Interview Preparation

1. Java Core

1. Stream lambda new thread。

```
Thread t1 = new Thread(()->{do somthing})

Thread t = new Thread(new MyRunnable());
        t.start(); // 启动新线程

Runnable r2 = () -> System.out.println("我爱北京故宫");

import java.util.*;
import java.util.function.Function;
import java.util.stream.Collectors;

public class GFG {
    public static void main(String[] args)
    {

        // Get the List
        List<String> g = Arrays.asList("geeks", "for", "geeks");

        Map<String, Long> result = g.stream().collect(
              Collectors.groupingBy( Function.identity(), Collectors.counting()));

        // Print the result
        System.out.println(result);
    }
}

List<Integer> list = Arrays.asList(3, 6, 9, 12, 15);

        // Using Stream map(Function mapper) and
        // displaying the corresponding new stream
        list.stream().map(number -> number * 3).forEach(System.out::println);

list.stream()
          .filter(num -> num % 5 == 0)
```

```
        .forEach(System.out::println);
```

2. Interface VS abstract class?

What are the differences between them?

- **Type of methods:** Interface can have only abstract methods. An abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also.
- **Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.
- **Type of variables**: Abstract class can have final, non-final, static and non-static variables. The interface has only static and final variables.
- **Implementation:** Abstract class can provide the implementation of the interface. Interface can't provide the implementation of an abstract class.
- **Inheritance vs Abstraction:** A Java interface can be implemented using the keyword "implements" and an abstract class can be extended using the keyword "extends".
- **Multiple implementations:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
- **Accessibility of Data Members**: Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.

- Interfaces **cannot** have **implemented methods**, abstract classes can
- **All interface variables are static and final**, abstract classes may have non-final, non-static variables
- Interfaces can only extend other interfaces, abstract classes may implement multiple interfaces and extend other classes
- Abstract classes may have private or protected members while interface members are public by default.

3. Hashmap vs Hashtable

HashMap is non-synchronized. It is not thread-safe and can't be shared between many threads without proper synchronization code whereas Hashtable is synchronized. It is thread-safe and can be shared with many threads.
HashMap allows one null key and multiple null values whereas Hashtable doesn't allow any null key or value.
HashMap is generally preferred over HashTable if thread synchronization is not needed.

why hashtable cannot contain null key

In order to successfully store and retrieve objects from a HashTable, the objects used as keys must implement **the hashCode method and the equals method**. **Since null is not an object**, it can't implement these methods. HashMap is an advanced version and improvement on the Hashtable. HashMap was created later.

4. Java 8 new features

1. Lambda expression

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

2. Functional interface

A functional interface is an interface that contains only one abstract method

3. Method References

ClassName::methodName

4. Stream API

default Stream<E> stream() : 返回一个顺序流
default Stream<E> parallelStream() : 返回一个并行流

5. Optional class

NullPointerException

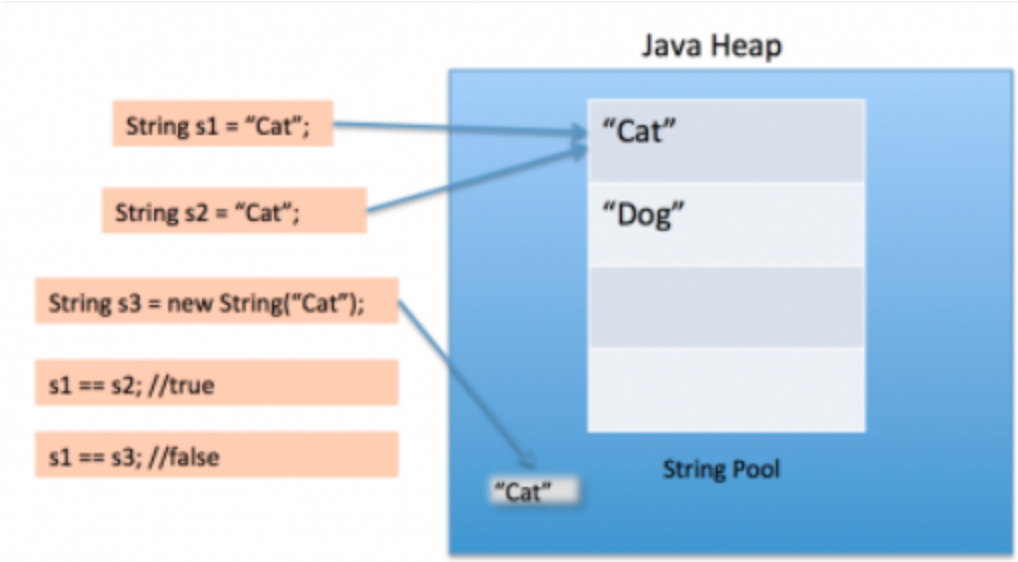
The Optional class in Java is a container that can hold, at max, one value and gracefully deals with null values. The class was introduced in the java.util package **to remove the need for multiple null checks** to protect against the dreaded **NullPointerExceptions during run-time**.

