ExecutorService executorService=ExBrookside School for Science
Fall Semester 20XX

Prepare: Md. Motaher Hossain
Email: md.motaherhossain04@gmail.com

Interview Prep
Java

—

Notes

Java Stream

Filter : Stream filter returns a stream consisting of the elements of this stream that match the given Logic

```
List<Integer> a=new ArrayList<>(Arrays.asList(1,2,3,4));
List<Integer> b=a.stream().filter(c-> c==2).collect(Collectors.toList());
System.out.println(a); //1,2,3,4
System.out.println(b); //2
```

Session

how a session is created?

Here you need to know that your HTTP request has no idea what is a session and the server has no idea what is a session. It is purely built by the web developer to store information of the client. A session in a server is implemented by multi threading or sub processes.

A session is a global variable stored on the server. Each session is assigned a unique id which is used to retrieve stored values. Whenever a session is created, a cookie containing the unique session id is stored on the user's computer and returned with every request to the server. If the client browser does not support cookies, the unique session id is displayed in the URL. Sessions have the capacity to store relatively large data compared to cookies. The session values are automatically deleted when the browser is closed. If you want to store the values permanently, then you should store them in the database.

How does a server identify which user belongs to which session?

To answer this I need to explain what is a cookie?

Its a file with a small amount of data stored in the client computer in a key value pair format which is read by both client and the server. The data contains information of user.

What is Cookie?

A cookie is a small file with the maximum size of 4KB that the web server stores on the client computer. Once a cookie has been set, all page requests that follow return the cookie name and value. A cookie can only be read from the domain that it has been issued from. For example, a cookie set using the domain www.guru99.com cannot be read from the domain career.guru99.com. Most of the websites on the internet display elements from other domains such as advertising. The domains serving these elements can also set their own cookies. These are known as **Third party cookies**.Third-party cookies are cookies that are set by a website other than the one you are currently on. For example, you can have a "Like" button on your website which will store a cookie on visitor's computer, that cookie can later be accessed by Facebook to identify visitor and see which websites he visited. A cookie created by a user can only be visible to them. Other users cannot see its value. Most web browsers have options for disabling cookies, third party cookies or both.

Who creates cookie?

Cookie is created by the server. It adds data and packages it in a cookie and is sent to the browser. Now every time a request occurs from the website and hits the server, the server requests the cookie from the browser. The browser sends the cookie through HTTP request. Since the cookie has the user information the server reads it and loads the web page based on the data from the cookie.

Similarly the session has a unique identifier called as session token Id/session ID. The server creates a variable representing some session information and stores this in the cookie. The rest of the process is just like explained above. This way the server identifies which user is accessing the website.

How "remember me" works?

The remember-me feature typically works by generating a unique cookie, associating it with the user in the database, and adding a **persistent cookie** (i.e. a cookie which is saved on disk by the browser) to the response once the user is logged in.

When the user opens the browser again and goes back to the app, the browser sends this cookie, and the server finds if any user has this cookie in the database. If the user is found, he's automatically authenticaed and a new session is started for this cookie.

Session Hijacking

//
https://www.netsparker.com/blog/web-security/session-hijacking/#:~:text=Session%20hijacking%20is%20an%20attack,ends%20when%20you%20log%20out.

To perform session hijacking, an attacker needs to know the victim's session ID (session key). This can be obtained by stealing the session cookie or persuading the user to click a malicious link containing a prepared session ID. In both cases, after the user is authenticated on the server, the attacker can take over (hijack) the session by using the same session ID for their own browser session. The server is then fooled into treating the attacker's connection as the original user's valid session.

Methods: Cross Site Scripting(XSS), Session side jacking, etc. follow the link for more.

Prevent

- Use HTTPS
- Set the HttpOnly attribute using the Set-Cookie
- Web frameworks offer highly secure and well-tested session ID generation and management mechanisms.
- Regenerate the session key after initial authentication. This causes the session key to change immediately after authentication, which nullifies session fixation attacks
-

RunTime vs CompileTime

Compile-time is the instance where the code you entered is converted to executable while Run-time is the instance where the executable is running.

