2. ArrayList	Vector ~~List~~
Not Synchronized preffered bcz can synchronized explicitly using Collection.synchronizedList	Synchronized
Not Thread Safe.	Thread Safe. (~~one~~ thread at a time)
Fast	Slow
Not a Legacy class.	Legacy class.
Iterator interface to traverse the elements	Iterator or Enumeration.

3 ArrayList	LinkedList
uses Dynamic array.	ussed doubley linked list.
implements only List	List and Queue
Manipulation is Solw	Fast

Storing and accesing is fast	Slower
No descendingIterator()	iterated in reverse
If no constructor is overloaded, it creates empty list of initial capacity 10.	No default capacity Empty list is created.
Memory overhead is low as index hold only actual data.	More as node needs to maintain the address of prev & next node

REST → Representational State Transfer

→ POST is not idempontent. Put is.

Idempotent means when you hit a server n no. of times with same req., your resonse should also be same and memory area given to that table same

URIs REST is an arch. style for developing app that can be accessed over n/ws.

POST → CREATE. GET → Retrieve

PUT → UPDATE. DELETE → Delete

```
@Override
@POST
@Path("/add")
```

Features of REST

→ based on Client Server Model

→ uses HTTP Protocol for fetching data query execution or any function.

→ resources are accessible by means of URIs.

→ follows statelessness concept where client req and response are not dependent on others and thereby provides assurance of getting req. sized data.

→ uses the concept of Caching to minimize the server calls for same type of requests.

→ medium of communication b/w client and server called Messging.

* Restful web service should not keep client state on server. This is statelessness.

It's responsibility of client to pass its context.

⇒ Java 8

→ Conciseness of the code.
→ Functional Programming which is enabled by Lambda expression.

Features

Lambda Exp	Stream API
Default method	Static method
Func. Interface	Optional
Date API	JS Engine
Method / Constructor References.	

Adv.

→ Consice code
→ Readable & reusable code
→ Testable code
→ Parallel operation

* Lambda exp is an anonymous func- (w/o name, return type, access modifier)

It helps to iterate, filter and extract data from collection. It saves code, just write implementation.

→ - Lambda Symbol

Eg: (a, b) → sop (a+b);

Provides Implementation to Func. Interface

BiConsumer < Integer, Integer > biConsumer
= (a,b) → sop(a+b);
biConsumer.accept(10, 5);

→ BiConsumer is predefined functional Interface, available in java.util.function package. It consumes and returns nothing.

* Yes we can create our own functional Interface.

Functional Interface is an interface with exactly one single abstract method & can have multiple Static or default methods

1. Create an Interface
2. Annotate with @FunctionalInterface
3. Define one Abstract method.
4. Define any no. of Static & default method

Java can implicitly identify functional interface, even without the annotation.

```
@FunctionalInterface
public interface FIDemo
{
    void singleAbsMethod();
    default void NMethod1()
    {   SOP("name1");
    }
    default void NMethod2()
    {   SOP("name2");
    }
}
```

⇒ Method Reference.

→ It is a replacement of lambda exp
→ It is used to refer method of functional interface to an existing method. mainly used for code reusability.

Whenever we have existing implementation of abstract method of our functional Interface then we go for method ref.

```
class Test{
    public static void testImp()
    {
        SOP("Test");
    }
}
Public class MethodRefDemo
{ public static void main(String[] args)
    {
        FunctionalInterfaceDemo funcDemo
            = Test :: testImp();
        funcDemo.singleAbsMethod();
    }
}
```

⇒ Default Methods.

→ It is a way of adding new methods to the Interface w/o affecting the implementation.

→ use of default method is "Backward Compatibility", means if JDK modifies any Interface the the class which implement the interface will break. but if you add default method in an interface then you'll be able to provide default implementation.

```
public interface InterfaceDemo
{
    default void printName()
    {   SOP("Welcome");
    }
}
```

```
public class Idemo implements InterfaceDemo
{
    public static v. m. (St.)
    {
        Idemo idemo = new Idemo();
        idemo.printName();
    }
}
```

output : Welcome

Defaut methods have dummy implemented but you can override and provide your own implementation.

```
Public class Idemo implements InterfaceDemo
{
    Public void printName()
    { SOP("Welcome to override");
    }
}
```

* Diamond Problem.

Class A → m1()

Class B → m1()

Class C implement A, B

multiple inheritance → ✗

* InterfaceName.super.methodName()

```
public interface Interface1
{
    default void m1()
    {  sop(" Interface1");
    {
}

public interface Interface2
{
    default void m1()
    {  sop("Interface2");
    {
}

public class demo implements
        Interface1, Interface2
{
    public void m1()
    {  Interface1.super.m1();
       demo.sup
    }
    p.s.v.m(String args[])
    {
       demo dem = new demo();
       dem.m1();
    }
}
```

Output : Interface1

⇒ Static Methods in Java 8.

Static methods were introduced in Interface so that you can call those methods with just interface name. No need to create class and its object.

```
public interface StaticDemo
{
    static void Meth()
    {   SOP(" Static method ");
    {
}
public class StaticClass implement
                StaticDmo {
    public static void main( )
    {
      StaticDemo.Meth();
    {
}
```

Output: Static Method

Syntax: InterfaceName.Static method

[Disadv].

→ Static methods are not available to 'implementing classes by default but you can call them using Interface name explicitly from the implementing classes.

⇒ Predicates.

predefined Functional Interface having only 1 abstract method

```
public boolean test(T t);
```

whenever we want to check some boolean cond. then we go for predicate.

⇒ Optional.

* It provides a way to deal with null value It is used to represent a value is present or not, available in 'java.util.package

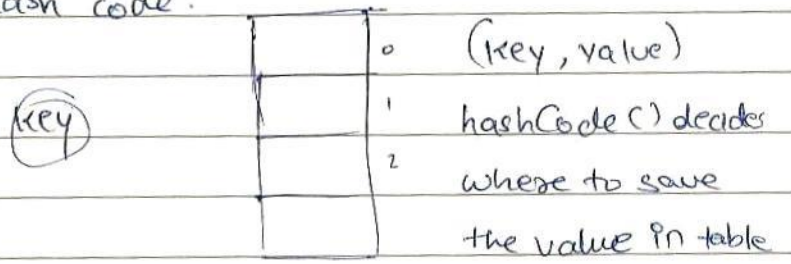
It is a single-value container that either contains a value or doesn't

```
Optional<Emp> e = repo.findById(id);
if(e.present())
{ return new ResponseEntity<>(e.get(), 'ok')}
else { return new " <>(" Not present");}
```

⇒ HashMap

It works on the principle of Hashing.

Hashing means using some algo. to map object data to some integer value, hashCode() method return you that hash code.



Hash Table

Eg: Phone Directory.

Key	Value
A	123
B	456
C	789
D	10 11 12

```
Map<String, String> phone = new HashMap();
phone.put("A", "123");
phone.put("B", "456");
```

To print:

```
sop(phone.get("A"));  → 123
```

To print all:

```
Set<String> keys = phone.keySet();
for(String i : keys)
{
    sop(i + " : " + phone.get(i));
}
```

→ Map.Entry.

Map is an interface. So, Entry is an interface within an interface.

```
Set<Map.Entry<String, String>> values
        = phone.entrySet();
for(Map.Entry<String, String> i : values)
{
    sop(i.getKey() + ": " + i.getValue());
    i.setValue("III");
}
```

HashMap	HashTable
- Introduced with 1.2	introduced with Java.
- Unsynchronized	Synchronized
- Not Thread Safe	Thread Safe.
- Works with ~~Single~~ MULTIPLE Thread.	~~Multiple~~ One
- Fast	Slow
- Allows 1 Null key, multiple null value.	No Null key or null value.
- Iterator	Iterator or enum.

⇒ Garbage Collection

In-use object or referenced object means that some part of your program still maintain pointer to that object.

Un-used object or unreferenced object means it is no longer referenced by any part of program

Garbage Collec. is an automatic process of looking at heap memory, and deleting the unused objects.
It removes the burden of manual memory allocation / deallocation.

Q Where are objects created in memory?
Ans Whenever an object is created, it's always stored in Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and references variables to objects in heap space.

Q Who manages Garbage Collector.
Ans JVM controls it and it decides when to run the GC. JVM runs it when it realizes the memory is running low.

* GC is a background / daemon thread.

Q How can the GC be requested?

Ans One can req. it but there is no guarantee that GC will be called.

There are 2 ways:

① System.gc() method

② Runtime.getRuntime().gc()

Q Purpose of finalize() method.

Ans - also called finalizer is defined in java.lang.Object.

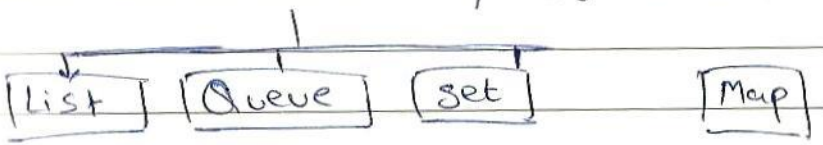
- It is called by GC just before collecting any object which is eligible for GC.
- It provides last chance to object to cleanup and free any remaining resource.

⇒ GC works in 2 steps:

1. Marking
2. Deletion

⇒ Collection framework.

Q. Collection Hierarchy. (java.util)



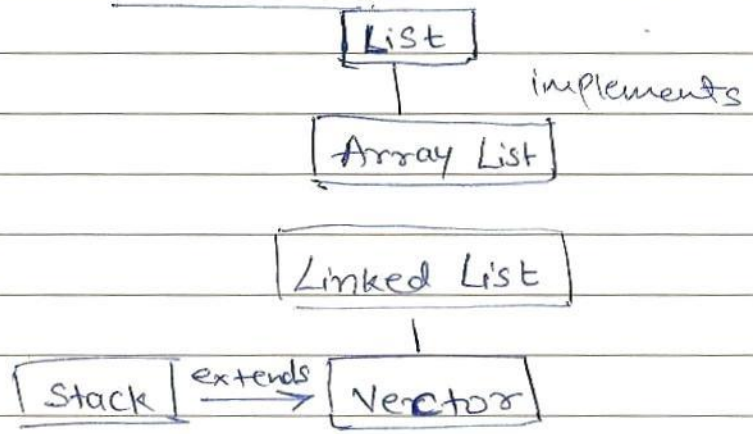
1. List - contains ordered elements
   - may include duplicates.
   - supports index-based search.

