void** main(String args[]){ Animal a=**new** BabyDog1();

a.eat();

}} Output: Dog is eating

#### There are two types of binding

1.static binding (also known as early binding).

2.dynamic binding (also known as late binding).

### static binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

### Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

**Can we Override static methods in java?**

We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is ‘No’.

If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class.

/\* Java program to show that if static method is redefined by a derived class, then it is not overriding. \*/

// Superclass **class** Base {

// Static method in base class which will be hidden in subclass

**public** **static** **void** display() {

System.***out***.println("Static or class method from Base");

}

// Non-static method which will be overridden in derived class **public** **void** print() {

System.***out***.println("Non-static or Instance method from Base");

}

}

// Subclass

**class** Derived **extends** Base {

// This method hides display() in Base

**public** **static** **void** display() {

System.***out***.println("Static or class method from Derived");

}

// This method overrides print() in Base

**public** **void** print() {

System.***out***.println("Non-static or Instance method from Derived");

}

}

// Driver class **public** **class** Test {

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

Base obj1 = **new** Derived();

// As per overriding rules this should call to class Derive's static

// overridden method. Since static method can not be overridden, it

// calls Base's display() obj1.*display*();

// Here overriding works and Derive's print() is called obj1.print();

}

}

Output:

Static or class method from Base

Non-static or Instance method from Derived

Following are some important points for method overriding and static methods in Java. **1)** For class (or static) methods, the method according to the type of reference is called, not according to the abject being referred, which means method call is decided at compile time. **2)** For instance (or non-static) methods, the method is called according to the type of object being referred, not according to the type of reference, which means method calls is decided at run time. **3)** An instance method cannot override a static method, and a static method cannot hide an instance method. For example, the following program has two compiler errors.

**4)** In a subclass (or Derived Class), we can overload the methods inherited from the superclass. Such overloaded methods neither hide nor override the superclass methods — they are new methods, unique to the subclass.

JAVA 8 Features

In which programming paradigm Java 8 falls?

* Object-oriented programming language.
* Functional programming language.
* Procedural programming language.
* Logic programming language

3. What are the significant advantages of Java 8?

* Compact, readable, and reusable code.
* Less boilerplate code.
* Parallel operations and execution.
* Can be ported across operating systems.
* High stability.
* Stable environment.
* Adequate support

5. What are functional or SAM interfaces?

Functional Interfaces are an interface with only one abstract method. Due to which it is also known as the Single Abstract Method (SAM) interface. It is known as a functional interface because it wraps a function as an interface or in other words a function is represented by a single abstract method of the interface.

Functional interfaces can have any number of default, static, and overridden methods. For declaring Functional Interfaces @FunctionalInterface annotation is optional to use. If this annotation is used for interfaces with more than one abstract method, it will generate a compiler error.

For Eg-:

|  |
| --- |
| @FunctionalInterface // Annotation is optional **public** interface **Foo**() {  // Default Method - Optional can be 0 or more **public** **default** String **HelloWorld**() { **return** "Hello World";  }  // Static Method - Optional can be 0 or more **public** **static** String **CustomMessage**(String msg) { **return** msg;  }  // Single Abstract Method **public** **void** **bar**();  }  **public** **class** **FooImplementation** **implements** **Foo** {  // Default Method - Optional to Override  @Override  **public** **default** String **HelloWorld**() { **return** "Hello Java 8";  }  // Method Override  @Override **public** **void** **bar**() {  System.out.println(“Hello World”);  }  } |

**public**

**static**

**void**

**main**

(

String

[]

args)

{

FooImplementation

fi

=

**new**

FooImplementation();

System.out.println(fi.HelloWorld());

System.out.println(fi.CustomMessage(“Hi”));

fi.bar();

}

6

. Can a functional interface extend/inherit another interface?

A functional interface cannot extend another interface with abstract methods as it will void the rule of one abstract method per functional interface. E.g:

|  |
| --- |
| **interface** **Parent** {  **public** **int** **parentMethod**();  }  @FunctionalInterface // This cannot be FunctionalInterface **interface** **Child** **extends** **Parent** { **public** **int** **childMethod**();  // It will also extend the abstract method of the Parent Interface  // Hence it will have more than one abstract method  // And will give a compiler error } |

It can extend other interfaces which do not have any abstract method and only have the default, static, another class is overridden, and normal methods. For eg:

**interface**

**Parent**

{

**public**

**void**

**parentMethod**

()

{

System.out.println(

"Hello"

)

;

}

}

@FunctionalInterface

**interface**

**Child**

**extends**

**Parent**

{

**public**

**int**

**childMethod**

()

;

}

7

. What is the

default method, and why is it required?

A method in the interface that has a predefined body is known as the default method. It uses the keyword default. default methods were introduced in Java 8 to have 'Backward Compatibility in case JDK modifies any interfaces. In case a new abstract method is added to the interface, all classes implementing the interface will break and will have to implement the new method. With default methods, there will not be any impact on the interface implementing classes. default methods can be overridden if needed in the implementation. Also, it does not qualify as synchronized or final.

What are static methods in Interfaces?

Static methods, which contains method implementation is owned by the interface and is invoked using the name of the interface, it is suitable for defining the utility methods and cannot be overridden.

9. What are some standard Java pre-defined functional interfaces?

**Runnable:** use to execute the instances of a class over another thread with no arguments and no return value.

**Callable:** use to execute the instances of a class over another thread with no arguments and it either returns a value or throws an exception.

**Comparator:** use to sort different objects in a user-defined order

**Comparable:** use to sort objects in the natural sort order

What are the various categories of pre-defined function interfaces?

**Function:** To transform arguments in returnable value.

**Predicate:** To perform a test and return a Boolean value.

**Consumer:** Accept arguments but do not return any values.

**Supplier:** Do not accept any arguments but return a value.

**Operator:** Perform a reduction type operation that accepts the same input types.

11. What is the lambda expression in Java and How does a lambda expression relate to a functional interface?

Lambda expression is a type of function without a name. It may or may not have results and parameters. It is known as an anonymous function as it does not have type information by itself. It is executed on-demand. It is beneficial in iterating, filtering, and extracting data from a collection.

As lambda expressions are similar to anonymous functions, they can only be applied to the single abstract method of Functional Interface. It will infer the return type, type, and several

arguments from the signature of the abstract method of functional interface

In Java 8, what is Method Reference?

Method reference is a compact way of referring to a method of functional interface. It is used to refer to a method without invoking it. :: (double colon) is used for describing the method reference. The syntax is class::methodName

For e.g.:

Integer::parseInt(str) \\ method reference str -> Integer.ParseInt(str); \\ equivalent lambda

18. What is an Optional class?

Optional is a container type which may or may not contain value i.e. zero(null) or one(not-null) value. It is part of java.util package. There are pre-defined methods like isPresent(), which returns true if the value is present or else false and the method get(), which will return the value if it is present.

What are the advantages of using the Optional class?

Below are the main advantage of using the Optional class:

It encapsulates optional values, i.e., null or not-null values, which helps in avoiding null checks, which results in better, readable, and robust code It acts as a wrapper around the object and returns an object instead of a value, which can be used to avoid run-time NullPointerExceptions.

What are Java 8 streams?

A stream is an abstraction to express data processing queries in a declarative way.

A Stream, which represents a sequence of data objects & series of operations on that data is a data pipeline that is not related to Java I/O Streams does not hold any data permanently. The key interface is java.util.stream.Stream<T>. It accepts Functional Interfaces so that lambdas can be passed.

What are the most commonly used Intermediate operations?

**Filter(Predicate<T>)** - Allows selective processing of Stream elements. It returns elements that are satisfying the supplied condition by the predicate.

**map(Funtion<T, R>)** - Returns a new Stream, transforming each of the elements by applying the supplied mapper function.= sorted() - Sorts the input elements and then passes them to the next stage.

**distinct()** - Only pass on elements to the next stage, not passed yet. **limit(long maxsize)** - Limit the stream size to maxsize. **skip(long start)** - Skip the initial elements till the start. **peek(Consumer)** - Apply a consumer without modification to the stream.

**flatMap(mapper)** - Transform each element to a stream of its constituent elements and flatten all the streams into a single stream

What is the most common type of Terminal operations?

* collect() - Collects single result from all elements of the stream sequence.
* reduce() - Produces a single result from all elements of the stream sequence o count() - Returns the number of elements on the stream. o min() - Returns the min element from the stream. o max() - Returns the max element from the stream.
* Search/Query operations o anyMatch() , noneMatch() , allMatch() , ... - Short-circuiting operations. o Takes a Predicate as input for the match condition. o Stream processing will be stopped, as and when the result can be determined.
* Iterative operations o forEach() - Useful to do something with each of the Stream elements. It accepts a consumer.

o forEachOrdered() - It is helpful to maintain order in parallel streams.

What is the difference between findFirst() and findAny()?

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **findFirst()** | | |  | | --- | | **findAny ()** | |

Returns the first element in the Stream Return any element from the Stream

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | Deterministic in nature | | |  | | --- | | Non-deterministic in nature | |

How are Collections different from Stream?

Collections are the source for the Stream. Java 8 collection API is enhanced with the default methods returning Stream<T> from the collections.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Collections** | | Data structure holds all the data elements | | External Iteration | | Can be processed any number of times | | Elements are easy to access | | Is a data store | | |  | | --- | | **Streams** | | No data is stored. Have the capacity to process an infinite number of elements on demand | | Internal Iteration | | Traversed only once | | No direct way of accessing specific elements | | Is an API to process the data | |

Lambdas in Java 8

What is method reference-

A method reference is the shorthand syntax for a lambda expression that executes just **ONE** method.

Here's the general syntax of a method reference:

Why wait,notify and notify all in object class not in thread class

Ans**Locks are made available on per Object basis**, which is another reason wait and notify is declared in Object class rather then Thread class.

In Java in order to enter critical section of code, Threads needs lock and they wait for lock, they don't know which threads holds lock instead they just know the lock is hold by some thread and they should wait for lock instead of knowing which thread is inside the synchronized block and asking them to release lock. this analogy fits with wait and notify being on object class rather than thread in Java.

Difference B/w start and run method

Start-Whenever we call start method new thread is created. while when we call run method no new thread is created directly. Code inside the run method execute the current thread.

2)We don’t call start method twice. If we call start method twice on thread object.once start second call of start method will give illegal state exception

3)Start method is time consuming while run is less time consuming

If we will not give @Id in entity class ..then exception will come

|  |  |
| --- | --- |
| Ans could not execute statement; SQL [n/a]; nested exception is |  |
| org.hibernate.exception.SQLGrammarException: could not execute statement", | |

Why countDownLatch is used? It increase or decrease the count?

*Ans Increase Java application performance using CountDownLatch. CountDownLatch class to synchronize the threads. A CountDownLatch is very similar to a count down timer, except that it has to be counted down explicitly.*

Steps

* **Instantiate a CountDownLatch object** in main method with the number of files to be compressed.
* **Spawn a new thread** for each file compression task, passing the file name and CountDownLatch object to each thread, and then start the thread.
* Once the thread has finished its task (compression), have it **decrement the CountDownLatch counter** before destroying itself.
* At the end of main function, **wait for CountDownLatch to hit zero**, after which the program ends.

This way, you ensure that the main function doesn’t end before the threads do, and avoid the risk of running zombie/orphaned threads.