Enum

Enum is a special class that represents a group of constants.

```
public static void main(String[] args) {
      Mobile[] phones=Mobile.values();
      for(Mobile phone:phones)
         System.out.println(phone);

      Mobile p=Mobile.samsung;
      System.out.println(p.showPrice());

      p=Mobile.valueOf("samsung");
      System.out.println(p);
   }

enum Mobile {
   samsung(44),oppo(46),vivo(49);
   int price;
   Mobile(int p){this.price=p;}
   int showPrice(){return price;}
}
```

Links: https://www.geeksforgeeks.org/enum-in-java/

https://www.geeksforgeeks.org/enum-customized-value-java/

https://www.tutorialspoint.com/java/lang/enum_valueof.htm

Follow this links;

1. By default enums have their own string values, the name.
2. We have to create parameterized constructor for this enum class to have custom values. Why? Because as we know that enum class's object can't be create explicitly so for initializing we use parameterized constructor. And the constructor cannot be the public or protected it must have private or default modifiers. Why? Cause we can't actually create it's object explicitly, as we can't call the constructor, so, there is no point of having a public constructor.
3. We have to create one getter method to get the value of enums.
4. enum can contain both concrete methods and abstract methods
5. values() method can be used to return all values present inside enum.
6. ordinal() method, each enum constant index can be found, just like array index.
7. valueOf() method returns the enum constant of the specified string value, if exists.

Random Notes

1. Local variables can't be static
2. final modifier is applicable for variable but not for objects, Whereas immutability applicable for an object but not for variables.
3. Initialize final variables at the time or declare or in the constructor.

Immutable

final modifier is applicable for variable but not for objects, Whereas immutability applicable for an object but not for variables.

Set instance variables private, remove setters, set methods final (to avoid override) to make a object immutable.

For more, follow the link:
https://www.journaldev.com/129/how-to-create-immutable-class-in-java

Clone

Object cloning: implement java.lang.Cloneable marker interface, call Abul.clone() to get a clone. It uses reflect API to assign all properties to a new object and return it.it uses shallow cloning.

To get deep clone, override clone() method and write code to make deep copy of each property for objects and use super.clone(), or just create a constructor, pass existing object and set all values from it. **(to avoid implementation)**

For more: https://www.journaldev.com/60/java-clone-object-cloning-java

Serializable could be utilized for deep clone. But costly

Serialize

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.

class A implements Serializable{}

The Serialization runtime associates a version number with each Serializable class called a SerialVersionUID, which is used during Deserialization to verify that sender and receiver of a serialized object have loaded classes for that object which are compatible with respect to serialization. If the receiver has loaded a class for the object that has different UID than that of the corresponding sender's class, the Deserialization will result in an InvalidClassException. A Serializable class can declare its own UID explicitly by declaring a field name.

It must be static, final and of type long.

i.e- ANY-ACCESS-MODIFIER static final long serialVersionUID=42L;

For more: https://www.geeksforgeeks.org/serialization-in-java/

Comparable and Comparator

A comparable object is capable of comparing itself with another object. The class itself must implement the java.lang.Comparable interface to compare its instances.

Unlike Comparable, Comparator is external to the element type we are comparing. It's a separate class.

Comparable is meant for objects with natural ordering which means the object itself must know how it is to be ordered.

A basic differentiating feature is that using comparable we can use only one comparison. Whereas, we can write more than one custom comparators as you want for a given type, all using different interpretations of what sorting means.

To summarize, if sorting of objects needs to be based on natural order then use Comparable whereas if you sorting needs to be done on attributes of different objects, then use Comparator in Java.

