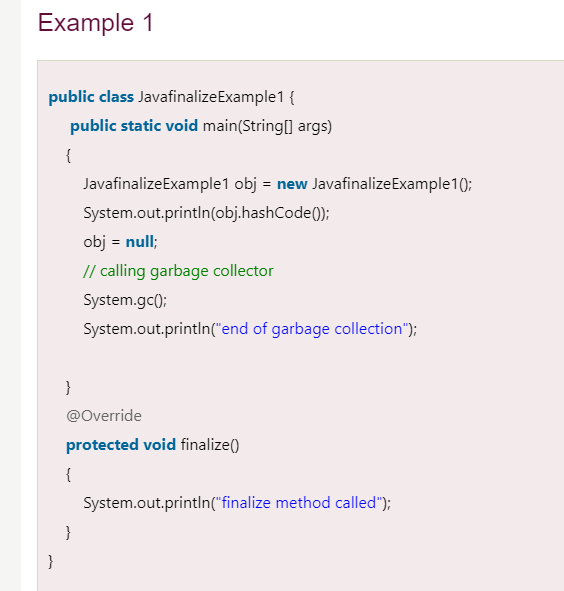
**Interview Question**

**Question 1 :**

**What is finalize()**

The Java **finalize() method** of [Object class](https://www.geeksforgeeks.org/object-class-in-java/) is a method that the [Garbage Collector](https://www.geeksforgeeks.org/garbage-collection-java/) always calls just before the deletion/destroying the object which is eligible for Garbage Collection to perform clean-up activity. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection,

****

**Question 2**

**How we will define heap memory**

Heap memory is finite memory but based on our requirements we can set maximum and minimum heap size i.e we can increase or decrease the heap size based on our requirements. We can do this by using the following java commands at runtime.