5. primaritive data type (8)

int, byte, short, long, float, double, boolean and char.
These aren't considered objects and represent raw values.
They're stored directly on the stack

6. String Constant Pool

String is immutable



7. Final

Variable: define constants
Method: prevent override
Class: 1) prevent inheritance, like Integer, String etc; 2) Make class immutable

8. Static

we'll create only one instance of that static member that is shared across all instances of the class

static variables are stored in the heap memory.

Method: Can directly call static method using Class name
A static inner class is a nested class which is a static member of the outer class.

***static* nested classes behave exactly like any other top-level class but are enclosed in the only class that will access it, to provide better packaging convenience.**

9. Difference collection VS stream

Streams differ from collections in several ways:

- **No storage**. A stream is not a data structure that stores elements; instead, it conveys elements from a source such as a data structure, an array, a generator function, or an I/O channel, through a pipeline of computational operations.

- **Functional in nature**. An operation on a stream produces a result, but does not modify its source. For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.
- **Laziness-seeking**. Many stream operations, such as filtering, mapping, or duplicate removal, can be implemented lazily, exposing opportunities for optimization. For example, "find the first String with three consecutive vowels" need not examine all the input strings. Stream operations are divided into intermediate (Stream-producing) operations and terminal (value- or side-effect-producing) operations. Intermediate operations are always lazy.
- **Possibly unbounded**. While collections have a finite size, streams need not. Short-circuiting operations such as limit(n) or findFirst() can allow computations on infinite streams to complete in finite time.
- **Consumable**. The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

10. Java generics

allow "a type or method to operate on objects of various types while providing compile-time type safety"

Generics means **parameterized types**. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types.

11.Connection pool to database

a **connection pool** is a cache of database connections maintained so that the connections can be reused when future requests to the database are required. Connection pools are used to enhance the performance of executing commands on a database.

Opening and maintaining a database connection for each user, especially requests made to a dynamic database-driven website application, is costly and wastes resources. In connection pooling, after a connection is created, it is placed in the pool and it is used again so that a new connection does not have to be established. If all the connections are being used, a new connection is made and is added to the pool. Connection pooling also cuts down on the amount of time a user must wait to establish a connection to the database.

Java Database Connectivity (or JDBC for short) is a Java API designed to connect and manage relational databases in applications programmed with Java.

JDBC connection pooling saves time when opening and closing database connections for every user.

12. Java8的特性，用lambda怎么遍历hashmap (thread pool)

```
Map<String, Integer> prices = new HashMap<>();
                    prices.put("Apple", 50);
                    prices.put("Orange", 20);
                    prices.put("Banana", 10);
                    prices.put("Grapes", 40);
                    prices.put("Papaya", 50);

prices.forEach((k,v)->System.out.println("Fruit: " + k + ", Price: " + v));
```

13. Internal Working of HashMap in Java

Hashing -> Buckets ->

Steps:

- Calculate hash code of Key {“vishal”}. It will be generated as 118.
- Calculate index by using index method it will be 6.
- Create a node object as :

In Case of collision:

map.get(new Key("vaibhav"));

- Steps:
    1. Calculate hash code of Key {“vaibhav”}. It will be generated as 118.
    2. Calculate index by using index method it will be 6.
    3. Go to index 6 of array and compare first element’s key with given key. If both are equals then return the value, otherwise check for next element if it exists.
    4. In our case it is not found as first element and next of node object is not null.
    5. If next of node is null then return null.
    6. If next of node is not null traverse to the second element and repeat the process 3 until key is not found or next is not null.
    7. Time complexity is almost constant for put and get method until rehashing is not done.
    8. In case of collision, i.e. index of two or more nodes are same, nodes are joined by link list i.e. second node is referenced by first node and third by second and so on.
    9. If key given already exist in HashMap, the value is replaced with new value.
    10. hash code of null key is 0.
    11. When getting an object with its key, the linked list is traversed until the key matches or null is found on next field.