For more: https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/

1. comparator for priority queue

```
PriorityQueue<ListNode> pq=new PriorityQueue<>((x,y)->Integer.compare(x.val,y.val));
```

2. comparator for array

```
int[] nums=Arrays.stream(nums).boxed()
        .sorted((a,b)-> (""+b+a).compareTo(""+a+b))
        .mapToInt(i -> i).toArray();
```

3. if " Class Abul{int a, int b}" . you have a list of abul and want to sort by a and then b

```
Comparator<abul> comparator = Comparator.comparing(ob -> ob.a);
comparator = comparator.thenComparing(Comparator.comparing(ob -> ob.b));
abulList=abulList.stream().sorted(comparator).collect(Collectors.toList());
```

4. Arrays.sort(array)

Collections.sort(list)

for this to work, the class of the elements must implement Comparable interface,

override compareTo method and define the way to compare. return negative,zero,positive

integer for less,equl,more outpu.

```
  @Override
 public int compareTo(Employee emp) {
    return (this.id - emp.id);
 }
```

or, write a comparator and pass it to the sort function

```
Comparator<abul> aComparator=new Comparator<abul>() {
   @Override
   public int compare(abul abul, abul t1) {
      return abul.a-t1.a;
   }
};
Collections.sort(list,aComparator);
```

5. Or ,sort 2d array with shorthand

Arrays.sort(array, Comparator.comparingDouble(o -> o[0]));

Or

Arrays.sort(nums, (a, b) -> a[0] == b[0] ? a[1] == b[1] ? a[2] - b[2] : a[1] - b[1] : a[0] - b[0]);

Notes

1. concat integer array

Arrays.stream(nums).mapToObj(String::valueOf).reduce((a,b)->a.concat(b)).get();

2. make the right 1 to 0

idx= idx - (idx & -idx)

// https://youtu.be/aAALKHLeexw?t=1585

3. Convert List to array

```
List<String> s=new ArrayList<>();
String[] k=s.toArray(new String[0]); // the '0' is nothing, pass any number in the parameter

List<int[]> out =new ArrayList<>();
int[][] a= out.toArray(new int[0][]);

Foo[] array = new Foo[list.size()];
list.toArray(array); // fill the array
```

Concurrency vs Parallelism

For more:
https://medium.com/@itIsMadhavan/concurrency-vs-parallelism-a-brief-review-b337c8dac350#:~:text=A%20system%20is%20said%20to,the%20phrase%20%E2%80%9Cin%20progress.%E2%80%9D

A system is said to be concurrent if it can support two or more actions in progress at the same time. A system is said to be parallel if it can support two or more actions executing simultaneously.

Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.

-- Concurrency means executing multiple tasks at the same time but not necessarily simultaneously--.

There are two tasks executing concurrently, but those are run in a 1-core CPU, so the CPU will decide to run a task first and then the other task or run half a task and half another task, etc. Two tasks can start, run, and complete in overlapping time periods i.e Task-2 can start even before Task-1 gets completed. It all depends on the system architecture.

Parallelism means that an application splits its tasks up into smaller subtasks which can be processed in parallel, for instance on multiple CPUs at the exact same time.

Parallelism does not require two tasks to exist. It literally physically run parts of tasks OR multiple tasks, at the same time using the multi-core infrastructure of CPU, by assigning one core to each task or sub-task.

Parallel Stream

Collections.parallelStream() and Collections.stream().parallel()

Both works at the same way, but parallelStream() could be overridden in a Collection type.

For more: https://stackoverflow.com/questions/43811182/parallelstream-vs-stream-parallel

https://www.geeksforgeeks.org/what-is-java-parallel-streams/

https://medium.com/javarevisited/java-8-parallel-stream-java2blog-e1254e593763

Ways to run a Thread

For more: https://www.baeldung.com/java-start-thread

Executor Framework vs Fork Join Pool

ForkJoinPool uses a work-stealing pattern, which means one thread can also execute a pending task from another thread. This improves efficiency in the case of ForkJoinTask as most of the ForkJoinTask algorithm spawn new tasks. Where the former provides a general-purpose thread pool

For more: https://javarevisited.blogspot.com/2016/12/difference-between-executor-framework-and-ForkJoinPool-in-Java.html#axzz6ltdx0wj2

ExecutorService and Callable

For more: https://howtodoinjava.com/java/multi-threading/executor-service-example/

Fixed threadpool is basically max how many runnable task be executed at the same time using stand by thread. Every task have to be runnablee.