**-Xmx** to set maximum heap size (max memory)

*java -Xmx512m heapMemory (m----mb)*

**-Xms**: we can use this command to set a minimum or initial heap size.

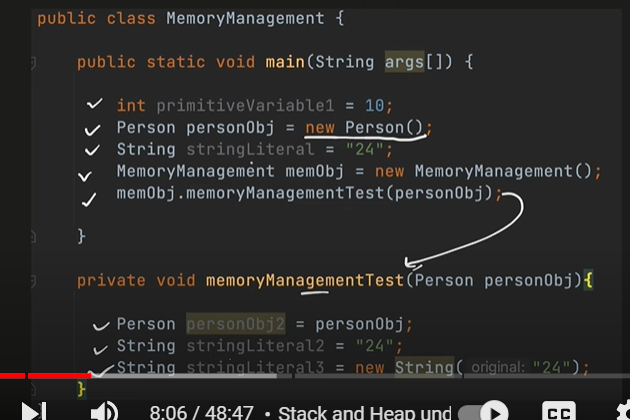
*java  -Xms64m heapMemory*

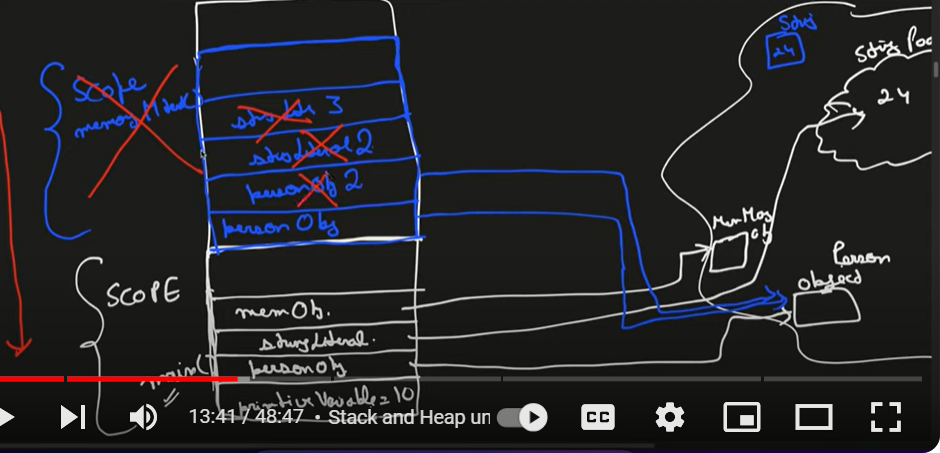
*Question 3*

**What is garbage collector and its types**

**Garbage collector is used to delete the unreferenced object from the heap.**

**Understand by examples**

****

****

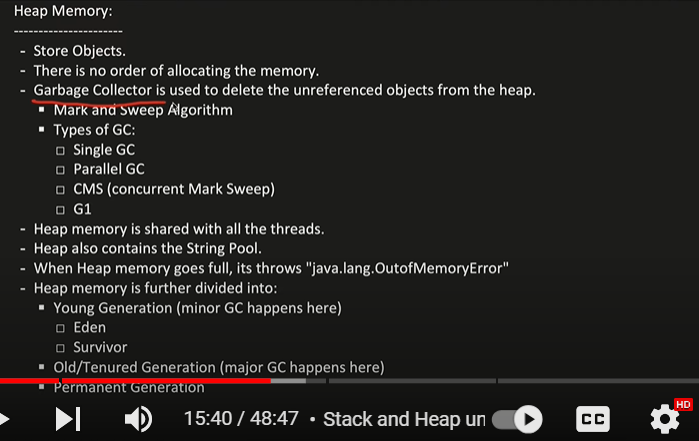
**Note :-**

**Each method will allocate in stack , called stack frame. Each stack will represent of one thread.**

**Object will be allocated in heap memory and reference variable will be assigned in stack.**

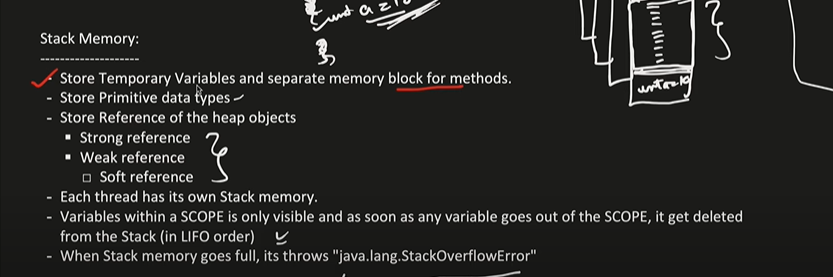
**So once scope is over that is method completed then it will remove from stack in lifo manner**

**But object will be there in heap then how it will be removed from heap that is where garbage collector come in picture**

****

**Note :**

* **JVM controls when to run the garbage collector.**
* **System.gc : - is the command to run the garbage collector.**
* **But JVM say there is no guarantee to run the garbage collector**
* **Jvm run garbage collector periodically**

****

**Each variable is visible within its own scope**

**Difference between strong reference and weak reference ?**

, If you have a strong reference to an object, then the object can never be collected/reclaimed by GC (Garbage Collector). If you only have weak references to an object (with no strong references), then the object will be reclaimed by GC in the very next GC cycle

In **weak reference** , it says I have the reference of object but if you (GC) run u can free the object and I may get null value.

**Soft reference :**  is a type of weak reference which told GC to free the space only when it is very urgent like heap is full

**What Are Soft References**? A soft reference object (or a softly reachable object) can be cleared by the Garbage Collector in response to a memory demand.

**Strong reference :** if strong reference is there object will never get deleted

Whenever no reference is available object is allowed to freedom the memory by garbage collector

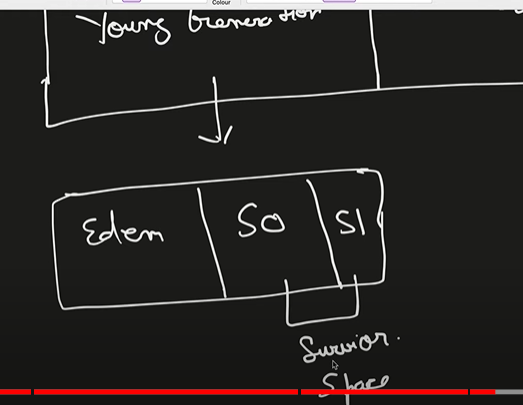
Heap memory

Heap memory is divided into two parts:

Young generation and old generation and there is one more known as non heap- metaspace



Young generation is further divided into three parts : eden , s0 and s1 ( **s0 and s1 called survivor space**)



Now when we create an object where its store first.

