**Core Java**

**OOPS**

1. What is immutability?

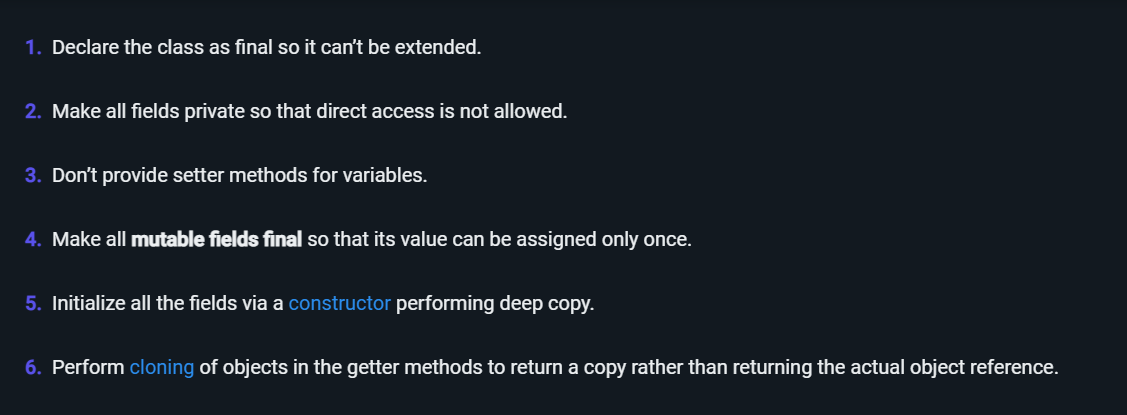
Immutability. If an object is immutable that we cannot change the state of that object.

The example of immutability

**caching.**

**thread safety. So if you class is immutable, so you do not need to worry about the thread safety in multithreaded environment.**

**How to make a class immutable?**

****

Public final class Ayush{

Private final String name;

Private final String address;

Private final HashMap<String ,String> metadata;

Public Ayush(String name, String address, HashMap<String,String> b ){

This.name=name;

This.address=address;

//perform deepcloning

Map<String ,String>temp=new Hashmap<String,String>();

For(Map.Entry<String,String> a:b){

Temp.put(a.getKey(),a.getValue());

}

this.metadata=temp;

}

//getter method

Public String getName(){

Return this.name;

}

Public Strign getAddress(){

Return this .address;

}

Public HashMap<String,String> getMetadata(){

Map<String ,String>temp=new Hashmap<String,String>();

For(Map.Entry<String,String> a:b){

Temp.put(a.getKey(),a.getValue());

}

Return temp;

}

}

**Link - [https://www.journaldev.com/129/how-to-create-immutable-class-in-java#:~:text=An%20immutable%20class%20is%20good,case%20of%20multi%2Dthreaded%20environment.](https://www.journaldev.com/129/how-to-create-immutable-class-in-java" \l ":~:text=An%20immutable%20class%20is%20good,case%20of%20multi%2Dthreaded%20environment.)**

[d](https://www.journaldev.com/129/how-to-create-immutable-class-in-java" \l ":~:text=An%20immutable%20class%20is%20good,case%20of%20multi%2Dthreaded%20environment.)

1. **Constructor**

* **It is used in java for initialising the object**
* **It is either parameterized or non-parameterised.**

**Class Ayush{**

**//instance variables**

**Int ayush=0;**

**//construtors 🡪access modifier ConstrutorName()}**

**Public Ayush(){**

**}**

**//paramterised**

**Public Ayush(int ayush)**

**{**

**This.ayush=ayush**

**}**

**//getter setter**

**}**

**What is deep Cloning and What is Shadow Cloning:**

**Clonning – Create a copy of object**

**Deep Cloning**

**Shadow Cloning**

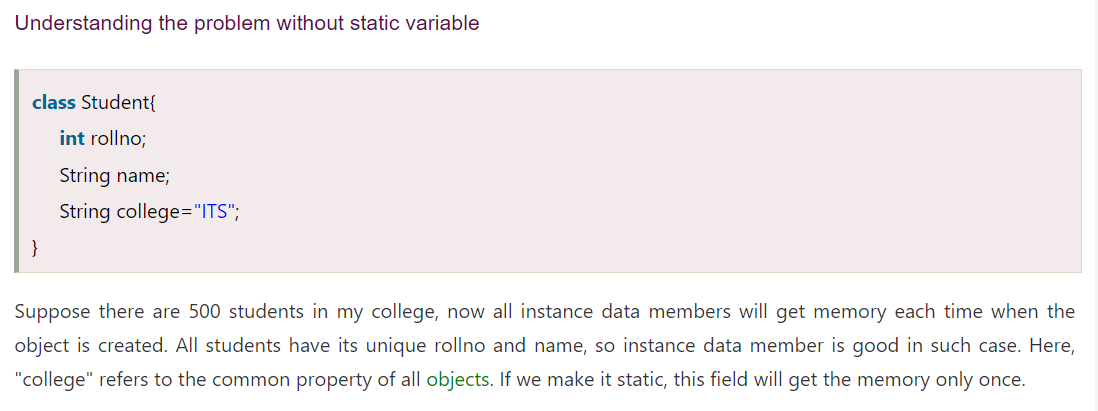
**Static Keywords**

**Static Variable**

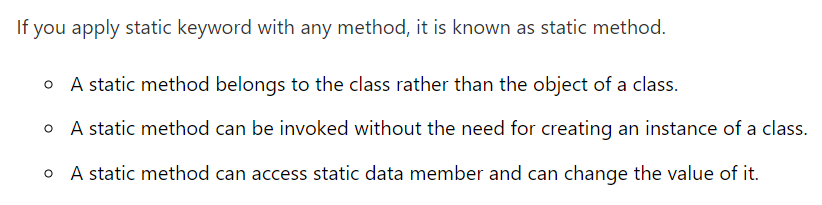
**It is used to refer the common property of the object.**

* The static variable gets memory only once in the class area at the time of class loading.

**Example**

****

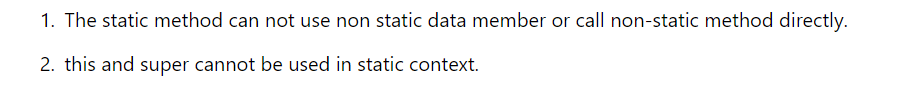
**Static method**

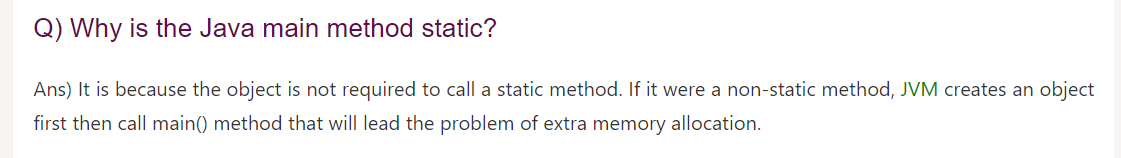
****

**Static instance change into the static member**

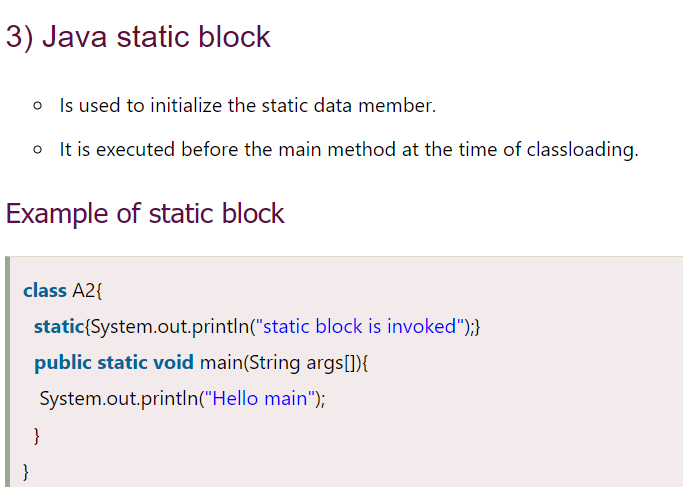
**Non Static variable will not be call in the static member it gives error for that non static mentber.**

