SPRING BOOT

Benefits of Spring Boot Over Spring

1. Dependency resolution.
2. Minimum Configuration.
3. Embedded Server for testing.
4. Bean auto Scan.
5. Health matrics.

Starter POM

- Meta-INF/Spring.factories

1. Enable 2. Disable

Based on @ Conditional @Configuration

Run() Internal Flow

- Create Application Context
- Check Application Type
- Register the annotated class beans with the Context
- Creates an instance of TomcatEmbeddedServletContainer and adds the Context

MICROSERVICES Architecture Spring Boot

28/5/24

Java 8 Features

* Lamda Expressions.
* Method references.
* Functional interfaces
* Stream API
* Default methods.
* Base64 Encode Decode.
* Static methods in interface
* optional class
* CollectionClass
* For.Each()
* Nashorn JavaScript Engine.
* Parallel Array Sorting
* Type and repeating Annotations.
* IO Enhancements
* Concurrency Enhancements.
* JDBC Enhancements etc.

1. Lamda Expression.

It provides clear & Concise way to implement SAM interface (Single Abstract Method) by using an expression. It provides implementation of functional interface (interface which has only one abstract method but can have multiple default and static method)ES

2. Method Reference.

1. Static Method Reference
2. Reference to an instance method.
3. Reference to a Constructor.

@Transient - For Calculating dynamically
Private Integer age;

16-6-24

Spring & Spring boot Annotations.

1. @Component

Indicates the annotated class is a "Spring bean / Container. Component. This annotation tells Spring Container to automatically create Spring bean.

2. @Autowired

This used to inject the bean automatically. This annotation is used in Constructor injection, Setter injection and field injection.

3. @Primary

This is give higher preference to a bean when there are multiple beans of the same type.

```
Class ~~Section~~ @Primary
                  @Component
Class Suzuki implements Car
Class Honda implements Car
```

Primary annotation creating bean for Suzuki.

4. @Qualifier

this annotation give the preference to which Class to create bean

```
@Component
Class Suzuki implements Car

@Component
Class Honda implements Car

main() {@autowired @Qualifier("honda")
        Car car;
```

Create bean for honda

}

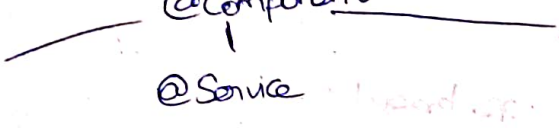
5. @Configuration.

6. @Bean - Explicitly creating object

```
public class Config {
    @Bean
    public Pizza vegPizza(){
        return new VegPizza();
    }
}
```

@Bean (name = "vegBean") - we able to give name to bean by name attribute

7. Stereotype Annotations



we able to create bean with this annotation. These Comp annotations are derived from @Component.

8 @Lazy

* By default Spring creates all singleton because eagerly at the startup / bootstraping of application

*Using @Lazy can load bean lazily (on demand)

* @ Lazy can used with @ Configuration, @ Component, @ Bean.

9 @ Scope

1. @ Singleton.
2. @ prototype
3. request.
4. Session.
5. application.
6. websocket.

* Last four annotations are only available with in a web aware application.

@Scope can use with @Component or @Bean

```
@Component
@Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class PrototypeClass() {

}
```

Spring Create Singleton Scope explicity

10. @value

```
@value("(")
private String openbracket;
```

11. @ propertySource and @propertySources

Those annotations is used with @Configuration classes

```
@Configuration
@PropertySource("Classpath: messages.properties")
Public class Config {

}
```

```
@Component
public class PropertySourceDemo{
  @value("${message}")
  private String message;
```

```
@PropertySources({
    @PropertySource("Classpath: message.properties"),
    @PropatySource("Classpath: label.properties")
    })
```

@ RestController

Spring 4.0 introduced this annotations.

@ ResponseEntity<ClassName>

@ ResponseStatus (value = HttpStatus.IM_USED)

Java Guides - 2:05:45

16-06-21

Java Guides - Spring boot Templates

URI Template Variable

```
@GetMapping(value = "/book/update/{id}")
Public Book getBook(int id){
} return book;
```

```
@RequestBody
(@PathVariable int id, Book book)
```

@SpringBootApplication internally Contains @SpringBootConfiguration, @EnableAutoConfiguration + @ComponentScan + @Configuration.

Types of Configuration

1. ~~@Conf~~ Java based
2. XML Based
3. Annotation based.

24-06-24.

FreeCodeCamp - Spring Boot & Spring Data JPA - 13 hrs

@Autowired on constructor is optional if there is only one Constructor.

Bean Scope :-

- Singleton
- Prototype
- Request
- Session
- Application
- WebSocket

29/6/24

Hibernate - Telugu

ORM Framework - Object Relational Mapping

ORM Tools

Hibernate, IBatis, Toplink

States

1. Transient
2. Persistant
3. Detached
4. Removed

Session.Save() vs Session.Load()

Session.load - proxy object(doesn't hit db is the result is not used in Java after fetching)

Save() - It will hit db everytime it doesn't care the result is used or not after fetch.

30/6/24

Spring boot Interview Questions - Java Techie part-1

Spring MVC

We need to create <datasource> object, SessionFactory, then inject datasource to session factory, hibernate properties, transaction Manager inject session factory to transaction Manager

1. Dependency Resolution/Avoid version Conflict
2. Avoid additional Configuration.
3. Embed Tomcat, Jetty (no need to deploy WAR file
4. Provid Production-ready features such as metrics, health checks

2. What all Spring boot Starter you have used / what are all modules you have worked on.

- Spring boot Starter web.
- Spring boot Starter data JPA
- * Spring boot Starter AOP.
- * Spring boot Starter Web Services
- * Spring boot Starter Security.
- * Spring boot Starter for Apache Kafka.
- * Spring boot Starter for Spring Cloud.
- * Spring boot Starter-thymleaf.

3. How will you run your Spring Boot Application?

plugin - Run from Terminal

Project path projectName %. mvn Spring-boot:run

4. What is the purpose of the @SpringBootApplication annotation?

@SpringBootApplication is Combination of

@EnableAutoConfiguration.

@ ComponentScan

@ Configuration.

@ComponentScan - Basically it couldn't load class other than main class packages. we explict to load if we doesn't maintain package hierarchy

@SpringBootApplication (ScanBasePackages = "com. businessCommon"

@ Enable AutoConfiguration

Application.properties - debug = true

7. How can you disable a specific auto-configuration class in spring boot?

@SpringBootApplication(exclude={classname.class})

We can also exclude in properties file.

Spring.autoConfigure.exclude = org.springfw... etc
Complete package with classname.

6. How can you customize the default configuration in Spring Boot

application.yml or application.properties
Default-8080

```
Server:
  port: 8090
```

9 How Spring boot run() method work internally

* Create Application Context
* Register bean in to Context
* Kicked up embedded tomcat Container to run your job

Inside Run()

prepareEnvironment() what environment contains all properties files .prop, .yml - OriginTrackedMapPropertySource

16. What is Command line runner in Spring boot

SOLID principles

S - Single Responsibility principles.

O - Open & closed principle.

L - Leskov's Substitution principle

I - Interface Segregation principle.

D - Dependency Inversion principle.

1. Single Responsibility principles - SRP

The Single responsibility principles states that every java class must perform a single functionality. Implements of multiple functionalities in a single class mashup the code.

2. Open - closed principles [OCP]

This states that according to new requirements the module should be open for extension but closed for modification.

3. Liskov Substitution principles LSP

It applies to inheritance in such a way that the derived classes must be completely substitutable for their base classes. In other words, if class A is a subtype of a class B, then we should be able to replace B with A without interrupting the behaviour of the program

4. Interface Segregation principles.

The principle states that the larger interfaces split into to smaller ones. Because the implementation classes use only the methods that are required. we should not force the client to use the methods that they donot want to use.

5. Dependency Inversion principle.

This principles states that we must use abstraction (abstract classes and interfaces) instead of concrete implementations. High level modules should not depend on the low-level module but both should depend on the abstraction.

18-7-24

Web Socket

It is a Computer Communication protocol, Providing full duplex communication channels over a single TCP connection. BI Directional Communication.

Two methods to initiate the web socket connecting using HTTP

* ws Method
* wss Methods - with encryption

20/7/24.

Web Socket

STOMP - Streaming Text oriented Messaging protocol.

Stomp is a simple text based protocol, it allows STOMP clients to talk with any message broker supporting the protocol

* Spring provides default support for it but you are open to choose any other messaging protocol such as RabbitMQ or ActiveMQ.

21-7-24.

Time Complexity.

Big-oH - O - Worst Case time Complexity.

O - Order of

1) Big Notations

i) Big oh Notation -> worst case, upper bound,

ii) Big Ω-omega Notation -> Best Case, lower bound.

iii) Big Θ notation - average case, both upper & lower bound.

ex-

Search for element - n=9

[1, 5, 7, 8, 9] - worst case O(n) - Size of array

n=1

[1, 5, 7, 8, 9] - Ω(1) - best case - Minimum time

Common Time Complexity

1) O(1) - Constant - N=10

2) O(Log N) - logarithmic - $Log_2 10 = 4$

3) O(N) - Linear - N=10

4) O(Nlogn) - Linearithmic - NXLogN $10 \times 4 = 40$

5) $O(N^2)$ - Quadratic. - $10^2 = 100$

6) $O(N^3)$ - Cubic - $10^3 = 1000$

7) $O(2^n)$ - exponential - $2^{10} = 1024$

8) O(n!) - factorial. - 10! - 3628800

9. $O(\sqrt{n})$

Space Complexity

The term Space Complexity is misused for auxiliary space at many place. Following are the Correct definition of Auxiliary Space and Space Complexity.

Auxilary Space is the extra space or temporary space used by an algorithm.

The space complexity of an algorithm is the total space taken by algorithm with respect to the input size. Space complexity includes both auxiliary space and space used by input.