**-----Whenever we create new object it will goes inside eden**

**Let**  say 5 object store in eden and after a periodic time GC is invoked.

Now GC will do two things there is something called **mark and sweep algorithm**



It will **marked** all the object which is no more referenced and need to get delete.

Let say o2 and obj3 there is no referenced and allowed to delete now there is **sweep algo it will do two things**

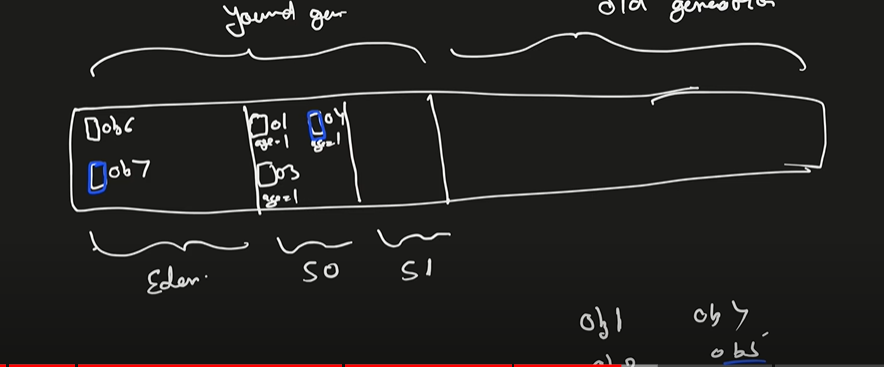
1. It will remove the obj2 and obj3
2. It will sweep remaining object into one of the surviving space that is memory (s0)



Age will get added to object in s0 means it age got increased

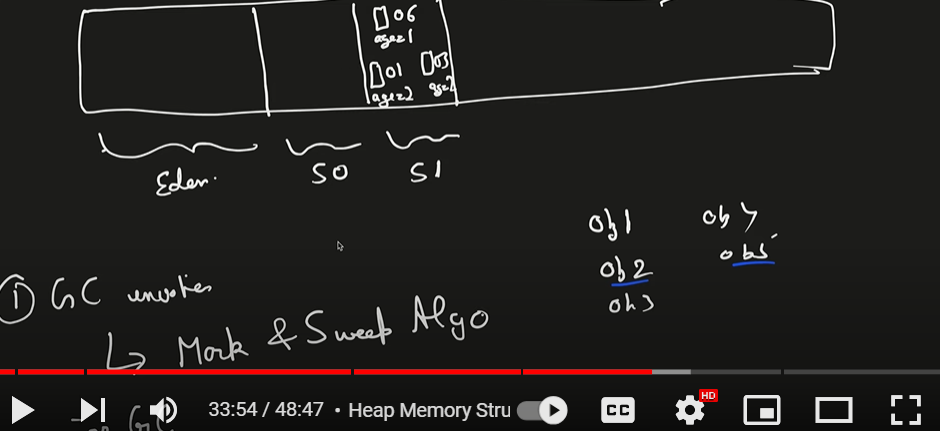
**This process is called minor GC :------ dead object will remove from eden and live object will move to s0 (survivoe space)**

Minor GC also checks the objects in a survivor space, and moves them to the other survivor space. Take the following sequence as an example: Eden has all objects (live and dead) Minor GC occurs - all dead objects are removed from Eden. All live objects are moved to S1 (FromSpace)



Now again new object came(obj6 and obj7) and it will get added in eden and again GC run first it will marked.