**This keyword can not used with the static context.**

****

****

**Static Block**

****

**Overload and OverRide the scenario**

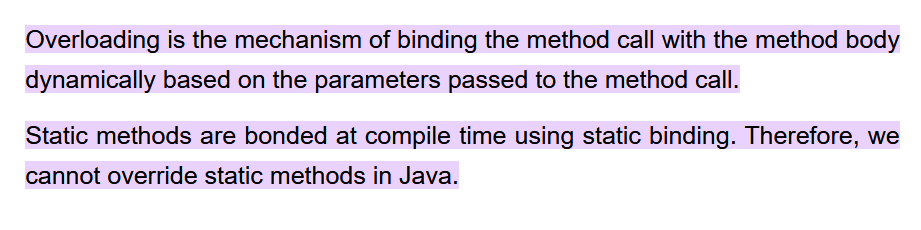
**Static**

**OverLoad Method-> Yes we can overload the static method**

**OverRide Method-> No we can not override the static method**

**Can We override a static method –**

**No Why?**

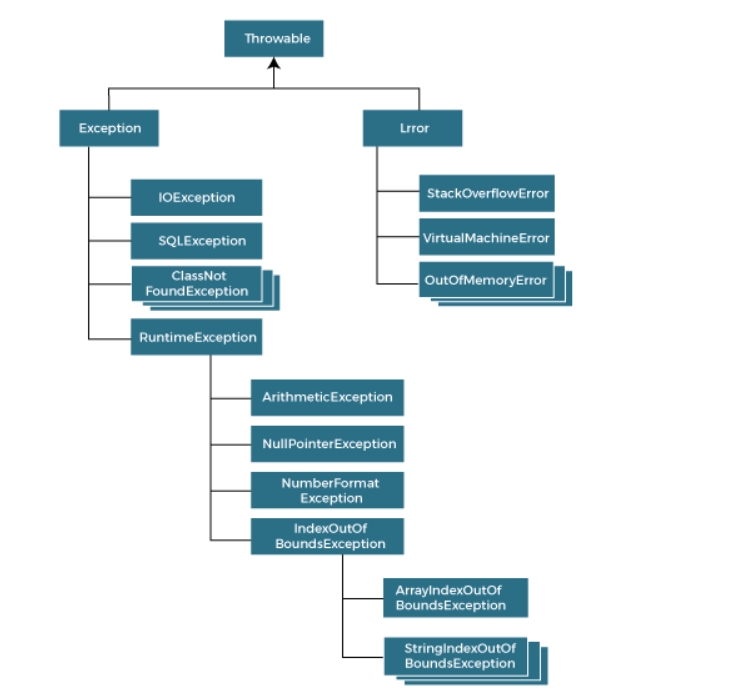
****

**Interface**

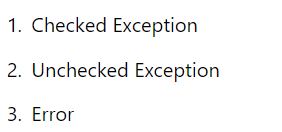
**Why Interface has Static method –**

**We can call the method of static by using the Interfacename and the**

**Exceptions In java**

****

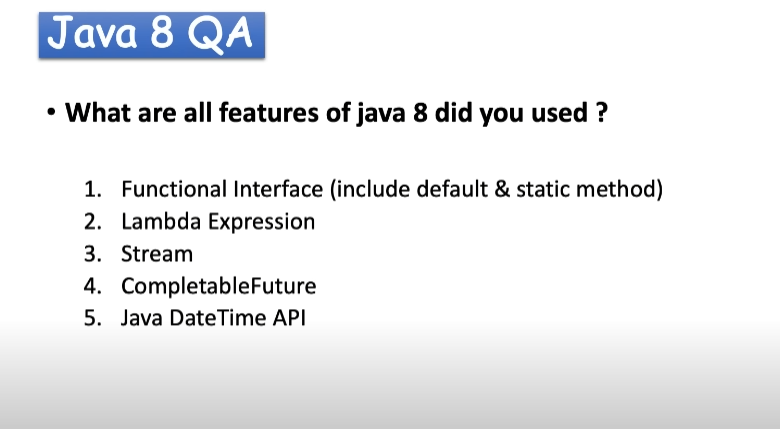
**Types of Exceptions**

****

**Q3. Working of linked hashmap and linked hashset**

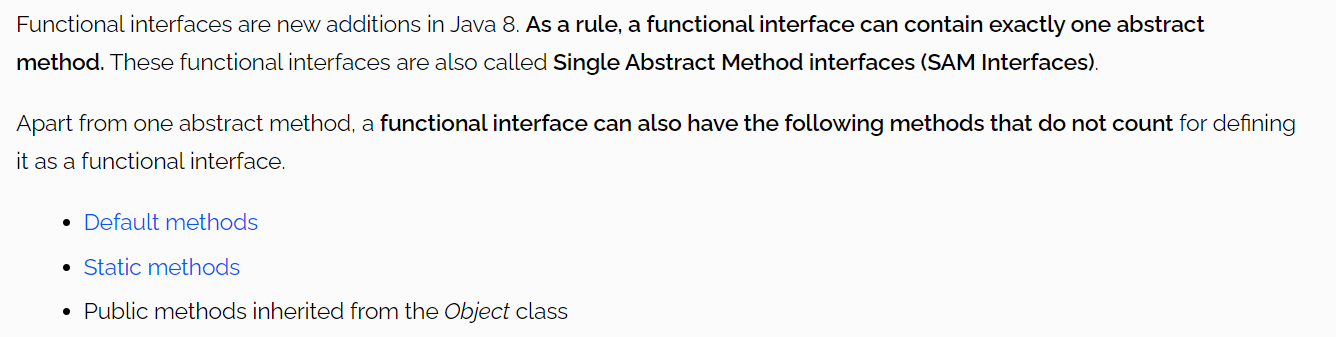
**https://www.youtube.com/watch?v=m6w\_N14\_IRs**