We will create a **main class, a compression class and a worker class** which will handle the pre-processing and post-processing (if any) and decrement the CountDownLatch

How two thread communicate

Ans **Thread communicate** via shared memory. ... Once the **thread** is accessing the object any **other thread** has to wait until this **thread** release the object. The accessing **thread** notifies all the waiting **thread** via notify or notifyall method defined in object class.Once the thread is accessing the object any other thread has to wait until this thread release the object. The accessing thread notifies all the waiting thread via notify or notifyall method defined in object class. ...

Threads can access everything just like that. A simple non threaded application has one thread.

What is interthread communication?

Ans **Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

o wait() o notify() o notifyAll()

what are commercial ways of garbage collection in market

What is the difference between system GC and runtime GC in Java?

**System**.**gc**() is effectively equivalent to **Runtime**.**gc**() . **System**.**gc**() internally calls**Runtime**.**gc**() . The only **difference** is **System**.**gc**() is a class method where as**Runtime**.**gc**() is an instance method. So, **System**.**gc**() is more convenient.

What are the garbage collection algorithms in Java?

Algorithm of garbage Collection

* 3.1. Serial Garbage Collector. This is the simplest GC implementation, as it basically works with a single thread. ...
* 3.2. Parallel Garbage Collector. It's the default GC of the JVM and sometimes called Throughput Collectors. ...
* 3.3. CMS Garbage Collector. ... Concurrent Mask Sweep
* 3.4. G1 Garbage Collector. ...
* 3.5. Java 8 Changes.

|  |  |  |  |
| --- | --- | --- | --- |
| Producer and consumer Problem  A a1=new A(); A a2=new A(); a1==a2 a1.equals(a2)  How concurent hashmap follow thread safety? | | | |
| **ConcurrentHashMap** is thread safe without **synchronizing** the whole map. Reads can happen | | |  |
| very fast while write is done with a lock. Both are **synchronized** version of HashMap, with | |  |
| difference in their core functionality and their internal structure |  |
| Difference between protected and default  The protected specifier allows access by all subclasses of the class in a program, whatever package they reside in, as well as to other code in the same package.  The default specifier allows access by other code in the same package, but not by code that is in subclasses residing in different packages.    If two duplicates key with different value .How we can store the value in hashmap  Ans Linklist  What are the classes of set interface |

Ans HashSet,LinkHashSet,TreeSet,EnumSet How Tree Sets internally Work?

Ans The data structure for the TreeSet is TreeMap; it contains a **SortedSet** & **NavigableSet** interface to keep the elements sorted in ascending order and navigated through the tree.As soon as we create a TreeSet, the [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) creates a TreeMap to store the elements in it and performs all the operations on it. It's working is similar to HashSet.

A lock allows only one thread to enter the part that's locked and the lock is not shared with any other processes.

A mutex is the same as a lock but it can be system wide (shared by multiple processes).

A [semaphore](https://stackoverflow.com/a/2350628/58553) does the same as a mutex but allows x number of threads to enter, this can be used for example to limit the number of cpu, io or ram intensive tasks running at the same time.

Why we called wait,notify and notifyall in synchronized block or method?

Ans-Illegal state Exception will come if we don’t call wait ,notify and notifyall method through synchronized context

Any Potential Race Condition between wait and notify method.

How to get how many thread are running?

Note that [activeCount()](http://docs.oracle.com/javase/6/docs/api/java/lang/Thread.html#activeCount()) only gives you the number of threads *in the local thread group*, but there are multiple thread groups in the JVM

|  |
| --- |
| thread.isAlive() |

Thread.getAllStackTraces() will give you a map where each Thread is key. You can then examine the state of each Thread and check .

Map<Thread, StackTraceElement[]> threads = Thread.getAllStackTraces();

You can get a set of running threads from Thread class with getAllStackTraces() method. Iterate that set and print current status using getState() API When we call run method directly then what happen?

Ans Run() method act as java method if we call directly run method

What is JIT Compiler?

The JIT compiler is enabled by default, and is activated when a Java method is called. The JIT compiler compiles the **bytecodes** of that method into native machine code, compiling it "just in time" to run. When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it.

) You have thread T1, T2 and T3, how will you ensure that thread T2 run after T1 and thread T3 run after T2?

How will you awake a blocked thread in java?

Ans If thread is blocked on IO then I don't think there is a way to interrupt the thread.On the other hand.If thread is blocked due to result of calling wait(), sleep() or join() method you can interrupt the thread and it will awake by throwing InterruptedException

3) Cyclic barrier and Countdown latch

Ans You can use cyclic barrier again when the barrier is broken but countdown latch is not reuse

Why we call wait ,notify and notifyall used inside syncronised method or block?

Ans If we don't call wait() or notify() method fro m synchronized context we will receive IllegalMonitorStateException.

2) Any potential race condition between wait and notify method in Java

4)Producer and consumer problem?

Ans Producer thread should wait if the queue is full and consumer thread should wait if the queue is empty

5)When we use notify andnotify All method?

Ans-If some thread is waiting for some condition to become true, you can use notify and notifyAll methods to inform them that condition is now changed and they can wake up

6) Difference b/w Notify and NotifyAll?

Ans Notify method sends notification to only one of the waiting thread. no gurantee which thread will receive notification.

Notify all method sends notification to all thread

7)Synchronized and wait and notify or Why wait and notify inside Object class

Ans 1) Synchronized is to provide mutual exclusion and ensuring thread safety of Java class like race condition while wait and notify are communication mechanism between two thread.

2)Locks are made available on per Object basis

1. Sleep and yeild are defined in thread class and wait is defined in Object class
2. Difference b/w Yield and sleep

Ans The major difference between yield and sleep in Java is that yield() method pauses the currently executing thread temporarily for giving a chance to the remaining waiting threads of the same priority to execute. If there is no waiting thread or all the waiting threads have a lower priority then the same thread will continue its execution.

Yield-

* + 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**.](http://howtodoinjava.com/java-5/java-executor-framework-tutorial-and-best-practices/)
  + 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.

Join

The join() method of a Thread instance can be **used to “join” the start of a thread’s execution to the end of another thread’s execution** so that a thread will not start running until another thread has ended. If join() is called on a Thread instance, the currently running thread will block until the Thread instance has finished executing.

Concurrent hashMap and Syncronised hashmap?

Ans) ConcurrentHashMap does not allow null keys or null values while synchronized HashMap allows one null key.

12) Tree Set and Tree map

Ans 1) TreeSet implements Set interface while TreeMap implements Map interface in

Java.

2)TreeSet stores only one object while TreeMap uses two objects called key and Value. Objects in TreeSet are sorted while keys in TreeMap remain in sorted Order.

3)TreeSet and TreeMap is that, former implements NavigableSet while later implements NavigableMap in Java.

4)Duplicate objects are not allowed in TreeSet but duplicates values are allowed in

TreeMap.

Why Concurrent HashMap does not allow null key and value?

Ans the main reason null keys are not allowed in Concurrent maps because of ambiguities that may be just barely tolerable in non-concurrent map can't be accomodated.The main one is that when map.get(key) returns null. You cant detect whether key explicitly maps to null or key is not mapped. In non concurrent map you can check this via containsKey(key) but in concurrent it might have change between calls.

if (m.containsKey(k)) { return m.get(k);

} else { throw new KeyNotPresentException();

}

Since m is a concurrent map, key k may be deleted between the containsKey and get calls, causing this snippet to return a null that was never in the table, rather than the desired KeyNotPresentException.

Normally you would solve that by synchronizing, but with a concurrent map that of course won't work. Hence the signature for get had to change, and the only way to do that in a backwards-compatible way was to prevent the user inserting null values in the first place, and continue using that as a placeholder for "key not found".

**Spring FrameWork-:**

What are the ways to create beans in Spring- Ans Three ways to configure bean- we define a bean in configuration XML.

By annotation- @Component

@Service

@Bean

Do we set isolation level in DB-

In hibernate islation level is set read commit by default. But manually we can set isolation level.Serializable is the best isolation level.

What to if our one microservice will fail and not give response to other microservice.How to do backtracking

Ans -We can do back tracking by saving the logs in cache and wait tillthe response of second microservice. Or We can use save in database or stored in kibana logging tool.In this way we do backtracking

How we can avoid dirty read

Ans And BY setting isolation level

What is advantage of using API gateway?

An API Gateway includes:

* Security o Caching
* API composition and processing o Managing access quotas o API health monitoring o Versioning o Routing

Spring Cloud- Spring Cloud on the balance load on service level by loadbalalncer where AWS balance load on the application level

Hystrix- If we got any error then we redirect to any default page through Hystrix.It will not show error message if service fails. It is given Netflix.

FeignClient-Through fiegn client we can interact two microservice.

@SpringBoot Annotation contain @EnableAutoConfiguration,ComponentScan,Configuration

@Configuration annotation indicates that **a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests** for those beans at runtime.

What is the use of @EnableAutoConfiguration annotation?

The @EnableAutoConfiguration annotation **enables Spring Boot to auto-configure the application context**. Therefore, it automatically creates and registers beans based on both the included jar files in the classpath and the beans defined by us.

@Autowired **: Spring** @Autowired **annotation is used for automatic injection of beans. Spring @Qualifier annotation is used in conjunction with**Autowired **to avoid confusion when we have two of more bean** configured **for same type.**

How to do patch for update and delete operation simulteanously?

Ans We need to mentioned operation type for for all the operation type inside JSON format. Suppose two list is there then we need to mentioned operation type for one list as update and other list json operation would be delete.

To support deleting, creating, and updating in a single call, that's not really supported by standard REST conventions. One possibility is a special "batch" service that lets you assemble calls together:

POST /batch

[

{ method: 'POST', path: '/items', body: { title: 'foo' } },

{ method: 'DELETE', path: '/items/bar' }

]

**What is Jdbc Template?**

Spring **JdbcTemplate** is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API. Like Exceptionhandling,,creation connection and closing connection

Types of Transaction

Declarative transaction

Programmatic Transaction

**What is Symmetric and asymmetric key in Spring security?**

**Ans Symmetric encryption** uses a single **key** that needs to be shared among the people who need to receive the message while **asymmetrical encryption** uses a pair of public **key** and a private **key** to encrypt and decrypt messages when communicating.

Difference between Public key and private key?

Ans the **public key** is used for data encryption while the **private key** is used for data decryption **Why we are using Yml file?**

One of the benefits of **using YAML** is that the information in a single **YAML file** can be easily translated to multiple language types.

Where is YML used

Ans It is a human-readable data-serialization language. It is commonly **used** for configuration files, but could be **used** in many applications where data is being stored (e.g. debugging output) or transmitted (e.g. document headers).

How to create composite key in hibernate?

Ans- First create a class with annotation @embedable. In that class both id field will be there.Then in another class we take the reference of that class and used @Embedded id above that.

How to maintain state in microservice?

There are actually two kinds of state. Application State that lives on the client and Resource State that lives on the server..It is easy to cache.

