Full Stack Java Developer Interview Questions and Answers

201 Full Stack
Questions & Answers

11th Edition

•••••

Mr. Kiran Jadhav
Full Stack Java Expert

“Dedicated to my Students & Job Seekers whose admiration inspires me to teach forever.”

-: About the Author:-

Mr. Kiran Jadhav holds the B.E. degree in Information Technology from Pune University, First Class with Distinction. He currently works for top investment bank as Solution Architect.

He has about 10+ years of industry experience in Development, Architecture and Deploying Enterprise-Level Full Stack technological product and solutions for Fortune 500 MNC's in the domain of top Fintech, Retail E-Commerce, Banking, Insurance and Education technologies.

He has worked at various corporate as Software Developer, Team Lead, Solution Architect, Program Manager and Corporate Trainer for Fortune 500 MNC's. Mr. Kiran Jadhav is teaching and interacting with more than 1000+ new candidates every month, including Software Developers and Foreign Candidates.

●●●●●

Table of Contents

Chapter 1. CORE JAVA

1. What is Java and list out its features?

Java is a high-level object-oriented programming language.

It is platform independent, robust, secure, and multithreaded programming language which makes it popular among other OOP languages.

Java developed by James Gosling in 1991. Initially they given name as "Oak" and later they renamed as "Java" in 1995. Now Java owned by Oracle.

It is widely used for Application Software, Web and Mobile Application Development.

Features:

1. Platform Independent :

Java is the first programming language that is Platform Independent. Once we have written program in Windows then we are able to run it on multiple platforms(Operating Systems), for example, Windows, Linux, Sun Solaris, Mac, Unix, etc. In other words, WORA- Write once Run anywhere. Java code can be executed anywhere on any systems.

2. Simple :

Java is easy to learn. The syntax of Java is based on C++ which makes easier to write the program in it.

Secondly, the concept of pointers has been completely removed from Java which leads to confusion for a programmer and pointers are also vulnerable to security.

3. Secure :

Java is best known for its security. With Java, we can develop virus free applications. In today's world everybody needs safety and security. Because of to the threat of hackers, people feel unsafe while using application over the Internet.To overcome such fears, Java provides safety and security features. Java provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, Oauth, JWT, Cryptography, JAAS, etc.

4. Object-Oriented :

In Java everything with in class and object. Oops have 4 pillars- Abstraction, Encapsulation, Polymorphism and Inheritance.

5. Multithreading:

Two or more thread simultaneously comes into execution and they will execute at a same time.

It consumes less memory and gives the fast and efficient performance.

6. Robust:

Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.

7. Dynamic:

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It allows to do runtime changes.

8. Distributed:

Java is distributed because it facilitates user to create distributed applications in Java, Example- Facebook, LinkedIn, Amazon, etc. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Java supports to distributed cloud platform like AWS, Azure, GCP OpenShift, etc.

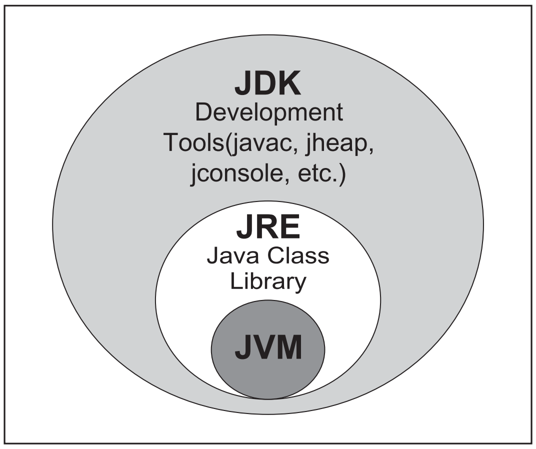
9. High Performance:

Java enables high performance with the use of just-in-time compiler. JVM can execute byte codes (highly optimized) very fast with the help of Just in time (JIT). Java application always give us good performance as compared to other programming languages.

2. Explain JDK, JRE, and JVM?

JDK, JRE, and JVM are core concepts of Java programming language.

