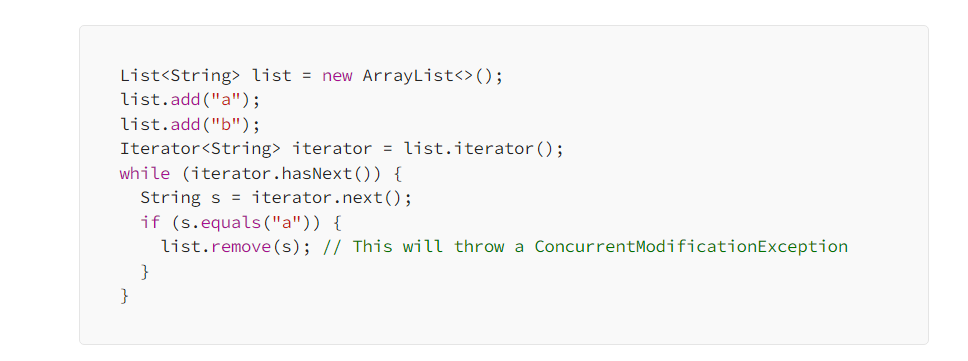
# Concurrent Modification Exceptions:

It’s a Run time exceptions that occurs when we try to modify the Collection i.e

1. Vector
2. List
3. Set
4. Map

If we try to modify, update, or delete the value at the time of iteration then we will get the run time exception that is Concurrent Modification Exception and this scenario is called **FAIL FAST.**

****To avoid ConcurrentModificationExceptions, you can:

1. Use thread-safe collections such as Vector or CopyOnWriteArrayList instead of non-thread-safe collections like ArrayList.
2. Use synchronization to ensure that only one thread can modify the collection at a time.
3. Use an iterator that supports the remove() (iterator.remove()) method and call that method instead of modifying the collection directly.

## How Remove() will work internally in iteration:

Iteration is a interface with 3 method we have => Next(), hasNext(), remove();

# Explain Solid Principles In java:

As we know at some point of time the application will grow we can reduce their complexity and save ourselves a lot of headaches further that can create the use and we can form the reusability components

**Single Responsibility:**

A class should only have one responsibility

Let say if we have a class BOOK then this class contains all the data fields method and logic related to book only

**Open for extension closed for Modifications:**

We are stopping the class modification here and give us a chance to do a bug free code

Let say we have payment interface and we have an abstract method pay

Different classes we have

1. UPI
2. Credit
3. Debit
4. Cash

All these are the implementing class of the pay interface so we are open for extension if we have to add new payment strategy then we will create one more class like bitcoin payment and implement the pay method accordingly

Main class logic of this {

Payment pay;

setPaymentStrategy(Payment pay){

this.pay = pay;

}

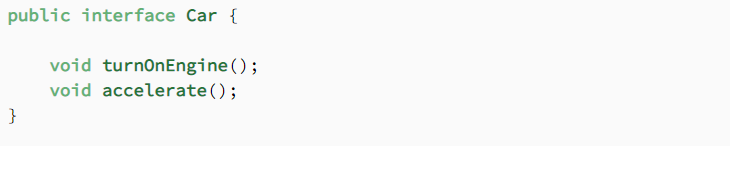
}

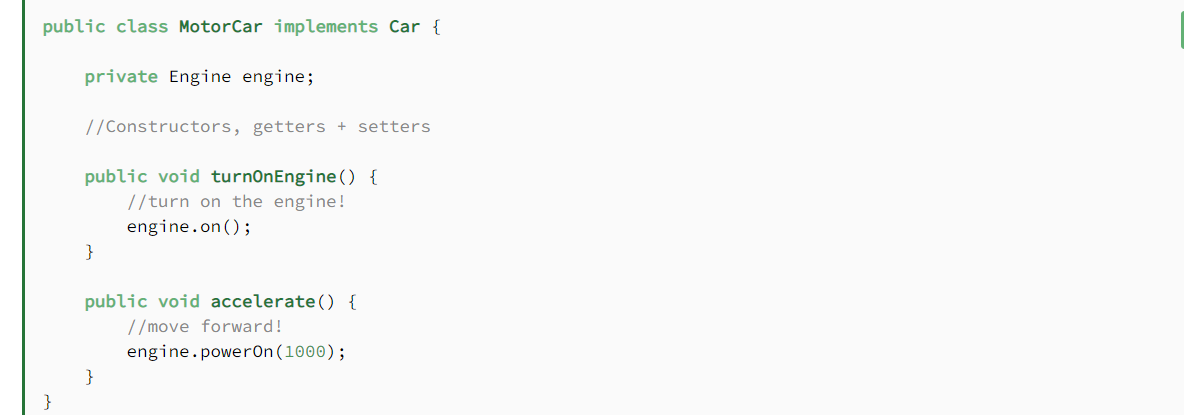
StrategyPattern{

setPaymentStrategy(new paypalPayement(10000));

Liskov Substitution:

If class A is a subtype of Class B we should be able to replace B with A without disrupting the behaviour of our program





Interface Segregations:

The I  in SOLID stands for interface segregation, and it simply means that **larger interfaces should be split into smaller ones. By doing so, we can ensure that implementing classes only need to be concerned about the methods that are of interest to them.**

For this example, we’re going to try our hands as zookeepers. And more specifically, we’ll be working in the bear enclosure.