**Difference between bean and component? Ans-:**

1. @Component **auto detects** and configures the beans using classpath scanning whereas @Bean **explicitly declares** a single bean, rather than letting Spring do it automatically.
2. @Component **does not decouple** the declaration of the bean from the class definition where as @Bean **decouples** the declaration of the bean from the class definition.
3. @Component is a **class level annotation** where as @Bean is a **method level annotation** and name of the method serves as the bean name.
4. @Component **need not to be used with the @Configuration** annotation where as @Bean annotation has to be **used within the class which is annotated with @ Configuration**.
5. We **cannot create a bean** of a class using @Component, if the class is outside spring container whereas we **can create a bean** of a class using @Bean even if the class is present **outside the spring container**.
6. @Component has **different specializations** like @Controller, @Repository and @Service whereas @Bean has **no specializations.**

#### **PUT POST**

PUT method requests for the enclosed entity be stored under the supplied [Request-URI.](https://restfulapi.net/resource-naming/) If the Request-URI refers to an already existing resource – an update operation will happen, otherwise create operation should happen

|  |  |
| --- | --- |
| PUT method is [idempotent.](https://restfulapi.net/idempotent-rest-apis/) So if you send retry a POST is NOT idempotent. So if you retry the request N request multiple times, that should be equivalent to times, you will end up having N resources with N  single request modification. different URIs created on server.  Use PUT when you want to modify a singular resource | |
| which is already a part of resources collection. PUT replaces the resource in its entirety. Use PATCH if request updates part of the resource. | Use POST when you want to add a child resource under resources collection. |
| PUT is idempotent, so you can cache the response.  Generally, in practice, always use PUT for UPDATE | Responses to this method are not [cacheable,](https://restfulapi.net/caching/) unless the response includes appropriate Cache-Control or Expires header fields. However, the 303 (See Other) response can be used to direct the user agent to retrieve a cacheable resource. |

PUT /questions/{question-id}

The

POST

method is used to request that the origin

serv

er accept the entity enclosed in the request as a

new subordinate of the resource identified by the

Request

-

URI in the Request

-

Line. It essentially means

that

POST

request

-

URI should be of a collection URI.

POST /questions

Always use POST for CREATE operations. operations.

**Differnace between PUT and PATCH**

**ANS 1)** In a PUT request, the enclosed entity is considered to be a modified version of the resource stored on the origin server, and the client is requesting that the stored version be replaced.

1) With PATCH, however, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version.

2)Also, another difference is that when you want to update a resource with PUT request, you have to send the full payload as the request whereas with PATCH, you only send the parameters which you want to update.

3) Suppose we have a resource that holds the first name and last name of a person.

If we want to change the first name then we send a **put** request for Update

{ "first": "Michael", "last": "Angelo" }

Here, although we are only changing the first name, with PUT request we have to send both parameters first and last.

In other words, it is mandatory to send all values again, the full payload

When we send a PATCH request, however, we only send the data which we want to update. In other words, we only send the first name to update, no need to send the lastname.

PATCH request requires less bandwidth than put

Why Put is idempotent?

From a RESTful service standpoint, for an operation (or service call) to be **idempotent**, clients can make that same call repeatedly while producing the same result. In other words, making multiple identical requests has the same effect as making a single request.

Which command line library we used to test the service like Postman?

Ans Http Apache client

Eureka Server- When we register Eureka Service application at that time we used @EnableEurekaServer

EurekaClient- In each and every microservice we used @EnableEurekaClient and

@EnableDiscoveryClient

Zuul register-@EnableZuulProxy it will enable zuul in the application

Then we have to configure zuul in our UI-

@EnableEurekaClient register all the the microservice. Then Zuul check whether these microservice is configured inside Eureka or not. If it registered then it will call the application with ZUUL URl with application name. Each microservice has there own application name. So ZUUL URL call the microservice with application name.

**What are fallback dependency in microservice?**

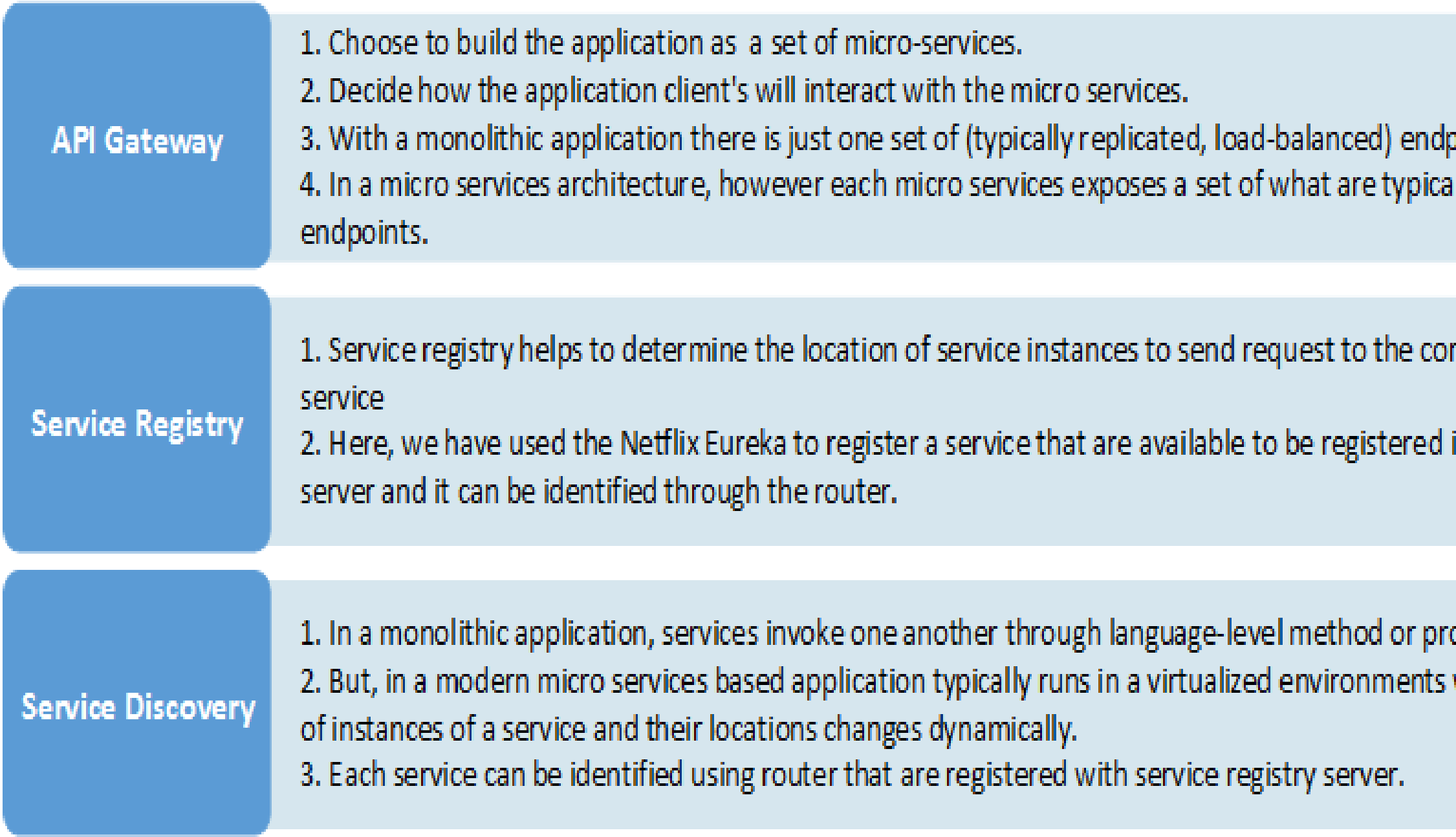
**Ans** <dependency<groupId>org.springframework.cloud</groupId> <artifactId>spring-cloud-starter-hystrix</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-hystrix-dashboard</artifactId> </dependency>

**Patterns in Micro services Architecture**

**Ans 1) Decomposition pattern- Decompose by Business Capability**

**Decompose By Subdomain**

**Strangler pattern**

1. **Integration pattern--- Api Gateway pattern**

**Aggregator pattern**

**Client Side Ui Composition pattern**

1. **DataBase pattern-------Database per Service**

**Shared Database per Service**

**Command Query Resposibilty Seqregation**

**Saga Pattern**

1. **Observability pattern—Log Aggregation-AWS Cloud Watch and PCF do this**

**Performance metrics**

**Distributed Tracing-**Spring Cloud Slueth, along with Zipkin server, is a common implementation.

**Health Check-**Spring Boot actuator

1. **Cross Cutting Concern pattern—External Configuation-**Spring Cloud config server provides the option to externalize the properties to GitHub and load them as environment properties.

**Service Discovery pattern-**A service registry needs to be created which will keep the metadata of each producer service. A service instance should register to the registry when starting and should de-register when shutting down. There are two types of service discovery: client-side and server-side. An example of client-side discovery is Netflix Eureka and an example of server-side discovery is AWS ALB.

**Circuit Breaker Pattern-**Netflix Hystrix is a good implementation of the circuit breaker pattern

12Factor of microservice-

* 1. Codebase
  2. Dependencies
  3. Config
  4. BackingService
  5. Build Release and run
  6. Processes
  7. PortBinding
  8. Concurrency
  9. Disposibility
  10. Dev/prod priority
  11. Logs
  12. 12 Admin Processes

**The SOLID Programming Principles of OOP**

* 1. Single Responsibility **Principle**. - There should never be more than one reason to for a class to change. Class focuses on single functionality addresses a specific concern
  2. Open Closed **Principle**. — Software entities (classes, modules, methods etc) should be open for extension but closed for modification.

Open for Extension means extend existing behaviour

Closed for Modification means Existing code remain unchanged. To completely avoid modification of existing base class

* 1. Liskov Substitution **Principle**.—We should substitute base class object with child class objects and this should not alter behaviour/characteristics of program
  2. Interface Segregation **Principle**- Client should not be forced to depend upon interfaces that they do not use. Interface Pollution----Large Interface, unrelated methods.

Classes have empty method implementation

Methods implementation throws UnsupportedOperationException.

Method implementation return null, default or dummy values 5) Dependency Inversion **Principle—High level Module should not depend upon low level Modules. Both should depend upon abstraction.**

**Abstraction should not depend upon details. And details should depend upon abstraction**

Difference between oauth1 and oauth2

Ans 1) Where OAuth 2.0 defines four roles, (client, authorization server, resource server, and resource owner,) OAuth 1 uses a different set of terms for these roles.

2) The OAuth 2.0 “client” is known as the “consumer,” the “resource owner” is known simply as the “user,” and the “resource server” is known as the “service provider”. OAuth 1 also does not explicitly separate the roles of resource server and authorization server.

**How to secure particular service method access by particular user role.**

The key points are:

* + Use of hasRole – note you need to append ROLE\_ to your roles.
  + We could also add PreAuthorize security to custom methods defined in the repository.

@PreAuthorize("hasRole('ROLE\_USER')")

public interface ParkrunCourseRepository extends CrudRepository<ParkrunCourse

, Long> {

@Override

@PreAuthorize("hasRole('ROLE\_ADMIN')")

ParkrunCourse save(ParkrunCourse parkrunCourse);

1. The terms “two-legged” and “three-legged” have been replaced by the idea of grant types, such as the [Client Credentials](https://www.oauth.com/oauth2-servers/access-tokens/client-credentials/) grant type and the [Authorization Code](https://www.oauth.com/oauth2-servers/access-tokens/authorization-code-request/) grant type.

**Spring**

**TypesHandler Mapper In Spring**

* 1. **BeanNameUrlHandlerMapping**
  2. **DefaultAnnotationHandlerMapping**

**3)ControllerBeanNameHandlerMapping**

1. **ControllerClassNameHandlerMapping**
2. **SimpleUrlHandlerMapping**  6) **RequestMappingHandlerMapping**

**Difference between Page and Slice?**

Ans A Page knows about the total number of elements and pages available. It does so by the infrastructure triggering a count query to calculate the overall number. As this might be expensive depending on the store used, Slice can be used as return instead. A Slice only knows about whether there’s a next Slice available which might be just sufficient when walking through a larger result set.