JDK (Java Development Kit) is used for developing Java applications, JRE (Java Runtime Environment) is used for running Java applications, and JVM (Java Virtual Machine) is used for executing Java bytecode, regardless of the platform or operating system.



JDK(Java Development Kit):

- JDK is for development purpose.
- JDK provides all the tools, executables, and binaries required to compile, debug, and execute a Java Program.
- JDK is a platform-specific software and that's why we have separate installers for Windows, Mac, and Unix systems. JDK contains JRE with Java compiler, debugger, and core classes.

JRE(Java Runtime Environment):

- JRE is for running the java programs.
- It provides a platform to execute java programs.
- JRE consists of JVM, Java binaries, and other classes to execute any program successfully. If you just want to execute a java program, you can install only JRE.

JVM(Java Virtual Machine):

- JVM is the heart of Java programming language.
- When we execute a Java program, JVM is responsible for converting the byte code to the machine-specific code.
- JVM is also platform-dependent and provides core java functions such as memory management, garbage collection, security, etc.

3. What is platform independent?

Platform means Operating System. Platform independent means Java program run on any operating system.

Example- Once we have written program in windows it will be run on any operation system like Mac, Linux, Sun Solaris, Unix, etc.

This adaptability trait makes Java a superior programming language.

4. Explain JVM Architecture

JVM (Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a Java code. JVM is a part of JRE (Java Runtime Environment).

Java Programmer can develop Java code on one system and can expect it to run on any other Java-enabled system, it's possible because of JVM.

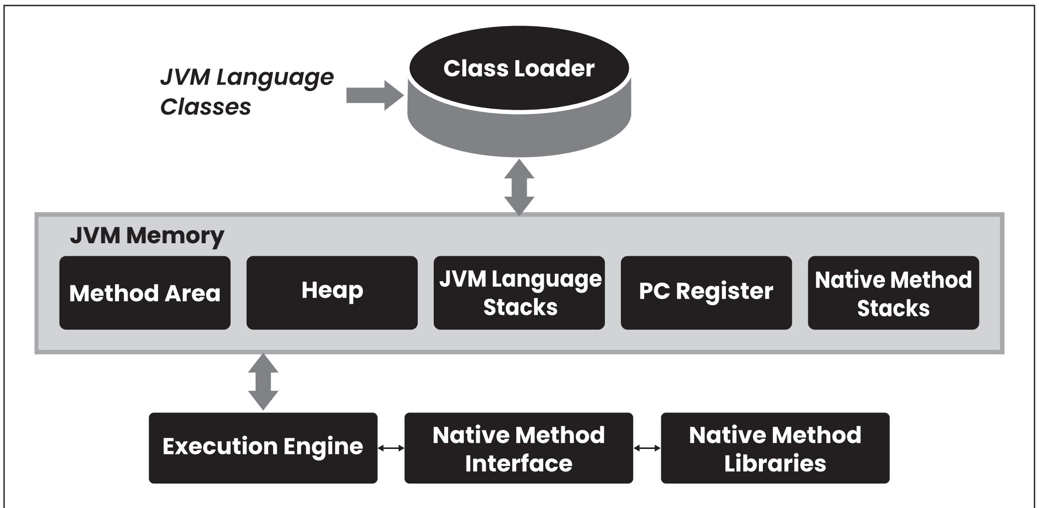


Fig. JVM Architecture

JVM Memory

- **Method area:** In the method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables. There is only one method area per JVM, and it is a shared resource.
- **Heap area:** Information of all objects is stored in the heap area. There is also one Heap Area per JVM. It is also a shared resource.

- **Stack area:** For every thread, JVM creates one run-time stack which is stored here. Every block of this stack is called activation record/stack frame which stores methods calls. All local variables of that method are stored in their corresponding frame. After a thread terminates, its run-time stack will be destroyed by JVM. It is not a shared resource.
- **PC Registers:** Store address of current execution instruction of a thread. Obviously, each thread has separate PC Registers.
- **Native method stacks:** For every thread, a separate native stack is created. It stores native method information.
- **Execution Engine:** Execution engine executes the ".class" (bytecode). It reads the byte-code line by line, uses data and information present in various memory area and executes instructions. It can be classified into 3 parts:
  1. **Interpreter:** It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required.
  2. **Just-In-Time Compiler (JIT):** It is used to increase the efficiency of an interpreter. It compiles the entire bytecode and changes it to native code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.
  3. **Garbage Collector:** It destroys un-referenced objects.