Let’s start with an interface that outlines our roles as a bear keeper:

# Immutable Classes in java:

The classes where we cannot change the value of it

Once the person object is created then the state of that object cannot be changed

If we are going to change the state of it and there is the reference of other classes as well then we need to make sure that it should return the defensive copy of that particular object so that we can’t change the state of our immutable class

Let take an example person:

Class Person{

Private Int id;

Private LocalDateTime ldt;

Private Address address;

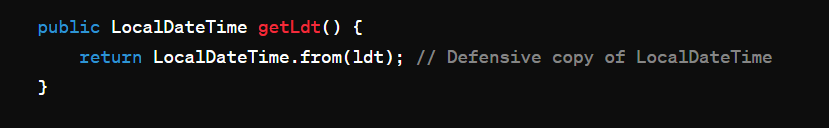
Public person(Int id, LocalDateTime ldt, Address address){

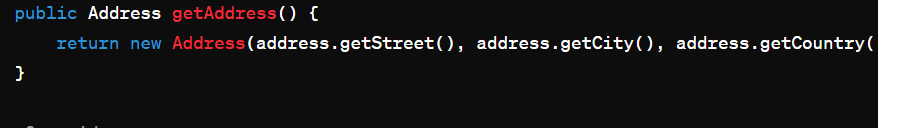
This.id = id;

this.ldt = LocalDateTime.from(ldt) // Defensive copy of LocalDateTImel

this.address = new Address(address.getStreet(), address.getCity());

}





That’s how we will have the defensive copy of other reference classes.

For Collections we will have defensive copies like

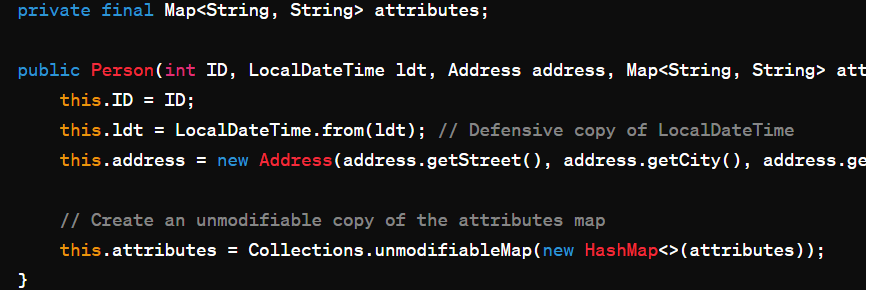
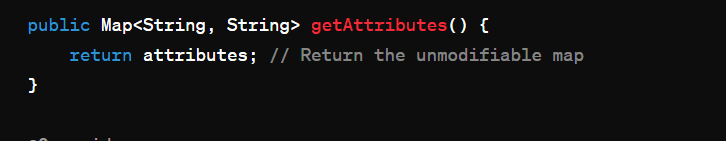
List<String> hobbies;

Constructor(){  
hobbies = Collection.UnmodifiableList(hobbies);

}

Here we will have the defensive copies;

For map :

# Query to find the employees in Each department:

SELECT emp\_department, Count(\*) from employees GROUP BY emp\_department

# Java8 changes in memory:

JVM defines various run time data area which are used to execution of a program. The data areas of thread are created during instantiations and destroy when the thread exists

Note: JVM destroy when JVM exits.

JVM memory divides in the following

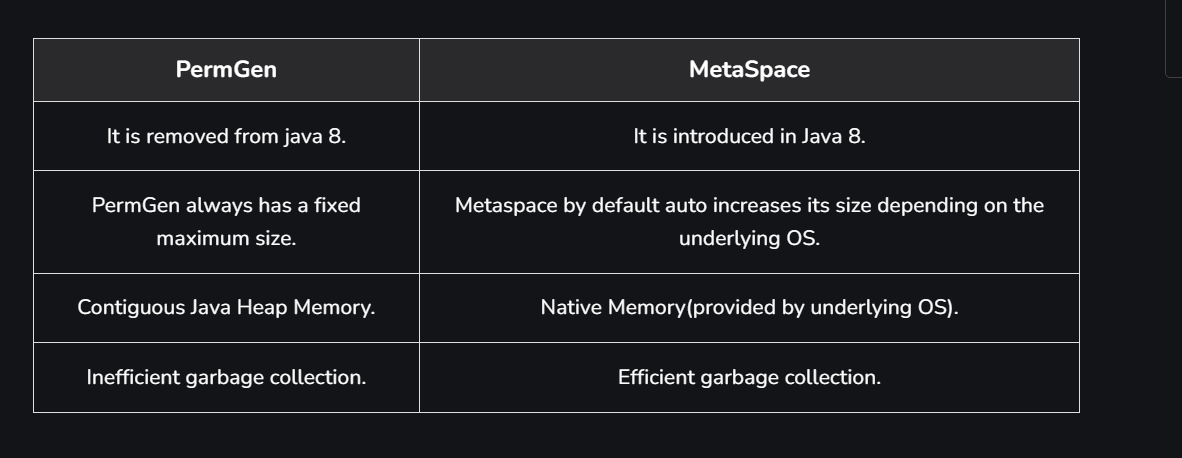
* Heap Area
* Method Area
* JVM Stack
* Native Method stack
* PC Register

All the objects are stored into the heap area The heap are is created when the JVM Starts.

The heap is generally divided into the two parts:

1. Young Generation – All the objects are stored in this generation only When the memory is fill then garbage collection happens this is called Minor Garbage Collections.
2. Old Generation – All the long lived object which has survived many garbage collections is stored in this area

JVM also contains Permeant generation memory apart from heap memory or “PermGen”



# **Write a program to find even numbers from a list of integers and multiply with 2 using stream java 8**

List.stream().filter(x -> x % 2 == 0)

.map(x -> x \* 2).collect(Collectors.toList());

# Why we need hashcode if we have equal what are the contract terms between them.

The hashcode() in java use to generate a unique integer representation for an object based on it’s content. The primary purpose of hashcode() is to support hashing based data structures such as HashMap<>, HashSet and HashTable.