**When we print Object of any class ..Then what It will print. And that variable denote?** As other have told you, using System.out.println with an object will call

to toString method on that object. If the class doesn't have it's own toString method, then it's a call to the super class's toString. If the super class call goes all the way back to java.lang.Object , the default toString method prints the name of the object's type (what class it is), followed by an @ sign, and the memory location of the object--the hexidecimal address of where that object is stored in memory.

**Caching works on Cache Aside pattern?**

If we need to know how many objects created of that class.Then how we will do?

Ans We can increase count variable inside that Pojo class whose object is created inside Instance Initialize Block

**What are the different types of IOC (dependency injection) ?**

There are three types of dependency injection:

* **Constructor Injection** (e.g. Pico container, Spring etc): Dependencies are provided as constructor parameters.
* **Setter Injection** (e.g. Spring): Dependencies are assigned through JavaBeans properties (ex: setter methods).
* **Interface Injection** (e.g. Avalon): Injection is done through an interface.

Index-

Indexes are **special lookup tables** that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table.

An index helps to speed up **SELECT** queries and **WHERE** clauses, but it slows down data input, with the **UPDATE** and the **INSERT** statements

**Single-Column Indexes**

A single-column index is created based on only one table column

**Unique Indexes**

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table.

**Composite Indexes**

A composite index is an index on two or more columns of a table.

**Implicit Indexes**

Implicit indexes are indexes that are automatically created by the database server when an object is created.

Difference between HQL And Criteria?

Ans-: HQL is to perform Both select and non-select operation on data but criteria is only for selection the data, we cannot perform non-select operations using criteria.

1. HQL is suitable for executing static query but criteria is suitable for executing dynamic query.
2. HQL doesn’t support pagination concept but we can achieve pagination with criteria
3. Criteria used more time to execute than HQL

Difference Between Delete, Truncate and Drop?

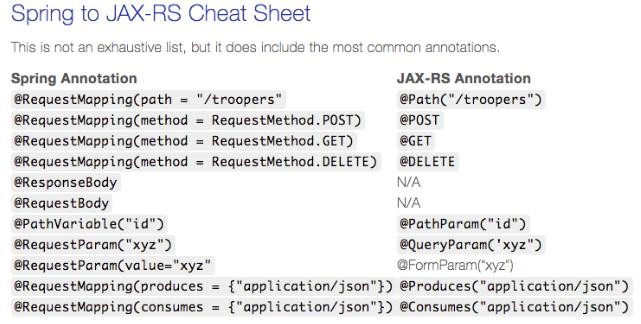
1)Drop and Truncate is DDL Command where Delete is a DML command

2)Drop and Truncate operation cannot be rolled back but delete operation can be rolled back 3)Truncate cannot use where clause but we can use where clause with delete to filter and delete specific records

4)Truncate is executed using a table lock and whole table is locked for remove all records but delete is executed using a row locks and each row in table is locked for deletion

5)In Truncate minimal logging in transaction logging so its performance wise faster but delete maintain log so it is slower than truncate.

JAX-RS is a [specification](http://download.oracle.com/otn-pub/jcp/jaxrs-2_0-fr-eval-spec/jsr339-jaxrs-2.0-final-spec.pdf) for implementing REST web services in Java, currently defined by the [JSR-370.](https://jcp.org/en/jsr/detail?id=370) It is part of the [Java EE technologies,](https://stackoverflow.com/a/37083274/1426227) currently defined by the [JSR 366.](https://www.jcp.org/en/jsr/detail?id=366) [Jersey](https://jersey.java.net/) (shipped with GlassFish and Payara) is the JAX-RS reference implementation, however there are other implementations such as [RESTEasy](http://resteasy.jboss.org/) (shipped with JBoss EAP and WildFly) and [Apache CXF](https://cxf.apache.org/) (shipped with TomEE and WebSphere).



Spring framework provides an XML-based approach for configuring beans. Later Spring introduced XML-based DSLs, Annotations, an

d JavaConfig-based approaches for configuring beans.

Spring provides many approaches for doing the same thing, and we can even mix the approaches and use both JavaConfig and Annotation based configuration styles in the same application. Spring Problem

* You need to hunt for all the **compatible libraries** for the specific Spring version and configure them.
* 95% of the times we configure the **DataSource**, **EntitymanagerFactory**,**TransactionManager** etc beans in the same way. Wouldn’t it be great if Spring can do it for me automatically.
* Similarly we configure SpringMVC beans like **ViewResolver**, **MessageSource** etc in the same way most of the times
* Annotation
* @Repository-n Spring 2.0 and later, the @Repository annotation is a marker for any class that fulfills the role or stereotype (also known as Data Access Object or DAO) of a repository.
* @Component-Spring 2.5 introduces @component. @Component is a generic stereotype for any Spring-managed component.

@Repository, @Service, and @Controller are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.

Therefore, you can annotate your component classes with @Component, but by annotating them with @Repository, @Service, or @Controller instead, your classes are more properly suited for processing by tools or associating with aspects. For example, these stereotype annotations make ideal targets for pointcuts.

Thus, if you are choosing between using @Component or @Service for your service layer, @Service is clearly the better choice. Similarly, as stated above, @Repository is already supported as a marker for automatic exception translation in your persistence layer.

Annotation | Meaning |

+------------+-----------------------------------------------------+

| @Component | generic stereotype for any Spring-managed component |

| @Repository| stereotype for persistence layer |

| @Service | stereotype for service layer |

| @Controller| stereotype for presentation layer (spring-mvc)

•

Spring boot solve this problem

Why we use Spring-boot-starter-parent?

The "spring-boot-starter-parent" is a special starter that provides useful Maven defaults i.e it adds all the required jars and other things automatically. It also provides a **dependencymanagement section** so that you can omit version tags for dependencies you are using in pom.xml

1. **Easy dependency Management**
   * First thing to observe is we are using some dependencies named like **spring-bootstarter-\***. Remember I said “95% of the times I use the same configuration. So when you add **springboot-starter-web** dependency by default it will pull all the commonly used libraries while developing Spring MVC applications such as **spring-webmvc, jacksonjson, validation-api** and **tomcat**.
   * We have added **spring-boot-starter-data-jpa** dependency. This pulls all the **springdata-jpa** dependencies and also adds **Hibernate** libraries because the majority of the applications use Hibernate as JPA implementation.
2. **Auto Configuration**
   * Not only the **spring-boot-starter-web** adds all these libraries but also configures the commonly registered beans like **DispatcherServlet, ResourceHandlers, MessageSource** etc beans with sensible defaults.
   * We also added **spring-boot-starter-Thymeleaf** which not only adds the Thymeleaf library dependencies but also configures **ThymeleafViewResolver** beans as well automatically.
   * We haven’t defined any of the **DataSource, EntityManagerFactory,**

**TransactionManager**etc beans but they are automatically gets created. How? If we have any in-memory database drivers like **H2** or **HSQL** in our classpath then SpringBoot will automatically create an in-memory **DataSource** and then registers **EntityManagerFactory, TransactionManager** beans automatically with sensible defaults. But we are using MySQL, so we need to explicitly provide MySQL connection details. We have configured those MySQL connection details in **application.properties** file and SpringBoot creates a **DataSource** using these properties.

1. **Embedded Servlet Container Support**

The most important and surprising thing is we have created a simple Java class annotated with some magical annotation **@SpringApplication** having a main method and by running that main we are able to run the application and access it at **http://localhost:8080/**.

**Where is the servlet container comes from?**

We have added **spring-boot-starter-web** which pulls the **spring-boot-startertomcat** automatically and when we run the main() method it started tomcat as an **embedded container** so that we don’t have to deploy our applica tion on any externally installed tomcat server.

By the way have you observe that our packaging type in **pom.xml** is **‘jar’ not ‘war’**. Wonderful!

**Ok, but what if I want to use Jetty server instead of tomcat?**

Simple, exclude **spring-bootstarter-tomcat** from **spring-boot-starter-web** and include **springboot-starter-jetty**.

**Hibernate**

ORM-ORM stands for **O**bject-**R**elational **M**apping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages

What are the key component of Hibernate-

* **Configuration** - Represents a configuration or properties file required by the Hibernate.
* **SessionFactory** - Configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated.
* **Session** - Used to get a physical connection with a database.
* **Transaction** - Represents a unit of work with the database and most of the RDBMS supports transaction functionality.
* **Query** - Uses SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.
* **Criteria** - Used to create and execute object oriented criteria queries to retrieve objects.

**List out the design patterns used in Hibernate framework.**

* Domain Model Pattern - An object model of the domain that incorporates both behavior and data,
* **Data Access Object (DAO) Design Pattern**,
* **Abstract Factory**,
* **Data Mapper**,
* Proxy for **lazy loading**,
* **Object-Relational Mapping** (**ORM**),
* Query Object for **Criterion API**,
* and **facade**.

•

What are the two key component of a hibernate Configuration Object

* **Database Connection:** This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.
* **Class Mapping Setup**

What is a configuration object in hibernate?

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate.

What is a SessionFactory in hibernate?

Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated The SessionFactory is a thread safe object and used by all the threads of an application.The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple SessionFactory objects.

What is Session in hibernate?

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

Hibernate Versioning

Once an object is saved in a database, we can modify that object any number of times right, If we want to know how many no of times that an object is modified then we need to apply this versioning concept.

When ever we use versioning then hibernate inserts version number as **zero**, when ever object is saved for the first time in the database. Later hibernate increments that version no by one automatically when ever a modification is done on that particular object.

In order to use this versioning concept, we need the following two changes in our application

* Add one property of type int in our pojo class
* In hibernate mapping file, add an element called version soon after id element

<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="str.Product" table="products">

<id name="productId" column="pid" />

<version name="v" column="ver" />

<property name="proName" column="pname" length="10"/>

<property name="price"/>

</class>

</hibernate-mapping>

first we must run the logic to **save** the object then hibernate will inset 0 (Zero) by default

in the version column of the database

First save logic to let the hibernate to insert zero in the version column, then any number

of update logic’s (programs) we run, hibernate will increments +1 to the previous value

• But if we run the update logic for the first time, hibernate will not insert zero..! it will try to increment the previous value which is NULL in the database so we will get

the exception.

Difference B/w JDBC and Hibernate

|  |  |
| --- | --- |
| **JDBC** | **Hibernate** |
| With JDBC, developer has to write code to map an object model's data representation to a relational data model and its corresponding database schema. | Hibernate is flexible and powerful ORM solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this. |
| With JDBC, the automatic mapping of Java objects with database tables and vice versa conversion is to be taken care of by the developer manually with lines of code. | Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS. |
| JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database, i.e. to select effective query from a number of queries to perform same task. | Hibernate provides a powerful query language Hibernate Query Language (independent from type of database) that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native SQL statements. It also selects an effective way to perform a database manipulation task for an application. |
|  |  |
| Application using JDBC to handle persistent data (database tables) having database specific code in large amount. The code written to map table data to application objects and vice versa is actually to map table fields to object properties. As table changed or database changed then it’s essential to change object structure as well | Hibernate provides this mapping itself. The actual mapping between tables and application objects is done in XML files. If there is change in Database or in any table then the only need to change XML file properties. |
| as to change code written to map table-toobject/object-to-table. |  |
| With JDBC, it is developer’s responsibility to handle JDBC result set and convert it to Java objects through code to use this persistent data in application. So with JDBC, mapping between Java objects and database tables is done manually. | Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects. It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost. |
| With JDBC, caching is maintained by handcoding. | Hibernate, with Transparent Persistence, cache is set to application work space. Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many times for same write. Automatic Transparent Persistence allows the developer to concentrate more on business logic rather than this application code. |
| In JDBC there is no check that always every user has updated data. This check has to be added by the developer. | Hibernate enables developer to define version type field to application, due to this defined field Hibernate updates version field of database table every time relational tuple is updated in form of Java class object to that table. So if two users retrieve same tuple and then modify it and one user save this modified tuple to database, version is automatically updated for this tuple by Hibernate. When other user tries to save updated tuple to database then it does not allow saving it because this user does not have updated data. |

Type Of View Resolver In Spring?