2. Set → no order
   → no duplicates
   → no index based search.

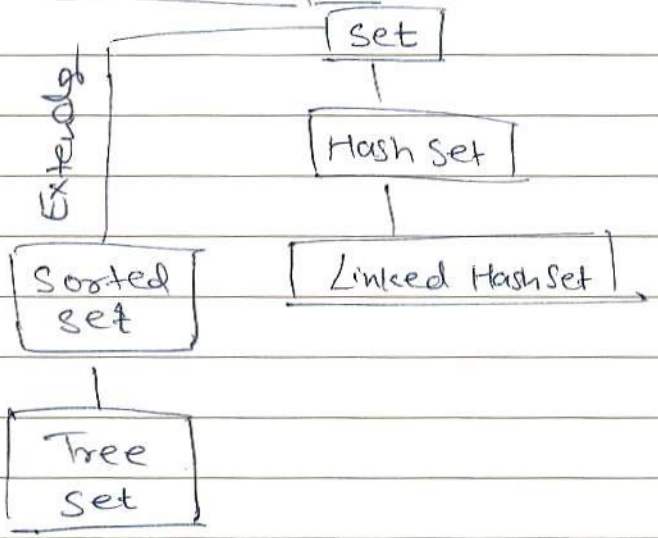
3. Queue → FIFO approach

4. Map → represents key, value pair
   → does not implement Collection.
   → only contain a unique ~~value~~ KEY
   → Can have duplicate values

List Interface



Set Interface



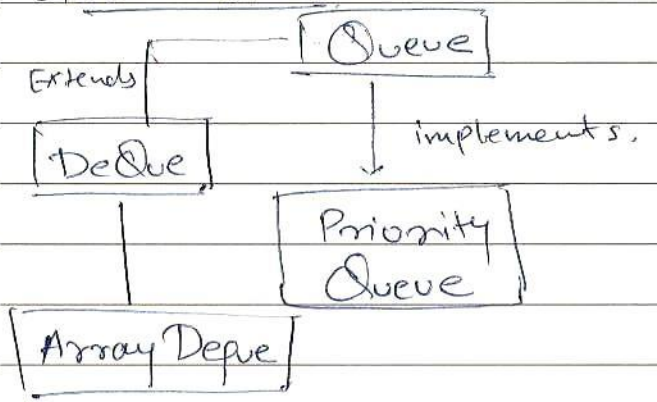
HashSet:- contains only unique elements
- only one null element can be add
- unordered set.

Linked HashSet:- ordered version of HashSet which maintaing a doubly linked list across all elements
- It preserves insertion order.

Sorted set:- All elements must implement comparable interface.
- sorted in an ascending order.

Tree set:- Uses a Tree for storage
- Objects in a Treeset are sorted and in ascending order.

Queue Interface

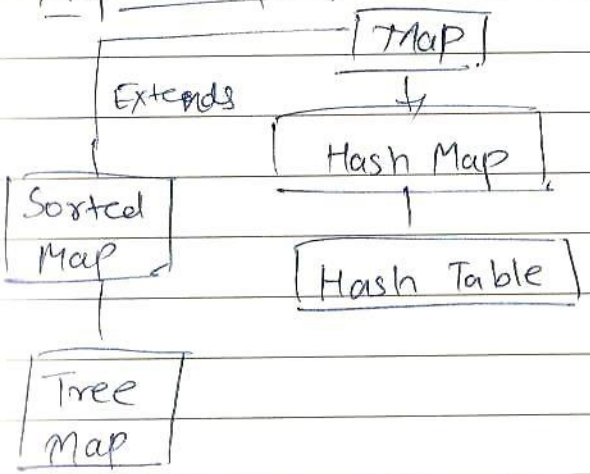


Priority Queue:
- queue with priority associated with each element.
- higher priority ele. is served before a low priority ele. irrespective of insertion order.

DeQue: double-ended queue elements can be added and deleted from both ends.

Array Dequeue: way to apply resizable-array in addition to the implementation of deque interface.

Map Interface



Q Why Map doesn't extend Collection Interface?

Ans Map interface follows key/value pair but Collec. interface is a collection of objects which are stored in a structured manner with a specified access mechanism.

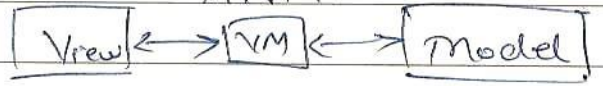
- Main reason is that add(E e) method of collection doesn't support key/value pair like Map put(K,V) method.

Q What is Angular?

Ans. Angular is an JS Binding framework which binds the HTML UI and JS Model?

This helps you to reduce your efforts on writing lengthy lines of code.

MVVM.



HTML Angular object

It helps to build SPA by using routing. It also has features like HTTP, DI, Input/output bcz of which do not need other framework

	(1.x) Angular JS	Angular (2,4,5,6,7,8,9)
1.	JS language	TS language
2.	Controller archi.	Component arch.
3.	Not mobile compliant	YES
4.	No CLI	CLI (ng)
5.	No Lazy loading	YES
6.	No SEO friendly	YES
7.	Not run on server side	YES

Q. What are directives?

Ans. Directive changes the behaviour of HTML DOM. They are angular syntaxes inside HTML.

3 types of Directives:-

① Structural Directives

- Change the DOM layout by adding and removing elements.

eg: ngFor

```
<tr> *ngFor="temp of Sales">
    <td>{{ temp.Name }}</td>
    <td>{{ temp.Amount }}</td>
</tr>
```

② Attribute Directives.

- Change the appearance and behaviour of HTML elements. like color, true/false

```
<div [hidden] = "Hide()">Hello</div>
```

③ Component Directives.

Directives with templates. It's like a user control.

```
@Component(
  {
    selector : "my-grid",
    templateUrl : './Utility.Grid.html'
  })
```

Q. What is NPM & Node_Modules?

NPM - Node Package Manager.

NPM is a package manager which makes installation of JS framework easy.

node_modules is the folder where all the packages are installed.

Q package.json file?

It has all the JS references needed for a package. So rather than installing one package at a time we can install all packages.

Q. What is Typescript? extends JS by

It is superset of JS. It added types to JS. It speed ups your development experience by catching errors and providing fixes before you even run your code. It is open source. It gives OOPs env. which convert to JS.

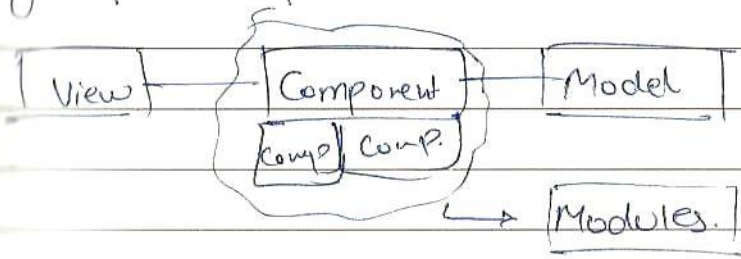
Q What is Aylar CLI?

Command Line Interface is used to create initial Aylar project template.

eg. ng new myApp.

Q. What are Component and Modules?

Component is where you write your binding code. Module logically groups components.



Q What is decorator?

Decorator defines what kind of AngUlar class it is

@Component → Angular Component

@NgModule → Angular Module.

Also termed as Annotations & Metadata

Q What are templates?

Templates is an HTML view of Angular in which we can write directives

2 types of defining template:

→ In-line

template: " test "

→ Seprate file

templateUrl: './Cust.Component.html'

Q. Types of Data Binding?

Data binding is how view and model communicate with eachother.

1. Expression or interpolation. {{ }}

Data flows from controller to view then we can mix it with HTML tag.

{{ Cust.custId }}

2. Property [] Pass the value from parent to child component

Data flows from controller to view but then get attached to some user input like text box. []

[(ngModel)]

3. Event Binding () Child → Parent

A click event which goes from View to ~~controller~~ component.

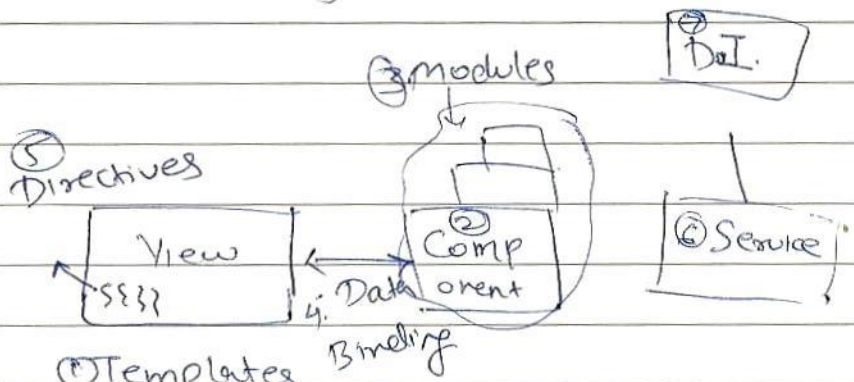
eg: (click) * send event *

4. 2-way Binding. [( )]

Data flows from controller's to View and view to controller.

Q. Architecture of Angular

1. Template 2. Component
3. Modules 4. Data Binding
5 Directives 6. Services
7. Dependency Injection 8. Metadata

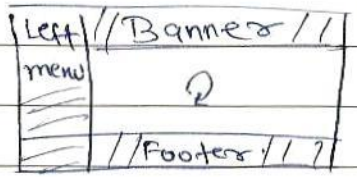


Service is a common logic which we want to inject all accross the project It can be validation, loggin utility, http services and so on.

DI injects the services

Q. What is SPA?

Single Page Applications where the main UI ~~page~~ gets loaded once and then the needed UI is loaded on demand.