While the **equals()** method is used to compare two objects for equality based on their contents, the **hashCode()** method is used to efficiently distribute objects across buckets in hash-based data structures. Here's why we need both **hashCode()** and **equals()** methods:

Hash-based data structures use the hash code of objects to determine their storage location in the underlying data structure (e.g., buckets in a hash table). When retrieving an object from the data structure, the hash code is used to quickly locate the bucket where the object is stored, thereby improving retrieval performance

To ensure consistency between **hashCode()** and **equals()** methods, the contract between these two methods must be maintained. According to the contract:

If two objects are equal according to the **equals()** method, their hash codes must be equal as well

if two objects have the same hash code, they may or may not be equal according to the **equals()** method. However, it's more efficient for hash-based data structures if objects that are considered equal also have equal hash codes

# What if heap memory is full and we need to optimise the code in java to save the heap memory what can we do?

1. Optimise Data Structure we need to user :
   1. Choose DS that minimize memory usage for the given use case For example use arrays instead of ArrayList If the size is fixed and known in advance as arrays have a smaller memory foot print
2. **Reduce String Usage:**
   1. Strings in Java are immutable and can consume a significant amount of memory, especially when manipulating them frequently. Use StringBuilder or StringBuffer for string manipulation operations that involve concatenation or modification to avoid creating unnecessary string objects.

# How to design the application from scratch from code?

Organized your code base into module,package,Function and domain boundaries For example We could have separate package for Controller,Services,and data domain(Repository), models and utilities

Keep business logic separate from the Data logic and presentation logic

Use the design pattern Such as DAO, MVC, and object creation pattern

* Implement data access logic in repository or DAO classes responsible for interacting with the database. Use ORM (Object-Relational Mapping) frameworks like Hibernate or JPA to abstract away database interactions and simplify data access.

1. We need to use technique such as query optimization , Lazy loading and cache to minimize the latency and response time to the user
2. Implement the filteration and pagination to limit the amount of data to be transfer over the network.

Infrastructure related Design :

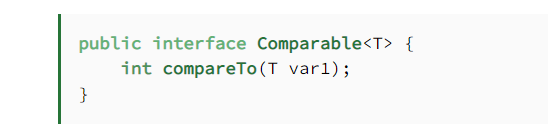
* + - 1. Design scalable solution architecture that can handle growing user loads and data volumes such as use of Load balancer to distribute the load among the instances of that service, Horizontal scaling that is basically increasing the instances of a particular service, use of caching and asynchronous process
* **Define Data Model:**
  + Design a robust data model that represents the entities, relationships, and constraints of the application domain. Use techniques such as entity-relationship modeling and normalization to ensure data integrity and consistency.

# How are you handling the load of 500k concurrent user?

# How to optimise the SQL queries and what are the approaches?

# Comparable VS Comparator

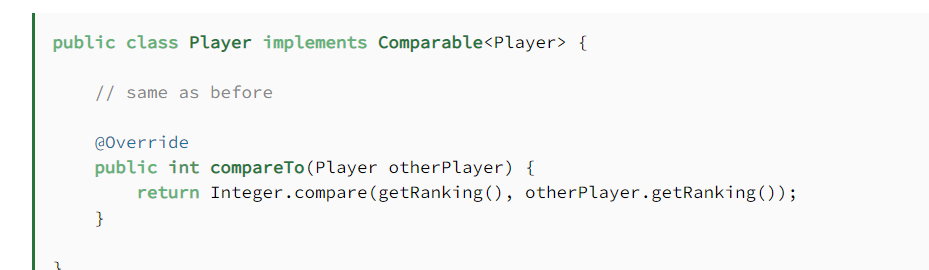
Sometimes when we write a new class we need to compare two objects of that class, It’s especially helpful when we want to use sorted collections There are 2 ways we can do this Comparable interfaces or With the comparator Interfaces.



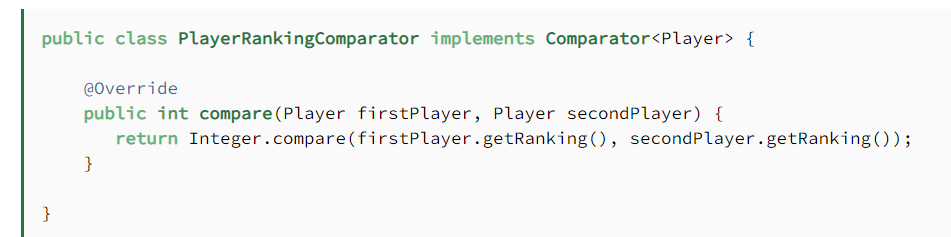
We should implement the interface for the object which we want to be sortedWe should implement the interface for the object which we want to be sorted

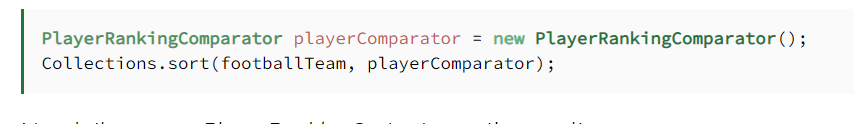
It has compareTo method which return int 3 values such as -1,0,1

**Comparable is an interface defining a strategy of comparing an object with other objects of the same type. This is called the class’s “natural ordering**

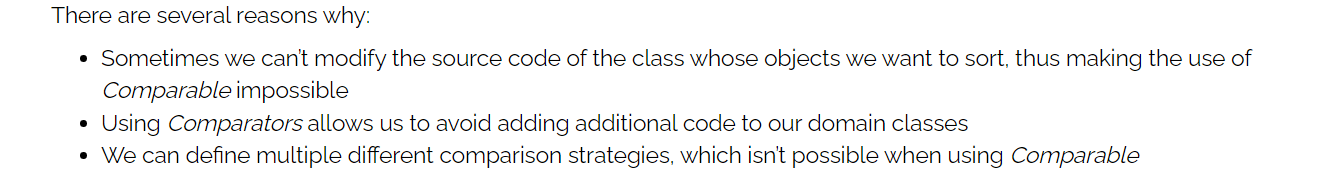


**The Comparator interface defines a compare(arg1, arg2) method** with two arguments that represent compared objects, and works similarly to the Comparable.compareTo() method.