Input Space + Auxiliary Space

```
int a,b;
get input for 2 vars - 2
c=a+b  - 1
  O(3) - Constant Space
```

Constraints

$\leq [10]$ - $O(n!)$

$\leq [20]$ - $O(2^n \times n^2)$

< 100 - $O(n^4)$

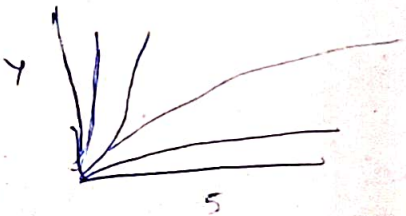
< 400 - $O(n^3)$

< 2000 - $O(n^2 \times \log n)$

$< 10^4$ - $O(n^2)$

$< 10^6$ - $O(n \log n)$

$< 10^8$ - $O(n)$, $O(\log n)$



Hashing

HashTable

Key	value
21	Madurai
32	Thrichy
43	Combatore
54	Chennai
62	Bangalore

Hashing

To store the above data as key.value pair but the key doesn't exceed 10, So use some hash functions(mathematical calculations) ex - %.10

	key	value
21%.10 =	1	Madurai
32%.10	2	thirchy
43%.10	3	Combatore
54%.10	4	Chennai
62%.10	2	- Bangalore
	1	

Duplicate key occur called Collisions

So we use strong hash functions.

22/07/24

DSA

1. Strivers A2Z DSA Course by Take U forward.
2. DSA by Abdul Bari

1. Strivers DSA Course

* Space complexity

  var a, b - input space. O(1) — O(3)

  c = a+b - auxiliary space O(2)

  int a[n] - O(n)

  1s = 10^8 operations - Time Complexity: Most of the servers take 10^8 operations in 1 second. eg - LeetCode, HackerRank, etc.

* patterns ✓

* Basic Maths for DSA

  N = 7789 → 7789 % 10 = 9 (1) — extraction of digits
  778 % 10 = 8 (2)
  77 % 10 = 7 (3)
  7 % 10 = 7 (4)
  0

1. Find count of digits using ↑ : len = 4
2. log10(n) = log10(7789) = 3.891... (int)(log10(n)) + 1 = Count
   int(log10(7789)) + 1 = 4

Time Complexity O(log10(n))

1) ReverseNum();
2) palindrome();
3) ArmStrong()

23-7-24

Recursion :- ✓

24-7-24

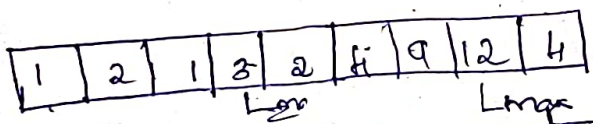
Hashing

* Pre storing & fetching something

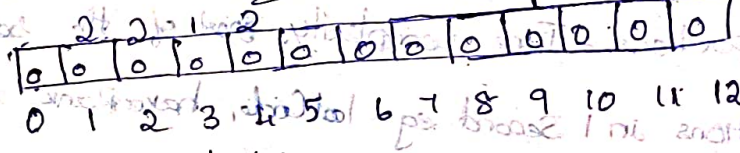
Ex - Arr - | 1 | 2 | 1 | 3 | 2 | - To find occurance
Lmax

1 - 2 times
2 - 2 times
3 - Times For this we have to loop n time O(n)

Hasing
n =



Ex Hash Arr



Pre Compute / Calculation.

For Ex ASCII value Max - 256

Hashing - 1) Division Method
2) Folding method.
3) Mid Square method.

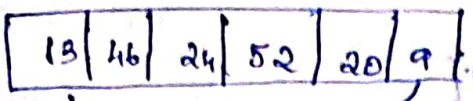
25-7-24

Sorting :-

1. Selection Sort
2. Bubble Sort
3. Insertion Sort.

1. Selection Sort.

Select min & Swap



- asc

9, 46, 24, 52, 20, 13

9, 13, 24, 52, 20, 46

9, 13, 20, 52, 24, 46

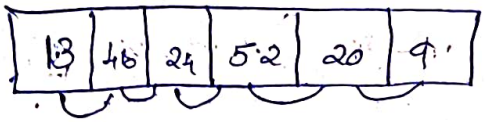
.....

TC: $n+n-1+n-2+n-3$ ---- Natural numbers 1, 2, 3, 4

$= \frac{n \times (n+1)}{2}$

$= \frac{n^2+n}{2} \quad 2 \quad = O(n^2)$

2. Bubble Sort - Max to last by adjacent Swaps



- b

13, 24, 46, 52, 20, 9 13, 24, 46, 20, 9 13, 24, 20, 46, 9

$6 < 6-6-1$

13, 24, 20, 9, 1

Tc $= n+n-1+n-2-\ldots$

Natural num - $\frac{n \times (n+1)}{2} = \frac{n^2+n}{2} = n^2$ $O(n^2)$

remove constant

13, 24, 46, 20, 9, 50

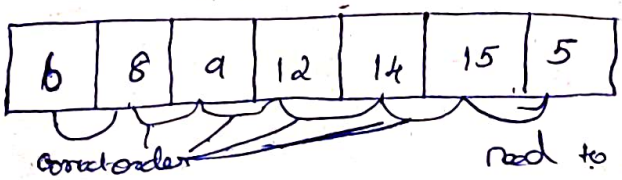
13, 24, 20, 9, 46, 50

worst case - $O(n^2)$

best case - $O(n)$

3. Insertion Sort.

Takes an element & place it in its correct position



need to swap as 5, 15 and 5 with other element before

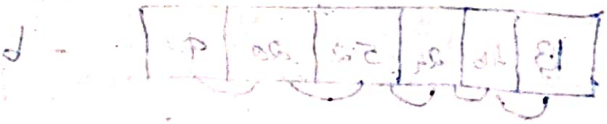
arr = 7, 2, 5, 3, 21, 8, 11, 1

```
int j, temt;
for(int i=0; i<=n-1; i++){
  j=i;
  while (j>0 && arr[j] < arr[j-1]){
    temp = arr[j];
    arr[j-1] = arr[j] = arr[j-1];
    arr[j-1] = temp;
  }
}
```

TC - $n \times \frac{(n+1)}{2} = O(n^2)$ - worst case

$O(n^2)$ - Avg case

$O(n)$ + best case if arr already is sorted.



26-7-24

Merge Sort :=

algo = Divide & Merge

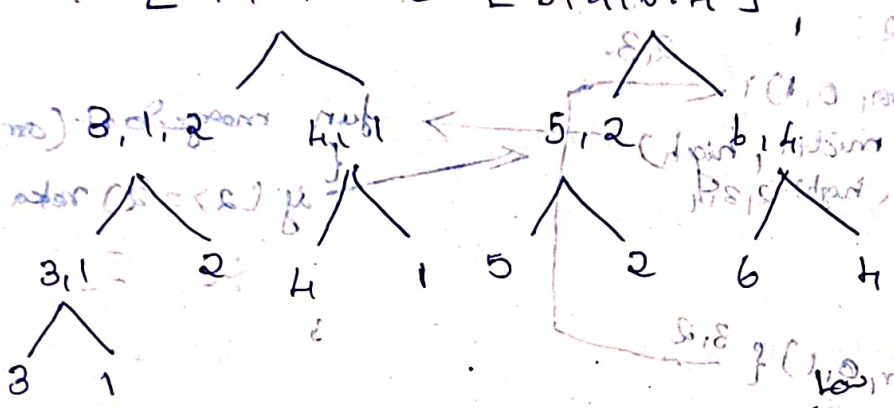
[3, 1, 2, 4, 1, 5, 2, 6, 4] / 2

5 or 4

4 or 5

[3, 1, 2, 4, 1, 5, 2, 6, 4]

[3, 1, 2, 4, 1] [5, 2, 6, 4]



arr = [10, 2, 5, 3, 9]

```
mergeSort(arr, low, high)
                    0+5/2 = 3
    mid = (low + high) / 2;
    mergeSort(arr, low, mid);
    mergeSort(arr, mid, high);
    merge(arr, low, mid, high);
```

arr = [3 | 2 | 4 | 1 | 3] - mergeSort(arr, 0, arr.length-1)

low high 3,2,4,1,3
fun mergeSort(arr, 0, 4){
if(low >= high) return; - Base case to stop recursion.
mid = (low+high)/2; 2
mergeSort(arr, low, mid);
mergeSort(arr, mid+1, high);
Sort(arr, low, mid, high)
2,3,4,1,3
0 1 2 3 4

2,3,4

1,3

1,3

fun mergeSort(arr, 0, 2){ 3,2,4
if(low >= high) return;
mid = (0+2)/2; 1
mergeSort(arr, 0, 1);
mergeSort(arr, mid+1, high)
Sort(arr, low, mid, high) 2,3,4

2,3.

fun mergeSort(arr,
if(2>=2) reta

3,2

3;2

fun mergeSort(arr, 0, 1){ 3,2
if(low >= high) return;
mid = (0+1)/2; = 0
mergeSort(arr, 0, 0);
mergeSort(arr, mid+1, high);
Sort(low, mid, high) = 2,3

func mergeSort(arr, 0, 0){
if(0>=0) return;
mergeSort(arr)
merge();
Sort();
}

fun mergeSort(arr, mid+1, high){
(low)
1>=1
if(low >= high) return;

}

⇒ fun mergeSort(arr, 3, 4){ 1,3

if(3>=4) return; mid=(3+4)/2; 3

mergeSort(arr, low, mid)

mergeSort(arr, mid+1, high);

merge(arr, low, mid, high);

}

fun mergeSort(arr, 3, 3){

if(3>=3) return;

...

}

2,2)

fun mergeSort(arr, 4, 4){

if(4>=4) ret;

...