Native Method Interface: The Java Native Interface acts as a bridge between Java method calls and native library calls.

Native Method Libraries: It is a collection of the Native Libraries (C, C++) which are required by the Execution Engine.

How is Java Platform Independent?

Any language is said to be platform-independent if, in terms of development and compilation, it can run on all accessible operating systems.

Bytecode makes Java platform-independent. In simple words, it is a code in a java virtual machine that is easily understandable by the machine.

The java compiler converts the original code into a .class file first. The code in the class file is in byte code, and JVM converts it into an object file. Then the final output appears on your screen.

The execution of a Java program consists of 5 steps:

1. Creation of a Java Program
2. Compiling a Java program,
3. Loading the program into the memory by Java Virtual Machine
4. Java Virtual Machine verification for bytecode
5. Java Program execution

Example: HelloWorld.java-> javac-> HelloWorld.class-> java-> Output

5. What is JIT compiler in Java?

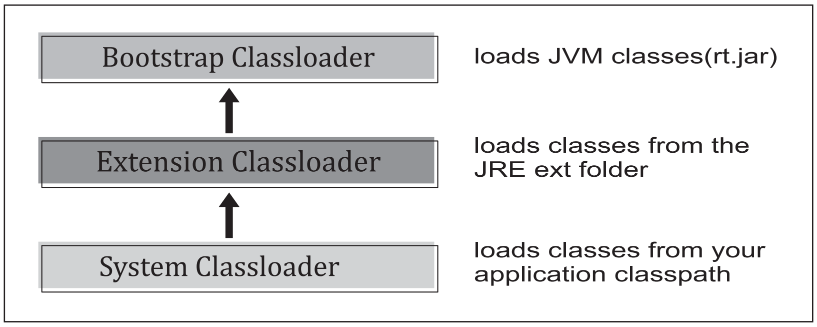
JIT stands for Just-In-Time compiler in Java. It is a program that helps in converting the Java bytecode into instructions that are sent directly to the processor. By default, the JIT compiler is enabled in Java and is activated whenever a Java method is invoked. The JIT compiler then compiles the bytecode of the invoked method into native machine code, compiling it "just in time" to execute. Once the method has been compiled, the JVM summons the compiled code of that method directly rather than interpreting it. This is why it is often responsible for the performance optimization of Java applications at the run time.

JIT compiler is the part of JVM which increases the speed of execution of a Java program.

6. What is ClassLoader in Java?

The Java ClassLoader is a part of the Java Runtime Environment that dynamically loads Java classes into the Java Virtual Machine.

Whenever we run the java program, it is loaded first by the classloader.


Bootstrap Classloader
loads JVM classes(rt.jar)
Extension Classloader
loads classes from the JRE ext folder
System Classloader
loads classes from your application classpath

There are 3 built-in classloaders in Java:

1. BootStrap ClassLoader
2. Extension ClassLoader
3. System ClassLoader

7. Why main() method is public, static and void in Java?

public : "public" is an access specifier which can be used outside the class. When main method is declared public it means it can be used outside class. It means access anywhere.

static : To call a method we require object. Sometimes it may be required to call a method without the help of object. Then we declare that method as static. JVM calls the main() method without creating object by declaring keyword static.

void: void return type is used when a method doesn't return any value. main() method doesn't return any value, so main() is declared as void.

main(): main() method is starting point of execution for all java applications.
String args[] are array of string objects we need to pass from command line arguments.

Every Java application must have at least one main method

8. Explain System.out.println().

It is used to print statement.

System is a class coming from java.lang package and .lang package by default available in Java, **out** is the object of PrintStream class.

println() is the method of PrintStream class.