// → loaded once

To Implement SPA we need to use Angular Routing.

Routing is a collection which has 2 things. URL and when this URL is called which component to load.

Routing helps to define the navigation of angular app. So if we want to move from one page to another and we want SPA then routing is needed

{ path: '~~Login~~ Home', component: HomeComp }
{ path: 'Login', component: LoginComp }

For implementing routing we need 3 steps:-

1. We need the collection saying which url will go to which component.

{path: 'Home', component: HomeComp}

2. We need to have the router outlet where exactly the navigation will load.

<router-outlet></router-outlet>

In HTML

<a [routerLink]= "['Home']"> Home

If routing from component.

route.navigate(['/Home']);

Q Lazy Loading.

It means demand loading

Loading only req. HTML, CSS, JS for better performance

To implement Lazy loading divide project into seprate modules.

{path: 'Cust', loadChildren: '../cust'}

Q. Dependency Injection.

It is an app design pattern where rather than creating object instances from within the component, Angular inject it via constructor.

Eg: constructor(public l: BaseLogger)

To implement DI we use providers

Advantage of DI is de coupling.

When we change at one place we do not have to change at other place

Q. Diff b/w ng serve and ng build?

ng serve build in memory.

ng build build on hard disk

↳ (for production)

Q. Ng build -prod

It compress JS file, remove comments, create GUIDs of JS files and make app ready for production.

Q. Life Cycle Hooks of Angular.

① ngOnChanges

- It is executed at the start, when a new component is created and also when one of the bound input property changes.

- Input properties are defined using @Input decorator

Parent Component: →

```
<app-child [message]="message">
</app-child>
```

Child comp. →

```
@Input() message: String
```

② ngOnInIt

- It is called after the creation of component and updation of i/p properties.

- It is fired only once. Initialization logic can be added for component.

Here you have access to every Input prop of the component. You can use them in http get requests to get data from backend server.

- None of the Child components are available at this point.

③ ngDoCheck

- Angular invokes this hook event during every change detection cycle. It is called even if there is no change in any prop.

- It is called event on a click on web-page which do not change anything

- It can be used for custom change detection.

④ ngAfterContentInit

- It is called after the component's projected content has fully initialized.

- It is called only once

- Angular updates properties decorated with @ContentChild and @ContentChildren before raising this hook. It is raised even if there is no content to project

- The content refers to external content injected from the parent component.

- ng-content element acts as a placeholder for the content from the parent.

parent:

```
<app-child>
<p> This content injected from parent. </p>
</app-child>
```

child:

```
<ng-content></ng-content>
```

⑤ ngAfterContentCheck

- This hook is similar to ngAfterContentInit. Both are called after the external content is initialized, checked and updated.
- Component only hook.
- The diff is that this hook is raised after every change detection cycle.

⑥ ngAfterViewInit

- The view refers to the view template of the current component and all its child components and directives.

- It is called after the Components View and all its child views are initialized.

- It is called during the first initialization of view.

⑦ ngAfterViewChecked

- Similar to ngAfterViewInit
- called after it checks and updates component's view and child view.
- Component only hook.
- The only diff is it is raised during every change.

⑧ ngOnDestroy

- Great place to do some cleanup work. It is called right before the objects will be destroyed itself by angular.
- Ex:
When you placed ngIf on a component and this ngIf is then set to false, ngIf will remove the component and ngOnDestroy is called.
- You can unsubscribe observable detach event handlers to avoid memory leaks.

Annotation	Decorator
1. They are only meta-data set on the class using Reflect Metadata library.	It is a function that is called on a class.
Used for creating an attribute annotations that stores array.	Decorator is a func that gets object needs to be decorated.
Hard-coded.	Not Hard-coded.

Example:

```
import { ComponentA
Annotation as compo
nent } from '@angular/
core';
```

```
import { Component }
from '@angular
/core';
```

Exception

It is an abnormal condition which occurs during the execution of a program and disrupts normal flow of the program. If not handled properly it can cause the program to terminate.

Eg:

```
public class Demo
{
    public String dogName = "Abc";
}

public class ExceptionDemo
{
    p.s.v.m( )
    {
        Dog dog = null;
        SOP(dog.dogName);
    }
}
```

O/P: Null pointer Exception.

How to handle Exception

Ans

Try: Encloses set of statements which can throw exception hence req to be monitored.

Catch: When exception occurs, this catches that excep. and to handle.

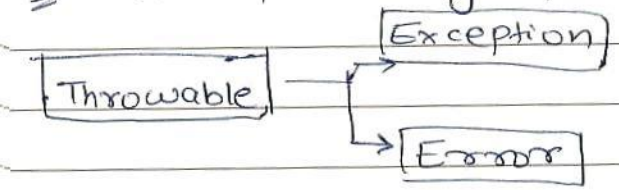
Finally: This always get executed regardless of excep. occurence. Clean up is done here.

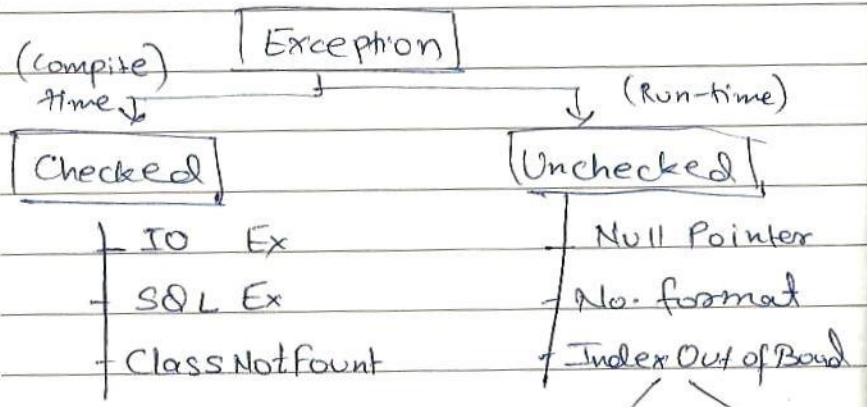
Eg:

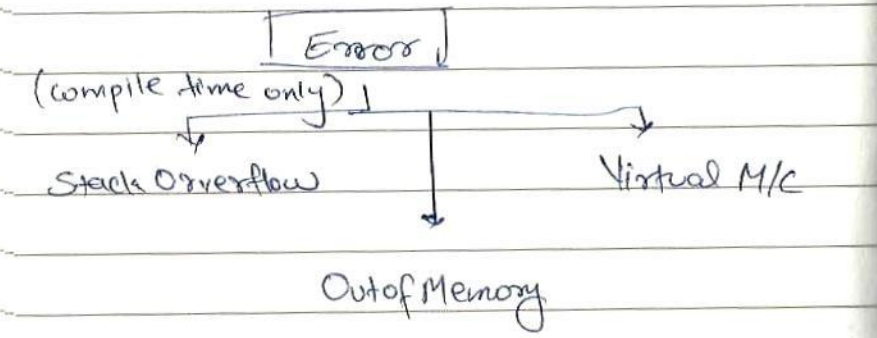
```
p.s.v.m( )
{
    Dog dog = null;
    try{
        SOP(dog.dogName);
    }
    catch (NullPointerException e)
    {
        dog = new Dog( );
        SOP(dog.dogName);
    }
}
```

O/P: Abc

Q. ~~Types of~~ Heirarchy of Exception?







Exception	Error
Recovery is possible	Not possible
Compiler have knowledge about checked excep.	No knowledge
Related to applicath	Related to env where app is running
Excep are checked and uncheked	Only checked
java.lang.Exception	java.lang.Error

Q. Can we write only try block?

A: No, either catch or finally is required.
It will throw error: insert finally to complete the statement.

Q What happen if excep is thrown inside Main method? throw NullPointer();

A: Main method do not have parent, Java Runtime Env. terminates and prints the excep. msg in console.

Q. Throw vs Throws

Throw	Throws
- throw keyword (real-scenario)	throws keyword (warning)
- Checked excep: cannot be propogated needs try catch.	can be propogated and ask the parent to handle
- used within the method.	with the method signature.
- Cannot throw multiple excep.	can declare multiple excep.

Q. Unreachable catch block error?

This occurs when:

```
catch (Exception e)
{ sop ( ); }
catch (NullPointerException e)
{ sop(); }
```

when super class is kept first and sub class later.

Q. Multi catch block? (| → Pipe)

Ans Introduced in Java 7, to reduce code duplication.

eg:

```
try
{ }
catch (NullPointerException | SQLException
e)
{ }
```

Q. Final, Finally, finalize?

Final: used to apply restriction on variable, method, and class. Final variable can't be changed. Method can't be overridden and class can't be inherited.

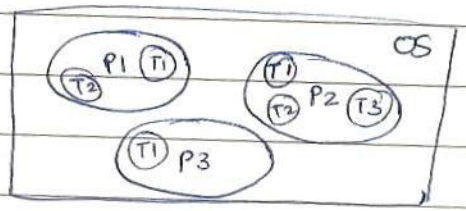
Finally: used with try-catch block.

Finalize: to perform clean up before the object is garbage collected.

Q. Multi-Tasking?

Performing multiple task at one time

1. Process Based
2. Thread Based



* Task Manager has process
* Thread are in ProcessExplorer GUI → right click on Process → properties → thread.

Q Multi-tasking vs Multi-Threading?

Process-Based:

executing several independent tasks simultaneously.

Thread Based:

executing several independent parts of same program simultaneously

Process are heavy weight.	Threads are light weight.
Req. own seprate address space.	share same addr ess space.
Req more overhead.	Less overhead
Inter process communicatn is expensive. Context Switching is costly.	- Less expensive.

Q. Thread?

Lightweight processes within a process.

Java creates thread using 'Thread Class'

All java program have at least 1 thread. i.e. main thread which is created by JVM at program's start when main() is called.

Types of thread:

① User Thread
② Daemon Thread

User Thread are created by Java developers - eg: main thread, all threads created inside main thread.