Sort Summary:

The comparable interface is good for natural sorting order or we need the default sorting ordering

# What are the method of Object class what can they do.

* Equals()
* getClass()
* hashcode()
* notify
* notifyAll()
* toString()
* wait()
* Clone()
* Finalize()

# Design Principle Of Micro services

* Modularity :
  + Service should be self contained and have a single well defined Purpose.
* Scalability:
  + Service should be able to scale independently and handle the increase load
* The System Should be Loosely Coupled and decentralized.
* Service should be Available all the time for that we need to define the HA of the System
* Service should able to handle the failure gracefully
* Service should be deployable independently
* Automation in the micro service for scaling and testing

# Design a application where you are getting millions of request how will you design the application:

Use of Load Balancer: A lb distribute the request across the multiple server to ensure that no single server get over load

**Choose the right architecture:**  Microservice, serverless, or monolithic architecture can all work depending on the specific use case. Microservices allow for more flexibility and can handle scale better but are more complex to manage. Serverless architecture can automatically scale to handle the load but may have limitations on customization.

**Use a distributed cache**: A distributed cache can help reduce the load on the database by storing frequently accessed data in memory. This can speed up the application’s response time and reduce the number of database queries.

**Optimize database performance**: Databases are often a bottleneck in highly scalable applications. To optimize database performance, use techniques like indexing, caching and partitioning.

**Implement asynchronous processing**: By using asynchronous processing, the application can handle multiple requests at the same time, improving performance and scalability.

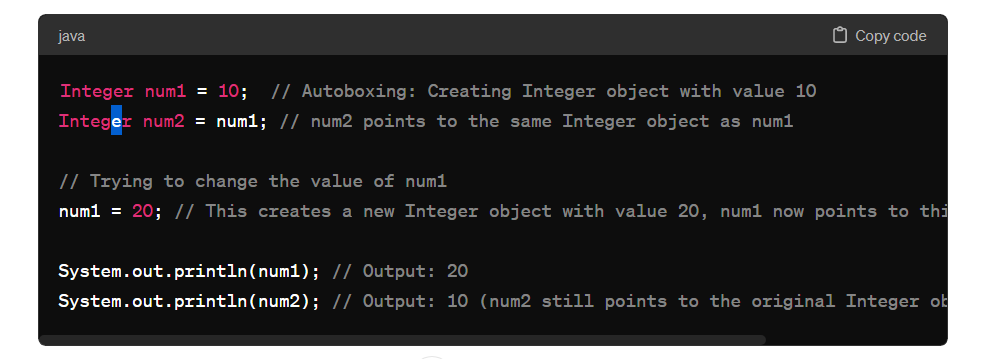
**Use Content Delivery Network (CDN)**: CDN is a network of servers that can distribute content globally, delivering it from the closest server to the user, thus reducing the latency and improving the application’s performance.

**Monitor and optimize**: Monitor the application’s performance and usage patterns and optimize the infrastructure accordingly. Implementing logging, monitoring, and alerting can help detect issues and optimize performance in real time.

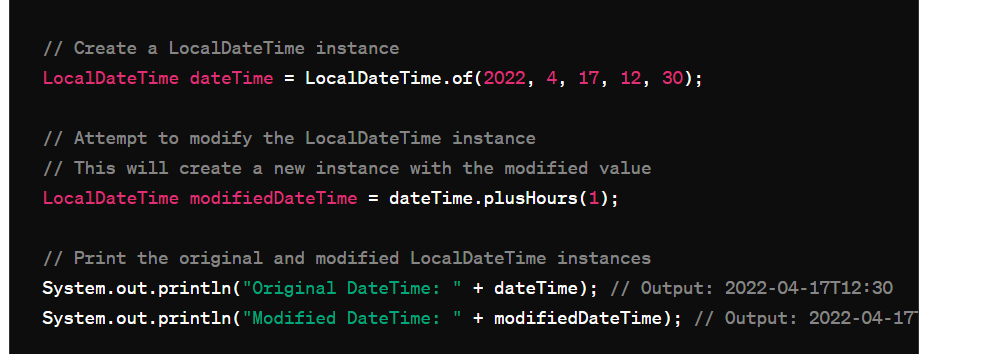
# Inbuilt Immutable Classes in Java:

String : The class represent the sequence of character and is one of the most commonly used classes in java Once a string object is created it’s content cannot be modified.

Integer,Long,Double,Float, Short,Byte, & Character These classes represent primitive data types and are also immutable. Once an Object of any of these classes is created it’s value cannot be changed;

Example: 

LocalDate, LocalTime, LocalDateTime, and Instant: These classes represent dates and times and are also immutable. They are part of the java.time package introduced in Java 8.



They don’t have the setter method and if we are refering the object with another instance and try to change it then it will create the new instances for the same.