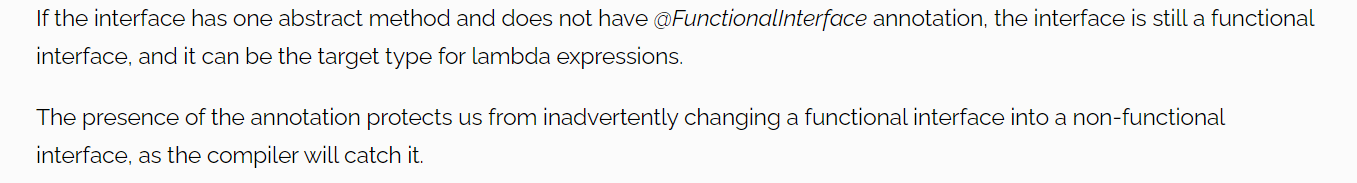
Java 8

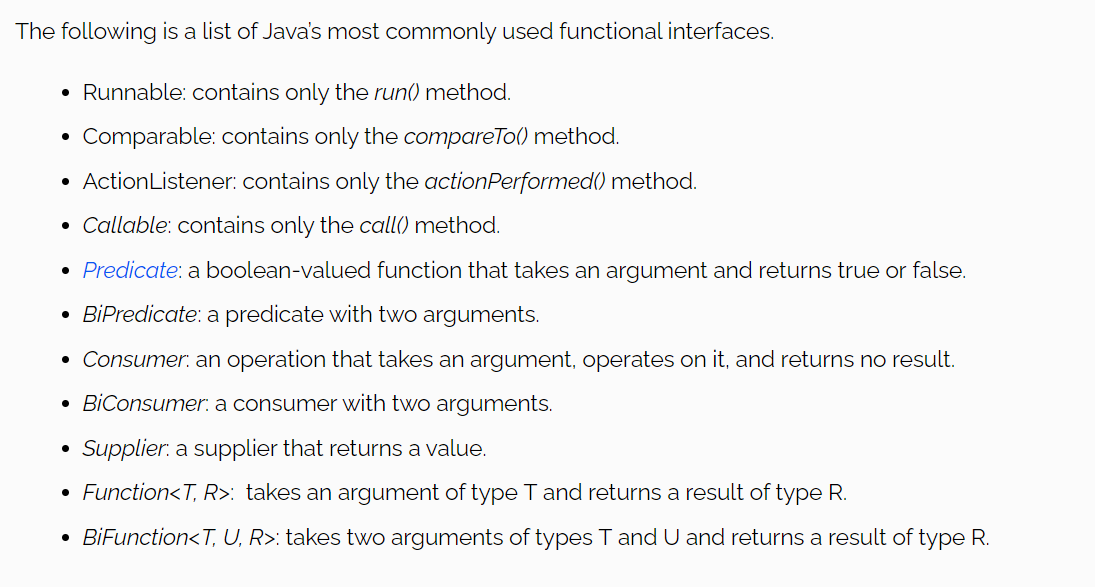


What is Lambda Expressions?

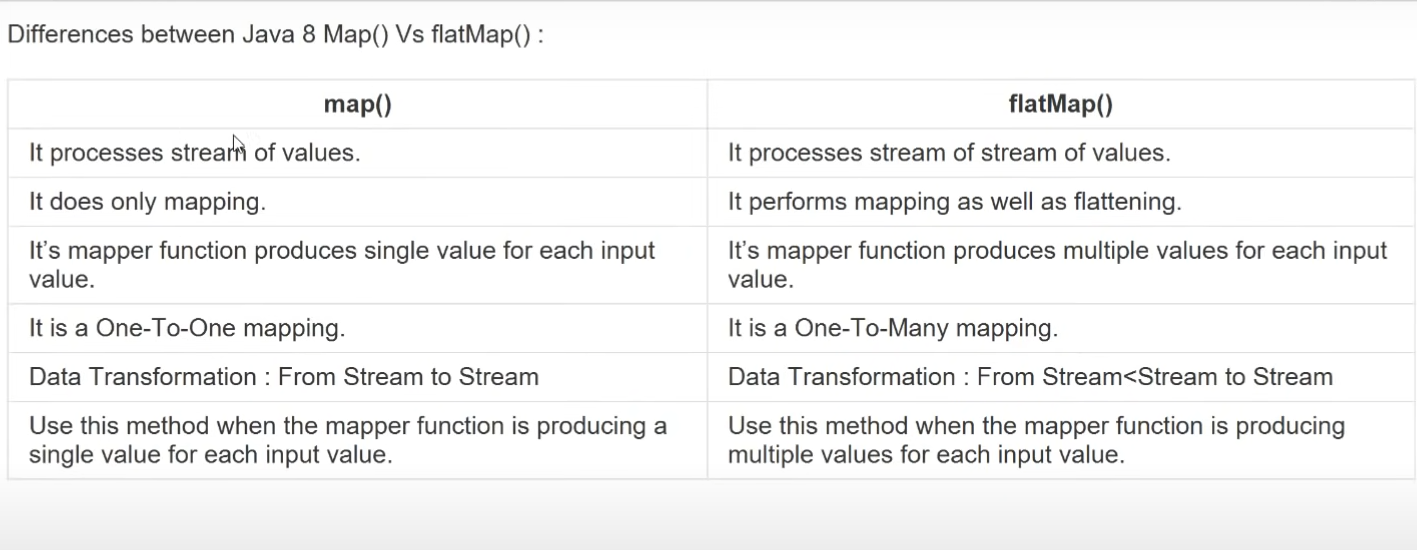
Functional Interface in java







Difference Between Map and Flatmap



**Spring Boot Annotations**

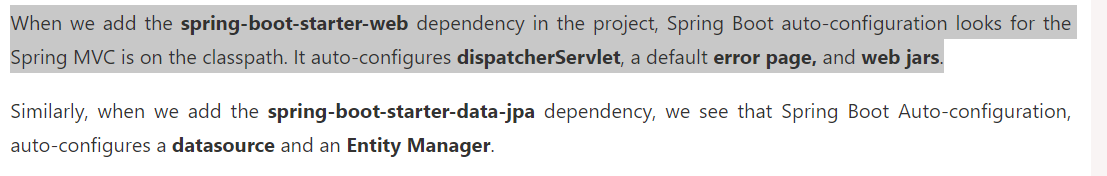
**@SpringBootAnnotations**

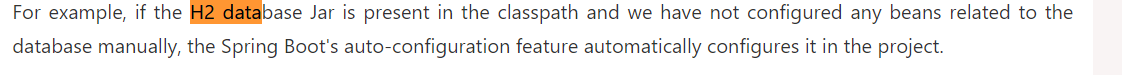
1. **@EnableAutoConfigurations**

**This annotation is used to automatically configure the dependency based on the class path.**

**(It enables the spring boot to automatic configure the(spring application based on the jar dependency ) application context in spring boot i.e It automatically created and register beans based on the included jar in the class path and the bean defined by us).**

**Examples**

****

****

1. **@ComponentScan**

**It can scan to your bean so that it can visible to the IOC container.**

**It simply scan the current package as well as the base package.**

1. **@Configurations**

**We used this configuration for java based configuration.**

**We can define the bean definition in this class so that the spring IOC can load this**

**@Configuration/@Bean**

**Both are used together to give java based configirurations.**

**It can be used by spring container as a source of bean definitions**

**@Configuration**

**Public class HelloWord{**

**@Bean**

**Public HelloWorld t(){**

**Return new HelloWorld();**

**}}**

**Kafka+RestTempate both are the example of the configuration class**

**Spring stereotype Annotations:**

**@Component :**

**It is used to create a spring bean automatically in the application context.**

**Spring will scan the bean and manage its life cycle from object creation to object destroy.**

**@Service**

**Where we can write the business logic**

**@RestController/@Controller**

**It is at the web layer where we can expose our APIs.**

**Difference between the RestController and controller**

**-------------------------------------------------------------------------**

|  |  |
| --- | --- |
| **Controller** | **Rest Controller** |
| **It is used for making classes as Spring mvc controller.** | **It is used for making the restful webservices.** |
| **Controller** | **RestController=Controller+Response Body** |
| **Officially added in spring 2.5** | **Added in spring 4.0** |

* **You do not need to write down the @ResponseBody on every handler method once you annotate the class with the @RestController.**