1. **AbstractCachingViewResolver :** Abstract view resolver that caches views. Often views need preparation before they can be used; extending this view resolver provides caching.
2. **XmlViewResolver :** Implementation of ViewResolver that accepts a configuration file written in XML with the same DTD as Spring’s XML bean factories. The default configuration file is /WEB-INF/views.xml.
3. **ResourceBundleViewResolver :** Implementation of ViewResolver that uses bean definitions in a ResourceBundle, specified by the bundle base name. Typically you define

the bundle in a properties file, located in the classpath. The default file name is views.properties.

1. **UrlBasedViewResolver :** Simple implementation of the ViewResolver interface that effects the direct resolution of logical view names to URLs, without an explicit mapping definition. This is appropriate if your logical names match the names of your view resources in a straightforward manner, without the need for arbitrary mappings.
2. **InternalResourceViewResolver :** Convenient subclass of UrlBasedViewResolver that supports InternalResourceView (in effect, Servlets and JSPs) and subclasses such as JstlView and TilesView. You can specify the view class for all views generated by this resolver by using setViewClass(..).
3. **VelocityViewResolver/FreeMarkerViewResolver :** Convenient subclass of UrlBasedViewResolver that supports VelocityView (in effect, Velocity templates) or

FreeMarkerView ,respectively, and custom subclasses of them.

1. **ContentNegotiatingViewResolver :** Implementation of the ViewResolver interface that resolves a view based on the request file name or Accept header.

**Servlet Filter-**A **filter** is an object that is invoked at the preprocessing and postprocessing of a request.

It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

|  |  |
| --- | --- |
| **Interfaces** | **Recommended Use** |
| Statement | Use the for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters. |
| PreparedStatement | Use the when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. |
| CallableStatement | Use the when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters. |

What is Transaction in hibernate?

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

What is Query in hibernate?

Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.

What is Criteria in hibernate?

Criteria object are used to create and execute object oriented criteria queries to retrieve objects.

|  |  |
| --- | --- |
| Name some of the properties you would require to configure for a databases in a standalone | |
| situation. |  |

* 1. Hibernate connection. driver class
  2. Hibernate dialects
  3. Hibernate.connection.url
  4. Hibernate.connection.username
  5. Hibernate.connection.password
  6. Hibernate.connection.poolsize
  7. Hibernate.connection.autocommit

What is the purpose of Session.beginTransaction() method?

Session.beginTransaction method begins a unit of work and returns the associated Transaction object.

openSession and getCurrentsession- Open Session

When you call SessionFactory.openSession, it always create new Session object a fresh and give it to you. You need to explicitly flush and close these session objects. 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 will provide you session object

which is in hibernate context and managed by hibernate internally. It is bound to transaction scope.

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.

Which method is used to add a criteria to a query?

Session.createCriteria creates a new Criteria instance, for the given entity class, or a superclass of an entity class.

Which method is used to create a HQL query?

Session.createQuery creates a new instance of Query for the given HQL query string.

Where Object/relational mappings are defined in hibernate?

An Object/relational mappings are usually defined in an XML document. This mapping file instructs Hibernate how to map the defined class or classes to the database tables. We should save the mapping document in a file with the format <classname>.hbm.xml.

Which element of hbm.xml is used to automatically generate the primary key values?

The **<generator>** element within the id element is used to automatically generate the primary key values. Set the **class** attribute of the generator element is set to **native** to let hibernate pick up either **identity, sequence** or **hilo** algorithm to create primary key depending upon the capabilities of the underlying database.

|  |  |
| --- | --- |
| Which element of hbm.xml is used to map a java.util.Collection property in | |
| hibernate? |  |

This is mapped with a <bag> or <ibag> element and initialized with java.util.ArrayList.

What is the difference between save() and persist() methods of session object?

session.save saves the object and returns the id of the instance whereas persist do not return anything after saving the instance.

What is default type of autowired? Autowired By Type Types Of Autowired?

Autowired By Type

Autowired by Name

AutoWired Byconstructor

Autowired By Autodetect

What is the difference between get() and load() methods of session object?

1)get() returns null if no data is present where as load throws ObjectNotFoundException exception in such case.

2)get() always hits the database whereas load() method doesn't hit the database.

3)get() returns actual object whereas load() returns proxy object.

What is lazy loading?

Lazy loading is a technique in which objects are loaded on demand basis. Since Hibernate 3, lazy loading is by default, enabled so that child objects are not loaded when parent is loaded.

LAZY Fetch = Doesn’t load the relationships unless explicitly “asked for” via getter EAGER Fetch = Loads ALL relationships

From the JPA 2.1 spec, the defaults are like so:

* OneToMany:*LAZY*
* ManyToOne: *EAGER*
* ManyToMany: *LAZY*
* OneToOne: *EAGER*
* Columns : *EAGER*

Example-

@OneToMany( mappedBy = "category", fetch=FetchType.LAZY ) private Set<ProductEntity> products;

What is HQL?

HQL stands for Hibernate Query Language. It takes java objects in the same way as SQL takes tables. HQL is a Object Oriented Query language and is database independent

Hibernate uses the following ways to retrieve objects from the database: 1) Hibernate Query Language (**HQL**)

2) Query By Criteria (**QBC**) and Query BY Example (**QBE**) using **Criteria** API 3) Native **SQL** queries Why we Use HQL?

* 1. HQL allows representing SQL queries in object-oriented terms—by using objects and properties of objects.
  2. HQL fully supports **polymorphic queries**. That is, along with the object to be returned as a query result, all child objects (objects of subclasses) of the given object shall be returned
  3. HQL contains many advance features such as pagination, fetch join with dynamic profiling, and so forth, as compared to SQL. 4)

Difference B/w HQL and SQL

The main difference between is **HQL uses class name instead of table name, and property names instead of column name**.

|  |  |  |
| --- | --- | --- |
|  | Caching- | Caching is all about application performance optimization and it sits |
| between your application and the database to avoid the number of database hits as | | |

many as possible to give a better performance for performance critical applications. 1. **The first-level cache - Session** (Earlier hibernate already provide this level of cache)

1. **The second-level cache -Session-factory-level** cache
2. **and the query cache.**

What is first level cache in hibernate?

The first-level cache is the Session cache and is a mandatory cache through which all requests must pass. The Session object keeps an object under its own power before committing it to the database.. The session cache caches object within the current session but this is not enough for long level i.e. session factory scope.

What is second level cache in hibernate?

Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache. The second-level cache can be configured on a per-class and per-collection basis and mainly responsible for caching objects across sessions.

-Across sessions in an Application

-- Across applications (different applications on same servers with same database)

-- Across clusters (different applications on different servers with same database) **Hibernate second level cache** uses *a common cache for all the session object of a session factory*. It is useful if you have multiple session objects from a session factory

**NOTE**: Be careful Caches are never aware of changes made to the persistent store by another application i.e. suppose one application deploy one server with using hibernate and get the data from database and put to the cache for further using purpose but another application deployed another server which does not using any ORM tool so it does mot know about Cache and direct interacting with database and may be update data of database. Now data in Cache is invalid. How to enable second level cache through annotation cache concurrency strategy is *NONSTRICT\_READ\_WRITE*, but depending on cache provider, other strategies can be followed like *TRANSACTIONAL, READ\_ONLY*,

@Cache(usage=CacheConcurrencyStrategy.READ\_ONLY, region="department").

@Cacheable

@Cache(usage = CacheConcurrencyStrategy.NONSTRICT\_READ\_WRITE)

Hibernate does not fire query twice. If you don't use second level cache, hibernate will fire query twice because both query uses different session objects.

What is Query level cache in hibernate?

Hibernate also implements a cache for query resultsets that integrates closely with the second-level cache.

This is an optional feature and requires two additional physical cache regions that hold the cached query results and the timestamps when a table was last updated. This is only useful for queries that are run frequently with the same parameters.

List<Dna> list = session.createQuery(

"from dna").setFirstResult(0).setMaxResults(50).setCacheable(true).list();

There are three possible strategies to use. Ininheritance in Hibernate

 **SINGLE TABLE STRATEGY**,- In Single table per subclass, the union of all the properties from the inheritance hierarchy is mapped to one table. As all the data goes in

one table, a discriminator is used to differentiate between different type of data**.** Advantages of Single Table per class hierarchy

* Simplest to implement.
* Only one table to deal with.
* Performance wise better than all strategies because no joins or sub-selects need to be performed.

**Disadvantages:**

* Most of the column of table are nullable so the NOT NULL constraint cannot be applied.
* Tables are not normalized

ANNOTATION-**@Inheritance(strategy=InheritanceType.SINGLE\_TABLE)**

 **WITH TABLE PER CLASS STRATEGY-**

In this case every entity class has its own table i.e. table per class. The data for Vehicle is duplicated in both the tables.

This strategy is not popular and also have been made optional in Java Persistence API.

Advantage:

Possible to define NOT NULL constraints on the table. Disadvantage:

* Tables are not normalized.
* To support polymorphism either container has to do multiple trips to database or use SQL UNION kind of feature.

ANNOTATION-**@Inheritance(strategy=InheritanceType.TABLE\_PER\_CLASS) //slightly more normalized**

 **WITH JOINED STRATEGY –**

It's highly normalized but performance is not good. Advantage:

* Tables are normalized.
* Able to define NOT NULL constraint. Disadvantage:

* Does not perform as well as SINGLE\_TABLE strategy
* Annotation-@Inheritance(strategy=InheritanceType.JOINED)//Highly normalized

Java's transient keyword is used to denote that a field is not to be serialized, whereas JPA's @Transient annotation is used to indicate that a field is not to be persisted in the database, i.e. their semantics are different.

Transient Persistence and Detached State

Every entity object are passed to three states of the object before saving and updating the row in the database table as per as given in the following picture.



**1. Transient State:** A New instance of a persistent class which is not associated with a ***Session***, has no representation in the ***database*** and no identifier value is considered ***transient*** by Hibernate:

Example-:

|  |  |  |
| --- | --- | --- |
| 1. | | UserDetail user = **new** UserDetail(); |
|  | 2. | user.setUserName("Dinesh Rajput"); |
|  | 3. | // user is in a transient state |

2 **Persistent State:** A persistent instance has a representation in the ***database*** , an identifier value and is associated with a ***Session***. You can make a transient instance persistent by associating it with a ***Session***:

**Example**

1. Long id = (Long) session.save(user);
2. // user is now in a persistent state
3. **Detached State:** Now, if we close the ***Hibernate Session***, the ***persistent*** instance will become a ***detached*** instance: it isn't attached to a ***Session*** anymore (but can still be modified and reattached to a new Session later though).

**Example**

1. session.close();
2. //user in detached state

**Difference between Transient and Detached States:**

Transient objects do not have association with the databases and session objects. They are simple objects and not persisted to the database. Once the last reference is lost, that means the object itself is lost. And of course , garbage collected. The commits and rollbacks will have no effects on these objects. They can become into persistent objects through the save method calls of Session object.

The detached object have corresponding entries in the database. These are persistent and not connected to the Session object. These objects have the synchronized data with the database when the session was closed. Since then, the change may be done in the database which makes this object stale. The detached object can be reattached after certain time to another object in order to become persistent again.