Daemon Thread is a low priority that runs in background. eg Garbage Collection. JVM does not care weather Daemon thread is running and terminates itself when all user threads finish their execution

Q. How to create user Thread?

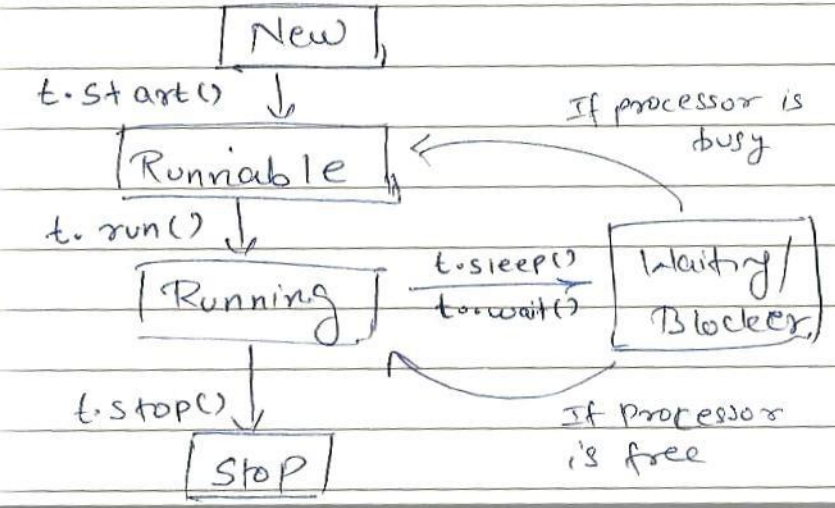
① Extending java.lang.Thread class
② Implementing java.lang.Runnable interface

Implement Runnable interface when just need run() method of thread otherwise extend thread for multiple method like stop, suspend, resume, interrupt, etc

In java, only one class can extended at one time, so if you don't want to extend other classes use Runnable.

* The o/p of thread is not in order because the order is decided by Thread Scheduler which is a part of JVM hence its vendor dependent and we can't expect exact execution order.
→ decide on 2 ways: priority, FCFS.

Q. Life Cycle of Thread.



New: When an object is created for the Thread class.

Runnable: Ready to run.

Running: If processor is available

Waiting/Blocker: When interruption occur

Stop: termination

Singleton Class

A class with one instance. only.

Eg: SOCKET and DB Connection.

```
public class Singleton {
    private static Singleton singleton = null;
    public string s;
    private Singleton()
    {   s = "Hello";
    }
    public Static Singleton getInstance()
    {   if (singleton = null) {
            Singleton = new Singleton();
        }
        return singleton; }
```

Eg: Runtime class, Service Locator

ways to Break Singleton.

// Marker Interface — Provides run-time type info. of object

It does not have any method. Classes implementing it do not have to implement any methods. It is used when class wants its instance to be Serialized or Deserialized.

Serializable

Object state is read from memory and written into a file or db.

⇒ Serialization (java.io.* package)

Convert Object → byte Stream.

Done using ObjectOutputStream.

```
public final void writeObject(Object obj)
                                throws Exception
```

⇒ Deserialization

Convert byteStream → Object.

using ObjectInputStream.

```
public final void readObject(Object o)
                                throws Exception
```

Transient

When we do not want an object to be serialized or we can say we do not want to save value of a variable we use transient keyword. It ignores the original value and saves default value

private transient String password;

* no use of using with static or final

HashSet TreeSet.

HashSet	TreeSet
-Implemented using HashTable.	using tree structure
- allows null object	do not allow.
- uses equal() to compare 2 objects	use compare()
- doesnot allows heterogeneous object	Allows.
- do not maintain order	Sorted order
- Fast	- Slow

Palindrome Program.

```
String original="abd" reverse="";
int length= original.length();
for(i= length; i>=0; --i)
{ reverse = reverse + original.charAt(i);
{
if(original.equals(reverse));
```

Streams

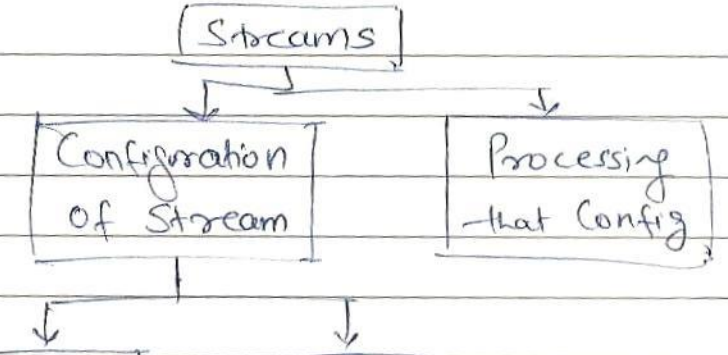
It is a iterator that allow processing on collec. of objects. Used when we want to process bulk objects of collec. If we need to perform operation on collec.

Eg: fetch all objects from collection of list whose value > 15.

```
List<Integer> newlist = alist.stream().
    filter(x -> x>=15).collect(
    Collectors.toList());
```

Q Diff b/w Streams and java.io.Stream?
Java io Stream is sq. of charac. used to be written in a file or read data from file.
Streams are used to perform ope on coll.

Q Diff B/w Stream and Collec?
To represent group of collec as single entity use collection.



Eg. list of even no.

```
= alist.stream().filter(x -> x%2==0).
    forEach(x -> sop(x));
```

Eg: Square of no. in list

```
= alist.stream().map(x -> x*x).
    forEach(x -> sop(x));
```

Processing that Config

If we want to collect Elements of stream after configuration.

Eg: list of evenno. ① Collect(.)

```
= alist.stream().filter(x -> x%2==0).
  collect(Collectors.toList());
```

② Count()

```
= alist.stream().count();
= alist.stream().filter(x -> x%2==0).
  count()
```

③ Sorted()

→ Natural sorting

```
= alist.stream().filter(i -> i>=20).
  sorted()
```

→ customized (asc/desc) order

```
= .sorted((i1, i2) -> i2.compareTo(i));
```

desc →

asc = `.sorted((i1, i2) -> i1.compareTo(i2));`

④ Min and Max

from (asc) (desc)

```
= alist.stream().min((i1, i2) ->
  i1.compareTo(i2)).get();
=
```

(5) forEach()

(6) toArray()

Used to copy elements present in stream to specific array.

Eg:

```
aStream = alist.stream().filter
          (i -> i >= 20);
Object[], intArry = aStream.toArray();

for (Object o : intArray)
{
    sop(o);
}
```

(7) of()

Any Group of value can be converted into stream not just list.

```
Stream.of(1, 11, 111, 1111).forEach(x->sop(x));

String[] name = {"abc", "defe", "xyz"};
Stream.of(name).filter(x->x.length
    >3).forEach(x->sop(x));
```

Parallel Stream.

- It divides code into multiple stream that executes parallely.
- The o/p of this stream is not in sequence
- Uset this only when order of execution doesn't matter. or tasks are independent.

# Singleton Class	Static Class
- It is a pattern.	It is a keyword
- Instance can passed as parameter.	Not
- It can implement interfaces, inherit and allow inheritence	No
- Allows OOP	No
- stored in Heap	Stored in Stack
- Objects can clone	No
- Have Constructors	No

ArrayList	HashMap
- implements List Interface	Map Interface
- stores values and maintain indexes	key/value pair
- order is maintained	Not maintained
- allows any no. of null values	Only 1 null key & any null value
- index-based data structure	Mapped

Similarity:
- Both are synchronized.
- Iterator is used.
- Both use get()
- Both allow null

Static	Final
applicable to nested static class, method, variable and block.	Class, method, variable
- No initialization req. for static variable at declaration.	Mandatory
- Variables can be modified.	No modificat^{n},
Methods	
- Can only access the static members of class and called by static methods.	Final Methods cannot be Overridden.
- Object cannot be created	Final class not inherited.

Q. Can we override static method?

A. No, bez method overriding is based on dynamic binding at run-time and static methods are bonded using static binding at compile-time.

Q. Can we overload static method?

A. Yes, but method signature must be different.

Q. Why String is Immutable?

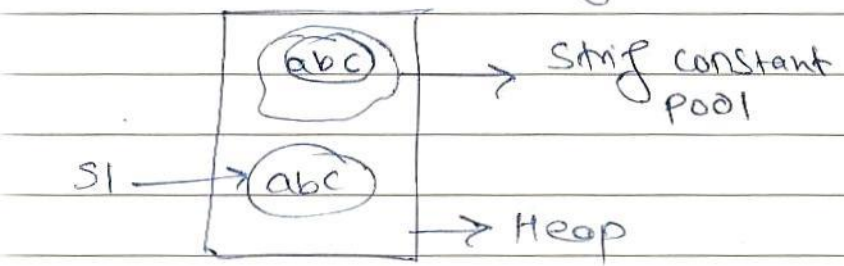
Immutable → unmodifiable.

For security reasons, string is immutable. String can be referenced by many variable at a time. If one variable changes the value, it affect all. Sometimes string store password and username which can't be modified.

String s1 = "abc";
s1 = s1 + "xyz"; (Not Possible)

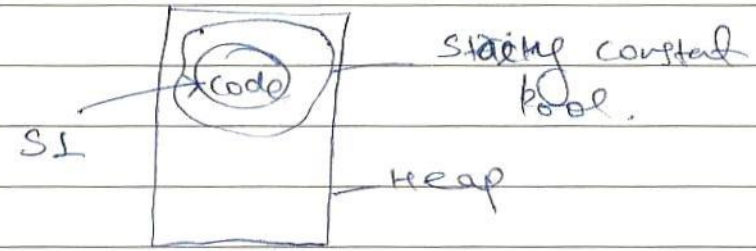
Q. Object creation in string?

A. String s1 = new String ("abc");



⇒ 2 objects are created.