# SQL Query to print the value if it exist otherwise Print the null and create the dynamic query for the same:

DELIMITER //

CREATE PROCEDURE GetWorkersWithSalary(IN salary\_value INT)

BEGIN

SELECT

\*,

CASE

WHEN salary = salary\_value THEN salary\_value

ELSE NULL

END AS Null\_Salary

FROM

worker;

END //

DELIMITER ;

# @Table Annotation In java:

* 1. Specify the name of the table that is mapped to the db with the help of ORM frame work

@Entity

@Table(name = “employees”)

Public class Employee{

}

* 1. Defining the unique constraint to one or more columns of the database table

@Entity

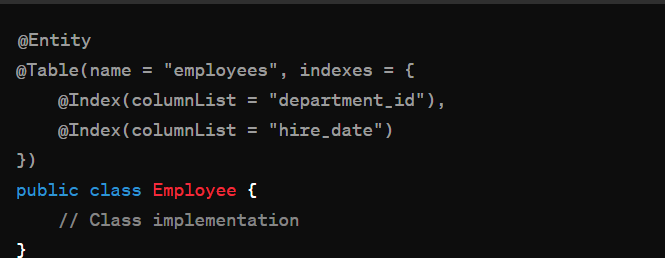
@Table(name = “employees”,

uniqueConstraints = {

@UniqueConstraint(ColumnNames = {“email”,”Phone”})

})

* 1. We can set the index using this annotation



# Clone Object Information and clone able:

In java it is used to define the shallow copy of the object and clone() provide a way to create a new object with same state by default it performs the shallow copy

Colne able interface act as marker interface and we have a clone() in Object class which we override to define logic

* + - 1. Proto type design pattern cloning is used
      2. Immutable object we will use cloning

# Difference btw Mono lithic & Micro services architecture:

Mono take all software components into one single project and microservice divide the application into separate self contained services.

1. Mono lithic app are not separable housing all functionality in a single codebase whereas micro services are modular each responsible for a specific set of task.
2. Deployment of micro app are independent to each other and
3. Mono communicate with in person call which increase the dependency in the classes whereas micro call other services with HTTP/REST protocol or message brokers.

When system is complex and we need to ensure the millions of user a scalable project then we will use micro services.

# Inter Communication Issues in Micro Services & Failure Roll over:

# Asynchronous & Event Based Communication in micro services:

We have 2 communication One is

1. Synchronous : REST, FEIGN,GRPC,
2. Asynchronous

Asynch Communication:

We can communicate with the help of queue We can send the message to the message broker and we will not wait for the response as soon as service receive the message from broker it will send the response.

If a services is down then in synch communication we will get the error but in async it’s more fault tolerance it will wait for the services to be up as soon as it’s up it will send the message to the services

If MB is down or restart then we will loose our data

Asynch removes the tight coupling from the services communication as we have to wait for the response

1. To make the fault tolerance **Make Replicas of MB**

## Types of Async:

1. Point to point
2. Pub sub

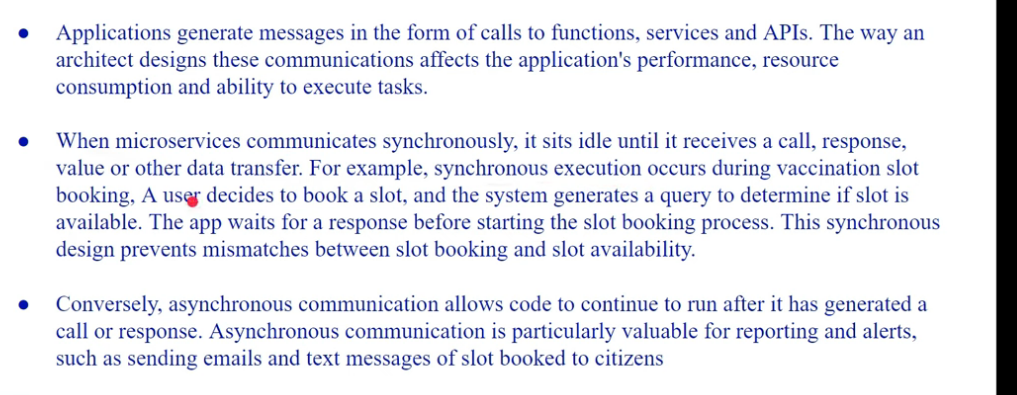
## Point to point:

* A queue will be used for this communication
* We have only 1 producer and 1 consumer that are interacting with each other with the help of queue
* One message send by producer will be consumed by one Consumer & message will be deleted after consumption
* If recierver service is down then message will be in the queue till the time receiver is up
* Popular choice are Rabbit MQ & Active MQ

## Pub – Sub model:

* In this model topic is used for the communication the broker will be used to store the message sent by the publisher and then subscribers that subscribe to that topic will consume that message.
* Unlike p to p the message will be ready to consume for all subscribers and the topic can have one or more subscriber. The message remains persistent in a topic until we delete it.
* Example are Kafka & AWS SNS etc.

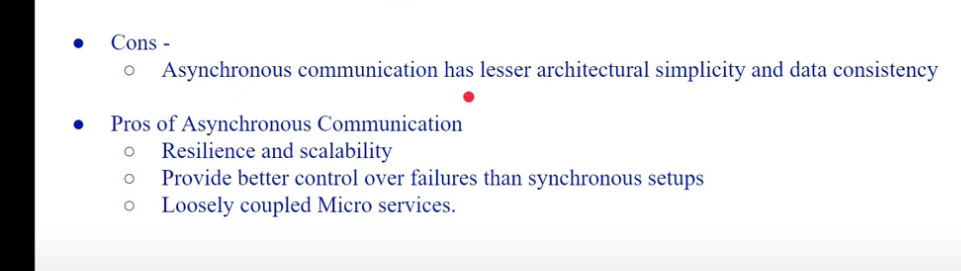
# How does Synch & ASync Communication Works?