**RestController= Controller+Response Body**

**@Repository**

**Where we can write the database logics.**

**Spring Core Annotations**

**@Configuration;**

**Class can be used by spring IOC Container which is source of bean defnitions.**

**@Configuration //then spring ioc can except that this class to define the couple of bean which will return the object**

**@Configuration**

**@ComponentScan**

**Public class ayush{**

**}**

**@Bean**

**Public class ayushTestBean{**

**}**

**@Bean**

**@Autowired**

**@Qualifier**

**@Lazy**

**@Value**

**@Property Source**

**@Configuration property**

**@Profile**

**@Scope**

**MultiThreading in java**

**How to Create thread in Java**

* **By using Thread class**
* **By using the Runnable Interface**

**Methods in Thread Class**

**Start()-New Thread is created and the code inside the run() method is executed.**

**Sleep()**

**This is used to stop the execution of current thread for the specific durartion of time.**

**It is in Lang package**

**It is has 4 methods in Thread Class**

**Void sleep(long millis) IllegalArgumentException**

**Void sleep(long millis, int nanos) IllegalArgumentException**

**Void sleep(long millis) InterruptedException**

**Void sleep(long millis, int nanos) InterruptedException**

**Exception it will through 🡪 IllegalArgumentException, InterruptedException**

**Code Examples**

**Public class A extends Thread{**

**@Override**

**Public void run(){**

**Try{**

**For(int i=0;i<5;i++){**

**Thread.sleep(1000);**

**}**

**}catch(InterruptedException){**

**Sop(“ayush”);**

**}**

**}**

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

**A a=new A();**

**Thread t=new Thread(a);**

**t.start();**

**}**

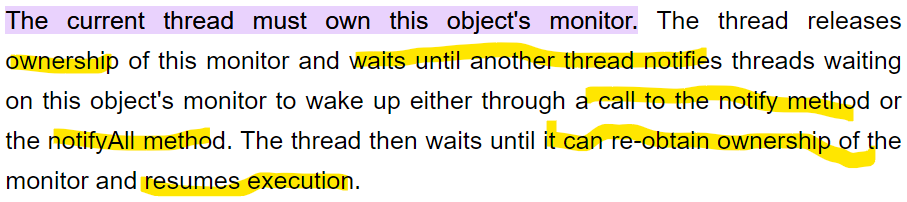
**}**

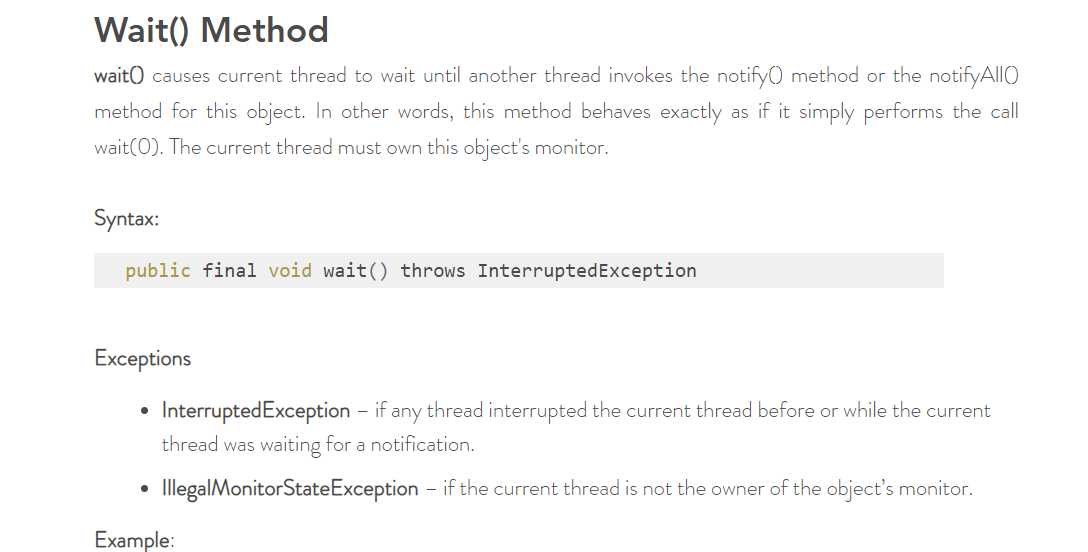
**Run()-no new thread is created and the code inside run method is executed.**

**Wait() method 🡪**

**The wait method is used to wait the current thread until the another thread uses the notify and notify All method.**

**It release the ownership of this monitors**

****

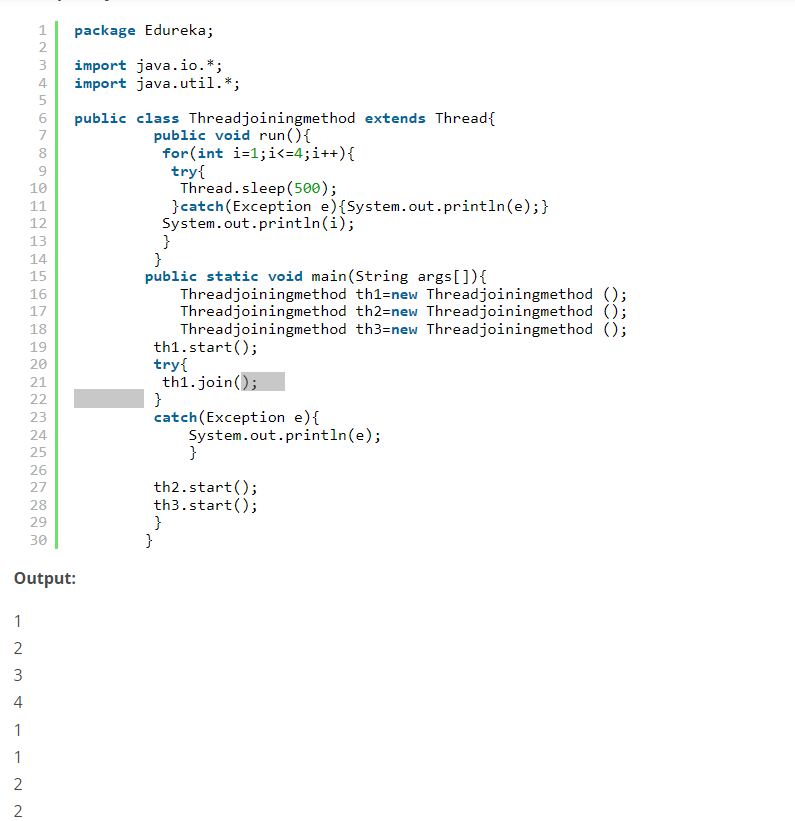
****

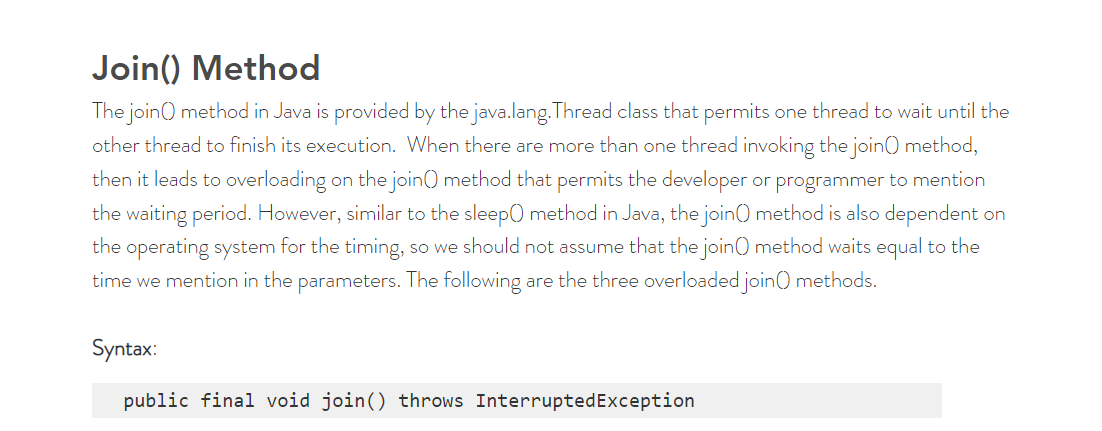
**Join Method**

[**https://www.edureka.co/blog/join-method-java**](https://www.edureka.co/blog/join-method-java)

Join method one thread to wait until another thread completes its execution. In simpler words, it means it waits for the other thread to die.