Let obj7 and obj4 are available for GC and sweep will run

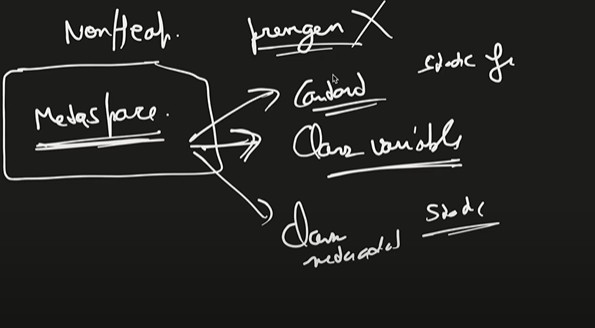


Alternatively we put the data in s0 and s1 and it age will increase

We need to put certain threshold and when object age will reach that much then it will get promotion and object will move to old generation

**In old generation it is called major GC**

**In old generation :**  gc run very less periodically because they have lot of references alive so long

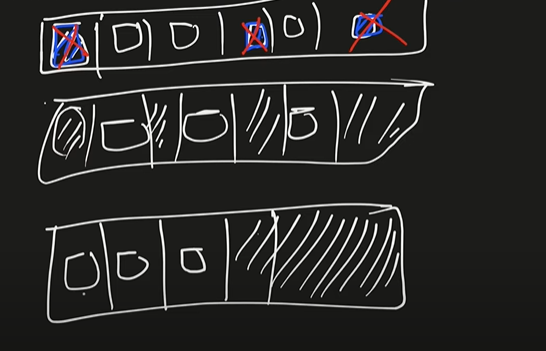


**Metaspace (outside of heap memory)** will store class variables, about class, constant

**Earlier metaspace called permgen**

**Major difference : permgen was part of heap memory and it is not extendable as** soon as it will filled up will get out of memory error and metadata is expandable as it is required.

**Mark and sweep with compaction**

****

**We can reduce the fragmentation by compaction**; we shuffle the memory content to place all the free memory blocks together to form one large ..

**Important**

**There are different version of GC**

**1-Serial GC 2- parallel GC (default) 3- CMS- concurrent mark and sweep 4- G1 garbage collector**

**Serial :**  it will create single thread to perform GC

D/A :---- work will be slow, GC is very expensive. As soon as GC run the application thread will pause so when GC is slow application is very slow.

**Parallel GC :**  multiple thread will work like when we have 2 core, 4 core

**CMS** : while your application thread are working concurrently GC thread also working. So application thread are not stopping by GC thread but this is not 100 percent guarantee. And also no memory compaction happen in this

**G1 garbage collector** : better version of CMS and also it give compaction

Currently java 8 use parallel GC

Question 4

**Best practices to create rest endpoints**

1. **Use Descriptive URIs :** Use nouns to represent resources
2. **HTTP Methods**: Use HTTP methods (**GET**, **POST**, **PUT**, **DELETE**, etc.) correctly
3. **Use HTTP Status Codes**: Return appropriate HTTP status codes to indicate the success or failure of a request (e.g., **200 OK**, **201 Created**, **404 Not Found**, **500 Internal Server Error**, etc.).
4. **Request and Response Formats**: Use standard formats for requests and responses, such as JSON or XML. Clearly document the format of request payloads and response bodies.
5. **Validation**: Validate input data on the server-side to ensure data integrity and security. Return appropriate error messages and status codes for validation failures.
6. **Authentication and Authorization**: Implement authentication and authorization mechanisms to control access to your API. Use standards like OAuth 2.0 for authentication
7. **Security**: Implement security best practices, such as using HTTPS, protecting against common security vulnerabilities (e.g., SQL injection, cross-site scripting)
8. **Pagination**: Implement pagination for large collections of resources to improve performance and reduce the amount of data transferred over the network.
9. **Rate Limiting**: Implement rate limiting to prevent abuse of your API and ensure fair usage by clients. Set appropriate limits based on your application's needs.
10. **Logging and Monitoring**: Implement logging and monitoring to track API usage, identify performance bottlenecks, and troubleshoot issues.
11. **Documentation**: Provide comprehensive documentation for your API, including endpoints,
12. **Testing**: Write automated tests for your API to ensure its correctness and reliability.
13. **Performance Optimization**: Optimize your API for performance by minimizing latency, reducing unnecessary data transfer, caching frequently accessed data, and scaling horizontally when necessary.