Example

1. **package** com.sdnext.hibernate.tutorial;

2.

1. **import** org.hibernate.Session;
2. **import** org.hibernate.SessionFactory;
3. **import** org.hibernate.cfg.AnnotationConfiguration;

6.

7. **import** com.sdnext.hibernate.tutorial.dto.UserDetails;

8.

1. **public** **class** HibernateTestDemo {
2. /\*\*
3. \* @param args
4. \*/
5. **public** **static** **void** main(String[] args)
6. {
7. UserDetails userDetails = **new** UserDetails();
8. userDetails.setUserName("Dinesh Rajput");
9. userDetails.setAddress("Noida City");
10. //Here 'userDetails' is TRANSIENT object

19.

1. SessionFactory sessionFactory = **new**

AnnotationConfiguration().configure().buildSessionFactory();

1. Session session = sessionFactory.openSession();
2. session.beginTransaction();

23.

1. session.save(userDetails);
2. //Here 'userDetails' is PERSISTENT object
3. userDetails.setUserName("User Updated after session close");

27.

1. session.getTransaction().commit();
2. session.close();
3. //Here 'userDetails' is DETACHED object
4. }

32. }

Merge and Update-Both are used to convert object from detached state to persistence state

For eg

------ 1 -----

1. SessionFactory factory = cfg.buildSessionFactory();
2. Session session1 = factory.openSession();

4

5

1. Student s1 = null;
2. Object o = session1.get(Student.class, new Integer(101));
3. s1 = (Student)o;
4. session1.close();

10

11 s1.setMarks(97);

12

13 Session session2 = factory.openSession();

14

1. Student s2 = null;
2. Object o1 = session2.get(Student.class, new Integer(101));
3. s2 = (Student)o1;
4. Transaction tx=session2.beginTransaction();

19

Session2.merge(s1);

### Explanation

* See from line numbers 6 – 9, we just loaded one object s1 into session1 cache and closed session1 at line number 9, so now object s1 in the session1 cache will be destroyed as session1 cache will expires when ever we say session1.close()
* Now s1 object will be in some RAM location, not in the session1 cache
* here s1 is in detached state, and at line number 11 we modified that detached object s1, now if we call update() method then hibernate will throws an error, because we can update the object in the session only
* So we opened another session [session2] at line number 13, and again loaded the same student object from the database, but with name s2
* so in this session2, we called **session2.merge(s1)**; now into s2 object s1 changes will be merged and saved into the database

Life Cycle of Jsp

* + Translation of JSP Page o Compilation of JSP Page o Classloading (class file is loaded by the classloader) o Instantiation (Object of the Generated Servlet is created). o Initialization ( jspInit() method is invoked by the container).
  + Reqeust processing ( \_jspService() method is invoked by the container). o Destroy ( jspDestroy() method is invoked by the container).

@AutoWired-:

You can use **@Autowired** annotation on properties to get rid of the setter methods. When you will pass values of autowired properties using <property> Spring will automatically assign those properties with the passed values or references.

@Autowired is spring-specific. @Inject is the standard equivallent. It is an annotation that tells the context (spring, or in the case of @Inject - any DI framework) to try to set an object into that field.

The compiler has nothing to do with this - it is the DI framework (spring) that instantiates your objects *at runtime*, and then sets their dependencies at the points you have specified - either via XML or via an annotation.

The @Autowired annotation is performing Dependency Injection.

If @Autowired is applied to

* a field: then the dependency is stored in this field • a setter: then the setter is invoked, with the parameter that is determined by the same algorithm like for the field dependency injection
* a constructor: then the constructor is invoked with the parameters determined by the same algorithm like for the field dependency injection

**Why we use jquery over java script?**

**JQuery** is a fast and concise **JavaScript** Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development Where we use abstract class and interface?

We can use abstract class where if some common functionality is needed then we can write inside code inside nonabstarct method inside abstract class.

Can we call constructor in abstract class

Yes we can create the object abstract class.So it will communicate with object. The class which is extend the object class get the functionality

DependencyInjection- We can create the object using new keyword But in spring due to depency injection it will create the object inside the container.

IOC Container-:

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

o to instantiate the application class o to configure the object o to assemble the dependencies between the objects

There are two types of IoC containers. They are:

1. **BeanFactory**
2. **ApplicationContext**

Dependency Injection

The Dependency Injection is a design pattern that removes the dependency of the

programs. In such case we provide the information from the external source such as XML file. It makes our code loosely coupled and easier for testing.

Cascade Type InJPA

1. **CascadeType.PERSIST** : means that save() or persist() operations cascade to related entities.
2. **CascadeType.MERGE** : means that related entities are merged into managed state when the owning entity is merged.
3. **CascadeType.REFRESH** : does the same thing for the refresh() operation.
4. **CascadeType.REMOVE** : removes all related entities association with this setting when the owning entity is deleted.
5. **CascadeType.DETACH** : detaches all related entities if a “manual detach” occurs.
6. **CascadeType.ALL** : is shorthand for all of the above cascade operations.

Query, Native query,Named query, TypeQuery

Query-Query refers to JPQL/HQL query with syntax similar to SQL generally used to execute DML statements(CRUD operations).

Native Query-Native query refers to actual sql queries (referring to actual database objects). These queries are the sql statements which can be directly executed in database using a database client.

Named Query- NamedQuery is the way you define your query by giving it a name. You could define this in mapping file in hibernate or also using annotations at entity level

TypeQuery- TypedQuery gives you an option to mention the type of entity when you create a query and therefore any operation thereafter does not need an explicit cast to the intended type.

Dispatcher servlet------------→HandlerMapping

Controller

ViewResolver

Dispatcher servlet send request to handlermapping. And handler mapping find the url to dispatcher servlet which controller they have to call. Then dispatcher servlet call the controller .after controller send to dispatcher servlet .And again dispatcher servlet send to view resolver—Andd then view display the page.

@Configuration

@PropertySource("classpath:application.properties")

@PropertySource("classpath:messages.properties")

@ComponentScan("com.infotech.orion")

**public** **class** RootConfiguration {

@Bean

**public** HibernateJpaSessionFactoryBean sessionFactory() {

**return** **new** HibernateJpaSessionFactoryBean();

}

@Bean

@ConfigurationProperties(prefix = "spring.datasource") **public** DataSource getDataSource() {

**return** DataSourceBuilder.*create*().build();

}

}

Application properties

spring.datasource.url=jdbc:mysql://localhost:3306/orion\_dist

spring.datasource.username=root spring.datasource.password=root spring.datasource.testWhileIdle=true

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

########################## Hibernate Configuration ######################## spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5

Difference b/w POST,PUT,delete,Get

Create = PUT with a new URI

POST to a base URI returning a newly created URI

Read = GET

Update = PUT with an existing URI

Delete = DELETE

POST Creates a new resource.

GET Retrieves a resource. PUT Updates an existing resource. DELETE Deletes a resource.

Name idempotent methods- POST is not idempotent and all other are idempotentz

Ans Get() and put()

Difference between put and post?

Ans PUT puts a file or resource at a specific URI, and exactly at that URI. If there's already a file or resource at that URI, PUT replaces that file or resource. If there is no file or resource there, PUT creates one. PUT is [idempotent,](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.1.2) but paradoxically PUT responses are not cacheable.

POST sends data to a specific URI and expects the resource at that URI to handle the request. The web server at this point can determine what to do with the data in the context of the specified resource. The POST method is not [idempotent,](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.1.2) however POST responses *are* cacheable so long as the server sets the appropriate Cache-Control and Expires headers.

ACID Properties

* [Atomicity](http://www.service-architecture.com/articles/database/atomicity.html)
* a transaction is a unit of operation - either all the transaction's actions are completed or none are
* atomicity is maintained in the presence of database software failures
* atomicity is maintained in the presence of application software failures
* atomicity is maintained in the presence of CPU failures
* atomicity is maintained in the presence of disk failures
* atomicity can be turned off at the system level

* [Consistency](http://www.service-architecture.com/articles/database/consistency.html)

Consistency is the ACID property that ensures that any changes to values in an instance are consistent with changes to other values in the same instance. A consistency constraint is a predicate on data which serves as a precondition, post-condition, and transformation condition on any transaction.

* [Isolation](http://www.service-architecture.com/articles/database/isolation.html)

The isolation portion of the ACID Properties is needed when there are concurrent transactions. Concurrent transactions are transactions that occur at the same time, such as shared multiple users accessing shared objects.

The safeguards used by a DBMS to prevent conflicts between concurrent transactions are a concept referred to as isolation

An important concept to understanding isolation through transactions is serializability.

Transactions are serializable when the effect on the database is the same whether the transactions are executed in serial order or in an interleaved fashion. As you can see at the top of the figure, Transactions 1 through Transaction 3 are executing concurrently over time. The effect on the DBMS is that the transactions may execute in serial order based on consistency and isolation requirements. If you look at the bottom of the figure, you can see several ways in which these transactions may execute. It is important to note that a serialized execution does not imply the first transactions will automatically be the ones that will terminate before other transactions in the serial order.

Degrees of isolation1:

* degree 0 - a transaction does not overwrite data updated by another user or process ("dirty data") of other transactions
* degree 1 - degree 0 plus a transaction does not commit any writes until it completes all its writes (until the end of transaction)
* degree 2 - degree 1 plus a transaction does not read dirty data from other transactions
* degree 3 - degree 2 plus other transactions do not dirty data read by a transaction before the transaction commits

* [Durability](http://www.service-architecture.com/articles/database/durability.html)

Durability refers to the ability of the system to recover committed transaction updates if either the system or the storage media fails. Features to consider for durability:

* recovery to the most recent successful commit after a database software failure
* recovery to the most recent successful commit after an application software failure
* recovery to the most recent successful commit after a CPU failure
* recovery to the most recent successful backup after a disk failure
* recovery to the most recent successful commit after a data disk failure

Spring Framework

In spring framework bean declared in ApplicationContext.xml can reside in five scopes:

* 1. Singleton (default scope)
  2. prototype
  3. request
  4. session
  5. global-session

Spring Application Context- Singleton And ProtoType

Web Aware Application Context- request, session,global -session

Singleton bean Scope- only one instance created per Spring Container. SO no matter how many times you called getBean method.

ProtoType bean Scope-In case of prototype bean scope, every getBean() call creates a new instance of

Spring bean.