It has a ***void*** type and throws **InterruptedException**. Joining threads in Java has three functions namely,

****

****

**Difference between Join And Wait method**

|  |  |
| --- | --- |
| **Wait** | **Join** |
| **Wait mainly used when you want inter Thread Communication** | **When you want one thread to start execution only if the first thread finishes its execution** |
| **Present in Object Class.** | **Present in Thread Class.** |
| **Wait must be called for Synchronised context** | **You can call join method with and without synchronised context** |
| **It released the lock.** | **It does not release the lock.** |

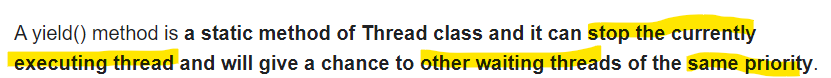
**How Thread can do intercommunication**

**By using the below methods**

* **Wait**
* **Notify**
* **NotifyAll**

**Yield Method-**

**------------------**

****

**Callable and Futrue Interface**

**Callable 🡪**

**It is used when you want to return the values after performing the operation in multithreading environement.**

**Future-> This is used to get the value which is return by the call method of the callable interface in java.**

**It is used by the help of the executer services.**

**Example**

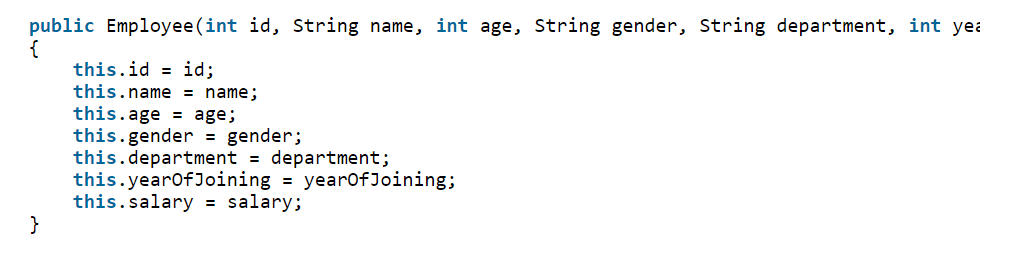
**Future<Integer> a=ExecuterService.submit(new Task());**

**a.get() ; //getting the value return by the callable call method**

**CompletableFurture Interface in Java 8**

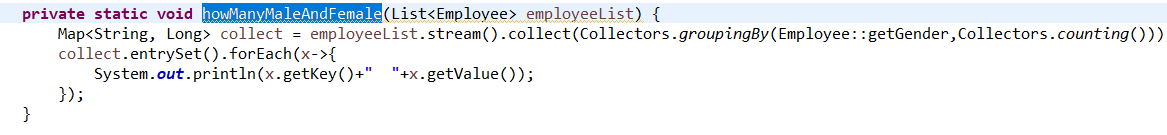
**This is basically used to perform the computational of asynchronous task.**

**Java 8 Coding Questions**

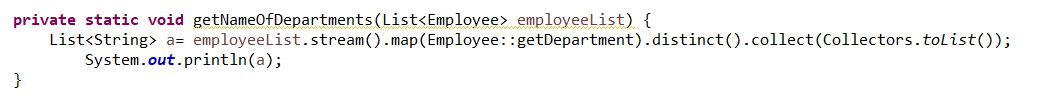


**Question :**

****

****

**Query 3.2 : Print the name of all departments in the organization?**

****

**Query 3.3 : What is the average age of male and female employees?**

**Map<Gender,Double> a=list.stream().collect(Collectors.groupingBy(Employee::genGender,Collectors.averagingDouble(Employee::getAge));**

**Query 3.4 : Get the details of highest paid employee in the organization?**

**Optional<Employee> highestPaidEmployee=**

**List.stream().collect(Collectors.maxBy(Collectors.comparingDouble(Employee::getSalary)));**

**Query 3.5 : Get the names of all employees who have joined after 2015?**

**List<Employee> aa=**

**List.stream().filter(x->x.getJoining>2015).collect(Collectors.toList());**

**Query 3.6 : Count the number of employees in each department?**

**Map<String,long> aa=**

**List.stream().collect(Collectors.groupingBy(Employee::getDepartment,Collectors.counting()));**

**Query 3.7 : What is the average salary of each department?**

**Map<String,Double> aaa=**

**List.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> aa=**

**List.stream().filter(x->x.getDepartment=”Product” && x.getGender=”Male”).min(Comparator.comparingInt(Employee::getAge));**

**Method 2:**

**List.stream().filter(x->x.getDepartment=”Product” && x.getGender=”Male”).collect(Collectors.minBy(Collectors.comparingInt(Employee::getAge));**

**Query 3.9 : Who has the most working experience in the organization?**

**Optional<Employee> aaa=**

**List.stream().collect(Collectors.maxBy(Collectors.comparingDouble(Employee::getYearOfJoinign)**

**Method 2: by using the sorted method and FindFirstmethod**

**Sort employee by year of joining and pick the first employee from the list.**

**Optional<Employee> aa=**

**List.stream().sorted(Comparators.comparingInt(Employee::getYearOfJoining)).findFirst();**

**Query 3.10 : How many male and female employees are there in the sales and marketing team?**

**Map<String,Long> aa=**

**List.stream.filter(x->x.getDepartment= =”Sales” a&& x.getDepartment==”Marketting”)collect(Collectors.groupingBy(Employee::getGeneder,Collectors.counting());**

**Query 3.11 : What is the average salary of male and female employees?**

**Map<String,long> aa=**

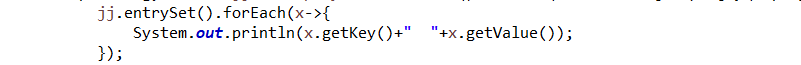
**List.stream().collect(Collectors.groupingBy(Employee::getGender,Collectors.avergingDouble(Employee::getSalary));**