**Question 5**

**Idempotent vs non Idmepotent method**

**Idempotent : If** we send request multiple times the responses always going to be same

Example :----- get,delete, put

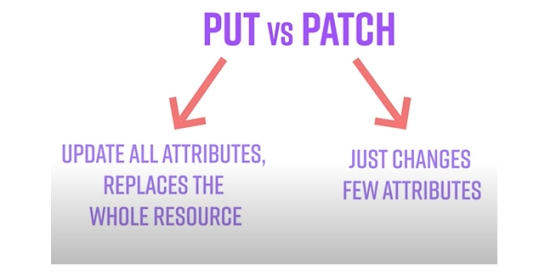
**Non-idempotent** :- example POST

In case of post first time it will create but second time it will give error 409 that’s means post request is not idempotent

Question 6

**Put vs post vs patch**

* Use PUT to update or replace an existing resource or create a new resource if it doesn't exist.
* Use POST to create a new resource on the server.
* Use PATCH to partially update an existing resource.



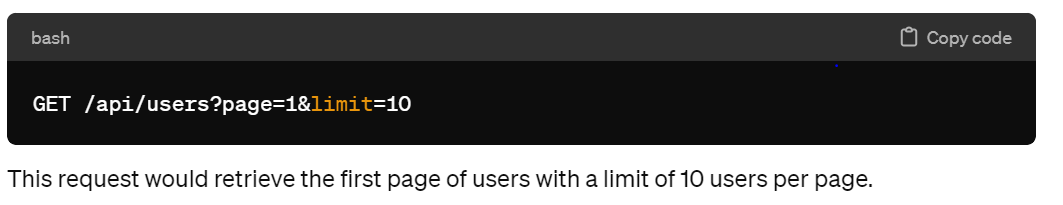
Question 7 :

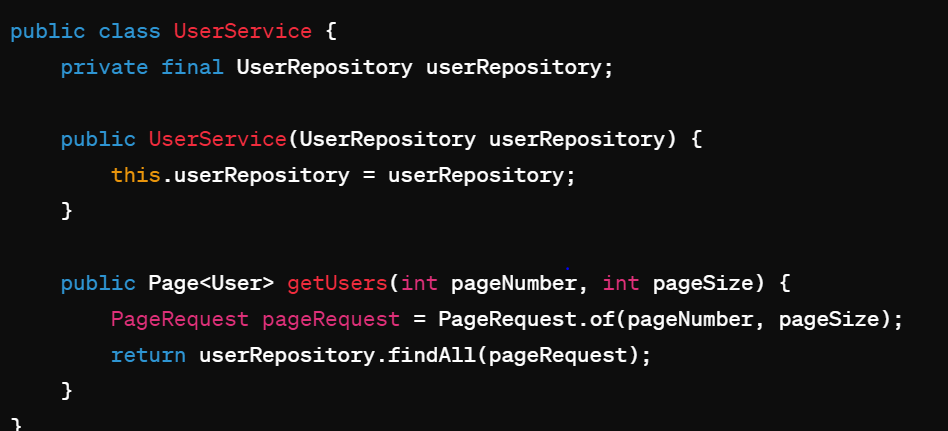
**Pagination and versioning in rest**

**Pagination is the practice of breaking large sets of data into smaller, more manageable chunks or pages**. Pagination allows clients to retrieve data incrementally, reducing the amount of data transferred and improving performance.

Commonly used pagination parameters include:

* **Limit**: Specifies the maximum number of resources to include in each page.
* **Offset/Page Number**: Specifies the starting position or page number of the requested page.
* **Next/Previous Page Links**: Links to the next and previous pages, allowing clients to navigate through the paginated results.

****

****

**Versioning**:

**Versioning is the practice of maintaining multiple versions of an API to support backward compatibility while introducing changes and updates**. Versioning allows clients to continue using older versions of the API without breaking their existing integrations, while also enabling them to adopt newer versions with updated features and improvements.