}

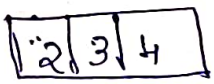
[2,3,4,1,3] 0 2 4

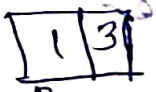
fun Sort(arr, low, mid, high){ TC = O(n) worst case

temarr = [ ]; or list or etc

int left = low;

int right = mid+1;





while(left<=mid && right<=high){

if(left<right)

temarr.add(arr[left]);

else

temarr.add(arr[right]);

}

2,3,4 1,3
l r

temp=1

2,3,4 1,3
l r

temarr=1,2

2,3,4 1,3
l r

tar=1,2,3

2,3,4
l

arr = 4, 6, 5, 7, 3, 9, 2

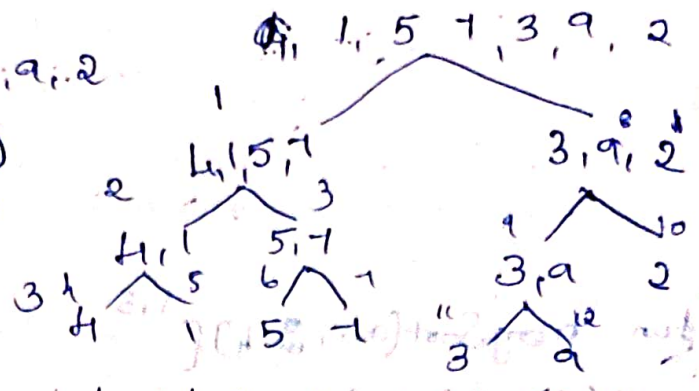
ms(arr, 0, len-1)

mid = (0+6)/2 = 3

ms(arr, 0, 3) {

(0+3)/2 = 1

} ms(arr, 0, 1)



dividing by 2

$\log_2 n$, 2^7

Sort merge

TC = $O(N * \log_2 n)$, $O(N \log N)$

SC = $O(N)$

Quick Sort :- DAC Divide & Conquer algo

int arr = [4, 6, 2, 5, 7, 9, 1, 3] ; TC = $O(N \log^N)$

1 2 3 4 5 6 7 8 SC = O(1)

1. Pick a pivot & place it at its Correct order in the sorted arr.

↓

pick any element

2. Smaller on left & larger on right

ex: [1] [4 6 5]

a. 1st elm on the arr

b. ~~last elm~~ on last or elm

c. median elm

d. random elm.

4, 6, 2, 5, 7, 9, 1, 3

low high.

4 > 4 ✗

6 > 4 ✓ - move low to 6

Find Smaller than right

3 < 6 ✓

1 < 3 ✓

move high to 1 & Continue Same.

6 > 4 ✗

5 > 4 ✓

move low to 5

0 1 2 3 4 5 6 7

arr = 4, 6, 2, 5, 7, 9, 1, 3

4 > 4 ×

6 > 4 ✓

3 < 4 ✓

4, 3, 2, 5, 7, 9, 1, 6

3 > 3 ×

2 > 3 ×

5 > 4 ✓

6 < 4 ✓

Swap

4, 3, 2, 1, 7, 9, 5, 6

4, 3, 2, 1, 7, 9, 5, 6

4, 3, 2, 1, 7, 9, 5, 6

1, 3, 2, 4, 7, 9, 5, 6

left smaller, right larger

list = {4, 6, 2, 5, 7, 9, 1, 3}

0 7

QS (arrlist, low, high)

```
{ if (0<7){
int pIndex= partition(arr,low,high);
QS(arr, low, pIndex-1);
QS(arr, pIndex+1, high);
}
```

fun partition (arr, low, high) {

```
pivot = arr[low];
i = low;
j = high;
```

```
while (i<j){                          i j   arr{4, 6, 2, 5, 7, 9, 1, 3}

  while (arr[i] <= pivot && i <= high - 1){    4<=4 ✓
       i++;  1, 3,                              6<=4 ✗
  }
  while (arr[j] > pivot && j >= low + 1){
  }  j--;                                       3>4 ✗
                                                9>4 ✓
  if (i<j){
    temp = arr.get(i);
    arr.get(j) = t
    arr.set(i, arr.get(j));
    arr.set(j, temp);
  }
}
```

1st 1<7

4, 3, 2, 5, 7, 9, 1, 6

6>7 ✗

3<7 ✓ swap

~~4, 3, 2, 6, 7, 9, 1, 5~~

4, 3, 2, 1, 7, 9, 5, 6

4, 3, 2, 1, 7, 9, 5, 6

---

p=4 i=0

4, 6, 2, 5, 7, 9, 1, 3

i=1 4<=4 ✓ 3>4 ✗ j=7 4, 3, 2, 5, 7, 9, 1, 6

6<=4 ✗

1<7 ✓

i=2 3<4 ✓ 6>4 ✓ j=6

i=3 2<4 ✓ 1>4 ✗

i=4 5<4 ✗

3<6

4, 3, 2, 1, 7, 9, 5, 6

i=4 1<4 ✓ 5>4 ✓ j=5

7<4 ✗ 9>4 ✓ j=4

7>4 ✓ j=3

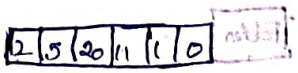
1>4 ✗

28-07-24.

DS

* Array.

To store Collection of items of same data types in Continous memory

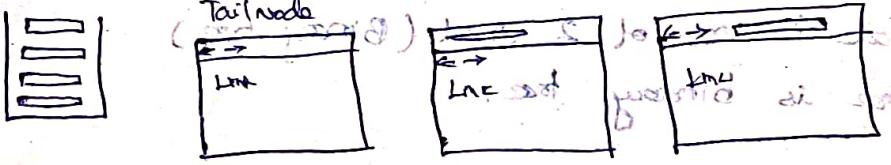


* Linked List

Series of nodes linked to each other.

* Stack

FILO or LIFO

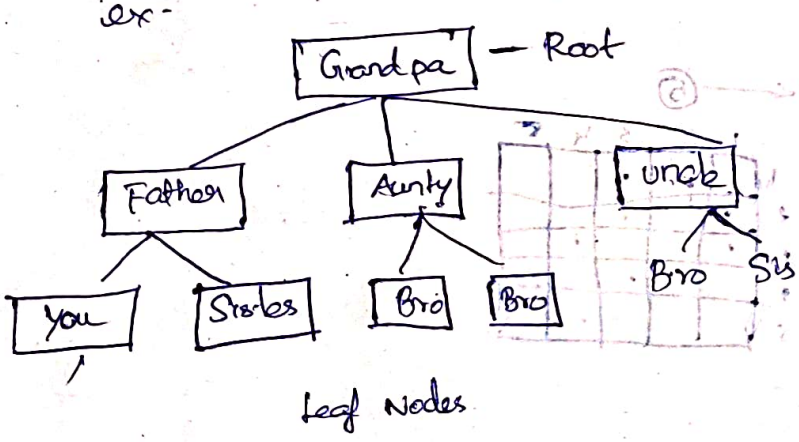


* Queue

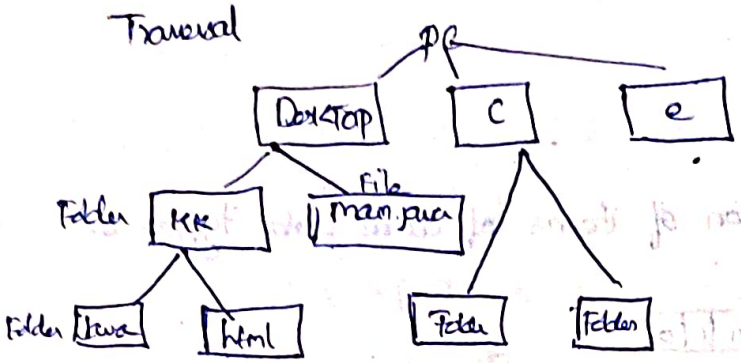
FIFO, Implelated using array, linked list

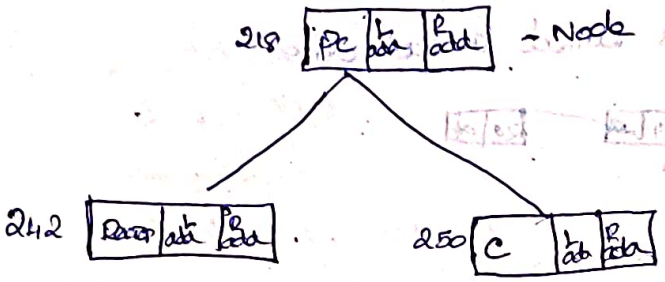
* Tree & Graph.

ex-



Traversal





Each node has max of 2 child (Binary tree)

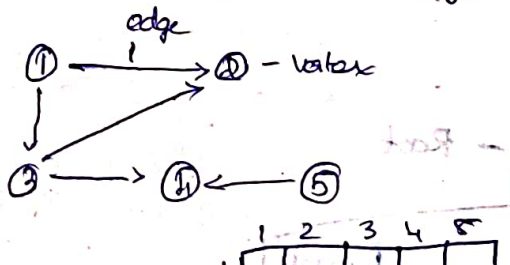
One of the tree is binary tree.

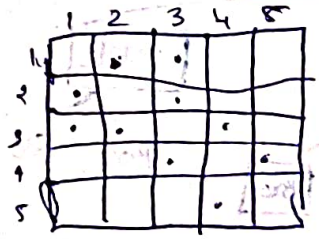
* Graph

Vertex, Edges

* Vertex like location / node / location
* Edges like road / path.

* Assume 5 nodes / vertex * Edges 1 2, 3 4, 5 4, 3 2, 1 3





12-10-24

Telusko - Java for programmers

* REPL - Read - eval - print loop (like write one line of code to begin).

2 Java is statically Typed Language eg (int num, String name).

3 AutoBoxing & unBoxing

* AutoBoxing

The automatic Conversion of a primitive type to its Corresponding wrapper class. ex - an int can be automatically converted to an integer

* unBoxing - Vice versa

4. Changing the value of non-static variable in the static methods - exception Compile time - "Cannot make a static reference to the non static field var"

Cannot use "this" keyword in static context

5. Types of TypeCasting

1. Implicit casting (widening Conversion)
2. Explicit Casting (Narrowing Conversion)

Implicit Casting

* Smaller datatype is automatically converted to a larger datatype.