**Query 3.12 : List down the names of all employees in each department?**

**Map<String,List<Employee> aa**

**List.stream().collect(Collectors.groupingBy(Employee::getDeapartment));**

**//convert this into set**

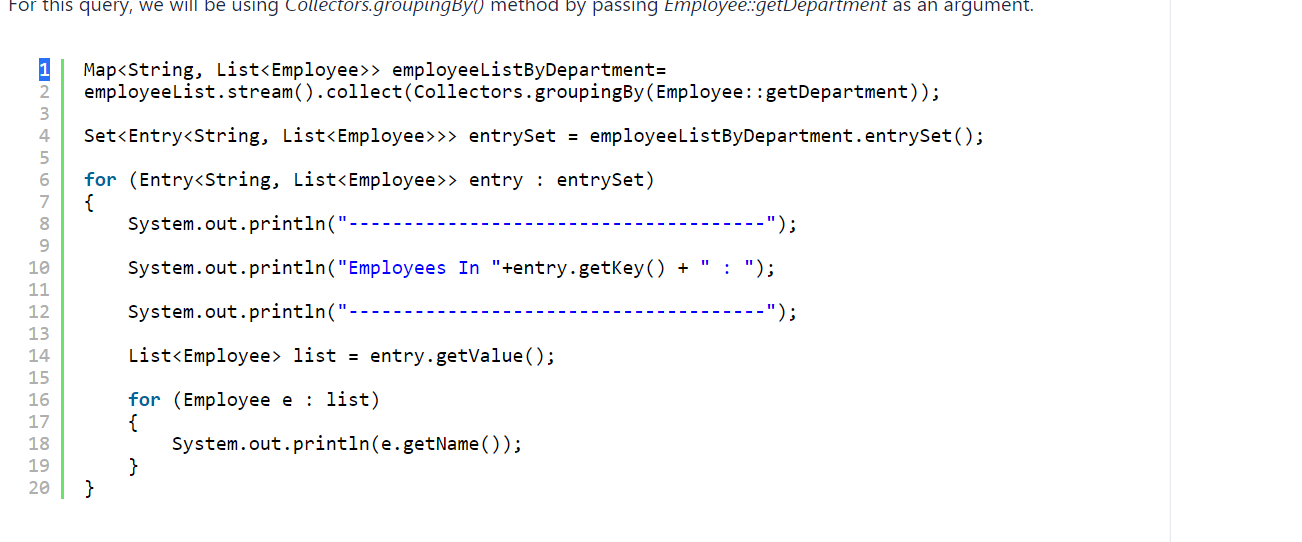
****

**aa.entrySet().forEach(x->{**

**print the key and values**

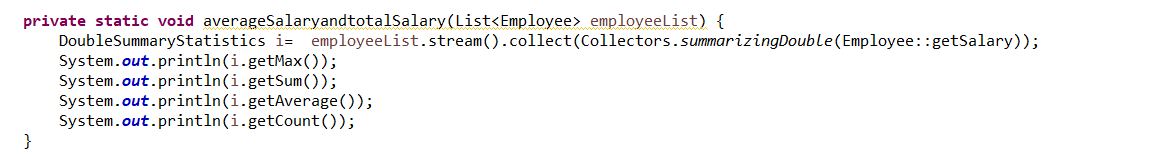
**})**

**//now we are processing the list and request here**

****

**Query 3.13 : What is the average salary and total salary of the whole organization?**

**//for finding the total salart and the average salary we have the Statistics method with us**

****

**DoubleSummaryStatistics a=**

**List.stream().collect(Collectors.summarizingDouble(Employee::getSalary);**

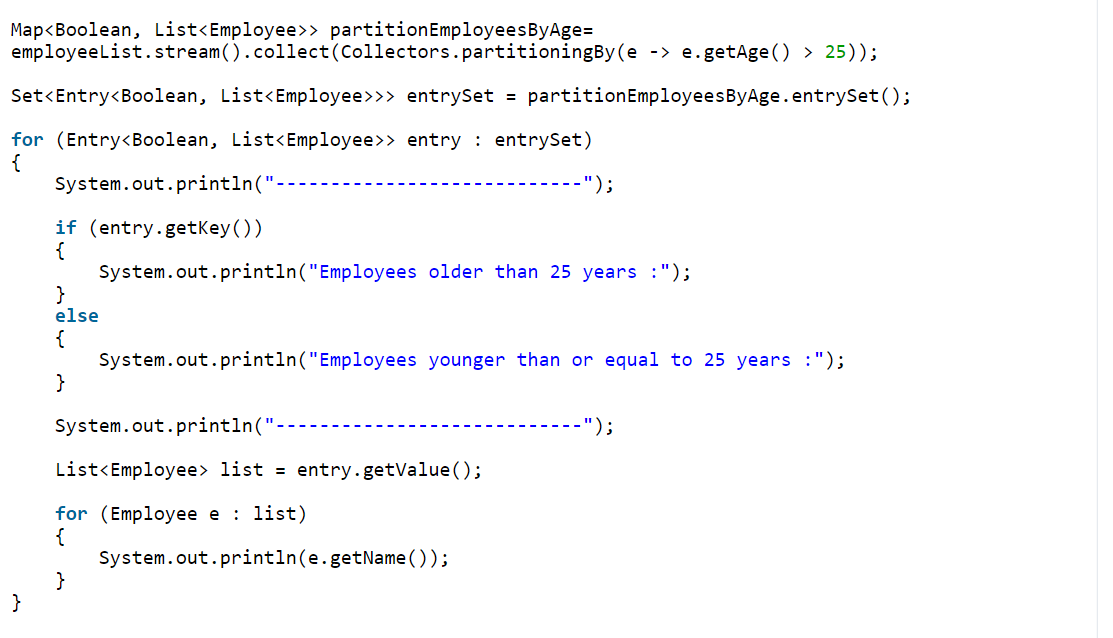
**a.getMax();**

**a.getMin();**

**a.getAverage();**

**a.getCount();**

**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>> aa=**

**List.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?**

**Coding Paradigm**

**1.How to reverse an arraylist**

**Collections.reverse(ArrayList);**

**2.Decreasing Order of TreeSet**

**TreeSet<>aa=new Treeset<>(Collections.reverseOrder());**

**System Designing Concept**

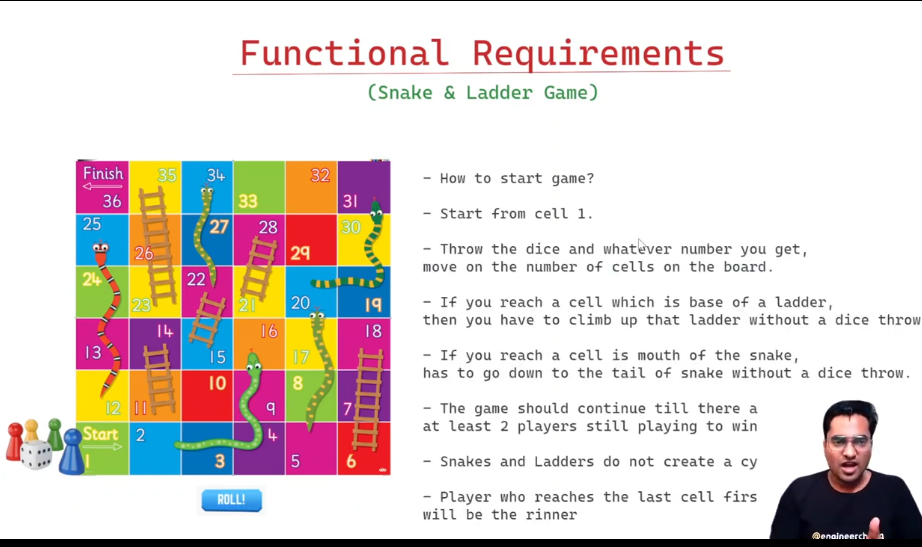
**Snake Ladder Games**

* **Requirements(Functional and Non Functional)**
* **Prioritisation**
* **HLD/tech Stack**
* **Interfaces(LLD)**
* **Component and its implantation.**