The shutdown() method doesn't cause immediate destruction of the ExecutorService. It will make the ExecutorService stop accepting new tasks and shut down after all running threads finish their current work.

executorService.shutdown();

Calling the method get() on a Future blocks the current thread and waits until the callable completes before returning the actual result.

In ExecutorService you can't submit 10 million tasks at once.

Do not be afraid, you can submit all of them into executor. The actual amount of tasks run in parallel is limited by the underlying thread pool size. That is, if you have pool size of 2, only two tasks are being executed at the time, the rest sit in the queue and wait for the free thread.

By default Executors.newFixedThreadPool() creates the Executor that has a queue of Integer.MAX_VALUE size, hence your millions of tasks would fit there.

ExecutorService es = Executors.newSingleThreadExecutor();

.newFixedThreadPool(2);

newScheduledThreadPool(10);

new ThreadPoolExecutor(10, 100, 5L, TimeUnit.MILLISECONDS,

new LinkedBlockingQueue<Runnable>());

Topics: InvokeAll, Future, Callable,

For more:
https://winterbe.com/posts/2015/04/07/java8-concurrency-tutorial-thread-executor-examples/

Fork/Join

Worker threads can execute only one task at a time, but the ForkJoinPool doesn't create a separate thread for every single subtask. Instead, each thread in the pool has its own double-ended queue (or deque, pronounced deck) which stores tasks.

a worker thread gets tasks from the head of its own deque. When it is empty, the thread takes a task from the tail of the deque of another busy thread or from the global entry queue, since this is where the biggest pieces of work are likely to be located.

This approach minimizes the possibility that threads will compete for tasks. It also reduces the number of times the thread will have to go looking for work, as it works on the biggest available chunks of work first.

For more: https://www.baeldung.com/java-fork-join

Asynchronous programming

Nothing but just calling a thread. Could be done with CompletableFuture .

```
new Thread(() -> {
   //Do whatever
}).start();

CompletableFuture.runAsync(() -> {
   // method call or code to be asynch.
});
```

For more:
https://stackoverflow.com/questions/1842734/how-to-asynchronously-call-a-method-in-java

CompletableFuture

For more: https://www.callicoder.com/java-8-completablefuture-tutorial/

Functional programming

Lamda Expression

An interface with single abstract method is called functional interface. An example is java.lang.Runnable

Lambda expressions basically express instances of functional interfaces . lambda expressions implement the only abstract function and therefore implement functional interfaces

```
interface FuncInterface{
   void abstractFun(int x);
   default void normalFun() { Sout("Hello");}
}
class Test{
   public static void main(String args[]){
      FuncInterface fobj = (int x)->System.out.println(2*x);
```

```
        fobj.abstractFun(5);
    }
}

arrL.forEach( (n) -> {System.out.println(n); });
```

For more: https://www.geeksforgeeks.org/lambda-expressions-java-8/

https://www.codejava.net/java-core/the-java-language/java-8-lambda-runnable-example

Wait() and Notify()

For more:
https://www.netjstech.com/2015/07/why-wait-notify-and-notifyall-called-from-synchronized-java.html

https://www.baeldung.com/java-wait-notify

wait() method is about thread releasing the object's lock and go to sleep where as notify/notifyAll methods are about notifying the thread(s) waiting for the object's lock. So, wait(), notify() and notifyAll() methods (as mentioned above) should be invoked on an object only when the current thread has already acquired the lock on an object. Thats why these are called from inside synchronized

In some cases, all waiting threads can take useful action once the wait finishes. An example would be a set of threads waiting for a certain task to finish; once the task has finished, all waiting threads can continue with their business. In such a case you would use notifyAll() to wake up all waiting threads at the same time.