* No Risk of data loss

```
int i = 100;
long l = i;
```

Explicit Casting

* larger data type to smaller data type.
* Risk of potential data loss

ex

double to float

long to int.

```
double d = 12.55;
int i = (int) d;

long l = 1000L;
int i = (int) l;
```

6. Upcasting & Downcasting

Upcasting

process of converting a subclass reference to a Superclass reference.

```
Animal animal = new Dog();
```

Down Casting

```
Dog myDog = (Dog) Animal;
```

7. Atomic Integer - Java.util.Concurrent.atomic

It is call class that represents an integer value that may be updated automically. Operations on an atomic integer are thread safe without need for explicity Synchronization for Concurrent programing.

* Constructor:

AtomicInteger(), AtomicInteger(int initialvalue)

* Basic Operations:

int get() - Gets current val

Void Set(int val)

int getAndSet(int val) Sets the new val and return old val

* Increment & Decrement

int incrementAndGet(): Atomical increse the curr val & return new val

Int decrementAndGet() -'' decrease by one

Int addAndGet(int data): "A adds the specified val and returns new val

* Compare and swap

boolean CompareAndSet(int expect, int update)

8. How you find memory leaks and steps to fix memory leaks

Tools

Visualvm, yourkit, or JProfiler

Memory Dumps

generate heap dump using tools like jmap

Garbage Collection Logs:

Enabling GC logging

9. Changes made in Java Concurrency model introduced in Java 8

* Fork / Join Framework - (Java 7)

It simplifies parallel programming by allowing task to be split in to smaller subtask recursively. It leverages the work stealing algorithm, improves performance in multicore system.

* CompletableFuture (Java 8)

It provides more powerful way to work with asynchronous programming. It supports non-blocking asynchronous & allows for Compositions of multiple futures.

* Streams API

It allows parallel processing of Collections through parallelStream() method. It makes easier to perform parallel ops without manage threads explicitly.

17-10-25

1. Class Main {
   A obj : new A()
   print(A); main.A@15dc9731;
}

prints to string method by default from Object class

2 Optional Class : java.util

To avoid nullpointerException.

of (T val) - returns an optional with non null value, throws nullPointerException. If value is null

ofNullable(T val) - Return an optional describing the value, or an empty optional if the value is null.

isPresent() - Returns true if value present otherwise false.

get() - Retrain the value if present, other throws NoSuchElementException

ifPresent(Consumer<? super T> action):

If a value is present, performs given action

orElse(T oth) - Returns the value if present, otherwise, returns other.

orElseGet (Supplier<? extends x> exceptionSupplier).

If value presents it returns otherwise it throws an exception created by the provided Supplier.

3. Date and Time API

Java.time, java.util, java.sql & java.text

Drawbacks of existing API

1. Thread safety.

existing Date and Calendar does not provide thread safety, leads to concurrency issues.

New Date & Time is immutable & thread safe.

2 Bad API design

existing Date & Calendar class not provide methods to perform basic day to day functionalities.

Java.time.LocalDate - year-month-date.
- LocalTime
- LocalDateTime - Date and time without timezone
- ZonedDateTime
- OffsetTime
- OffsetDateTime
- Clock
- Instant

java.util.Date, java.util.Calendar	java.time
Mutable	Immutable
Limited with timezone or Calendar	ZonedDateTime, OffsetDateTime
Non-thread safe	Thread safe
Combines date and time in a single class	Date and time are separated class
Limited built in formatting	Extensive support for formatting with DateTimeFormatter
Poor support for adding & subtracting	

3. Iterator and Spliterator ?

Java.util	Java.Util
Iterator	Spliterator
• Introduced in Java 1.2	Introduced in Java SE 8
• used for Collection API	It is used for Stream API
• next(), hasNext() etc	tryAdvance()
• Iterates only in sequential Order	• Iterates in parallel & Sequential order.

* Spliterator :- Interface

This interface is designed for traversing & partitioning elements of a source. It is used for parallel processing, enabling to multiple threads

* parallel processing :-

Supports effecient parallel traversal of data structures, making it suitable for use with Java Streams API.

* Try Split Method :-

Allows for the splitting of a data source in to two parts for parallel processing.

* Characterization - :- Ordered, SIZED, SUBSIZED

```
List<String> list = Arrays.asList("A", "B", "C", "D", "E");
Spliterator<String> spliterator = list.spliterator();
Spliterator<String> split = spliterator.trySplit();
split.forEachRemaining(System.out::println);   C, D, E
spliterator.forEachRemaining(System.out::println);   A, B
```

* Try Advance CD

18/10/24

Re-enhant - Java.util.Concurrent

A reanhant lock, also known as "recursive" lock, used in multithreading to allow a thread to acquire the same lock without causing a dead lock.

Recursive Acquisition.

- A thread can acquire the lock multiple times, Each acquisition must be matched by a acosponding release

Ownership Tracking.

The lock keeps track of which thread currently hold it and how many times it has been acquired by that thread

Only the owning thread can release the lock.

Prevent Deadlocks:

Fairness:

Some implements can provide fairness, ensuring that threads acquire the lock in the order they requested it

It is like Synchronized

Static

Synchronized(this){ lock.lock() Implicit

}

lock.unlock();

Only the method or block will implicit

* Locks are Explicit

We can lock in one method and can release in another method

lock.tryLock()

lock.tryLock()

Static void accessResource(){

lock.lock()

lock.lock()

int num = lock.getHoldCount()

if(){

accessResource();

}

lock.unlock();

lock.unlock();

}

lock l = new ReentrantLock(true) - By default false

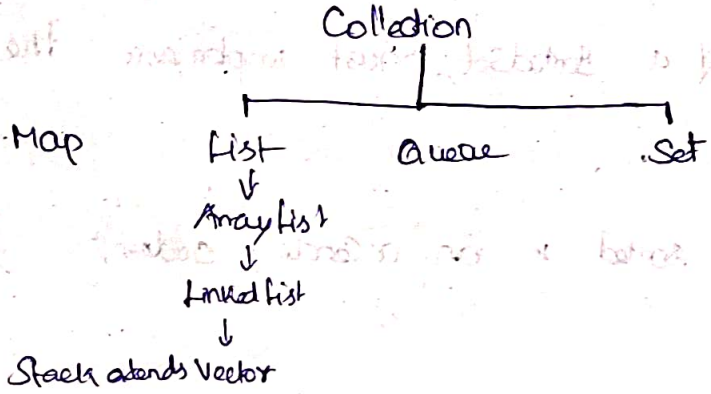
↓

Maintains Order

	Adv	Disadvantage
Fair	Equal chance for all thread	Slower
unFair	Faster (more throughput)	possible thread starvation

```
boolean lockAq = lock.tryLock(5, Seconds);
                                  ↳ act as unbar
if (lockAq)
    // Some task
```

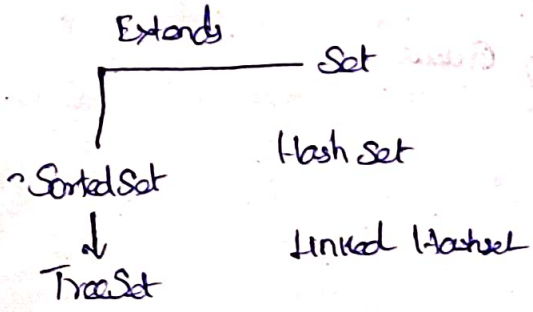
Collections



Vector

* It is synchronized
* Maintains the insertion order
* Thread safe
* Vector increasing its size by doubling the array size.
* It's a legacy class.

Set



HashSet

* It implicitly implements a hashtable
* Contains only unique elements
* only one null element can be added
* It unordered as set

Linked Hashset

* Ordered version of Hashset which maintains a doubly-linked list across all elements.

* It preserves the insertion order.

* Sorted Set

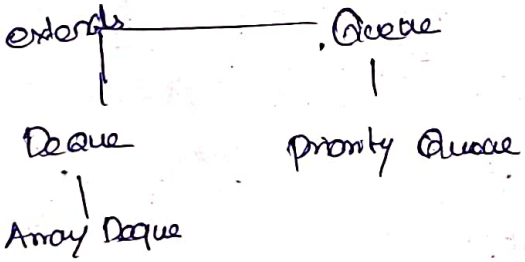
* All element of a SortedSet must implements the Comparable interface.

* It's a set sorted in an ascending order.

* TreeSet

* Uses a Tree for storage (Self-balancing binary Search tree like Red-black Tree)

* Object is a Treeset are stored in a sorted and ascending order.



* Priority Que

It is Que with priority associated with each element.

* High priority element is served before a low priority element irrespective of their insertion.

* DeQue Double Ended Queue

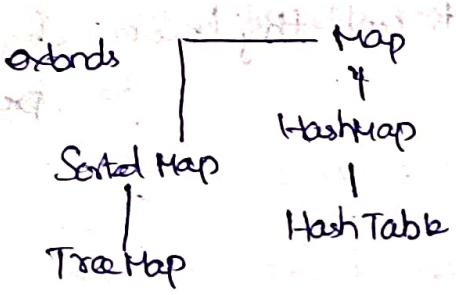
* Elements can be added and removed from either end.

* Array DeQue

* Way to apply resizable array in addition to implementation of the deque interface.

* NO Capacity restrictions

* Map Interface does not extends Collection



4. BlockingQueue - java.util.blockingQueue

It supports operation that wait for the queue to become non-empty when retrieving an element and wait for space to become available in the queue when storing an ele particularly used in producer - Consumer.

Blocking ops

Retrieval - If a thread tries to take an element from an empty Queue, it wait until an element becomes available.

Insertion - If a thread tries to add element to a full Queue, it will wait until space available.

Capacity - It can be bounded with a fixed capacity or unbounded (with no fixed limit)

* ArrayBlocking - FIFO
* LinkedBlockingQueue - A linked queue impl. both bounded or unbounded. FIFO
* PriorityBlockingQueue - orders elements according to natural order
* DelayQueue -- Holds elements until they become eligible for processing.