**Functional Requirement**

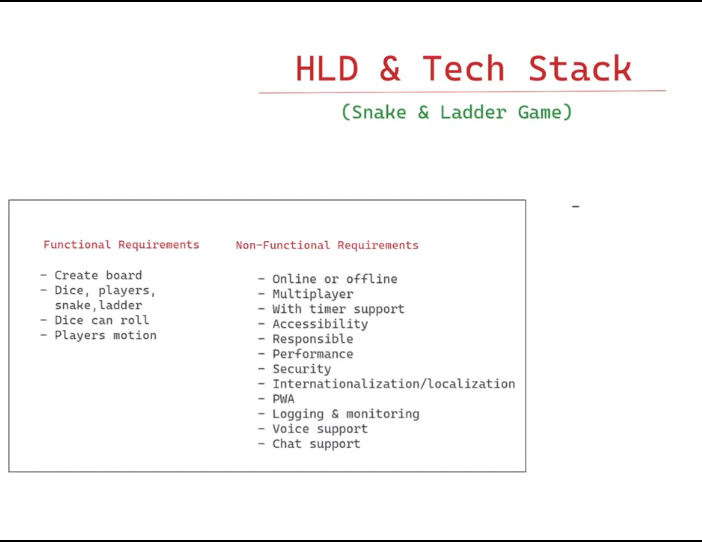
**Think Like a Child**

* **How to start game**
* **What is starting point**
* **Dice roller**
* **What if you reach to any ladder what happens**
* **What if you you reach at the snake mouth place**
* **What is the winning condition**
* **How many players are there at least to start the game.**

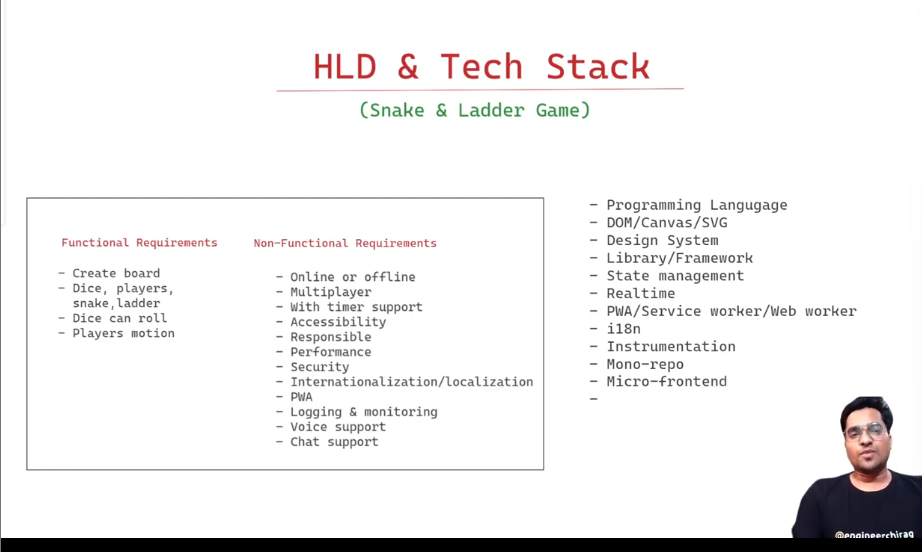
****

**Non Functional Requirement**

* **Game is online or offline**
* **Multiplayer(max player)**
* **Performance**
* **Accessibility**
* **Platform(Web,Tablet and mobile)**
* **Responsiveness**
* **Secuirty**
* **Localisation/Internalisation**
* **Caching(What is cached or what is not cached)**
* **Theming Support(colors)**
* **PWA**
* **Auth**
* **Logging/Monitoring**
* **SEO(Search Engine Optimsation)**
* **Voice Support**
* **Chat Support**
* **2d and 3D support**
* **CI/CD pipeline**