Garbage Collection

n the first step, unreferenced objects are identified and marked as ready for garbage collection. In the second step, marked objects are deleted. Optionally, memory can be compacted after the garbage collector deletes objects, so remaining objects are in a contiguous block at the start of

the heap. The compaction process makes it easier to allocate memory to new objects sequentially after the block of memory allocated to existing objects.

All of HotSpot's garbage collectors implement a generational garbage collection strategy that categorizes objects by age. The rationale behind generational garbage collection is that most objects are short-lived and will be ready for garbage collection soon after creation.

The heap is divided into three sections:

Young Generation: Newly created objects start in the Young Generation. The Young Generation is further subdivided into an Eden space, where all new objects start, and two Survivor spaces, where objects are moved from Eden after surviving one garbage collection cycle. When objects are garbage collected from the Young Generation, it is a minor garbage collection event.

Old Generation: Objects that are long-lived are eventually moved from the Young Generation to the Old Generation. When objects are garbage collected from the Old Generation, it is a major garbage collection event.

Permanent Generation: Metadata such as classes and methods are stored in the Permanent Generation. Classes that are no longer in use may be garbage collected from the Permanent Generation.

For more: https://stackify.com/what-is-java-garbage-collection/

Vector vs Arraylist

Vector is synchronized, whereas access to an ArrayList is not.you can use the Collections.synchronizedList function to create a synchronized list, thus getting you the equivalent of a Vector.

List<String> synlist = Collections .synchronizedList(new ArrayList<String>());

equals() and hashCode() methods

equals(Object otherObject) – he default implementation of equals method is defined in Java.lang.Object class which simply checks if two Object references (say x and y) refer to the same Object

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

It returns the hashcode value as an Integer. Hashcode value is mostly used in hashing based collections like HashMap, HashSet, HashTable….etc. This method must be overridden in every class which overrides equals() method.

For more: https://www.geeksforgeeks.org/equals-hashcode-methods-java/

https://stackoverflow.com/questions/2265503/why-do-i-need-to-override-the-equals-and-hashcode-methods-in-java

https://www.geeksforgeeks.org/internal-working-of-hashmap-java/

Bit Manipulation

For more:
https://www.hackerearth.com/practice/basic-programming/bit-manipulation/basics-of-bit-manipulation/tutorial/

Database Lock

InnoDB implements standard row-level locking where there are two types of locks, shared (S) locks and exclusive (X) locks.

MyISAM locking occurs at the table level, To achieve a very high lock speed . This is not as desirable as page or row locking for concurrency in a mixed read/write environment. However, deadlock cannot occur with table locking as it can with page or row locking.

For more: https://www.sqlshack.com/locking-sql-server/

DB Select…for Update

For more: https://www.cockroachlabs.com/docs/stable/select-for-update.html

Update Locking mechanism in Mongo

A transaction (t1) can be subject to a writeConflict if another write operation modifies the same document (D1) after the transaction starts and before the transaction itself tries to modify it. This can happen whether or not the other write operation is in a transaction.

So the write operation has document level lock

For more:

https://www.linkedin.com/pulse/handling-locks-mongodb-jerwin-roy/

https://www.mongodb.com/blog/post/how-to-select--for-update-inside-mongodb-transactions