2. SpringBoot

Javabean

所谓JavaBean，是指符合如下标准的Java类：

>类是公共的

>有一个无参的公共的构造器

>有属性，且有对应的get、set方法

Spring advantages:

> **The main goal of Spring Boot is to quickly create production ready Spring-based applications without requiring developers to write the same boilerplate configuration again and again**

- dependency injection

- database transaction management

- integration with other Java frameworks

- Web MVC framework for building web applications

Spring need a lot of configurations

Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added.

Why do we need Spring Boot Auto Configuration?
-> Spring based applications have a lot of configuration.
-> When we use Spring MVC, we need to configure
- **Component scan,**
- **Dispatcher Servlet**
- **View resolver**
- **Web jars(for delivering static content) among other things.**

-> When we use Hibernate/JPA, we would need to configure a
- **data source**
- **entity manager factory/session factory**
- **transaction manager among a host of other things.**

Above examples are typical with any Spring framework implementation or integration with other frameworks.

Springboot offers

1.
2.
3.
4. DI( )

springboot annotation
@co

differences between @RequestParam and @PathVariable?

@RequestParam used for accessing the values of the **query parameters** where as @PathVariable used for accessing the values from the **URI template.**

both @RequestParam and @PathVariable are used to extract values from the HTTP request

1) As the name suggests, @RequestParam is used to get the request parameters from URL, also known as query parameters, while @PathVariable extracts values from URI.

2) `@RequestParam` is more useful on a traditional web application where data is mostly passed in the query abatements while `@PathVariable` is more suitable for RESTful web services where URL contains values, like `http://localhost:8080/book/9783827319333`, here data, which is ISBN number is part of URI.

3) `@RequestParam` annotation can specify default values if a query parameter is not present or empty by using a defaultValue attribute, provided the required attribute is false.

4) Spring MVC allows you to use multiple `@PathVariable` annotations in the same method, provided, no more than one argument has the same pattern.

Read more:
https://javarevisited.blogspot.com/2017/10/differences-between-requestparam-and-pathvariable-annotations-spring-mvc.html#ixzz7WRZwU0bU

Read more:
https://javarevisited.blogspot.com/2017/10/differences-between-requestparam-and-pathvariable-annotations-spring-mvc.html#ixzz7WRZgqpDA

1. Using @RequestParam to get Query parameters

`http://localhost:8080/eportal/orders?id=1001`

To accept the query parameters in the above URLs, you can use the following code in the Spring MVC controller:

```
@RequestMapping("/orders")
public String showOrderDetails(@RequestParam("id") String orderId, Model model){
  model.addAttribute("orderId", orderId);
  return "orderDetails";
}
```

URL: `http://localhost:8080/eportal/trades?tradeId=2001`

```
@RequestMapping("/trades")
public String showTradeDetails(@RequestParam String tradeId,
                               Model model){
 model.addAttribute("tradeId", tradeId);
 return "tradeDetails";
}
```

2. Using @PathVariable annotation to extract values from URI

URL: `http://localhost:8080/book/9783827319333`

Now, to extract the value of ISBN number from the URI in your Spring MVC Controller's handler method, you can use `@PathVariable` annotation as shown in the following code:

```
@RequestMapping(value="/book/{ISBN}", method= RequestMethod.GET)
public String showBookDetails(@PathVariable("ISBN") String id,
                              Model model){
  model.addAttribute("ISBN", id);
```

```
   return "bookDetails";
}
```

URL
http://localhost:8080/springmvc/hello/101?param1=10&param2=20

The above URL request can be written in your Spring MVC as below:

```
@RequestMapping("/hello/{id}")
public String getDetails(@PathVariable(value="id") String id,
@RequestParam(value="param1", required=true) String param1,
@RequestParam(value="param2", required=false) String param2){
.....
}
```

Types of dependency injection, why field injection is not recommended?

3 types constructor injection, method injection, and property injection

https://cloudolife.com/2021/02/27/Programming-Language/Java/FAQs/Field-injection-is-not-recommended-and-Injection-guidelines-in-Java-Spring/#:~:text=The%20reasons%20why%20field%20injection.in%20unit%20tests)%20without%20reflection.

Which Is the Best Way of Injecting Beans and Why?

constructor injection with programmatic validation of arguments is preferable.

1. constructor injection, as it lets you implement application components as immutable objects and ensures that required dependencies are not null.
2. constructor-injected components are always returned to the client (calling) code in a fully initialized state.