****

**High Level Design and Tech Stack :**

****

**Language --javascript,**

**Dom/Canvas/SVG🡪**

**SVG is most used in the 2d**

**Canvas have a very rich functionality.**

**Design System – material Design System**

**Library/Framework -> Angualr**

**State management ->**

**RealTime🡪Web Socket**

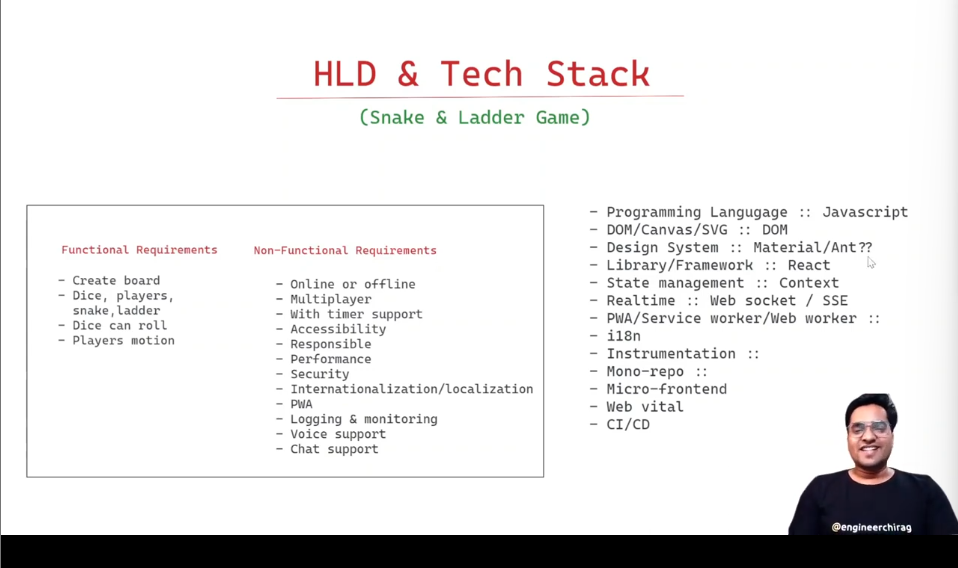
**Web Workers-🡪**

**I18n**

**Instrumentation🡪**

**Mono repo🡪**

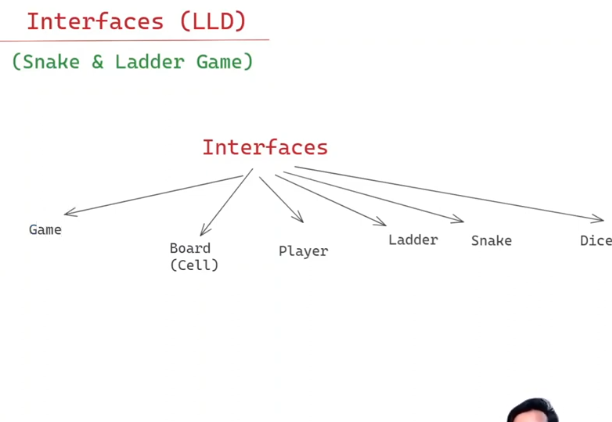
**CiCD Actions-🡪**

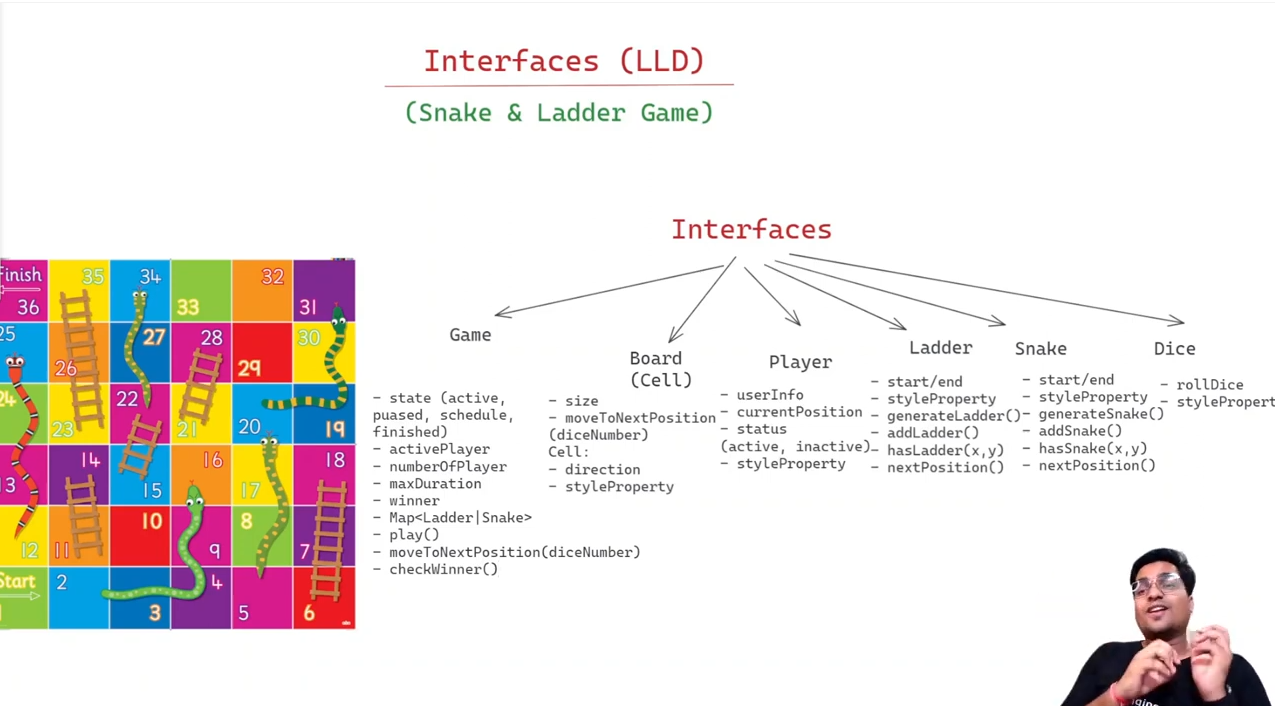
****

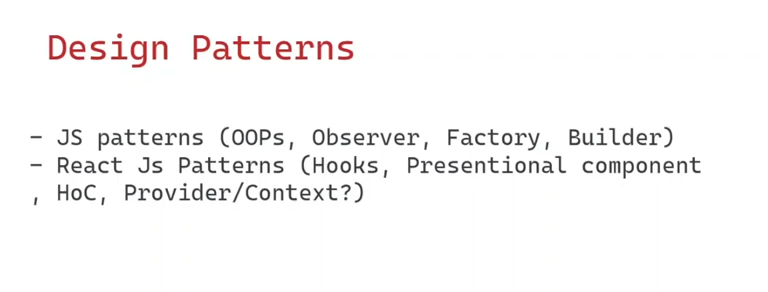
**Low**

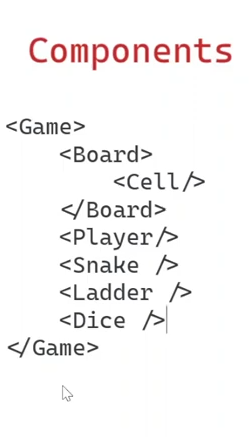
**Low level Design**

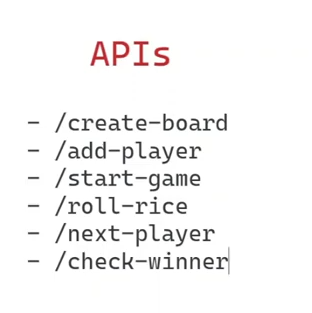
**Interfaces**

****

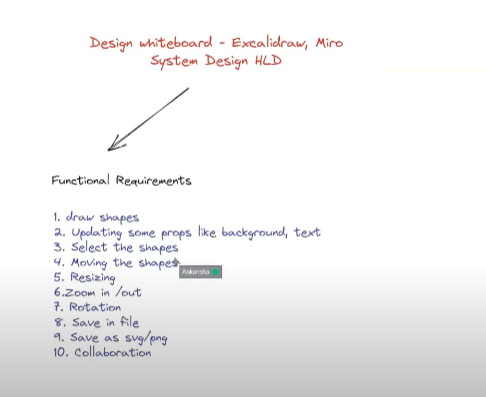
****

****

****

****

**2.Design White Boards**

****

**Design low Level Deisgn of Tic Tac Toe**

**Undo functions**

**Timed**

**Spectate**

**Statistics**

**Tournment**

**Ai**

**Rating Change**

**Game Board**

**2 d matrix 🡪0,1,empty int [i][j]**

**Winner – “First”**

**“Second”**

**“Draw”**

**“Undergo”**

**Behavior the board currently**

**Intiatlise()**

**getBaord()**

**getWinner()**

**getCurrentPlayer()**

**makeMove(Move m)—int player, int I, intj**

**User – user id and statistics**

**Game –**

**Game id**

**UserId1, 2**

**List<Moves>**

**Code for Tic Tac toe**

**Public class TicTacToe{**

**Private final int [][] board;**

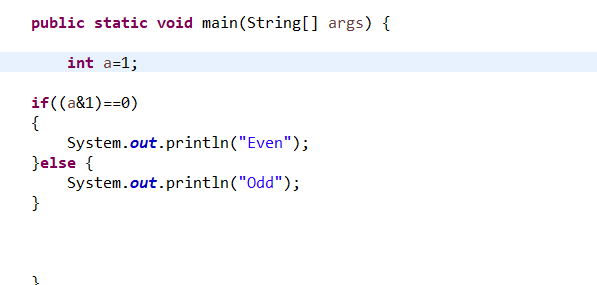
**Public TicTactoe(int n)**

**}**

**Coding By youtube**

**Bitwise operations**

**Even odd Program using the bitwise operations**

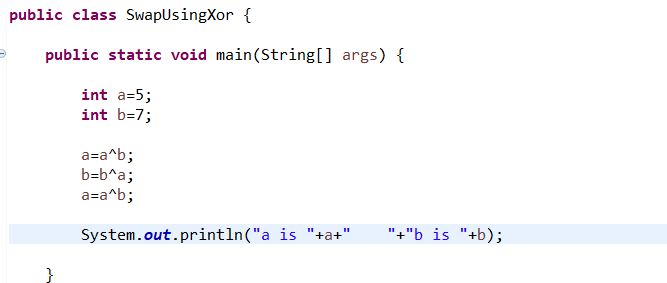
****

**If(a and 1==0🡪even else odd)**

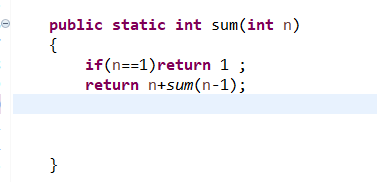
**2.Swap Two Number using XOR**

**Property of XOR Is**

**When both number are same then it gives o otherwise 1**

****

**Sum of n number using Recusrion**

****