5. Difference between Synchronized Collection & Concurrent Collection

* Both provides thread safety.
*

Synchronized	Concurrent
* Utility Collections.SynchronizedList() Collections.SynchronizedMap()	* ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue etc.
* entire segment will lock will leads to slower perf	* Single segment only be locked
* explicit synchronization	* implicit synchronization

6. Is a & Has a Relationship

↓

Inheritance

extends

Association

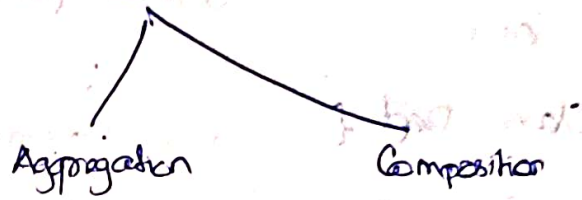
refer var, new

No-blood relation

not tightly Coupled

tightly Coupled

```
Class car {
  speed();  -> if we changing something it
}                will impact here.
Class SportsCar {
  speed();
}
```



Aggregation, association, Composition

1. Association

It is a general relationship 2 Classes. It can be one-to-one, one to many, many to many. The objects have their own life cycle and can exists independently

```
Class P {
    Str name;
  public P (name) {
      this.name = name;
  }

Class C {
    Str Street;

  public C (Str Str) {
  }
}
```

```
Class emp {
    pvt P p;
    pvt C c;

  public emp(P p, C c) {
      this.p = p;
      This.c = c;
  }
```

2. Aggregation - Weak Relationship

Specialized form of association that represents "whole part" i.e. the part can exist independently of the whole. This means the life cycle of the part and whole are not highly coupled.

ex-1

```
Class Dept {
    ....
}

Class Company {
    Name
    Put list<Dept> dept;
}
```

ex-2

```
Class Team {
    List<Player> players;
}

Class Player {
}
```

ex-3

```
Class Car {
    List<Passenger> ~~Engine~~;
}
Class Passenger {
}
```

3. Composition - Strong Relationship

Both objects cannot exists independently. one obj cannot exists without owner object.

```
public Class Car {
    Private Engine engine;
}
```

1. CompletableFuture - class - Java.util.Concurrent - Java 8

It allows us to perform non blocking Computations.. you can run tasks asynchronously and continue with other operations without waiting for task to complete.

Chain of operations

1. thenApply(), thenAccept(), thenCombine()

Completing Stage - Manual completion.

1. Completed Normally, Completed exceptionally, not Completed

Error Handling

1. handle(), exceptionally(), whenComplete()

Combining future

1. allOf(), anyOf()

2/11/24

1. Types of Innerclass ?

* Non static inner class (Member class)

It is defined within another class, and can access all members (including private members) of the outer class.

To create an instance of non-static inner class, you need an instance of outer class.

```
OuterClass outer = new OuterClass();
OuterClass.InnerClass inner = outer.new InnerClass();
```

* Static Inner class

It cannot access non static members of the outer class directly, it can only access static members.

A static inner class can be instantiated without needing an instance of the outer class

* Local Inner class

It is defined within a method or a block and can access local variables (final or effectively final) as well as the outer class members.

The scope of a local inner class is limited to the method or block in which it is defined

4/11/24

1. Executor Service

Types of thread pools.

1. Fixed Threadpool - Internally maintains blocking queue.
2. Cached Threadpool
3. ScheduledThreadpool - Delay Queue
4. SingleThreadedPool

Constructor and LifeCycle methods:

```
public static ExecutorService newFixedThreadpool(int nThreads){
    return new ThreadPoolExecutor(nThreads, nThreads, 0L, TimeUnit.MILLISECONDS,
                                  new LinkedBlockingQueue<Runnable>());
}

public ThreadPoolExecutor(int corePoolSize, int maximumPoolSize,
                          long keepAliveTime, TimeUnit unit,
                          BlockingQueue<Runnable> workQueue,
                          ThreadFactory factory, RejectedExecutionHandler
                                                 handler);
```

Param	FixedThreadPool	CachedThreadpool	ScheduledThreadpool	SingleThreaded
CorePoolSize	Constructor-arg	0	Constructor-arg	1
maxPoolSize	Same as corepoolsize	Integer.MAX_VALUE	Integer.MAX_VALUE	1
KeepAliveTime	0 Seconds	60 Seconds	60 Seconds	0

Pool	Queue Type	Why?
• FixedThreadPool	LinkedBlockingQueue	Threads are limited, thus unbounded Queue to store all tasks
• SingleThreadExecutor	LinkedBlockingQueue	Note: Since Queue can never become full, new threads are never created
• CachedThreadPool	SynchronousQueue	Thread are unbounded, thus no need to store the task. It is a single slot.
• ScheduledThreadPool	DelayedWorkQueue	Special Queue that deals with Schedules/ time – delays.
• Custom	ArrayBlockingQueue	Bounded Queue to store the tasks. If Queue gets full, new thread is created

Rejection Handler - Types of Rejections

Abort Policy	Submitting new tasks throws RejectedExecutionException (RuntimeException)
Discard policy	Submitting new tasks silently discards it
DiscardOldestPolicy	Submitting new tasks drop existing oldest task, and task is added to the Queue.
CallerRunPolicy	Submitting new tasks will execute the task on the caller thread itself. This can create feedback loop where caller thread is busy executing the task and cannot submit new tasks at fast pace.

Life Cycle

- Service.ShutDown(); -> Initiate shutdown

If we add new Task after ShutDown() will throw RejectedExecutionException();
Service.Execute(newTask());

- Service.IsShutDown(); -> will return true - if shutdown has began.

* Service.IsTerminated(); -> will return true if all Tasks are Completed including Queued ones;

* Service.awaitTermination(10, TimUnits.SECONDS);

This method is used to block the calling thread until either all tasks have completed execution after a shutdown request or the timeout occurs, or the Current thread is interrupted, which ever happens first

* Service.ShutDownNow() {- Returns List<Runnable>

Will shutdown initiate and return all the Queued tasks which are not since executed and let other task to Complete it's execution which is already Started.

```
// Cancel the task
future.cancel(mayInterruptIfRunning: boolean);
```

This boolean parameter determines whether the thread executing the task should be interrupted. If true, and if the task is currently running, it will be interrupted. If false, the task will be allowed to complete if it is already running.

// Returns true if task was cancelled
* future.isCancelled();

// Returns true is task is completed (successfully or otherwise)
*
future.isDone();

6-11-24

1. Advantages of Java being partially object-oriented

1. Flexibility in programming paradigms
2. Ease of learning and adoption.
3. Simplicity and performance.
4. More efficient memory usage
5. Object oriented features when needed.
6. Backward Compatibility
7. Interoperability - Easy integration with other languages.
8. Improved Tooling & Libraries.

2. When NullPointerException() will occur while AutoBoxing & unBoxing

```
Integer i = null;          NullPointerException will occur
int num = i;   -> unboxing
```

```
int i = null; //X - AutoBoxing only work with non-null values,
Integer ii = i;   Because null cannot be autoBoxed into a primitive type
```

3. Difference Between HashMap & TreeMap in terms of ordering?

HashMap	TreeMap
* Does not maintains order and useful for quick access	1. Maintains, natural order or a specified Comparator Keys are stored in red black tree. which keeps them sorted.
* Insert, Delete, and access; Average-Case time Complexity is O(1). In worst case degrade to O(n)	* Insert, Delete, access - Avg and worstcase time Complexity is O(log n) because of underlying red black tree
* Requires more space compared to treemap due to the need for additional Storage for the hash table and handling collisions.	* Generally more memory intensive than HashMap due to overhead of treenodes

4. Maven vs Gradle?

5. Maven & Local Repositories in maven? Central

6. M2- Folder?

7-11-24

1. Key Components of JVM?

1. Classloader 2. Runtime Data accas 3. Execution Engine

1. Classloader loads class files in to JVM.

2. Runtime data areas store data needed while the program runs, like memory for variables & Code.

3. Execution engine actually runs the instruction in the class files.

2. Types of GC

Mark Sweep, Mark-Compact, & Generational Copying

10-11-24

HTML-5 Features

1. Semantic Elements
   - <header>
   - <footer>
   - <article>
   - <section>
   - <nav>
   - <aside>
   - <main>
   - <mark>
   - <progress>
   - <output>

2. Audio and video Support.

   Without the need for third-party plugins like flash.

   - <audio controls>

     <source src="" type="audio/mp3">
     </audio>

   - video

     <video controls>

     <source src="" type="">

     </video>

3. Canvas Elements

   To draw graphics, render images, and create animations on the fly using JavaScript.

4. Local Storage & Session storage

5. Geolocation Api

6. New Form Controls.

- attributes New input types
  - email
  - url
  - tel
  - date, time, datetime-local, month, week
  - range
  - Search
  - number
- New attributes
  - required
  - placeholder
  - autofocus

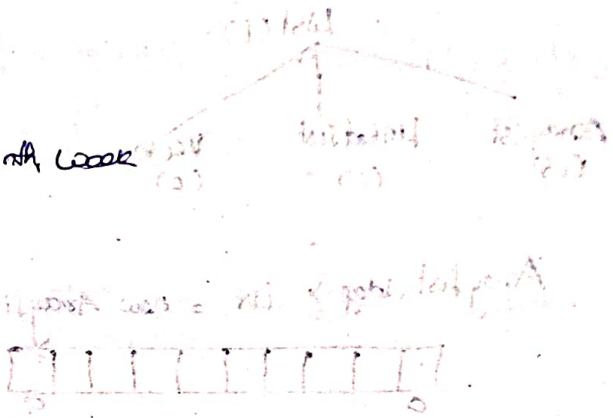
7. Web workers

8. WebSockets

9. offline web applications (AppCache)

10. Custom Data attributes

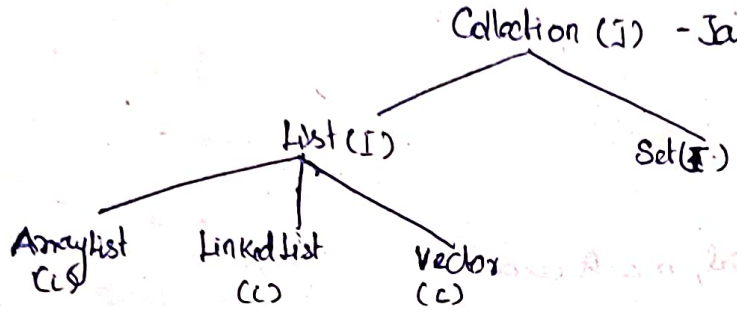
11. SVG and MathML Support



11-11-22

Collection.