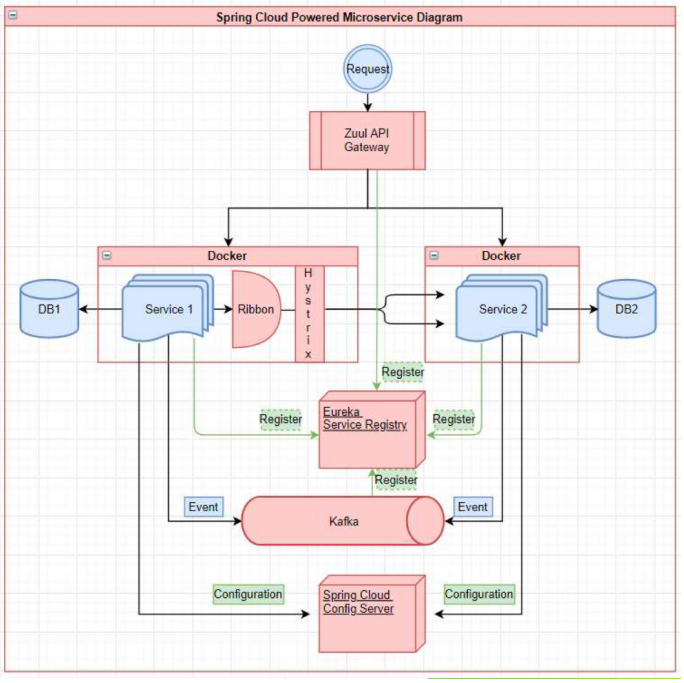
3. Microservice, kafka, cloud

Microservice

Microservice architecture

Component

- Spring Cloud
- Zuul API Gateway
- Eureka
- Ribbon Load Balancer
- Hystrix Circuit Breaker
- Config Server
- Kafka
- Docker



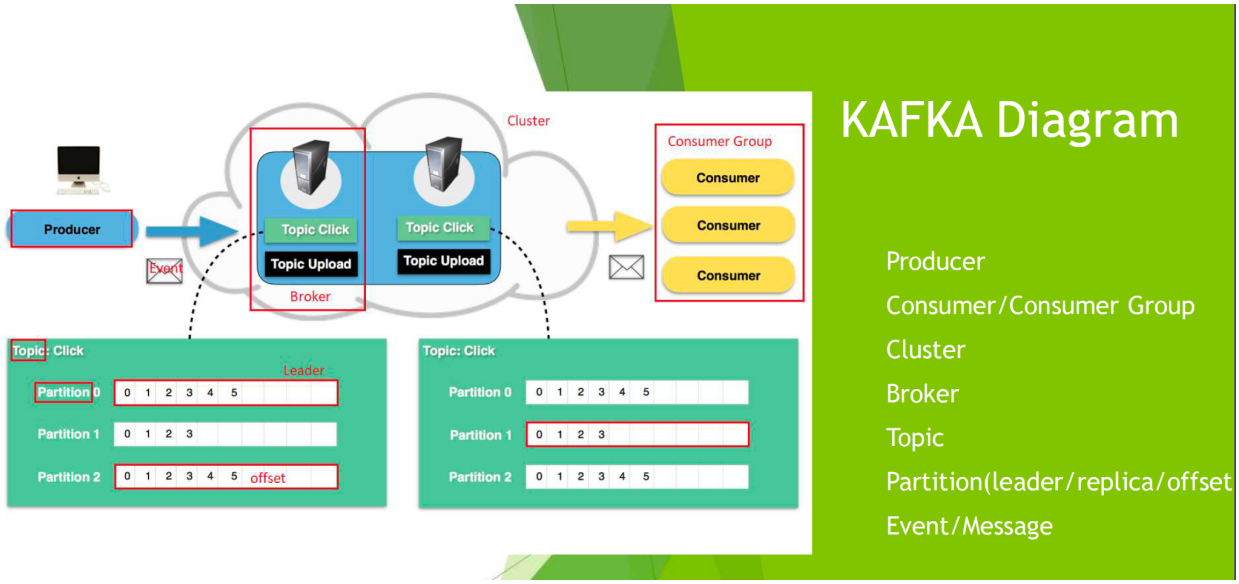
Kafka message queue, eureka
What are the components and the role.