Performance impact in sync communication if failure in any of the service

Note: to save yourself from failure we can have the load balancer and more instances of the services

ASYNCHRONOUS:

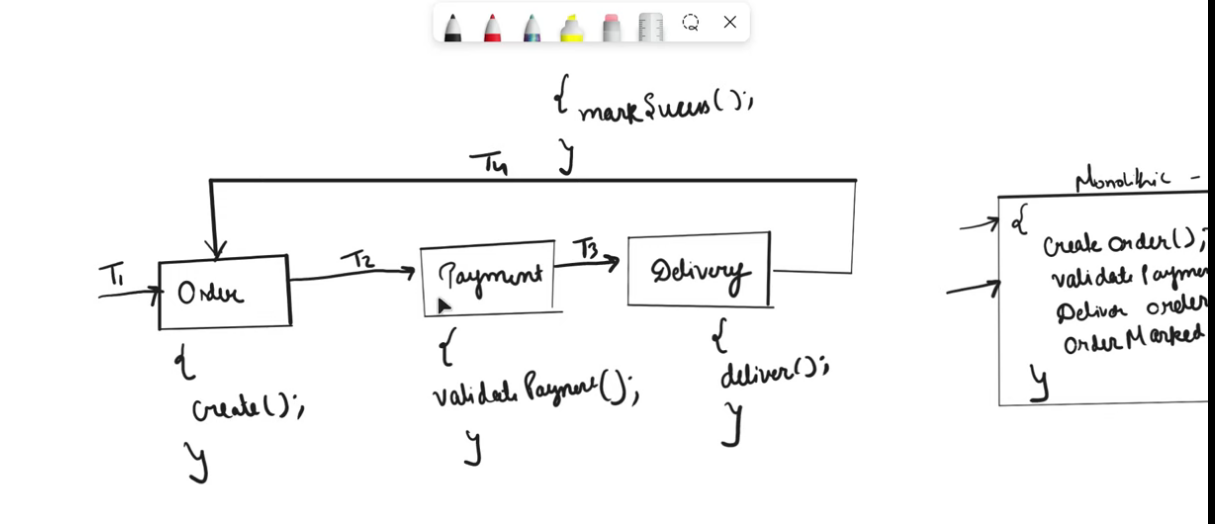


# When to use Which Micro service Communication:

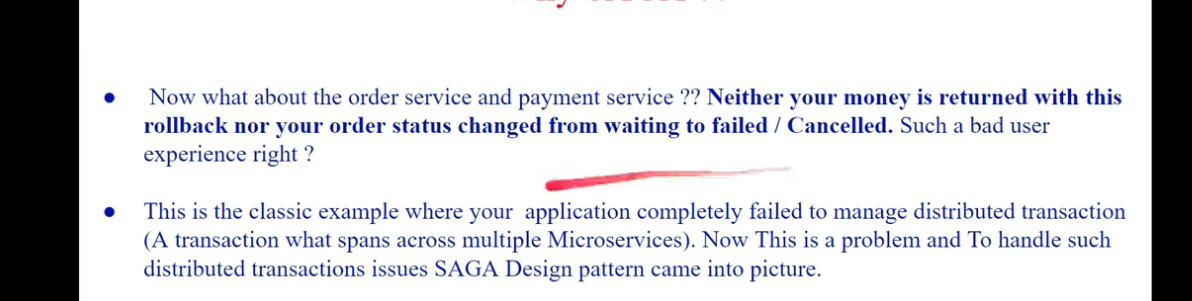
* When we start from scratch we need to go with synch communication to optimise for speed of evolution
* If your service starts to become complex and architecture grows the start adding the asynchronous communication between the services
* Figure it out if response is needed at the time of call only or we can have the response later then we can have asynchronous communication

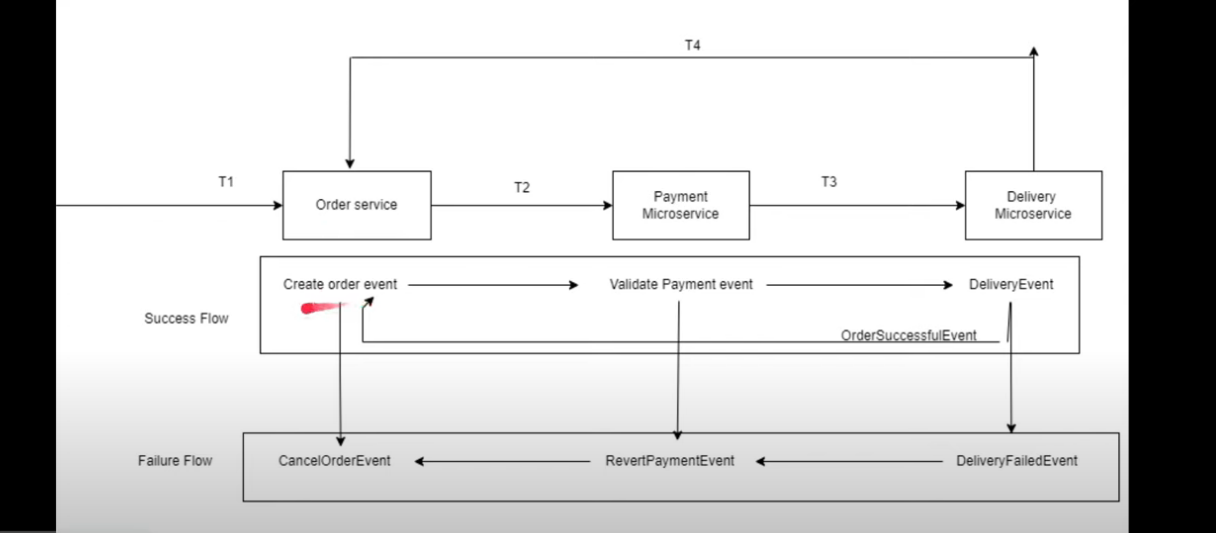
# SAGA Design Pattern:

* When we have a distributed services across the network and each service running on different IP address when
* we have a monolithic Service system then everything is atomic in nature all the tables we have in a single program the if we want to order something using zomato then whole process wrap it up using same application



At some point of time T3 is not executed as partner is not assigned for delivery we can marked as fail but local txn is roll backed in the services of Delivery service but not for Order & Payment txn



How SAGA Change the flow of your execution and handle the events:  


Here we are managing the event between the micro service to have the flow info and have the failure info as well

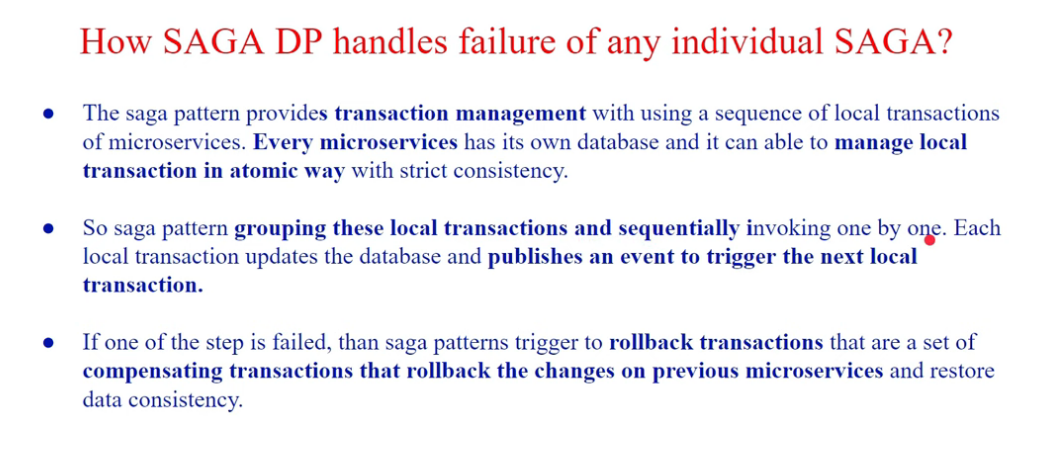
T1: Our execution start likes Order Service called have the local txn perform and generate the order event

T2: Payment Validation happen and generate the payment event

T3: Find delivery partner Not able to find it time out occurs from the delivery service

Now we have the reverse failure event flow where end micro services are the event generator and first one are the consumers of those event

Deliver fails we will generate event of failure into the another queue and let every other service to consume this message and let them roll back.



# Two ways to implement SAGA:

1. Choreography
2. Orchestration

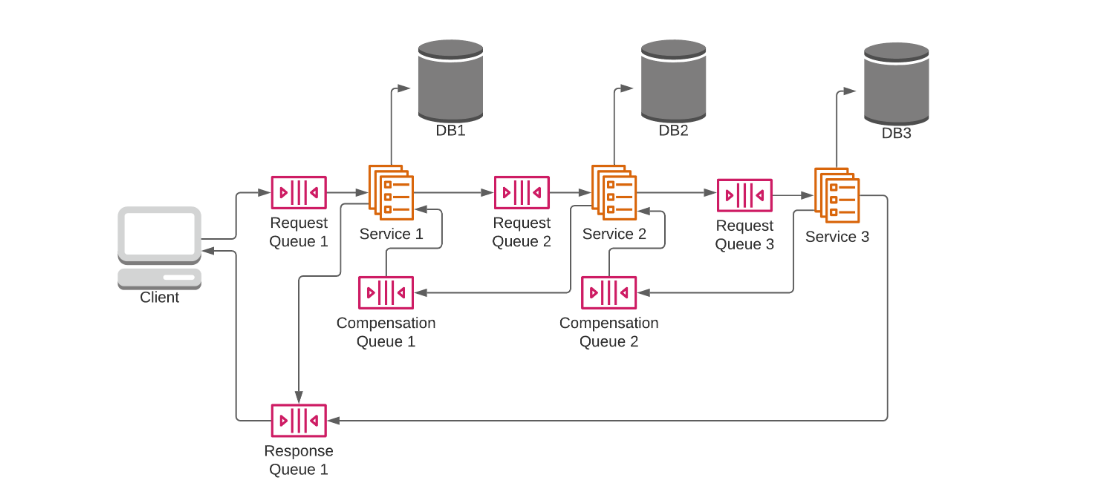
## Choreography Session:

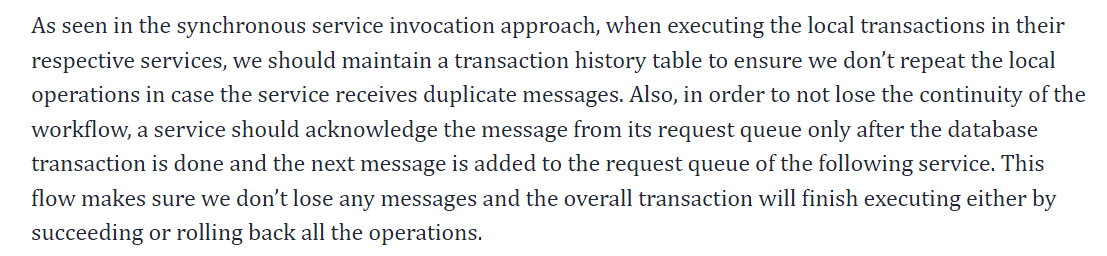
In this approach each of the service is communicate with each other with the help of queue we are maintaining the service operation here