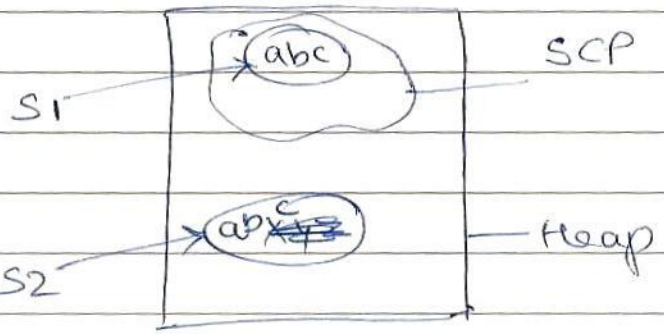
A. String s1 = "code";



⇒ 1 object created.

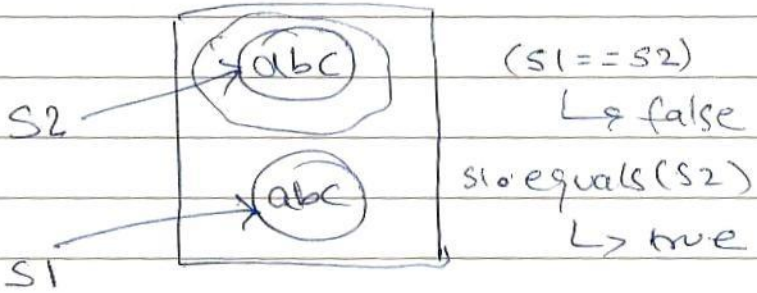
A. ⇒ "new" always created new object > String Contanst pool is free from Garbage collector. Even if no pointer is on an object of pool, it will not be collected by GC.

* String s1 = "abc";
String s2 = new String("abc");



→ 2 objects

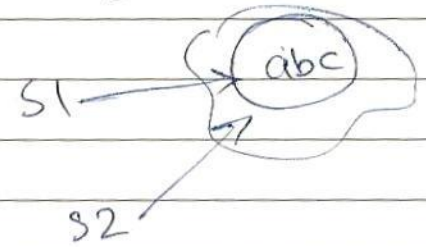
* String s1 = new String("abc");
String s2 = "abc";



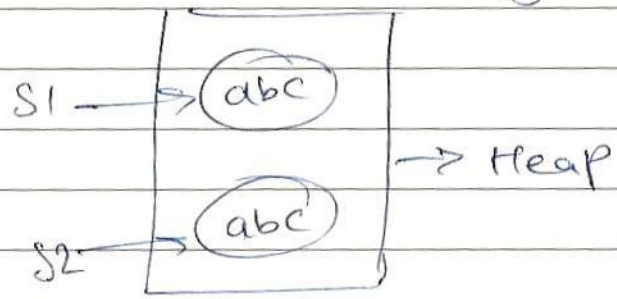
→ 2 objects

== compares memory address.
.equals compares value

* String s1 = "abc";
String s2 = "abc";



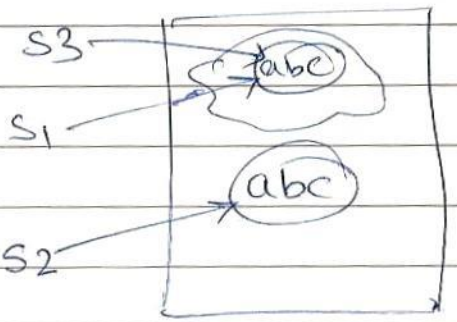
* String s1 = new String("abc");
String s2 = new String("abc");



Intern() in String?
It put String into the String constant pool if it is not present, else the String from the pool is returned.
returned
and a reference to the String object is

String s1 = "abc";
String s2 = new String ("abc");

String s3 = s2.intern();



Q. StringBuffer?

It is used to create mutable String object. It is Synchronized It is less efficient.

Q. String Builder?

It is used to create mutable string object. It is non-sychronized It is more efficient.

** Synchronized i.e. thread Safe. Two threads can't call the same method simultaneously.

Q. Synchronization in Java?

It is the capability to control the access of multiple threads to any shared resource.
Used when we want to allow only one thread to access the shared resource.

* If a super class method is throwing an unchecked exp. the it can be overridden in a sub class with same excep or any of the unchecked excep but not be overriden in a class with checked excep.

The sub class method cannot throw any any checked excep not covered by throws clause of the super class.

* printStackTrace is used to print the detailed info about the excep occured

Q. INPUT: Strng = "AABCACBAD"
OUTPUT: String = A4B2C2D1.

```
Class Example {
  public static void hash(Strng s){

    char c[] = s.toCharArray();
  HashMap<Character, Integer>
      hmap = new HashMap
      <Character, Integer>();

  for(int i=0; i<c.length; i++){

    if(hmap.containsKey(c[i]))
  {
    hmap.put(c[i],
          hmap.get(c[i]) +1);
  }
  else {
    hmap.put(c[i], 1);
  } {
}
```

```
for(Map.Entry e : hmap.entrySet())
{
    SOP(e.getKey());
    SOP(e.getValue());
}
    O/P: → A4 B2 C2 D1.
```

Q Deadlock
When 2 or more thread try to access an object that is acquired by another thread.

Q. How to detect? (Tool)
→ Use ibm portal. It allows us to upload a thread dump & analyze.

→ Use jConsole or Visual VM. It shows which threads are getting locked on which object.

Q:

```
Class Account
{
    double balance;
    void deposit ( double amount)
    {
        balance += amount;
    }
    void withdraw (double amount)
    {
        balance -= amount;
    }
    void transfer (Account from,
        Account to , double amount)
    {
        sync (from);
        sync ( to) ;
        from.withdraw(amount);
        to.deposit(amount);
        release (to);
        release (from);
    }
}
```

① How to avoid deadlock?

① Avoid Unnecessary locks.

② Avoid nested locks.

③* Use Thread.join() - it is used with max time you want to wait for another thread to finish.

④ Use Lock Ordering.

⑤ Lock Time-Out.

Q. Inter-thread Comm.?

Comm. blw synchronized threads It avoids thread-polling. Func used for this are:

- wait(): cause the current thread to wait until another thread invokes notify() or notifyAll().

- notify(): wakes up the single thread in waiting state.

- notifyAll(): wakes up all threads in waiting state.

Q. Context-switching?
current state of the thread is saved to be restored & executed later. By this, multiple processes can share same CPU.

Q. join()? eg. t1.join(): current thread will wait for t1 to complete its task, not terminate
causes the current thread to ~~stop~~ running until other thread finishes its execution. Has void type. and throws InterruptedException

Q. wait() sleep()

wait()	sleep()
- Not a static method	Static
- Called only from Synchronized context.	Not req.
- Releases lock during Synchronizatn	No
- until notify(), notifyAll().	until time expires, or call interrupt()
- multi-thread Shyn.	time-Sync.

Q. Race condition?
When 2 or more threads compete together to get certain shared resource. If thread A is reading a data & thread B is deleting same data
2 types:
-> Read-Modify-Write.
-> Check-then-act.

Q. How to avoid?
=> Synchronization.
=> Mutual Exclusion.

Q. Callable Interface?
A thread doesn't return anything when it completes the execution. So. Java Callable Interfaces allow it to do

public Object call() throws Exception()

Q. Yeild() - when current thread pauses and gives chance to other threads of same or higher priority to run.

Callable	Runnable
It returns an Object	returns void.
can throw Exceptn	can't throw Ex.
Thread can't be created.	can be created

Internal implementatn of HashSet.

When we create an object of HashSet it internally creates an instance of HashMap with default capacity 16.

HashSet uses a constructor HashSet (int Capacity) that represents how many elements can be stored in HashSet. The capacity may increase automatically.

* When we pass duplicate elements to add(), it internally returns False.

Internal implementatn of HashMap?

capacity = no. of bucket * load factor

→ Bucket is element of HashMap array used to store nodes. 2 or more nodes can have same bucket.

index = hashCode(key) & (n-1).

n → no. of bucket / size of array. by default, 16.

SOAP	REST
Simple Object Access Protocol.	Representational State Transfer.
- Tight Coupling	- Loose Coupling.
data exchange format is XML.	supports XML, JSON, HTML, etc.
uses Web Service Definition Lang.	not have any standard language.

Reads Cannot be cached.	can be cached.
For transport can use HTTP or TMO.	HTTP.
Cannot us REST	Can use SOAP

Q. try-with-resources ?

try-with resources implicitly closes all the resources used in the try-block. It was introduced in Java 7 to auto-close resources like file I/O streams, DB Connec., n/w connec, etc

Benefits:
- Avoids resource leaks.
- removes redundant statements.
- improves redability of code

Disadv:
It requires resources to be declared locally within try block not outside.

This issue is resolved in Java 9.

Design Patterns.

They are well-proved soluth for specific problems.

Creational	Structural	Behavioral
- FactoryMethod	- Adaptor	- Chain of Resopnsibility
- Singleton	- Proxy	
- Abstract	- Decorator	- Iterator

Creational patterns are ways of creating objects. used at the time of instantiatn of a class (creation of object).

Factory Method Pattern says just define an interface or abstract class for creating objects but let the subclass decide the type of object. Also known as Virtual Constructor. It promotes loose coupling.

Abstract Factory Pattern says that define an abstract class or interface for creating families of related objects without specifying their concrete sub-classes.

Usage:

When system needs to be independent of how its objects are created, composed and represented.

- works around superclasses.

Structural Pattern are used to provide solutions and efficient Standards regarding class compositions and object structures. They depend on Inheritance & Interfaces to allow multiple objects or classes to work together.

→ They are responsible for how classes & objects can be composed to form larger structures.

Adaptor Pattern converts the interface of a class into another interface based on the requirement.

When an object requires to utilize an existing class with an incompatible interface.

Proxy means an object representing another object. Proxy pattern provides control for accessing the ~~object~~ original object.

Protective Proxy - control access to real subject.

Virtual Proxy - used to instantiate expensive obj

Caching Proxy - cache expensive obj to real subject

Remote Proxy - distribute object comm.

Smart Proxy - implement log calls and reference to the object.

Behavioral Design Pattern is concerned with the interaction and responsibility of objects. The interaction b/w objects should be easy and still loosly coupled.