There are different approaches to API versioning, including:

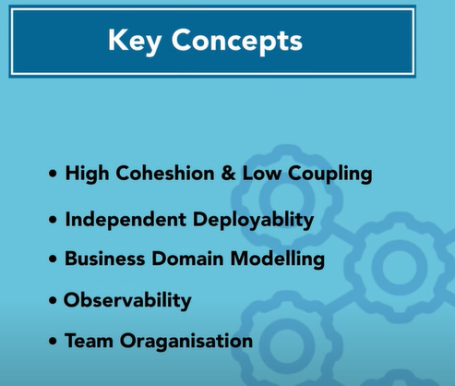
* **URI Versioning**: Version information is included in the URI path (e.g., **/api/v1/users**).
* **Query Parameter Versioning**: Version information is included as a query parameter (e.g., **/api/users?version=1**).
* **Header Versioning**: Version information is included in a custom header (e.g., **Accept: application/json; version=1**).

**Question 8: --**

**Difference between monolithic and microservices**

**Microservices :** service that does one thing very well and interact to other service

In a microservices architecture, the application is broken down into smaller, loosely coupled services that are independently deployable and scalable.

****

1. **High cohesion and low coupling**

All the functionalities which is related to that service sit inside that service that’s why we want our service to be highly **cohesive**

**Coupling** means if the services are too much depend on each other , know too much about each other share a common source or database that means services are coupled with each other

1. **Business domain model**

In m/s architecture the services are divided as per business domain or business feature

**Example :-- order,payment product**

1. **Independent deployability**

You should able to deploy our services independent of each other.

1. **Independent scaling**

we can increase the instance of order if order is increasing

1. **Observability**

**We can**  easily see if any of the service is degraded or not

1. **Resilience**

**If one of the service is down ,** it should not effect other system, the system should handled fault tolerance so that failures should not travel the system. This ensure even part if the system is failing but whole system will not come down.

1. **team organization**

Question 9

**saga orchestrator vs sage choreography**

Both Saga Orchestration and Saga Choreography are patterns used in distributed systems to manage long-running transactions involving multiple service

**Saga Orchestration**:

* In Saga Orchestration, there is a central component known as the Saga Orchestrator or Coordinator that controls the execution of the saga.

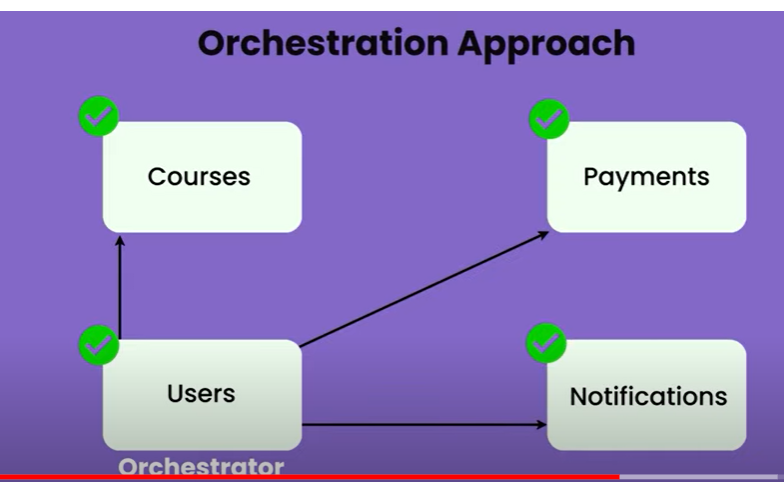
**Saga Choreography**:

* In Saga Choreography, there is no central orchestrator. Instead, each service participating in the saga is responsible for coordinating its own actions and reacting to events emitted by other services