After the successful transaction it will generate the message in the queue for another service 2 and same goes on

If any point of time any service got failed then we will generate the message into the compensation queue to roll back the transaction and it will further give the notice to the other services for giving compensation to the user if the payment is deducted from his/her side

Below is the diagram of the choreography :



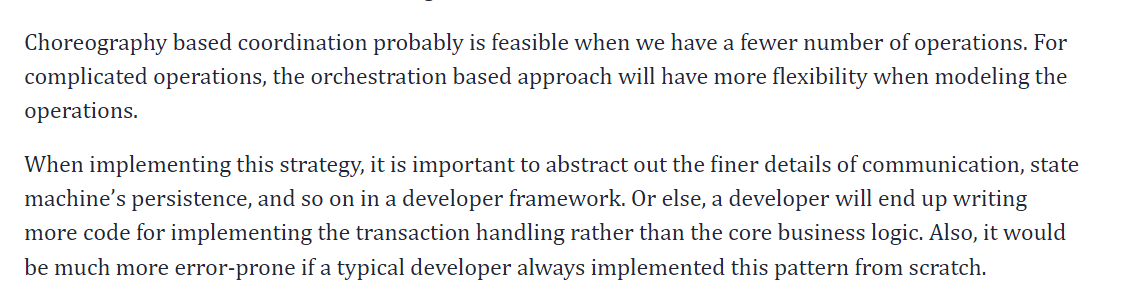


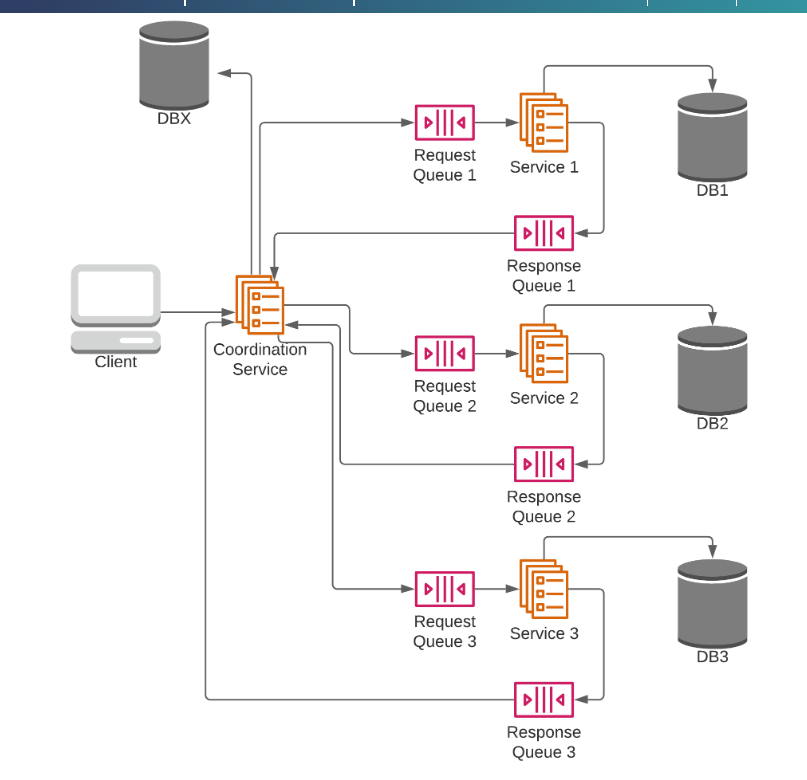
## Orchestration:

It’s a Service where all other services are communicating for further txn of a particular work flow

Advantages:

* Good for complex workflow there can be many participant with the orchestrator and there are new participants as well that can join the same
* Doesn’t introduce the cyclic dependencies
* There is no confusion between the communication of the microservice which one is listen to which command the command give by orchestrator is the only thing that is going to process





# If we have 2 micro services and We have to maintain the Transaction between the services if transaction failed in between how we will able to handle it

**one of the best ways to solve the problem of distributed transactions is to avoid them completely.**

TO ensure that we are having the communication between the micro services and performing the transaction then there are some approaches that we follow to solve this problem:

1. Implement the circuit breaker to check the response time from the called service if the response time convert it into the time out then circuit becomes open and we can conclude and call the “**Fall-back method”**  to give the default and final response to the user

# Use case of Abstract Class over Interfaces:

1. Partial implementation :   
   You have some method that can be implemented by the class and other method needs to implemented by the subclasses
2. Flexiblity in design:

Abstract class give more flexible design as compare to Interfaces They allow for a mix of abstract and concrete method in java

1. **Shared State or Fields**: You need to maintain state or fields that are shared among all subclasses. Abstract classes can have instance variables, constructors, and other features that interfaces cannot contain. This is useful when you have common data or behavior that all subclasses need access to.

# Relational Model Constraints:

Constraint in database:

1. Schema Based Constraint:

* Must be an atomic Value
* Perform data type check in the give table the age column have the data type integer but it’s storing the value of character which violated the domain constraint.



1. Key Constraints: An attribute that can be uniquely identified each tuple in a relation is called key

* Primary key has to be unique in nature
* Every relationship has at least one SUPERKEY that is a combination of all attributes. 