Chain of Responsibility Pattern, sender sends a request to a chain of objects. The reques can be handled by any object in the chain.
- It reduces coupling.
- Adds flexibity while assigning the responsibi
- Allows set of classes to act as one.

OOPS Concept.

1. Object: Object can be defined as an instance of object. class. An object contains an address and takes space in memory. Any entity that has state and behaviour is object.

2. Class: Collection of object. It can be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

3. Polymorphism: We can perform a sigle action in different ways.

→ Compile-time Polymorphism
also known as Static poly. achieved by function overloading.
Method Overloading is when there are multiple functions with same name but different parameters or type.

```
class Helper
{
    static int Multiply(int a, int b)
    {
        return a * b;
    }
    static double Multiply(double a, double b)
    {
        return a * b;
    }
    static int Multiply(int a, int b, int c)
    {
        return a * b * c;
    }
}
```

→ ~~Run-Time Poly.~~

→ Run-time - Poly.
also known as Dynamic Method.
achieved by Method overriding.
When sub-class has same name method as of super-class, same arguments & return type.
An overridden method is called through the reference variable of superclass.

Upcasting:- If the reference variable of parent class refers to the object of Child class.

```
class A{}
class B extends A{ }
  A a = new B();
```

Example:

```
class A{
    public void m1()
    {
        sop("Parent Class");
    }
}
class B extends A
{
    public void m1()
    {
        sop("Child Class");
    }
}
P.s.v.m(String args[])
{
    A a1 = new A();
    a1.m1();
    A a2 = new B();
    a2.m1();
}
```

Output:- Parent Class
Child Class

11. Abstraction
A process of hiding the implementation details and showing only functionality.
2 ways of achieving:
- Abstract Class
- Interface

Abstract Class

- declared with abstract keyword.
- Can have abstract & non-abstract method which should be implemented in extended class.
- cannot be instantiated.
- can have constructors & static method
- can not have final methods. Can have final variable.
- Abstract method do not have implementation.
- we cannot define an object, object is defined at Runtime

```
abstract class A
{
    abstract void run();
}
Class B extends A
{
    void run() {
    SOP ("runn"); }
}
P.S.V.M.
{ A a = new B();
  a. run(); }
}
```

O/P => runn

Interface.

It is a blueprint of a class. It has static constants & abstract methods. All the variables are public, static, final by default. Object is not created, not even at runtime.

Example –

```
Interface I
{ int A();}
Class A implement I
{ public int TA()
  { return SOP("A");
  }
}
P.S.V.M.
{ I i = new A();
  i. m();
}
```

O/P – A

multiple inheritance can be achieved in java using Interface

Example:

```
Interface Printable
{ void print(); }

interface Showable
{ void show(); }

Class A implements Printable, Showable
{
    public void print()
    { sop("print"); }
    public void show();
    { sop("show"); }

    p.s.v.m ( )
    {
        A obj = new A();
        obj.print();
        obj.show();
    }
}
```

O/P → Print
Show

1) Encapsulation: Wrapping code and data into a single unit.
It can be created by making all data members of class private. Now use getter and setter.

```
public class Student
{
    private String name;

    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
}
```

Another class
```
Student s = new Student();
s.setName("abc");
```

6. Inheritance - When 1 object acquires all the prop and behaviour of parent obj Used for Method Overriding and code reusability. ~~1. B b = new B();~~

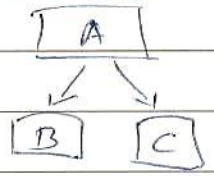
-Single Inheritance.

→Multi-level

```
Class A
{  void X()
   {  sop("X"); }
}

Class B extends A
{  void Y()
   {  sop("Y"); }
}

Class C extends B
{  void Z()
   {  sop("Z"); }
}

P.S.V.m( )
C c = new C();
c.Z();
c.Y();
c.X();
```

up → Z Y ~~X~~

3. Hierarchical.



7. Coupling.

Coupling refers to knowledge or dependency of another class. Strong coupling is when a class has detailed information of another class. Loose Coupling is when a class has least dependency. In java, public, private, protected are used to display visibility levels. Interface is an eg of weak coupling as it has no implementation

8. Cohesion

It refers to the level of a component which performs a single well-defined task. Weak Cohesive method will split the task eg. java.util. package. Highly cohesive eg, java.io package as it has related classes & interface

Association

The relationship b/w the objects.

- One - to - One
- One - to - Many
- Many - to - One
- Many - to - Many

Abstract	Interface
- Can have abstract & non-abstract method.	abstract method, static, default
- Can extend only 1 abstract class	Can implement many interfaces.
- can be public, private, protected	public
- contains constructor	No
Fast	Slow
- can have instance variable	no instance variable
- non-final varible & final	by default, static -final variable.
Overloading	Overriding
- Defination	Defination
- Gold or extend the method behavior	Change
- compile time poly	Run-time poly.
may or may not need inheritance	needs inheritance

Q Servlet?

It is a server side technology to extend the capability of web servers by providing support for dynamic response and data persistance.

javax.servlets & javax.servlet.http packages provide interface & class to write your own servlet

All servlet must implement 'javax.servlet.Servlet.

- extend HTTPServlet class

GET	POST
(2048 characters) limited data can be sent	Large data.
Not secured	Secured
Idempotent	Non-Idempotent
Cachable	non-cach
data is sent with the URL.	

Q Life cycle of Servlet?

Loading → Instantiated → Initialization → Request → Destroye

Q Cookies? → Cookies are text data sent by server to the client and it get saved at the client local m/c

Q. Steps to connect to the dB?

- Registring the driver class.
- Creating connection
- Creating statement
- Executing Query
- Closing Connection.

```
try {
    Class.forName("      ");
    Connection con = DriverManager.
        getConnection(url:"      ",
        user:"  ", password:"  ")
    Statement st = con.createStatement();
    SOP("  Connected Created");
}
Catch (ClassNotFoundEx e)
{  e.printStackTraces(); }
catch(SQLEx e)
{ e.printStackTraces(); }
```

* Type of Decorator in Angular?

1. Class Decorator
   - @Component, @NgModule.
2. Property Decorator
   - @Input and @Output
3) Method Decorator
   - @HostListener
4. Parameter Decorater
   - @Injectable.

// MySQL //

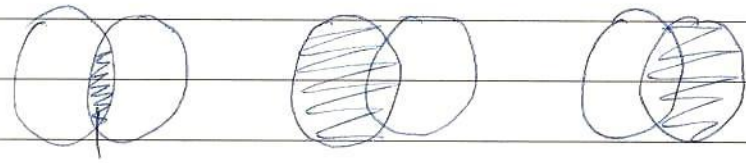
Constraints can be used to specify the limit on data type of table. specified while creating & altering. Eg. NOT NULL, UNIQUE, PRIMARY KEY, etc

Foreign key refers to the primary key of another table, maintains referential integ.
PRIMARY KEY (ORDER_ID), FOREIGN KEY (PERSON_ID) REFERENCES Persons(PERSON ID)

Order by used to sort the data.
Select id from Table where cond. order by id asc/desc.

# Group by used to organise data
select COUNT(ID), country from Cust where Group by Country;

Select T1.c1, T1.c2, T2.c1 from T1 Inner join T2 on T1.c = T2.c



inner join Left join Right Join

DELETE	TRUNCATE
- delete a row in the table	- deletes all rows in a table.
- DML	- DDL
- Rollback possible	- Not possible
- Slow	- Fast

// Unique key, uniquely identifies a single row in the table. Multiple values are allowed. Null values are allowed. Duplicates not allowed.

// INDEX, performance tuning method. Allow faster retrival of records from the table. Creates an entry for each value.

→ Unique Index do not allows the field to have duplicate index.

→ Clustured Index reorders the physical order of the table and searches on the basis of key value. Each table can have 1 clustered index.

→ Non Clustured do not alter the physical order and maintain logical order of data. Table can have many such index.

Normalization is process of organising data to avoid duplication and redundancy

1NF → Each table cell should have a single value and all records should be unique

2NF → should be 1NF. and should have single column primary key. (divide 1 table in 2)

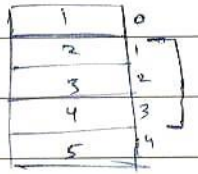
3NF → should be 2NF and must not have any transitive func. dependency (divide table in 3 table).

BCNF → Should be 3NF and can have only 1 candidate key.

Q. Find 3rd highest salary?

Select Salary from Emp order by Salary desc Limit 2, 1.

Limit → 1st value is starting index
→ 2nd value is count.



Limit 1, 3

Q. Find duplicate rows from a table

Select *, COUNT(empId) from Emp Group by empId having count (empId) > 1.

Q. Copy all rows of a table to new table

Create table empl as select * from emp;

Q. fetch list of employee working in same dept.

Select DISTINCT e.ID, e.Name, e.Dept from Empe, Empel where e.Dept = e1.Dept and e.ID != e1.ID

Q. Internal implementation of HashMap

Insertion.

① Get key, value pair.

② Using key, create hash code.

③ Do bitwise & to get index:

Index = hashCode(key) & n-1

④ Create a node object at index

| HashValue | key | Value | Next Node Pointer |

* Buckets is one element of HashMap array index.

Fetch the data:

① Get key.

② Create Hash Code

③ Calculate Index and compare the first element's key with given key. If both are equal then return the value, otherwise check for next element if exists.

Composite key is a combination of 2 or more columns in a table that can be used to uniquely identify each row in table. Cannot be null.

Candidate key is the minimal set of attributes that can uniquely identify a row. Must contain unique value. It can contain NULL values. A table can have multiple candidate key and atleast 1 candidate key. Primary key can be a candidate key

Marker Interface is used as a tag interface that inform the compiler by a message so that it can add some special behaviour to the class implimenting it. It is useful if we have info. about the class and that info never changes.

Built-in Interface are:-

1. Cloneable :- It generates a copy of an object with different name. To use clone(), we must implement Cloneable interface else it will throw ClassNotSupportedException.