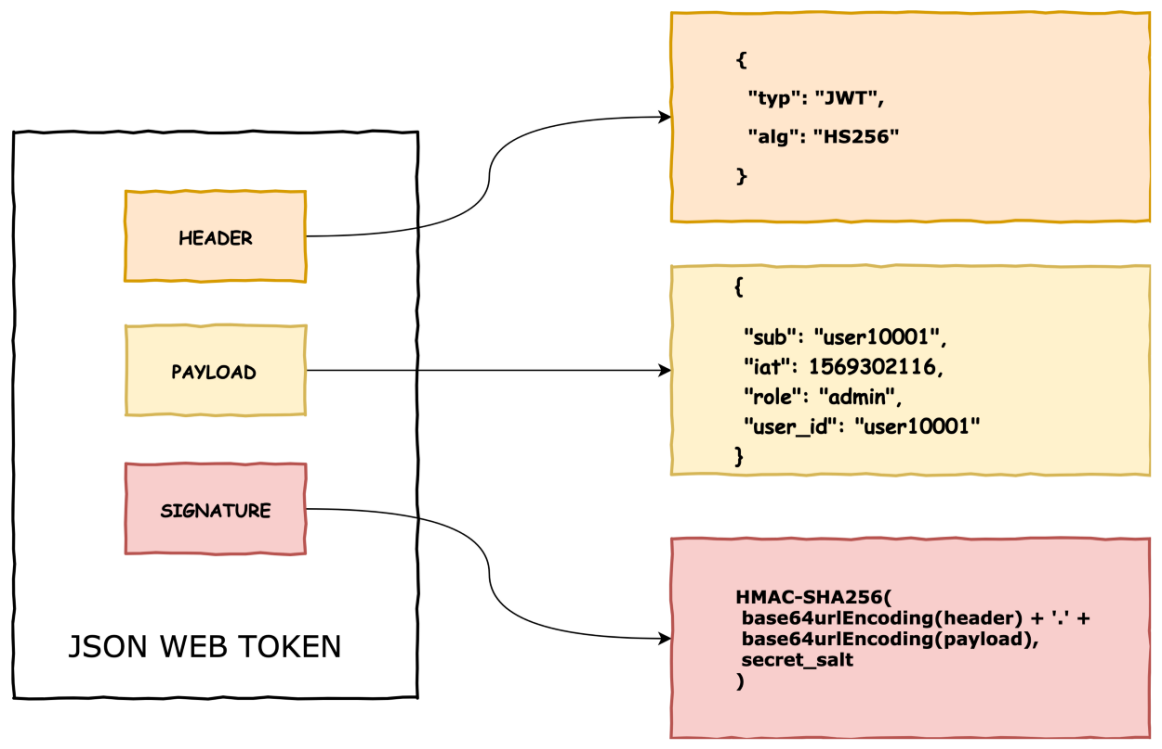
JWT

JSON Web Token (JWT) is an open standard (RFC 7519) for securely transmitting information between parties as JSON object.

It is compact, readable and digitally signed using a private key/ or a public key pair by the Identity Provider(IdP). So the integrity and authenticity of the token can be verified by other parties involved.

The purpose of using JWT is not to hide data but to ensure the authenticity of the data. JWT is signed and encoded, not encrypted.

JWT is a token based stateless authentication mechanism. Since it is a client-side based stateless session, server doesn't have to completely rely on a datastore(database) to save session information.



For more:
https://medium.com/@sureshdsk/how-json-web-token-jwt-authentication-works-585c4f076033

https://flaviocopes.com/jwt/

Spring

@SpringBootApplication

As per the Spring Boot doc, the @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan with their default attributes.

@SpringBootApplication annotation provides a load of defaults (like the embedded servlet container), depending on the contents of your classpath and other things. It also turns on Spring MVC's @EnableWebMvc annotation, which activates web endpoints.

@ComponentScan

@ComponentScan enables Spring to scan for things like configurations, controllers, services, and other components we define.

Alternatively, Spring can also start scanning from the specified package, which we can define using basePackageClasses() or basePackages(). If no package is specified, then it considers the package of the class declaring the @ComponentScan annotation \

```
@Configuration
@ComponentScan(basePackages = {"com.baeldung.annotations.componentscanautoconfigure.healthcare",
 "com.baeldung.annotations.componentscanautoconfigure.employee"},
 basePackageClasses = Teacher.class)
```

@EnableAutoConfiguration

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.