9. Can we have multiple classes in single file ?

Yes, we can have multiple classes in single file but it not recommended. We can have multiple classes in single file but only one class can be made public. If we try to make two or more classes public in single file then we get compilation error.

10. What is variable and how many types of variables in Java?

Variable: It is a storage. It help us to store some values/data.

In Java 3 types of variables:

1. Instance variable: Declared with class level

2. Static Variable: Declared with class level and must be static.When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level. All instances of the class share the same static variable.

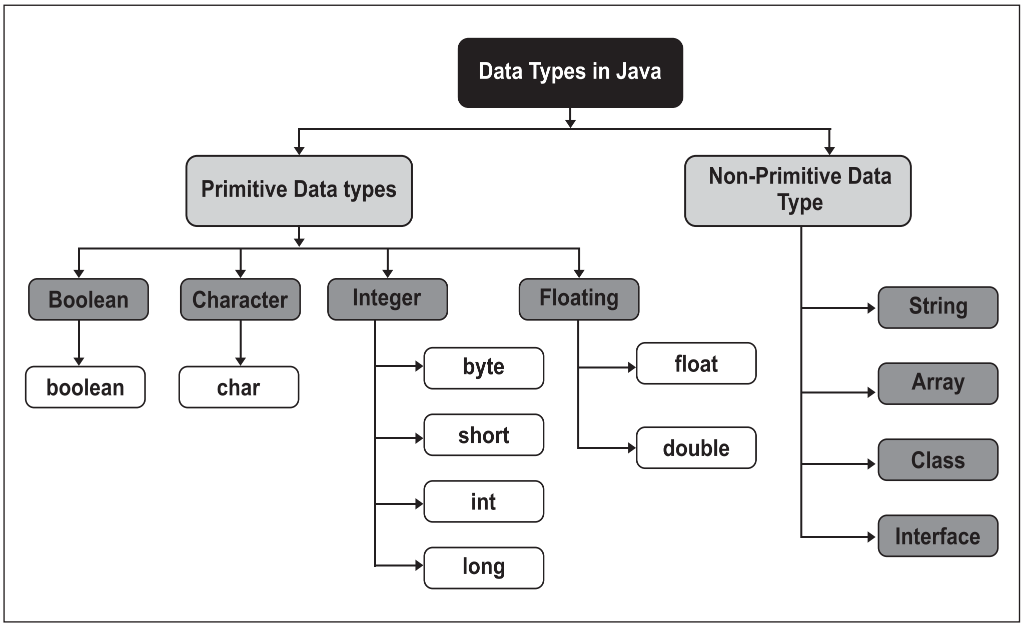
3. Local Variable: Declared inside method/block.

Example: int empId = 121;

Here, empId is variable.

11. What are the data types in Java?

Data types specify the different sizes and values that can be stored in the variable.

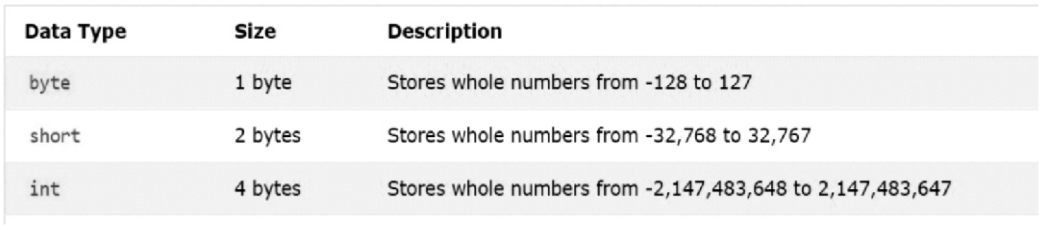


There are two types of data types in Java:

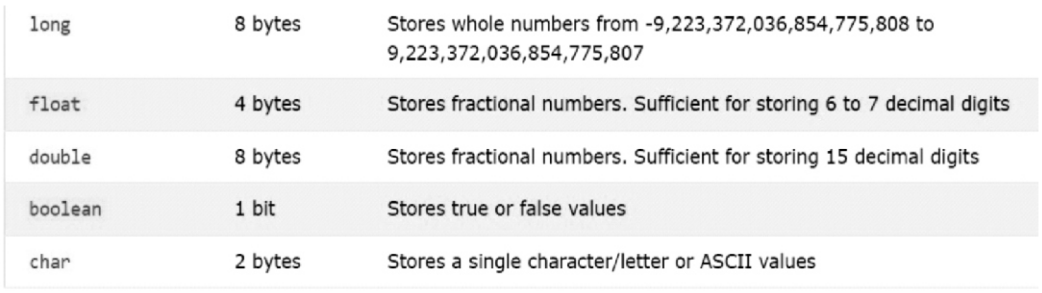
1. Primitive data types:
The primitive data types include byte, short, int, long, float, double, char and boolean.

2. Non-primitive data types: The non-primitive data types include String, Array, Class and Interface.

There are 8 Primitive Data types in Java:



Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647



long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

12. What do you understand about a ternary operator?

A ternary operator is a conditional operator responsible for deciding which values will be assigned to a variable.

The ternary operator (? :) consists of three operands. It is used to evaluate Boolean expressions. It is the only conditional operator that accepts three operands. It can be used instead of the if-else statement. It makes the code much more easy, readable, and shorter.

Example:

```
public class TernaryOperatorEx {
  public static void main(String[] args) {
    int marks = 68;
    System.out.println(marks >= 60 ? "Eligible for Campus" :
"Not Eligible for Campus");
  }
}
```

Output:

```
Eligible for Campus
```

13. What is Enum in Java?

In Java, an enum (short for enumeration) is a type that has a fixed set of constant values. We use the enum keyword to declare enums. It was introduced with the release of Java 5.

Enum represents a group of constants (unchangeable variables, like final variables). To create an enum, use the enum keyword and separate the constants with a "," separated.

NOTE Enum is by default public, static & final- Will always write enum in capital letters.

Example:

```
enum Size {
  SMALL, MEDIUM, LARGE, EXTRALARGE
}
```

14. What is Type Casting in Java?

Type casting means convert one datatype type into another datatype.

In Java there are 2 types of Type Casting:

1. **UpCasting (Widening):** lower to higher.
   Example: int to double
2. **DownCasting (Narrowing):** higher to lower.
   Example: double to int

15. What is autoboxing and unboxing in Java?

Autoboxing: Converting primitive data type into their corresponding wrapper classes.
Example: int to Integer, double to Double.

Unboxing: Unboxing is the reverse of autoboxing. It converts wrapper classes into a primitive data type.
Example: Integer to int, Double to double

16. What is an object?

Object is an instance of a class. And object can be created by using the new keyword.
Ex: Employee emp = new Employee();
Here, emp is an object. And object always start with lower case letter.

17. Explain 4 Pillars in Oops?

Oops stands for object-oriented programming. In oops everything comes under class and objects.

There are 4 Pillars:

1. Abstraction
2. Encapsulation
3. Polymorphism
4. Inheritance

1. **Abstraction:** Abstraction means hiding internal details and showing functionality.

   For example: phone call, we don't know the internal processing, but we know how to make call.

   In java, we use abstract class and interface to achieve abstraction.

2. **Encapsulation**: The wrapping up of a data into a single unit.
   Encapsulation refers to combining data and associated functions as a single unit.

   For example- Capsule, it is wrapped with different medicines.

   A Java POJO class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private. Provide public getter and setter methods to access and update the value of a private variable.

3. **Polymorphism:** Polymorphism means the ability to take more than one form. Polymorphism is Greek word; Poly means many, and morphism means form many form. And form means function, many functions it means many methods.

   Polymorphism uses methods to perform different tasks. This allows us to perform a single action in different ways.