2. Serializable: (1) Serialization → Object to byte Stream i.e. object state is read from memory and written into file or DB.

(2) Deserialization → Byte Stream → Object i.e. byte state reading from a file & written back into memory.

3. Remote interface: It mark an object as remote that can be accessed from another machine. It is used with Remote Method Invocation.

Types of Garbage Collectors.

1. Serial GC:- It works with a single thread. It will freeze all the applicath thread and create a single thread to perform GC until GC completes.
Disadv: Cannot be used for multi-thread application.

2. Parallel GC: → It is default GC in Java 8. Also called Throughput Collector. It also freezes the running threads of app. but it uses multiple threads to perform GC. Adv. We can mention →
No. of threads, GC can use.
Max. pause GC can take.
Disadv: It pauses the app during minor ope. also.

3. CMS GC :→ The GC uses multiple threads to scan the heap memory consistently to the mark objects that are unused and then sweep the marked objects. App is paused in 2 cases only.

1→ while performing GC, if there is any change in heap memory.

2→ while marking the referenced object.

Disadv

It used more CPU.

4. G1 (first) GC :→ Introduced in JDK7

It is default GC for Java 9. It is used for heap memory larger than 4GB.

G1 partitions the heap space in multiple equal sized regions. It mark the heap region which has ~~info~~ objects that are in use throughout the heap. That is how it has the info about regions that contain most useless objects and garbage collection collects first perform collection on that region only.

Statement	Prepared Statement
1 executes the query only once.	- when we want to execute multiple times.
- It is static, cannot pass parameters at runtime.	- Dynamic, can pass parameters at runtime.
- cannot read or write binary data.	- can read or write binary data.
- performance of exe. is slow	- fast
- Do not prevent SQL injection	- prevents
- Uses textual protocol for comm.	- Binary protocol.

// Create View in SQL.

CREATE VIEW _VIEWNAME_ AS SELECT C1, C2, FROM _TableName_ where COND;

// Stored Procedures : They are a sequence of SQL St. that access the relational DBMS. It can be stored in DB server.

Procedures are reusable SQL codes that we store in DB. We can direcly call procedures instead of writing query again & again.

Prepared St:- They are queries that contain the placeholders instead of actual values. It cannot be stored in Database.

Functions are reuseable codes which runs certain SQL commands and return an value.

```
CREATE PROCEDURE procedureName (Parameter List)
  BEGIN
      body;
  END
```

Adv. of MySQL.

- Flexibility
- Performance
- Indexing & Searching
- Query Caching
- Replication
- Security

Java Generics.

It is a way to store a specific type of objects only in a class, method or interface. It make code stable by eletecting the bugs at compile time.

Advantages:

→ Type-Safety.
→ Type-casting not req.
→ Compile-Time checking

Eg: `ArrayList<String> list = new ArrayList<String>();`

Static keyword

Only one instance of static member is created and shared across all instance of the class.

`int a=1;`

shared → Object1, Object2

⇒ Static Variable

When we declare a variable, single copy of that var. is created and shared among all instance of class. or diff class.

- It is stored in a heap memory.
- We can access them directly using class name.
- We don't need to initialize

⇒ Static Methods.

- They are resolved at compile time
- Cannot be overridden
- Cannot use 'this' or 'super' keyword

Instance Method can access:
- Instance Var.
- Instance Method
- Static Var.
- Static Method.

Static Method can access:
- Static Var.
- Static Method
- Need object reference to access inst. var & method

⇒ Static Block.

We use static block when the static variable needs some additional logic except the assignment or it is error prone and need exceptional handling.

A class can have multiple static blocks.

Resolved in the same order in which they are present.

⇒ Static Class

* Nested classes that are non-static are inner classes

Nested classes which are static are static nested classes.

- Static class can access to static members of outer class only.
- Increases encapsulation, readability

HashMap	Concurrent HT.
- Non-synchronized.	Synchronized
- High Performance	Low
- Allow null	Do not allow null
- throw excep during Iteration. (runtime) excep	No excep, if 1 thread is Iterating the object other thread also Iterate

Q. HashMap Implementation [DZONE website]

A HashMap is a map used for store mapping of key-value pair. It works on Hashing principle. It is a data structure which allow us to store object and retrieve object in constant time O(1) if we know the key.

In Hashing, hash func. is used to link key and value. Hash map maintains an array of the bucket. But when we store or retrieve any key-value pair, it calculates the index of the buck for each operath. By using 'key', the hash value is calculated using hash(key), private method of HashMap if hash value is too large then it converts the value into a smaller hash value.

Note: If the hash value of key object returns the integer that is greater than array size then ArrayOutOfBoundException will occur. So, Hashmap reduces the hash value between 0 and n-1.

index = hash(key) & (n-1)

This index is used by HashMap to find bucket location and can never generate exception as index value is always 0 to n-1.

Q. How put() works?

Firstly, all key objects are checked if there is null. The hashcode for null is always 0(zero). So the value is stored in t[0] position. Then hash value is calculated using hashcode(). Then index is calculated using hashcode value. for storing

Entry objects.
The hashcode() method might return very high or low hash code value, so the object's hash code is passed through hash() function to bring hash value in range.

indexFor(hash, table length) gives exact index position.

Q. How Collisions are Resolved?
If 2 objects have same hash code value the objects are stored in Linked List form. When an object needs to be stored in a particular index, Hashmap checks whether it is empty or not, If not, its next attribute is checked & the object becomes the next node in Linked List.

Q. Concurrent HashMap Implementation
It allows concurrent access on the map. Part of map or segments is only locked while adding or updating map and not while reading the value.
Multiple threads can access at the same time
Conc.HM is internally divided in segments of size 32., so at max 32 threads can work at a time. Uses Reentrant Lock - Atomic Ope, Lock-free, Scalable

Q. ArrayList Implementation
When we create an ArrayList by default constructor, the JVM creates arraylist in heap memory with default size 10. But arraylist is flexible in size.

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

We can initialize the capacity during creation of Arraylist, but if more ele. are added, size will increase.

```
ArrayList<Integer> list = new ArrayList<Integer>(15);
```

The size of arraylist grows automatically on the basis of load factor and current capacity. By default the load factor is 0.75f

Threshhold = (Current capacity) * Load factor

By default,

= 10 × 0.75

= 7

Arraylist increases the capacity after each threshold. If the size of the current element is greater than the threshold. JVM creates a new Arraylist and copies the old value in new one with new size.

It is a space & time consuming process

HashSet Implementation

HashSet stores unique ele. It is backed by a HashMap. It doesn't maintain insertion order. It is not thread safe.

Whenever we create an object of HashSet, it internally creates an object of HashMap. This reference variable gets initialized when the HashSet() constructor is called.

HashMap stores the entries in key-value pairs but HashSet is a set of unique elements. Hence, hashset are made key-value pair using a dummy object called PRESENT.

Initial capacity of Hashset is 16 and load factor is 0.75f.

If the element is not present in the backup map then add() method adds the element. If the ele is present, the method call leaves the set unchanged and add method returns false else true.

Q. Why wait(), notify(), notifyAll() are in Object. Class not in Thread class?

In Java, object is shared among the threads and facilitates inter-thread comm. Threads have no knowledge of the status of other threads. They just need to call the methods on the object

Q How to achieve thread-safety without using synchronization.

→ Volatile keyword.
→ Atomic Variable
→ final keyword.

Q Concrete Class.

A class that has an implementation for all its methods. It can extend abstract class, implement interface
It is a complete class. It can be instantiated.

Q Instance Variable

They are defined in class but outside the body of methods. They are created when an object is instantiated and they are accessible to all constructors, methods and blocks in class.
These variables are destroyed when the object is destroyed. They are used to reversing memory of data that the class needs. If not assigned, these variables are given a default value.
It include access modifiers. It is not neccessary to initialize the variable before use.

Q Local Variable

They are defined within the programm of blocks. These variables are created when a block, method or constructor is started and variable will be destroyed once it exists the block, method or constructor.
They are used to decreasing dependencies b/w components. i.e decreasing complexity

These variables must be assigned with a value by code. It is necessary to initialize local variable before use. It does not include any access modifiers.

Wrapper Class.

The wrapper class provides the mechanism to convert primitive data type to object called boxing and object to primitive data type called unboxing.

Uses:

1. Serialization
2. Synchronization
3. To change the value in method.
4. java.util. package provides the utility class to deal with objects.
5. Collection framework works on objects

Angular Pipes.

Pipes allows its users to change the format in which data is being displayed on the screen. They do not alter the data but change how they appear to user.

- Can be defined using '|' symbol.
- Can be chained with other pipes
- Can be provided with arguments by using colon (:) sign

Some predefined pipes are → DatePipe, UpperCase, LowerCase, etc

DatePipe: {{ dateToday | date | uppercase}}

Custom Pipes - convert the data in the formate that you desire. Steps:-

1. Create a TS Class with export key word.
2. Decorate it with @Pipe and pass the name property to it.

3. Implement the pipe transform interface in class.
4. Implement the ~~pipe~~ transform method
5. Return the transform data with the pipe
6. Add this pipe class to the ~~data~~ declaration array of the module

ng g pipe <name of pipe>

Pure Pipes.

Pipe that use pure functions. This pipe doesn't use any internal state and o/p remains the same as long as the parameters passed stay the same.
Angular calls the pipe only when it detect a change in the parameter being passed.
Single instance of pipe is used in all comp.

Impure Pipes.

For any change detection cycle, impure pipe is called regardless of the change in I/p fields. Multiple pipes are created for these pipes. I/p passed can be mutable.

Digest Cycle

This process is a func. that repeatedly checks for ~~e~~ changes in the variable and their values for a scope watchers.
With every ~~process~~ scope watcher you create a corres. watcher func for particular scope will be added to the digest loop at the same time itself.
The func. executes automatically every time the cycle runs, comparing the watcher's old value to new value. The cycle runs to check the changes until all watcher's stop changing.
This process is Digest Cycle.
Angular js uses concept of watcher and Listener.