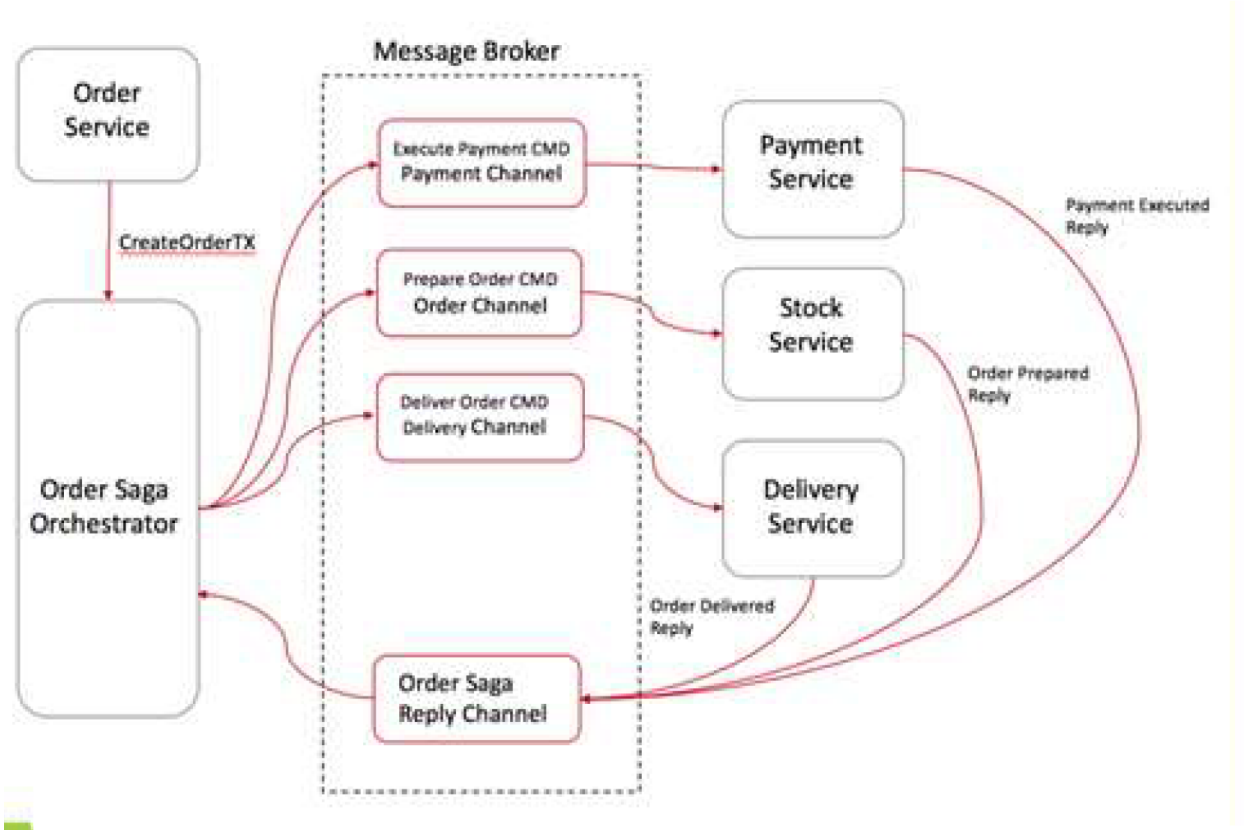
Here are 8 core components are Microsoft Architecture:

1. **Clients**
2. **Identity Providers**
3. **API Gateway**
4. **Messaging Formats**
5. **Databases**
6. **Static Content**
7. **Management**
8. **Service Discovery**

use lightweight communication mechanisms like REST and HTTP.

Kafka





<https://aws.amazon.com/msk/what-is-kafka/>

Kafka is primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.

producer -> queue -> consumer
What is the kafka
Producer , consumer, group producer, consumer,
Topic, partition, event name

Cloud

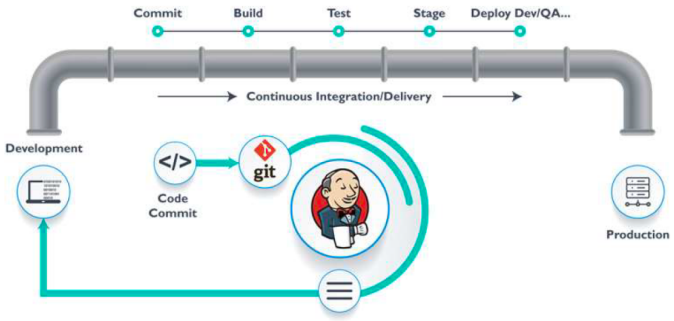
S3, EC2, RDs

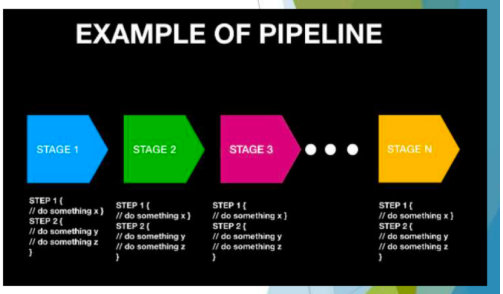
Eureka Server is an application that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server