Annotation For Bean Scope-@Scope("prototype")

JSP

Scriplet and Declarative

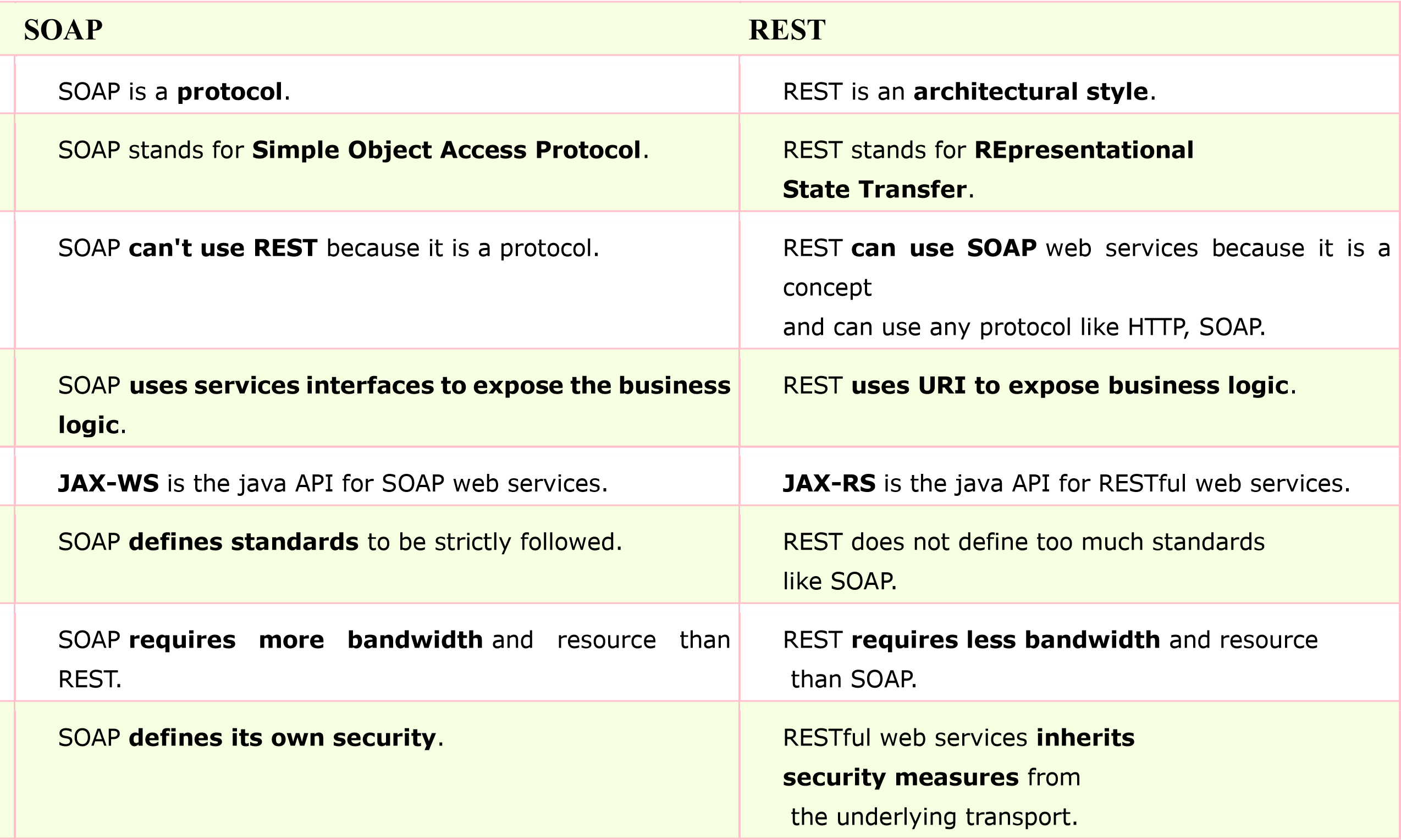
Jsp Scriplet tag- The jsp scriptlet tag can only declare variables not methods.

2)The declaration of scriptlet tag is placed inside the \_jspService() method.

Decllarative Tag-The jsp declaration tag can declare variables as well as methods.

2)The declaration of jsp declaration tag is placed outside the \_jspService() method.

JSP Implicit Object-:

There are 9 jsp implicit objects-:

1)out

2)request

3)response

4)Config

5)Application

6)session

1. PageContext
2. Page 9) Exception

-Differnec B/w SOAP and Rest WebService

Ans Soap-1)An XML based protocol

2)Uses WSDl for communication between Consumer and producer

3)Invokes service by calling RPC method

4)Does not result human readable result

5)transfer is over http. And also uses http protocol such as FTP,SMTP

6)Javascript can call SOAP but it is difficult to implement

7)Performance is not great comapred to REST

8)Business logic is exposed by interface

REST-An architectural style protocol

2)Use XML and JSON to send or receive data

3)simple call services via URL path

4)Result is readable which is just plain XML or JSON

5)Transfer is over Http only

SOAP

**permits XML**

data format only.

REST

**permits different**

data format such

as Plain text,

HTML,

XML, JSON etc.

)

SOAP is

**less preferred**

than REST.

REST

**more preferred**

than SOAP.

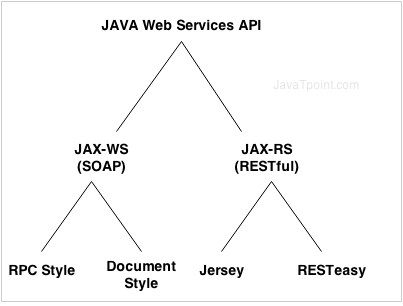
6)Easy to call from Javascript

7)Performance is much better compared to SOAP less cpu intensive and leaner codeetc

8)Url is used

There are two main API's defined by Java for developing web service applications since JavaEE 6.

1. **JAX-WS**: for SOAP web services. The are two ways to write JAX-WS application code: by RPC style and Document style.
2. **JAX-RS**: for RESTful web services. There are mainly 2 implementation currently in use for creating JAX-RS application: Jersey and RESTeasy.



@ManyToMany(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)

@JoinTable(name = "app\_user\_matrix", catalog = "orion\_dist", joinColumns = {@JoinColumn(name = "user\_id", nullable = **false**, updatable = **false**) },

inverseJoinColumns = { @JoinColumn(name = "app\_code", nullable = **false**, updatable = **false**) })

**private** Set<AppMaster> appMasterRef = **new** HashSet<AppMaster>(0);

@ManyToMany(fetch = FetchType.***LAZY***, mappedBy = "appMasterRef") **private** Set<UserMaster> userMasterRef = **new** HashSet<UserMaster>(0);

XML- It is an extensible markup language

JSON-It's a human readable object serialisation format, less verbose than XML.

JAXB- the "Java Architecture for XML Binding" in the Java ecosystem is the primary mechanism for turning XML data into objects which you can then interact with, and vice versa. It's important to realise that it's an API and not an implementation, so it mostly defines a set of annotations and simple classes / interfaces in the javax.xml.bind package. To do anything useful with JAXB you need an implementation. There's a reference implementation included in the Glassfish application server. Most application servers will have some kind of implementation of JAXB.

JACKSON- a library for data binding. It supports both XML and JSON as document formats, and implements the JAXB API. You can use Jackson as your implementation of JAXB, or you can just use the Jackson API directly.

Jersey-media-moxy-Jersey is an implementation of JAX-RS. One aspect of JAX-RS is passing documents around - often XML or JSON. To do that Jersey needs to know what underlying library to use for data-binding or stream processing

What is a Web Service Endpoint?

In simple terms, a **web service endpoint** is a web address (URL) at which clients of a specific service can gain access to it. By referencing that URL, clients can get to operations provided by that service.

The following words will help you get an idea of what an endpoint is.

* Port: a unique endpoint with its own address
* Protocol: a specific way to interact
* Message: a piece of abstract lingo that aids communication
* Port Type: what operations this port can perform
* Operation: an action that the service can do for you
* Service: a group of endpoints that share something in common
* Complexity Of

What to if our one microservice will fail and not give response to other microservice.How to do backtracking

Ans -We can do back tracking by saving the logs in cache and wait tillthe response of second microservice. Or We can use save in database or stored in kibana logging tool.In this way we do backtracking

**DataBase**

Second maximum Salary-

SELECT sal

FROM emp

ORDER BY sal DESC

LIMIT 1, 1;

If 4th salary 3,1 .You have use n-1

To delete a coloumn in a table

ALTER TABLE table\_name

DROP COLUMN column\_name

Eg-ALTER TABLE Persons

DROP COLUMN DateOfBirth

To add a coloumn in a table

ALTER TABLE table\_name

ADD column\_name datatype

Eg-:ALTER TABLE Persons

ADD DateOfBirth date

To change the data type of the coloumn

ALTER TABLE table\_name

ALTER COLUMN column\_name datatype

Eg-ALTER TABLE Persons

ALTER COLUMN DateOfBirth year

To craete table

CREATE TABLE Persons

(

ID int IDENTITY(1,1) PRIMARY KEY,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

INSERT INTO Persons (FirstName,LastName)

VALUES ('Lars','Monsen')

Update Query

UPDATE Customers

SET ContactName='Alfred Schmidt', City='Hamburg'

WHERE CustomerName='Alfreds Futterkiste';

**Cursors in SQL**. A **cursor** is a temporary work area created in system memory when a **SQL** statement is executed. A **cursor** is a set of rows together with a **pointer** that identifies a current row. It is a database object to retrieve data from a result set one row at a time

**Difference between Procedure and Trigger?**

1. We can execute a stored procedure whenever we want with the help of the exec command, but a trigger can only be executed whenever an event (insert, delete, and update) is fired on the table on which the trigger is defined.

1. We can call a stored procedure from inside another stored procedure but we can't directly call another trigger within a trigger. We can only achieve nesting of triggers in which the action (insert, delete, and update) defined within a trigger can initiate execution of another trigger defined on the same table or a different table.
2. Stored procedures can be scheduled through a job to execute on a predefined time, but we can't schedule a trigger
3. Stored procedure can take input parameters, but we can't pass parameters as input to a trigger.
4. Stored procedures can return values but a trigger cannot return a value.
5. We can use Print commands inside a stored procedure for debugging purposes but we can't use print commands inside a trigger.
6. We can use transaction statements like begin transaction, commit transaction, and rollback inside a stored procedure but we can't use transaction statements inside a trigger.
7. We can call a stored procedure from the front end (.asp files, .aspx files, .ascx files, etc.) but we can't call a trigger from these files
8. Stored procedures are used for performing tasks. Stored procedures are normally used for performing user specified tasks. They can have parameters and return multiple results sets.
9. The Triggers for auditing work: Triggers normally are used for auditing work. They can be used to trace the activities of table events

**How to find the duplicate element attribute name in the database and the count how many times duplicate name is coming**

**Ans**

|  |
| --- |
| **SELECT username, email, COUNT(\*)** |
| **FROM users**  **GROUP BY username, email**  **HAVING COUNT(\*) > 1** |

#### **MAVEN AND GRADLE**

**Difference between Maven and Gradle?**

**Gradle**

It is a build automation system that uses a Groovy-based DSL (domain-specific language).

It does not use an XML file for declaring the project configuration.

It is based on a graph of task dependencies that do the work.

In Gradle, the main goal is to add functionality to the project.

It avoids the work by tracking input and output tasks and only runs the tasks that have been changed. Therefore it gives a faster performance.

Gradle is highly customizable; it provides a wide range of IDE support custom builds Gradle avoids the compilation of Java.

**Maven-**

It is a software project management system that is primarily used for java projects..

It uses an XML file for declaring the project, its dependencies, the build order, and its required plugin..

It is based on the phases of the fixed and linear model.

In maven, the main goal is related to the project phase..

It does not use the build cache; thus, its build time is slower than Gradle.

Maven has a limited number of parameters and requirements, so customization is a bit complicated.

The compilation is mandatory in Maven.

**How to improve performance of application?**

1. Compress Data to **Improve** Web **Application Performance**. ...
2. Monitor Activities and Resolve Bottlenecks in Real Time. ...
3. Distribute Traffic with a Load Balancer. ...
4. **Optimize** Security Protocols. ...
5. Deliver Content Faster by Caching. ...
6. Keep Your Software Versions Updated. ...