A watcher is a func. that returns a value being watched for a scope. We can track model properties, component state on a scope, computed values, third party components. If the returned value is diff from the previous value then angular calls listener.
The listener is usually used to update the UI.
$scope.watch is used to track the scope variable and value.

The listener can change the content of another variable or set content of an HTML ele or anything.

ENUM.

An enum class represents a group of constants like final variable.

```
enum Level {
    Low, Medium, High}
Level myvar = Level.Low;
```

```
Class A{
    void m1()
    { sop("A"); }
}
Class B{
    void m1()
    { sop("B"); }
}
public class T{
    P.S.V.M( )
    { B b = new B();
      b.m1(); }
```

o/p → B

~~sop(b)~~

```
A a = new A()
a.m1()
```

o/p → A

```
A a = new B()
a.m1()
```

o/p → B

```
B b = new A()
b.m1
```

o/p → compile-time error

Fail Fast & Fail safe

Iterators in java are used to iterate over the Collection objects.

Fail fast iterators immediately throws ConcurrentModification Exception if there is structural modification of the collec.

Eg: ArrayList, HashMap.

St. modificath means adding, removing any element from collec. while a thread is iterating over that collection.

Fail-safe doesn't throw any expections if a collection is st. modified while iterating over it. Because they operate on the clone of collecth not on original collec.

Eg: ConcurrentHashMap, CopyOnWriteArrayList

→ Fail-fast requires low memory.
Fail-safe requires more memory.

Cloning.

Cloaniy is a way to create exact copy of an object. clone() is used. Cloneable Interface must be implem

Shallow Copy

When we do a copy of an entity to create another entity, the changes in 1 entity are reflected in others New memory allocation doesn't happens. only reference is copied. It is less expensive.

Deep Copy

When we do a copy of an entity to create another, the changes in one entity are not reflected in other. New memory allocation happens. Reference is not copied. Each entity has its own independent references. Highly expensive

⇒ An enum class have attributes & methods
Enum constants are public, static, final.
It cannot create objects.
It cannot extend class.
It can implement interface.

Solid Principle – to create more maintainable, understable and flexible s/w.

1. Single Responsibility
A class should only have one responsibility
- fewer test cases
- Less dependency
- easier to search.

2. OPEN/CLOSED
classes should be open for extension & closed for modification.

3. Liskov Substitution
If class A is a subtype of class B, we should be able to replace B with A w/o disrupting the behaviour of our pgm

4. Interface Segregation
Larger interface should be split into smaller ones. By this we ensure that implementing classes need to be concerned for the methods required

5. Dependency Inversion
Refers to decoupling of s/w modules.
Highlevel modules should not depend on low-level module. Both should depend on abstraction.

flatMap (Stream API)
Stream flatMap() is used to flatten a Stream of Collections to a Stream of objects (String / Integer).

⇒ finalList = [[1, 2, 3], [3, 6, 9], [2, 4, 5]]

```
List<Integer> intList = finalList.stream()
        .flatMap(list -> list.stream())
        .collect(Collectors.toList());
```

Sop (intList);

O/p → [1, 2, 3, 3, 6, 9, 2, 4, 5]

Stream API

1. Count of M and F emp in an org.

= elist.stream().collect(Collectors.groupingBy(Emp::getGender, Collectors.counting()));

2. Name of all dept

= elist.stream().map(Emp::getDept).distinct().forEach(System.out::println);

3. Avg age of M & F emp

= elist.stream().collect(Collectors.groupingBy(Emp::getGender, Collectors.averagingInt(Emp::getAge)));

4. Name of emp joined after 2020

= elist.stream().filter(e → e.getYOJ() > 2020).map(Emp::getName).forEach(System.out::println);

Synchronized HM	Concurrent HM
- At object level.	At segment level.
- Lock for read/ write operation	Lock for write ope.
- ConcurrentModification Exception is thrown if a thread tries to modify an existing SyncHM which is being iterated.	No Excep.
- Single thread can modify the map & block other threads	Multiple thread can modify the map.
- Bad performance	Better Perform-

Type of References in Java

1. Strong Ref. :

MyClass obj = new MyClass();

GC only when obj points to null

obj = null;

2. Weak Ref:

If JVM detects weak ref, object is marked for GC. Eg. DB Connection.

3. Soft Ref:
Even if an object is free for GC, JVM do not allow it to be collected until it is in need of memory.

4. Phantom Ref:
Before removing the obj from memory, JVM puts them in a 'reference queue', after calling finalize() on them.

CountDown Latch
A synchronization aid that allows one or more threads to wait until a set of ope. being performed in other threads completes.
Creating an object of CDL by passing a count. The thread which is dependent on other threads to start, waits until every other ~~other~~ thread has called countdown
All threads which are waiting on await() proceed together once count reaches to zero.

2. Merge 2 sorted array into one
Input → B[N] , C[M] = A[M+N]

```
i=0 , j=0
for k=→ 0 to (N+M-1)
if (i==N)
    A[k]= c[j]
    j+=1
else if (j==M || B[i]<=c[j])
    A[k] = B[i]
    i+=1
else
    A[k] = c[j]
    j+=1
```

Types of Functional Interface?
1. Consumer: It accepts 1 argument. It has no return value.
Consumer<Integer> consumer = (value) -> sop(value);
Bi-consumer takes 2 arguments

2. Predicate: It accepts single value.
It returns a boolean answer.

public interface Predicate <T>
{ boolean test (T t); }

3. Function: It accepts a single value.
It returns a value.

4. Supplier: Does not take any value.
It returns 1 value.

public interface Supplier <T>
{ T get(); }

Eg: Fibbonacci Series.

SPRING BOOT

@Controller is used to declare common web controllers which can return HTTP response. mostly used with @RequestMapping

@RestController is used to create controllers for Rest API which can return JSON.

Q. Static Block | Instance Block

- Executes during Class Loading
- Can use static variable
- execute only once during program execution when the class loads into memory

- Executes during class instantiation.
- static & non-static
- whenever there is a call to constructor.

Q Life cycle of Thread?

(1) New - newly created Thread
(2) Runnable - either Ready or Running.
(3) Blocked - waiting to acquire lock to enter or re-enter a synchronized block.
(4) Waiting - waiting for other thread to complete

without any time limit.

⑤ Timed-waiting. - waiting for other thread to complete for specific period.

⑥ Terminated - execution has completed.

Q How to kill a thread?

① Periodically checking a boolean flag.
Initially the flag is false, when we want to stop the thread set flag = TRUE.
Use volatile boolean flag to make our code thread safe.
or we can use atomic boolean class.

② By interrupting the thread.
Whenever Thread.Interrupted() is called it sets a flag known as interrupt status = TRUE and thread stops performing.

Q Volatile keyword.
It is used so that the value of a variable can be modified by diff threads. It is used to make class thread safe. The volatile keyword can be used with primitive type or obj.
It does not cache the value of the variable and always read the variable from the main memory. It gaurantees visibility and ordering.
★ It is an alternative way of achieving synchronization in java.
- can only be used with variable.

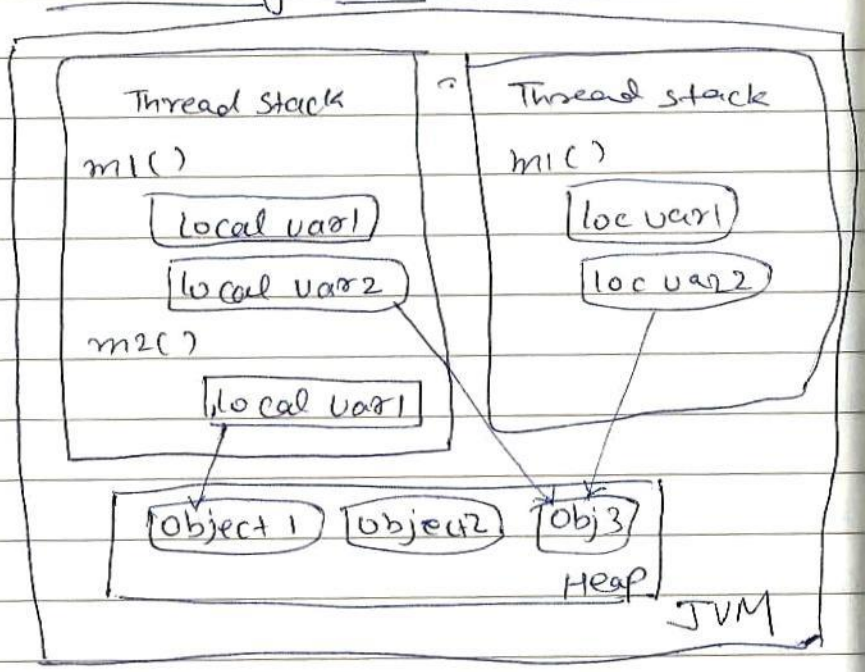
Volatile	Synchronization
- used for variable.	- Blocks and methods
- thread can't be put in wait state	- can be
- Improve thread performance	- degrade
- synchronize value of 1 variable at a time.	- all variable.
- not subject to compiler optimizath	- subject to compiler optimization.

Atomic Operation

~~Small~~ operation which either perform full task or not perform at all.

```
private AtomicInteger counter = new
                        AtomicInteger(0);
-> counter.getAndIncrement();
```

Java Memory Model.



- Each thread running in JVM has its own thread stack which contains info about methods the thread has called.
- Thread stack has all local variables for each method.
- A thread can only access its own thread stack. Local var created by a thread are not visible to other threads

- Heap contains all the objects created in Java app., regardless of which thread created the object.
- ~~Objects~~ Local variable may contain reference to an object
- Objects on the heap can be accessed by all threads that have a reference to that object and all the member variable of that object.

Equals and Hashcode

```
@Override
public boolean equals(Object object)
{
    if(Object == this)
    { return true; }

    if(object == null || object.getClass()
                    != this.getClass()
    { return false; }

    Person person = (Person) object;

    return person.name.equals(name or this.name)
        && person.id == pid;
}
@Override
public int hashcode()
{
    return Objects.hash(name, id);
```