knows all the client applications running on each port and IP address. Eureka Server is also known as Discovery Server.

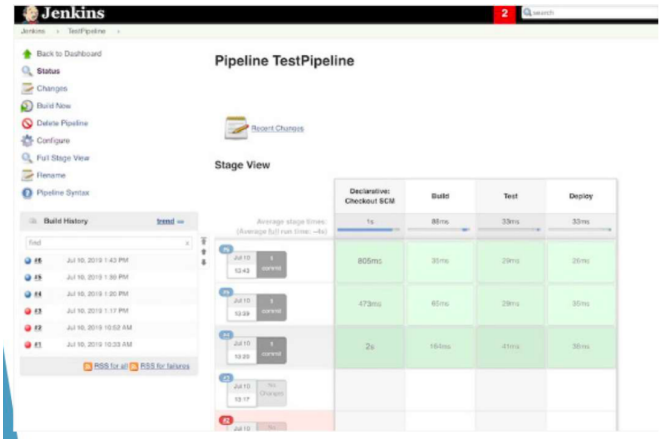
CI/CD

CICD





Jenkins Pipeline



READ:

https://opensource.com/article/19/9/intro-building-cicd-pipelines-jenkins

```
//Jenkinsfile (Declarative Pipeline)
pipeline {
    agent { docker { image 'maven:3.3.3' } }
    stages {
        stage("Build") {
            steps {
                echo "Building ${env.Build_ID}"
                javac HelloWorld.java
                mvn clean package ./HelloWorld
            }
        }
        stage('Deploy') {
            steps {
                sh './deploy-script' //run deploy script
            }
        }
        stage('Test') {
            steps {
                mvn test HelloWordTest
            }
        }
    }
    post {
        always {
            echo 'Generate Report File'
            junit 'build/reports/reports.xml'
        }
        success {
            echo 'This will run only if successful'
        }
        failure {
            mail to: 'team@example.com',
             subject: "Failed Pipeline: ${currentBuild.fullDisplayName}",
             body: "Something is wrong with ${env.BUILD_URL}"
        }
    }
}
```

4. Singleton

Final class Singleton{
	Private static Singleton obj = new Singleton();
	Private
}

Eager

```
final class Singleton{
  private static Singleton obj = new Singleton();
  private Singleton(){};
  Public static Singleton getInstance(){
    return obj;
  }
}
```

Lazy

```
final class Singleton{
//The Java volatile keyword guarantees variable visibility across threads,
meaning reads and writes are visible across threads.
  private static volatile Singleton obj = null;
  private Singleton(){};
  public static Singleton getInstance(){
    if(obj == null){
      // thread safe
      synchronized(Singleton.class){
        if(obj == null){
          Obj = new Singleton();
        }
      }
     return obj;
    }
  }
}
```

5. Other

how to debug in your project

IDE:
Break points
Run the application local
Use postman to call api to reach to the methods you want to debug
Or
Use unit test to

Log files

What are HTTP methods?

POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively.
GET -> retrieve data,
POST -> create data,
PUT-> update data,
DELETE -> get rid of the data.

What are HTTP Status codes?

https://www.restapitutorial.com/httpstatuscodes.html

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

Informational responses (100–199)
Successful responses (200–299)
Redirection messages (300–399)
Client error responses (400–499)
Server error responses (500–599)

STATUS CODE RANGE	MEANING
100 – 199	Informational Responses. For example, 102 indicates the resource is being processed
300 – 399	Redirects For example, 301 means Moved permanently
400 – 499	Client-side errors 400 means bad request and 404 means resource not found
500 – 599	Server-side errors For example, 500 means an internal server error

Connection Pool

a connection pool is **a cache of database connections maintained so that the connections can be reused when future requests to the database are required**. Connection pools are used to enhance the performance of executing commands on a database.