Sort the list in descending order

List<Integer> list = Arrays.asList(10, 4, 2, 6, 5, 8); list = list.stream()

.sorted(Comparator.reverseOrder()) .collect(Collectors.toList());

**Output:**

[10, 8, 6, 5, 4, 2]

**Query 3.1 : How many male and female employees are there in the organization?**

Map<String, Long> noOfMaleAndFemaleEmployees= employeeList.stream().collect(Collectors.groupingBy(Employee::getGender,

Collectors.counting()));

System.out.println(noOfMaleAndFemaleEmployees); **Output:**{Male=11, Female=6}

**Print the name of all departments in the organization?**

employeeList.stream()

.map(Employee::getDepartment)

.distinct()

.forEach(System.out::println);

**Query 3.3 : What is the average age of male and female employees?**

Map<String, Double> avgAgeOfMaleAndFemaleEmployees=

employeeList.stream().collect(Collectors.groupingBy(Employee::getGender,

Collectors.averagingInt(Employee::getAge)));

System.out.println(avgAgeOfMaleAndFemaleEmployees);

**Query 3.4 : Get the details of highest paid employee in the organization?**

Optional<Employee> highestPaidEmployeeWrapper=

employeeList.stream().collect(Collectors.maxBy(Comparator.comparingDouble(Emp loyee::getSalary)));

**Query 3.5 : Get the names of all employees who have joined after 2015?**

employeeList.stream()

.filter(e -> e.getYearOfJoining() > 2015)

.map(Employee::getName) .forEach(System.out::println); **Count the number of employees in each department?**

Map<String, Long> employeeCountByDepartment=

employeeList.stream().collect(Collectors.groupingBy(Employee::getDepartment,

Collectors.counting()));

**What is the average salary of each department?**

Map<String, Double> avgSalaryOfDepartments=

employeeList.stream().collect(Collectors.groupingBy(Employee::getDepartment,

Collectors.averagingDouble(Employee::getSalary)));

**Query 3.8 : Get the details of youngest male employee in the product development department?**

Optional<Employee> youngestMaleEmployeeInProductDevelopmentWrapper= employeeList.stream()

.filter(e -> e.getGender()=="Male" && e.getDepartment()=="Product Development")

.min(Comparator.comparingInt(Employee::getAge));

**Who has the most working experience in the organization?**

Optional<Employee> seniorMostEmployeeWrapper=

employeeList.stream().sorted(Comparator.comparingInt(Employee::getYearOfJoini ng)).findFirst();

**How many male and female employees are there in the sales and marketing team?**

Map<String, Long> countMaleFemaleEmployeesInSalesMarketing= employeeList.stream()

.filter(e -> e.getDepartment()=="Sales And Marketing")

.collect(Collectors.groupingBy(Employee::getGender, Collectors.counting()));

**Query 3.11 : What is the average salary of male and female employees?**

Map<String, Double> avgSalaryOfMaleAndFemaleEmployees= employeeList.stream().collect(Collectors.groupingBy(Employee::getGender,

Collectors.averagingDouble(Employee::getSalary)));

System.out.println(avgSalaryOfMaleAndFemaleEmployees);

**Query 3.12 : List down the names of all employees in each department?**

Map<String, List<Employee>> employeeListByDepartment= employeeList.stream().collect(Collectors.groupingBy(Employee::ge tDepartment));

**What is the average salary and total salary of the whole organization?**

DoubleSummaryStatistics employeeSalaryStatistics=

employeeList.stream().collect(Collectors.summarizingDouble(Employee::getSalar y));

**Query 3.14 : Separate the employees who are younger or equal to 25 years from those employees who are older than 25 years.**

Map<Boolean, List<Employee>> partitionEmployeesByAge= employeeList.stream().collect(Collectors.partitioningBy(e -> e.getAge() > 25));

**Query 3.15 : Who is the oldest employee in the organization? What is his age and which department he belongs to?**

Optional<Employee> oldestEmployeeWrapper =

employeeList.stream().max(Comparator.comparingInt(Employee::getA ge));

**LIFE CYCLE OF MAVEN**-:

It has 23 phases-:

1)Validate**-** Validates whether project is correct and all necessary information is available to complete the build process.

2)Initialize-Initializes build state, for example set properties

3)generate-sources-Generate any source code to be included in compilation phase.

1. process-sources-Process the source code, for example, filter any value.
2. generate-resources-Generate resources to be included in the package.
3. process-resources-Copy and process the resources into the destination directory, ready for packaging phase.
4. compile-Compile the source code of the project.
5. process-classes- : Post-process the generated files from compilation, for example to do bytecode enhancement/optimization on Java classes.
6. generate-test-sources-: Generate any test source code to be included in compilation phase.
7. process-test-sources-: Process the test source code, for example, filter any values.
8. test-compile-:Compile the test source code into the test destination directory.
9. process-test-classes-Process the generated files from test code file compilation.
10. test-Run tests using a suitable unit testing framework(Junit is one).
11. prepare-package-Perform any operations necessary to prepare a package before the actual packaging.
12. package-Take the compiled code and package it in its distributable format, such as a JAR, WAR, or EAR file.
13. pre-integration-test-Perform actions required before integration tests are executed. For example, setting up the required environment.
14. integration-test-Process and deploy the package if necessary into an environment where integration tests can be run.
15. post-integration-test-Perform actions required after integration tests have been executed. For example, cleaning up the environment.
16. verify-Run any check-ups to verify the package is valid and meets quality criterias.
17. install-used as a dependency in other projects locally.
18. deploy-Copies the final package to the remote repository for sharing with other developers and projects.

**What is Rest maturity model?**

Richardson used three factors to decide the maturity of a service i.e. [URI,](https://restfulapi.net/resource-naming/) [HTTP Methods](https://restfulapi.net/http-methods/) and [HATEOAS](https://restfulapi.net/hateoas/) (Hypermedia). The more a service employs these technologies – more mature it shall be considered.

**Level of maturity according Richardson model**

**And Hypermedia,Http,URI**

Richardson described these maturity levels as below:

* + [Level Zero-](https://restfulapi.net/richardson-maturity-model/#level-zero):These services have a single URI and use a single HTTP method (typically POST).
  + [Level One-](https://restfulapi.net/richardson-maturity-model/#level-one):Level one of maturity **makes use of URIs** These services employ many URIs but only a single HTTP verb – generally HTTP POST. They give each individual resource in their universe a URI. Every resource is separately identified by a unique URI – and that makes them better than level zero.
  + [Level Two-](https://restfulapi.net/richardson-maturity-model/#level-two): Level two of maturity **makes use of URIs and HTTP.**Level two services host numerous URI-addressable resources. Such services support several of the HTTP verbs on each exposed resource – Create, Read, Update and Delete (CRUD) services. Here the state of resources, typically representing business entities, can be manipulated over the network.

[Level Three-](https://restfulapi.net/richardson-maturity-model/#level-three):Level three of maturity **makes use of all three i.e. URIs and HTTP and HATEOAS**.This is the most mature level of Richardson’s model which encourages easy discoverability and makes it easy for the responses to be self-explanatory by using HATEOAS.

What is HATEOAS. Why it is used?

Ans HATEOAS stands for **Hypertext As The Engine Of Application State**. It means that hypertext should be used to find your way through the API

The term “***hypermedia***” refers to any content that contains links to other forms of media such as images, movies, and text.

The following are the two popular formats for specifying JSON REST API hypermedia links:

#### *RFC 5988 (web linking)*

[RFC 5988](https://tools.ietf.org/html/rfc5988) puts forward a framework for building links that defines the relation between resources on the web

#### *JSON Hypermedia API Language (HAL)*

[JSON HAL](https://en.wikipedia.org/wiki/Hypertext_Application_Language) is a promising proposal that sets the conventions for expressing hypermedia controls, such as links, with JSON or XML. It is in the draft stage at this time.

**Ahad:**

1->What is servlet;

2->comparator vs comparable.

3->runnable.

4->callable

5->default properties of interface: public and final.

5.1->Why interface methods are by default public

6->concurrent hash map vs map

HashMap<Integer, String> hashMap = new HashMap<>();  
hashMap.put(2, "Ahad");  
hashMap.put(3, "Samad");  
hashMap.put(4, "Siddiqui");  
for (Map.Entry<Integer, String> v : hashMap.entrySet()) {  
 hashMap.put(100, "Fa");  
}  
System.*out*.println(hashMap);

7->synchronization

8->java 8 feature.

* Lambda
* Stream api
* Static method in interface
* Default method in interface.
* Predicate
* forEach
* functionalInterface
* Optional

9->string pool,string buffer,string builder

10->Optional Class.

import java.util.Optional;

public class Java8Tester {

public static void main(String args[]) {

Java8Tester java8Tester = new Java8Tester();

Integer value1 = null;

Integer value2 = new Integer(10);

//Optional.ofNullable - allows passed parameter to be null.

Optional<Integer> a = Optional.ofNullable(value1);

//Optional.of - throws NullPointerException if passed parameter is null

Optional<Integer> b = Optional.of(value2);

System.out.println(java8Tester.sum(a,b));

}

public Integer sum(Optional<Integer> a, Optional<Integer> b) {

//Optional.isPresent - checks the value is present or not

System.out.println("First parameter is present: " + a.isPresent());

System.out.println("Second parameter is present: " + b.isPresent());

//Optional.orElse - returns the value if present otherwise returns

//the default value passed.

Integer value1 = a.orElse(new Integer(0));

//Optional.get - gets the value, value should be present

Integer value2 = b.get();

return value1 + value2;

}

}

11->mock vs injectMock.

12->What is garbage collector in java.

13->**Memory leak and there cause in java,detection and prevention.**

## **Preventing Memory Leak**

While writing code, remember the following points that prevent the memory leak in Java.

* Do not create unnecessary objects.
* Avoid String Concatenation.
* Use String Builder.
* Do not store a massive amount of data in the session.
* Time out the session when no longer used.
* Do not use the System.gc() method.
* Avoid the use of static objects. Because they live for the entire life of the application, by default. So, it is better to set the reference to null, explicitly.
* Always close the ResultSet, Statements, and Connection objects in the finally block.

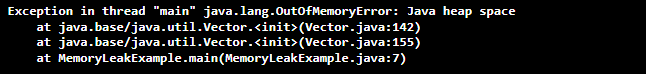
## **Creating Memory Leak**

Let's create a simple Java program that creates a memory leak.

**MemoryLeakExample.java**

1. **import** java.util.Vector;
2. **public** **class** MemoryLeakExample
3. {
4. **public** **static** **void** main(String[] args)
5. {
6. Vector v1 = **new** Vector(314567);
7. Vector v2 = **new** Vector(876543987);
8. System.out.println("There is no memory leak in this program.");
9. }
10. }

**Output**



In the above program, we have created two Vector objects and passed large numbers to them. When we run the above program, it shows **java.lang.OutOfMemoryError.** Because it does not occupy space in the memory. If the program prints the statement **There is no memory leak in this program,** it ensures that the program runs successfully.

### Detecting Memory Leak

Detecting memory leaks is a difficult task. To simplify the task, there are many tools available that perform static analysis and detect memory leaks:

* JProbe
* AppPerfect
* Visual VM
* Jprofiler
* YourKit
* GCeasy
* JRockit

## **Fixing Memory Leak**

There are the following solutions to the memory leak problem:

* **Using JVM Tools:** There are many tools available that optimizes the code and show the memory status.
* **Using Heap Dump:** It is a technique that is the solution to the memory leak problem. It is a snapshot of all objects that reside in the memory at a certain time. It also optimizes memory usage in a Java application. It is stored in binary format in **hprof**
* **Using Eclipse Memory Leak Warnings:** If you are using the Eclipse framework to develop a Java application, eclipse regularly shows the waring and errors whenever it encounters any causes of memory leak.

14->What are the different ways threads can communicate?

Wait()

notifyAll()

notify()