For example, when we define the spring-boot-starter-web dependency in our classpath, Spring boot auto-configures Tomcat and Spring MVC. However, this auto-configuration has less precedence in case we define our own configurations.

We can use exclude to disable a list of classes that we do not want to be auto-configured:

```
@Configuration @EnableAutoConfiguration(excludeName =
{"org.springframework.boot.autoconfigure.jdbc.JdbcTemplateAutoConfiguration"})

@EnableAutoConfiguration(exclude={JdbcTemplateAutoConfiguration.class})
```

Actuator

if something goes wrong, we always need to analyze the logs and dig through the data flow of our application to check to see what's going on. So, the Spring Actuator provides easy access to those kinds of features. It provides many features, i.e. what beans are created, the mapping in the controller, the CPU usage, etc. Automatically gathering and auditing health and metrics can then be applied to your application.

For more: https://dzone.com/articles/top-10-spring-boot-interview-questions

Shutdown is an endpoint that allows the application to be gracefully shutdown. This feature is not enabled by default. You can enable this by using management.endpoint.shutdown.enabled=true in your application.properties file. But be careful about this if you are using this.

Disable web server in the Spring Boot application?

The major strong point in Spring is to provide flexibility to build your application loosely coupled. Spring provides features to disable the web server in a quick configuration. Yes, we can use the application.properties to configure the web application type, i.e. spring.main.web-application-type=none.

Spring Data JPA vs Hibernate

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

This dependency includes JPA API, JPA Implementation, JDBC and other needed libraries. Since the default JPA implementation is Hibernate, this dependency is actually enough to bring it in as well.

IOC vs DI

In short, IoC is a much broader term that includes, but is not limited to, DI

The term Inversion of Control (IoC) originally meant any sort of programming style where an overall framework or run-time controlled the program flow

Before DI had a name, people started to refer to frameworks that manage Dependencies as Inversion of Control Containers, and soon, the meaning of IoC gradually drifted towards that particular meaning: Inversion of Control over Dependencies.

Inversion of Control (IoC) means that objects do not create other objects on which they rely to do their work. Instead, they get the objects that they need from an outside source (for example, an xml configuration file).

IoC is a design principle which recommends the inversion of different kinds of controls in object-oriented design to achieve loose coupling between application classes. In this case, control refers to any additional responsibilities a class has, other than its main responsibility, such as control over the flow of an application, or control over the dependent object creation and binding (Remember SRP - Single Responsibility Principle). If you want to do TDD (Test Driven Development), then you must use the IoC principle, without which TDD is not possible. Learn about IoC in detail in the next chapter.

Dependency Injection (DI) means that this is done without the object intervention, usually by a framework component that passes constructor parameters and set properties.

Dependency Injection (DI) is a design pattern which implements the IoC principle to invert the creation of dependent objects.

IoC is a generic term meaning that rather than having the application call the implementations provided by a library (also know as toolkit), a framework calls the implementations provided by the application.

DI is a form of IoC, where implementations are passed into an object through constructors/setters/service lookups, which the object will 'depend' on in order to behave correctly.

IoC without using DI, for example would be the Template pattern because the implementation can only be changed through sub-classing.

DI frameworks are designed to make use of DI and can define interfaces (or Annotations in Java) to make it easy to pass in the implementations.

IoC containers are DI frameworks that can work outside of the programming language. In some you can configure which implementations to use in metadata files (e.g. XML) which are less invasive. With some you can do IoC that would normally be impossible like inject an implementation at pointcuts.

The IoC container is a framework used to manage automatic dependency injection throughout the application, so that we as programmers do not need to put more time and effort into it

Extra

Mean/Max Heap

Similar to binary tree (not BST), with additional 2 properties

1. Have to be complete , except last level , Last level have to be complete from left to right
2. Every node have to be less or equal to child values(for MinHeap) , greater or equal to child (for MaxHeap)

3. The child for ith node is 2i+1 and 2i+2
4. Parent for ith node is floor((i-1)/2)