1. ArrayList



ArrayList<Integer> list = new ArrayList<>(); Default Capicity is 10

list.add(1);
. . . .
list.add(10);
list.add(11); -> exceeds the current capacity.

When its exceeds the current capacity it will grow or resize

Formula.

till java6 $CurrentCapacity \times \frac{3}{2} + 1;$

After java6 1.5x time increases it size

10 10/2 = 5

CurrentCapacity + CurrentCapacity/2 = 15

ArrayLists are generally called as Dynamic Array, Growable Array, Resizable Array;

After java 17 - Doubling its size to reduce number of times to resize

CurrCapacity/or newCapacity = oldCapacity × 2;

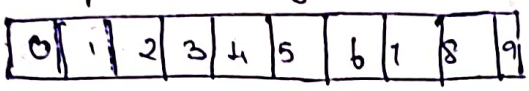
* Doubling strategy provides good balance between minimizing resizing.

* Generally it maintains internal array (elementData) to hold

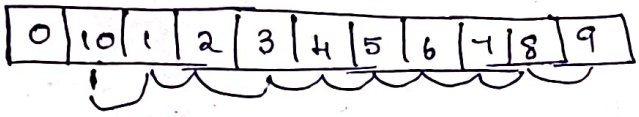
elements. When ever the size is full, it creates a new array that is twice as large & copies the elements over to this new array.

Note: once 'new' array is created the old array will be elegible for garbage collection.

11-11-24. Adding O(1) if adding in last, O(n) or O(n/2) if adding in middle

list = 

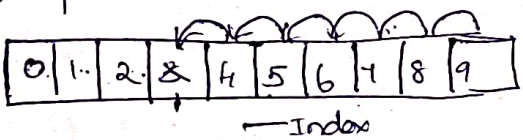
list.add(1, 10)



- Shift operation

Inserting elements in the middle of the arraylist takes more time than a linkedlist, still you are going to use Arraylist a lot in your project.

Remove opr - O(n/2)



list.remove(3);

Again all the elements are to be shifted, still takes more times than arraylist.

* Retrieval

list.get(5); , list.get(1000); (Index) O(1);

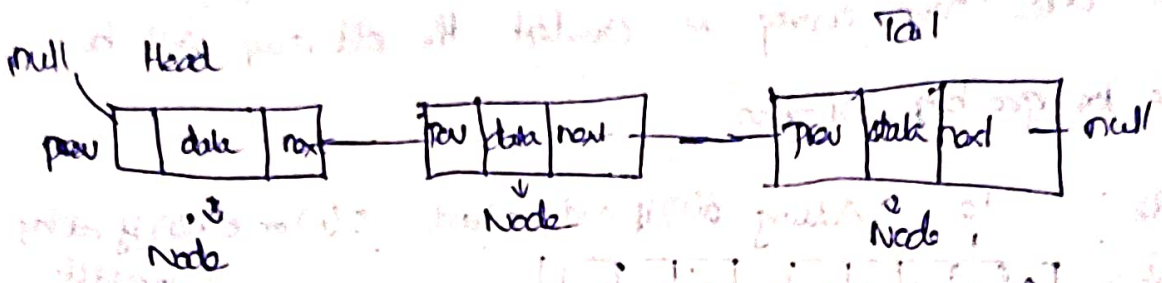
Public class ArrayList<E> extends AbstractList<E> implents List<E>, RandomAcess, Cloneable, Java.io.Serializable{

}

Marker interface, it helps to access the elements in random way.

12-11-24.

LinkedList



20/11/24

1. What access modifiers can be used for class?

public & ~~protected~~ Default. ✓

private & protected - ✗ cannot use

2. Access Modifiers used for methods? All ✓

3. " " " " variables? All ✓

21/11/24

1

Checked Exceptions

* IO Exception
* SQL Exception
* File Not Found Exception
* Invocation Target Exception
* Clone Not Support Exception
* Class Not Found Exception
* Instantiation Exception.

Unchecked Exceptions

* Arithmetic Exception
* ArrayIndex Out of Bounds Exception
* Class Cast Exception
* Index Out of Bound Exception
* Nullpointer Exception
* Number Format Exception
* String Index Out of Bounds
* Un Supported Operation Exception.

2. Methods Defined in Throwable?

1. void printStackTrace();
2. String getMessage();
3. String toString();

23/11/24.

1. Types of inner class?

1) Local Inner class - class inside a method.
2) Mamber Inner class - Non static class inside a class.
3) Anonymous Inner class
4) Nested class - Static inner class

2. Anonymous Inner class.

When this class compiles it Create 2 files

1. InnerClass.class
2. InnerClass$1.class

* We can't define an interface anonymously.
* We can't define static methods, fields or class.

3. Enumeration is introduced in java 1.5.

4 24 -11-24

11. Methods in Collection interface.

1. add(E e) - True is elm was added.
2. addAll(Collection e) - R True if all elm was added.
3. Clear() - clear all emp void
4. Contains(Object e) - Boolean
5. ContainsAll(Collection<> e) - Boolean
6. isEmpty(): - Boolean
4. Iterator(); - Iterator<T>.
8. remove(object o); - boolean
9. removeAll(Collection<?> c) - boolean
10. retainAll(Collection<?> c) - boolean
11. Size() - int
12. toArray() - to object[]
13. toArray(T[] a) - To Specific type.
14. forEach(Consumer<?> super E, action)

2. Methods in List Interface ?

1. add(E e) - add to Last, R boolean
2. add(int index, E element) - Insert at specified position
3. addAll(Collection<? extends E> c) - Add all elm from the specified Collection to the end. R - boolean
4. addAll(int index, Collection) - R - boolean
5. Clear() - R - void
6. Contains(objed e);
7. ContainsAll(Collection<?> c);
8. get(int index);
9. indexOf(object o) - Returns first occurance index, if not present R -1;
10. lastIndexOf(objed o) - Returns last occurence - R int
11. remove(object o) - Remove first occurance. R - boolean
12. remove(int index) - Remove elm from specified position;
13. removeAll(Collection c) - R - boolean
14. retainAll(Collection c);
15. Set(int index, E elm) - Return previous elm;
16. Size();
17. SubList(int fromIndex, int toIndex); - Return List<E>
18. toArray()
19. toArray(T[] a);

3. Methods Specific to LinkedList ?

1. getFirst(); - Elm
2. getLast(); - Elm
3. removeFirst(); Elm
4. removeLast(); - Elm
5. addFirst() - void
6. addLast() - void

26-11-24

Record Classes or Records

Introduced in Java 14 as preview feature & became a standard in Java 16. It is primarily used to store immutable data.

* Automatically generates common methods such as equals(), hashCode(), toString()
* Fields of a record class are implicitly final.
* Reduce boilerplate code.

```
public record person(int age, String name) {
}
```

* name(), age() are automatically created (equivalent to getName() & setName())

Key points

1. Immutable
2. No Setter methods
3. Compact Constructor
4. Subtyping & Inheritance

Records cannot extend other class (except Object) but implements interface.

5. Equality & Hashing

equals() and hashCode() methods are generated (data) based on the fields, not their reference.

GIT & GIT Hub

GIT is a distributed version control system used to track changes in source code during software development.

Features

1. Distributed version control.
2. Tracking changes
3. Branching and Merging
4. Collaboration.
5. History & Revisions
6. Remote Repositories

Commands

git config --global user.name "KathickkannanNR" - git ac name

git config --global user.mail "mail id" - Associated with gitac

This commands are used to configure GIT with our identity like who made the changes to other developers also works with this same branch.

git clone -b "https://github.com/kathickkannanNR/learnings.git

↓

For clone particular branch

Once we created new file in local file

File2.txt - U - untracked file.

It will show as untracked file

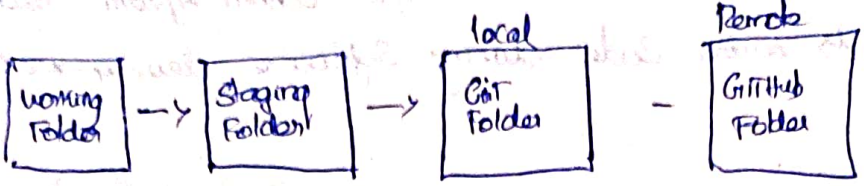
To add a new folder to git folder (which is located in our local system) we have to use git add File2.txt

ls1

File2.txt - U

Once added - git add file2.txt ->

File2.txt - A - This file goes to staging area



git Commit -m "Committed" -> it moves Committed files to local git folder

~~git Commit~~

git push origin main -> will reflect in github.Com files added ✓

Branch

https://github.Com/Kathaccannan/Learnings

* To add a new origin which is created or existing in GitHub or a new folder in local directory, we have to use

git remote add origin https://github.Com/Kathaccannan/learning2.git

* To Rename Branch ex(Master to main)

git branch -M main

"move/rename"

-M allow renaming the branch if the target branch name doesn't already exists. If main already exists this command will fail

once branch name was changed, update the remote to track the new branch

git push --set-upStream origin main

* git branch -a (all) - To get list of branch

This command is used to list all branches in our git repository, including both local & remote branches.

git branch -l → local branches
git branch -r for remote branches

* git checkout branchname

This command is used to switch the branch.

* git diff branchname

To compare this branch name with current branch what are all the changes done.

* git branch branchname

To create a branch in local.

Pull Request - It is mainly used once developers completed their code and asking / creating a pull request to merge on main/dev branch to leads. Once request is sent, TL will check what we done once it verified lead will approve pull request.

7/12/24.