**Orchestration approach ----- command and control approach ( like in music system)**

**Choreography approach ------trust and verify**

**User service :**  is going to act as orchestrator



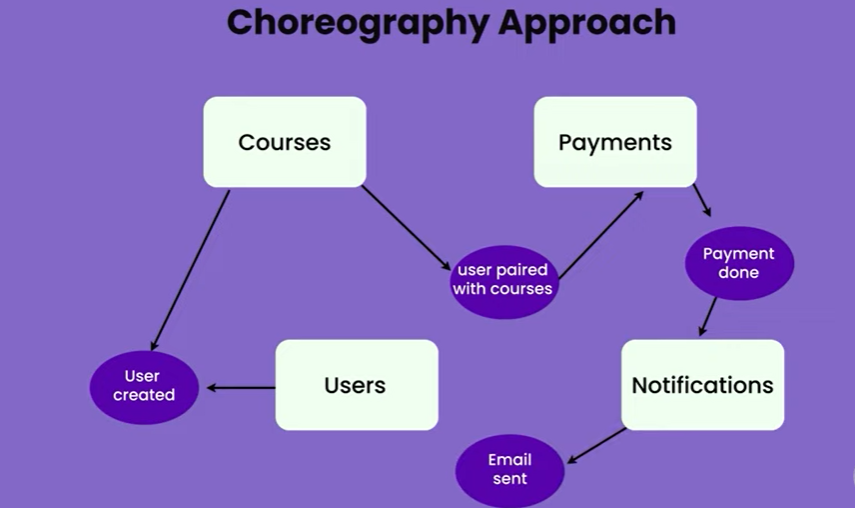
User service has control , it will talk to all service and give the command :----

U take course, u do payment, u send notification

**Basically user service has all the control and command. It is talking to diff service and telling what to do** So this is how orchestration approach look like

**How choreography looks like ?**

**There**  is no orchestrator



None of the service acting as orchestrator , all other service emits events and all other services reacting to those event to complete the whole business flow.

Each service is a system apart with its own database, and it is very hard to maintain data consistency. Saga pattern can help us maintain the data consistency among the microservices architecture efficiently. Event Sourcing ensures that all changes to business entity's state are stored as a sequence of events.

Question 10

**How do u do load balancing ?**

**client side load balancing :----- spring-cloud-starter-loadbalancer**

**server side load balancing : api gateway**

**Graphical user interface, text, application

Description automatically generated**

**Question 11 :**

**How do you configure api gateway**

Gateway

We have non functional requirement across m/s

Security

Tracing

Service aggregation :--- client may need call to various m/s

Rate Limits

Rate limiting is **a strategy for limiting network traffic**. Rate limiting can help stop certain kinds of malicious bot activity. It can also reduce strain on web servers.

A **rate** limiter that **limits** the number of requests received by the API within any given second. For most APIs, Stripe allows up to 100 read operations per second

**Note**

These all are cross cutting concern and we can configure this in spring cloud gateway and all m/s will be redirected through the gateway

If the productService want to use couponService, it need to pass through gateway, the gateway will route to couponservice get the response back and hands it to productservice. In the process it can apply any cross cutting concern

Diagram

Description automatically generated with medium confidence

If multiple instance of m/s is running gateway will do server side load balancing for us

Note :-- preprocessing and post processing occur in gateway

**Step -1 create gateway service**

**Note :---**

our gateway will communicate to eureka server to get other info for this will add eureka discovery client

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

It will register in eureka server by name gateway-service

Graphical user interface, application, Word

Description automatically generated

Configure the routes

We will configure ProductService and CouponService in properties file of gateway service

Notes:--

Predicate properties we will deal with path of m/s. within the coupon-service which all path is there

we can have multiple predicates so we will use [0]

Graphical user interface, text, application

Description automatically generated

**Important point:**

Now we don’t want feignClient directly talk with eureka server , we want to route from gateway service

Graphical user interface, text, application

Description automatically generated

**Gateway in action**

So that it can fetch info about ProductService and CouponService

Graphical user interface, text, application

Description automatically generated

**Now for m/s will be there**

EurekaServer

ProductService

CouponService

Apigatewayservice

Put this property in both m/s ProductService and CouponService

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Now everything is happening through gateway