1. ByteStream Java.io

Designed to handle raw binary data (img, audio files, video files, etc) but also used for text data. It handles data as bytes (8 bits)

InputStream	OutputStream
FileInputStream	FileOutputStream
BufferedInputStream	BufferedOutputStream
DataInputStream	DataOutputStream
ObjectInputStream	ObjectOutputStream

2. Character Stream Java.io

Designed for handling character data (text). Automatically handle the conversion between characters & bytes using default or specified encoding. Handle 16-bit characters

Reader	Writer
FileReader	FileWriter
BufferedReader	BufferedWriter
InputStreamReader (byte str to char)	OutputStreamWriter (char to byte strm)

3 PushBackStream java.io

It allows for unread or "pushing back" a single byte or a sequence of bytes back in to input stream.

4. FilterStream

It is a type of input and output stream they are used to wrap around other stream to provide additional functionality or processing.

FilterStream is subclass of InputStream, and outputStream

FilterInputStream FilterOutputStream

5. How can avoid serialization in child class

```
Class parent implement Serializable {

}
Class child extends parent {
    int ....;
    int ....;
    private void writeObject(ObjectOutputStream out) throws IOException {
        throw new NotSerializableException();
    }
    private void readObject(ObjectInputStream in) throws IOException {
        throw new NotSerializableException();
    }
```

23/12/24

Springboot Caching

- To basic Inmemory Cache (Springboot starter-Cache)
- @SpringbootApplication
  @EnableCaching
- By default springboot use basic In Memory Cache (like ConcurrentHashMap CacheManager)
- If we need some advance caching mechanism like (redis, EhCache, Hazelcast) then we need to add dependency in pom.xml.

1. @Cacheable({"books", "isbns"}); - Method level annotations
2. @Cacheable(CacheNames = "books", key = "#isbn")
   public Book findBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed)
3. @Cacheable(CacheNames = "book", condition = "#name.length() < 32", sync = true)
   public void findBook(String name);
4. @CachePut(CacheName = "book", key = "#isbn")
   void updateBook(ISBN isbn, Description description);
5. @CacheEvict(CacheNames = "book", allEntries = true)
   void loadBook(InputStream batch);
6. @Caching - To specify multiple annotation of same type?
   - CachePut or CacheEvict

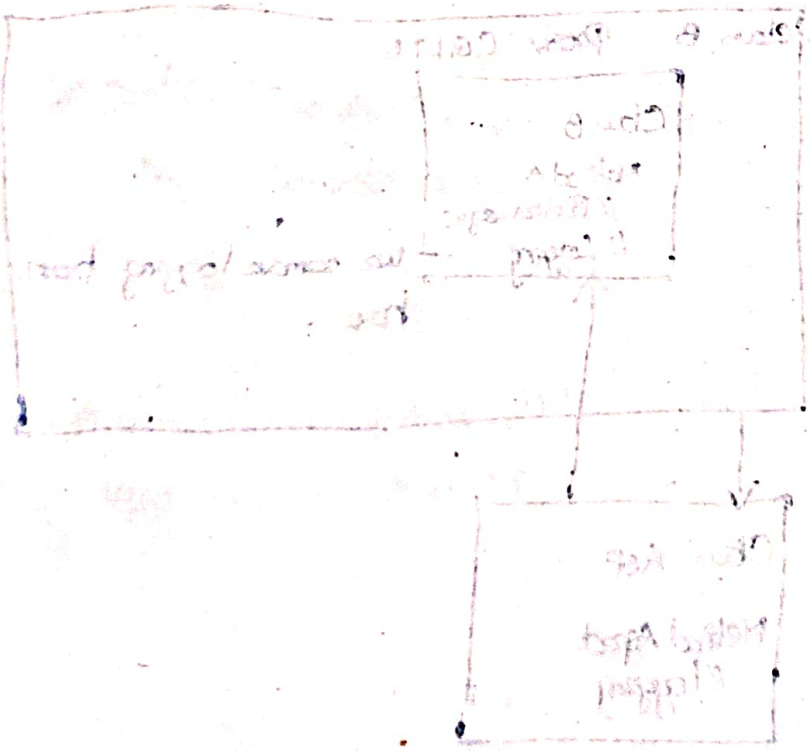
   @Caching(evict = {@CacheEvict("primary"), @CacheEvict(CacheNames = "secondary", key = "#p0")})
7. @CacheConfig - class level

25/12/22

1. What error will throw while Circular Dependency issue?

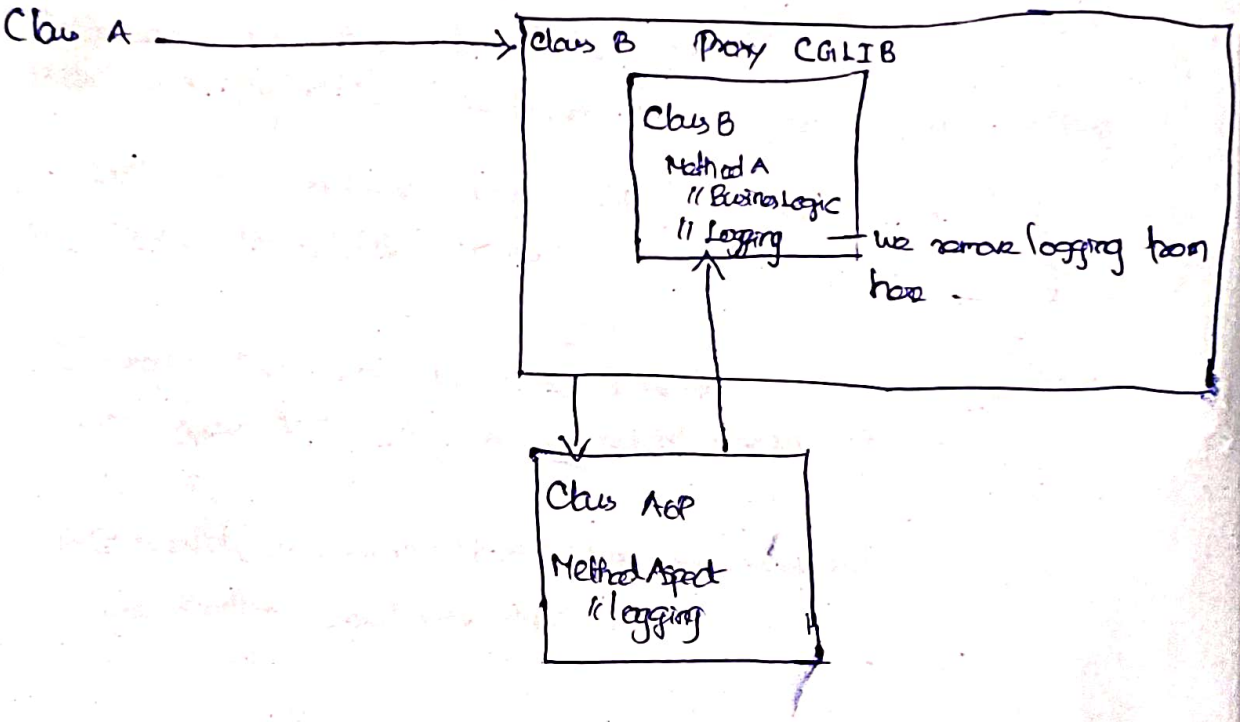
2. How to Create Custom Scope?

```
CustomScope impl Scope{
@Override
Get (Str name, ObjectFactory<?> objFact){ }
@Override
remove (Str name){ }

registerDestructionCallback (name, Runnable test){ }

resolveContextualObject( key);

getConversationId(){ }
```



AOP - Aspect oriented Programming

* Unit of modularity is aspect
* It enables the modularization of Concerns (such as transaction management) that cut across multiple types and objects. Such Concern are often termed "Cross cutting Concerns" in Aop like (log)



Aop use proxy layer of our actual class to run aspects.

Types of Aspects

1. @Before
2. @After
3. @AfterReturning
4. @AfterThrowing
5. @Around - In this will be do both before and after logging or others

```
@Around("execution(* Com.Springboot.Service.UserService.findUser(**))")
public Object aroundAspect(ProceedingJoinPoint joinPoint) throws Throwable {
    System.out.println("Aspect log called"); // executes first (before method call)
    Object result = joinPoint.proceed(); // executes the actual method (joinpoint)
    Sysout("After user found);
    return result;
}
```

6.
```
@PointCut("execution(* Com.*.*.*.*(..))")
void SomeMethod() {
}

@Around("SomeMethod()")
void anotherMethod() {
}
```

Designator

1. within - @PointCut("within(Com.Springboot.demo.Service.ProductService)")

   no need to add **.* (.) like this.

2. @within - works based on ~~annotations with in class level~~ Types

   @PointCut("@within(~~Com.Springboot.demo.Service~~ Com.Springframework.Stereotype.Service)")

3. execution - For matching methods execution join points

4. @annotation - Based on annotations

@PointCut("@annotation(org.springframework.web.bind.annotation.PostMapping())")

5. @Target

@PointCut("target("Com.CodeSnippet.Com.Service.ProductService)")

If ProductService is autowired in any of the class, and using the object to call a method that method is a joinpoint.

```
Class Controller {
    @Autowired
    @ProductService Service;

    Public void getprod() {
        Service.getproduct();
    }
}
```

↳ Joinpoint

6. this

"this" is used. When the CGLIB proxy class implementing the particular interface which are mentioned.

@PointCut("this(Com.example.Service.MyService)")

MyService is an interface.

If MyServiceImpl implements MyService, then advice will intercepted every method on MyServiceImpl.

6/a/25

1. Producer - Consumer problem (Bajja Bounded problem)

There are two threads producer and consumer. Producer produces some tasks, Consumer consumer Consumes the the tasks simultaneously. But if producer or Consumer is faster or slower than each other, there might be Chance Consumer try to consume empty taskslist or producer if fast - it will produce more task so Consumer is unable to Consume.

So using ArrayBlockingQueue will solve this problem efficiently.

7/1/25

1. Unnamed patterns & variables

We can use _ (underScore) instead of unusing variable name like

```
try{
  // some code
}catch(Exception _){
  System.out("error occured");
}
```

2. Sequenced Collections - Java 21.

addFirst(), addLast(), removeFirst(); removeLast(); reversed()

3. Relax Binding in springboot

app.prop:- db.servername = postgres

```
Class x{
  @value("${db.servername or dbServerName}")
```

No need to give exact property name. Springboot will match the name with different case like CamelCase, kebab-case, SnakeCase or in upper / lower case separated by UnderScore.

4. FixedRate vs FixedDelay

@Scheduled(fixedDelay = 5000)

	Interval start : From start of prev execution	From end of the prev execution
Task overlap	May cause overlap if task takes longer than interval	No overlap: waits for previous task to complete
Execution trigger	Executes at a fixed interval starting from the start of the last execution.	Executes after a fixed delay from the completion of the last execution
Use Case	Task that should run periodically regardless of exe time	Tasks that should not overlap & should run after a delay from prev exec.

11-01-25

Hibernate

Fetch Modes

- FetchMode.SELECT (Default)

Hibernate will generate a seperate SQL query for fetching the associated entities or collections.

Use Case - It will fetch data lazily & avoid large join operations

* Seperate query is executed for each association, which may lead to N+1 problem in some cases

- FetchMode.JOIN

It uses SQL Join to fetch the associated entities in single query. This can be more efficient than multiple SELECT queries but may result in a more complex SQL query.

Use case - When we want to eagerly fetch associated entities in a single query without multiple queries

- FetchMode.SUBSELECT

Hibernate will use subselect to fetch associated entities in a batch. It fetches collections in one query.

```
class Dept {
    @OneToMany(mappedBy = "dept", fetch = FetchType.LAZY)
    pvt Set<Emp> emp;     @Fetch(FetchMode.SUBSELECT)
}
@Entity
Class Emp {
    @ManyToOne(fetch = FetchType.LAZY)
    pvt Department dept;
}
```

First it will prepare main query, and it will prepare SubSelect query.

16-01-25

Allocating initial heap size

Java -xms8g -xms1g MyApp

Allocating max heap size

To enable Heap Dump

Java -xx:+HeapDumpOnOutOfMemoryError -xx:HeapDumpPath=heapdump.hprof MyApp

To enable garbage collection logs

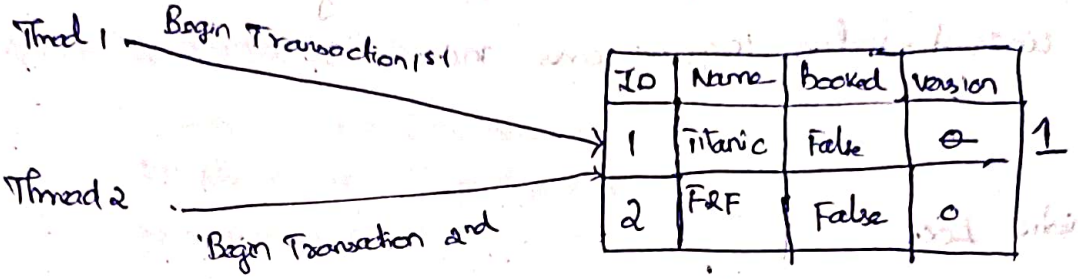
Java -xlog:gc* MyApp

26/1/25

Optimistic & pessimistic locking

This locking mechanism is for concurrent table access/modification.

Optimistic



Thread 1 Begin transaction first and status changed to true & version no updated 0 to 1. If another thread trying to update booked status (already version num is updated to 1) so version mismatch will occur. Optimistic lock will throw an exception (another thread already updated / deleted record.

Not lock resources during data access instead version number, timestamp, checksum is stored with data.

Pessimistic

Assume conflicts are likely and prevents them by locking the resource during the entire transaction.

1) A transaction locks a record when it read it

2) Other transaction are blocked from accessing the locked resource until the lock is released

3) locks can be applied at different level (eg - row, page, table)

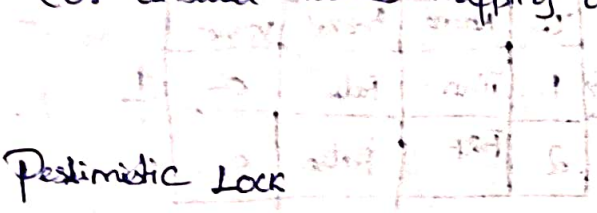
For optimistic

In entity class use this annotation for version check.

```
@Version
int version;
```

By default it is transient

If version number is mismatched exception will be throwed. (Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect)



Pessimistic Lock

In Repository

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
Optional<product> findById(long id);
```

Types of Locks:

1. READ
2. WRITE
3. OPTIMISTIC
4. OPTIMISTIC_FORCE_INCREMENT
5. PESIMISTIC_READ
6. PESIMISTIC_WRITE
7. PESSIMISTIC_FORCE_INCREMENT
8. NONE

STORED PROCEDURES

Create procedure Schema.procedureName (IN id int, out total int,)
update_Stock
Begin

. . . .

end.

```
@NamedStoredProcedureQueries ({
  @NamedStoredProcedureQuery(
        name = "updateStockProduct",
        procedureName = "update_Stock",
        parameters = {
         @StoredProcedureParameter (mode = ParameterMode.IN, name = "id", type = Integer.class),
        @StoredProcedureParameter(mode = ParameterMode.OUT, name = "total", type = Integer.class)
        }
  )
})

@Entity
public class product{

}

@Repository
Interface productRepo extends JPARepo<product, Integer>{
    @Procedure(name = "updateStockProcedure")
    Integer updateStock(Integer productId);

    @Query(value = "SELECT get_total_price(:productId)", nativeQuery = true)
    Double getTotalPrice(int productId);
```

To call function

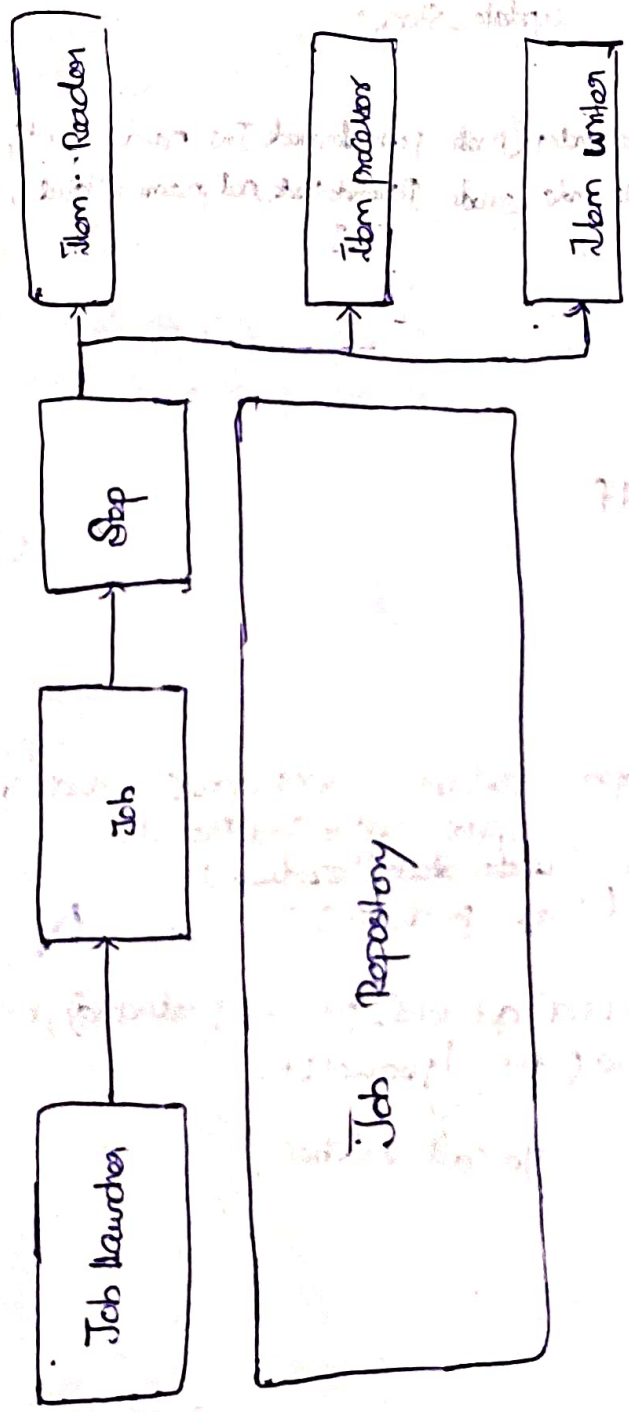
}

29/01/25

Spring Batch

* Chunk processing
* Scheduled Execution
* Error Handling & Retry
* Restartability
* Job Monitoring

Architecture



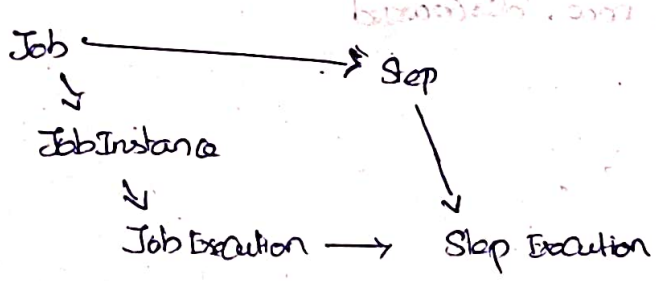
1. Job Launcher

It is a interface for launching a job with given set of parameters

```
public interface JobLauncher {

    public JobExecution run(Job job, JobParameters jobParameters) throws
                                JobExecutionAlreadyRunningException, JobRestartException,
                                JobInstanceAlreadyCompleteException, JobParametersInvalidException;

}
```

2. Job

A sequence of task execute to certain purpose.
Job will have multiple steps



3. Step

1. ItemReader

It is an interface it helps to read data from file, XML, Database etc

```
public I ItemReader {

    T read(); throws ...

}
```

2. Item writer

3 Job Repository

It is the persistence mechanism for all the three types mentioned earlier.

30/1/25

.Prop

spring.batch.job.enabled = false;

By default spring batch will start the job while application startup, so we need to restrict that.

@EnableBatchProcessing is now discouraged