**Filter Introduction**

**Note**

The api gateway allow us to do cross cutting concern like security using filter classes

Whenever request come from any m/s to api gateway it will invoke filter where we can do pre and post processing logic

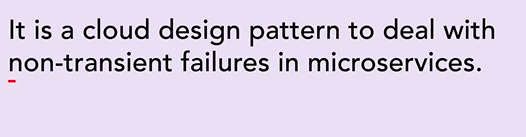
Chart, waterfall chart

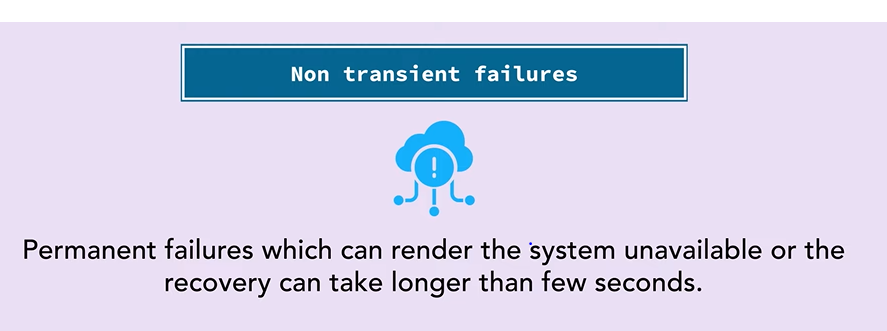
Description automatically generated

Question 12

**Uses of circuit breaker**

Circuit breakers are a pattern used in distributed systems to improve resilience and fault tolerance by preventing cascading failures



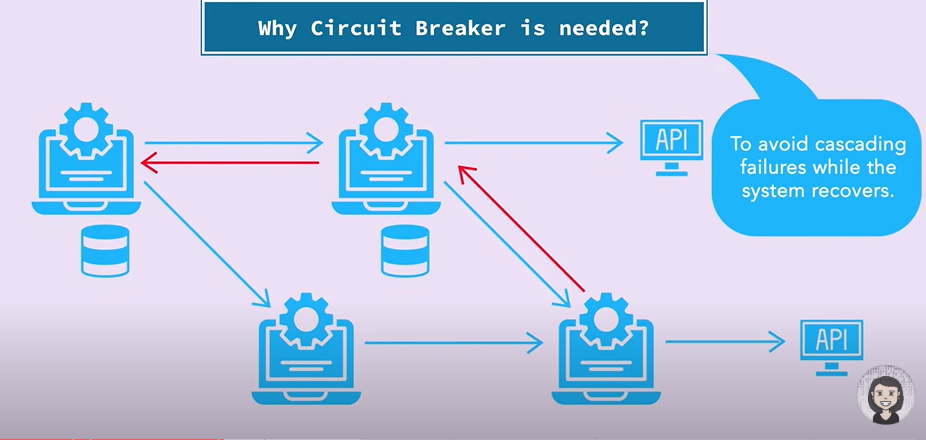
****

Example :

Timeout or delays where api do not response or take longer time to response resulting in timeout

Connection to databases failing

Service bombarded with lot of request and it failing due to memory or cpu issue

****

**Question 13**

**Non functional requirement in microservices**

**Scalability**:

* Microservices architectures should be able to scale horizontally to handle increased loads and traffic.

**Performance**:

* NFRs should define performance targets for response times, throughput, and latency for different microservices and APIs.

**Availability and Reliability**:

* Microservices should be designed for high availability and fault tolerance to ensure uninterrupted service in the event of failures or outage

**Resilience**:

* NFRs should define resilience requirements, such as the ability to gracefully handle failures, recover from errors, and maintain partial functionality during outages.

**Security**:

* Microservices should enforce security measures to protect sensitive data, prevent unauthorized access, and mitigate security threats.

**Monitoring and Logging**:

* NFRs should include requirements for comprehensive monitoring and logging to track system health, performance metrics, and security incidents.

**Manageability**:

* Microservices architectures should be easy to manage, deploy, and maintain, even as the system grows in complexity.