   2 types of Polymorphism:

Compile Time (Method Overload)	**Run Time** (Method Override)
1. Method overload means same method name, but different parameters.	1. Method override means same method name and same parameters.
2. Method overload possible with in Single class	2. Method Override never possible with in single class, we must need to use Inheritance concept.
3. Static method allows to overload	3. Static method can't allow to override
4. Final method allows to overload	4. Final method can't allow to override

   Compile Time : Both methods must have different arguments. The number of arguments, sequence and type of arguments is different. The Java Compiler handles Compile-time Polymorphism.

   Run Time: The number of arguments, sequence and type of arguments is same. JVM handles the Runtime polymorphism.

4. **Inheritance:** One class can acquire the properties of another class.

Inheritance promotes the reusability of code.To inherit from a class, use the **extends** keyword.

In Oops there are 5 types of Inheritance:

1) Single Inheritance
2) Multilevel Inheritance
3) Multiple Inheritance
4) Hybrid Inheritance
5) Hierarchical Inheritance

Java does not support to Multiple and Hybrid Inheritance.

Because of Ambiguity problem, we can't extend's 2 class at a same time. Instead of multiple inheritance Java support to interfaces and by using interface will implements 2 or more interfaces with in single class by "," separated.

Benefits of Inheritance:

- Code reusability
- We can achieve Runtime Polymorphism using Inheritance.

18. What is Multiple Inheritance? And why Java does not support it?

If a child class inherits the property from multiple classes is known as multiple inheritance.

Java does not allow to extend multiple classes. The problem with multiple inheritance is that if multiple parent classes have the same method name, then at runtime it becomes difficult for the compiler to decide which method to execute from the child class.

Therefore, Java doesn't support multiple inheritance. The problem is commonly referred to as Diamond/Ambiguity Problem.

Instead of Multiple inheritance Java supports to interfaces, and in class will implements n number of interfaces by "," separated.

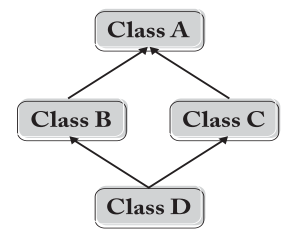


Fig.Multiple Inheritance

19. What happens if we try to run below Inheritance code?

- **First Scenario: Child Class Object Creation**

```
class Department {
  void get(int deptId, String deptName) {
    System.out.println("\n Department Id: " + deptId
      + "\n Department Name: " + deptName);
  }
}

class Employee extends Department {
  void set(int empId, String empName, double empSalary) {
    System.out.println("\n Employee Id: " + empId
      + "\n Employee Name: " + empName + "\n Employee Salary: "
      + empSalary);
  }
}

public class InheritanceScenarioEx {

  public static void main(String[] args) {

    Department department = new Employee();

    department.get(101, "HR");
    ((Employee) department).set(121, "SWARA", 96000);
  }

}
```

Output:

```
Department Id: 101
Department Name: HR
Employee Id: 121
Employee Name: SWARA
Employee Salary: 96000.0
```

NOTE Department department = new Employee();

By Creating child class object we are able to call methods from parent as well as child class.

- **Second Scenario: Parent Class Object Creation**

```
class Department {
void get(int deptId, String deptName) {

    System.out.println("\n Department Id: " + deptId
      + "\n Department Name: " + deptName);
  }
}

class Employee extends Department {
  void set(int empId, String empName, double empSalary) {
    System.out.println("\n Employee Id: " + empId
      + “\n Employee Name: " + empName + "\n Employee Salary: "
      + empSalary);

  }
}

public class InheritanceScenarioEx {
  public static void main(String[] args) {
    Employee employee = (Employee) new Department();
    employee.get(101, "HR");
    employee.set(121, "SWARA", 96000);
  }
}
```

Output:

```
Once we handle compile time error using cast-
Employee employee = (Employee) new Department();
then, it will give us Runtime Exception.
Exception in thread "main" java.lang.ClassCastException
```

Employee employee = (Employee) new Department();

Here, creating parent class object we are not able to call child class methods. And it will give us ClassCastException at Runtime.

20. Differentiate Abstract Class and Interface.

Interface	Abstract Class
1. All methods are abstract till JDK 7. But from JDK 8 interface supports 2 non-abstract methods- static and default	1. Abstract class support to abstract & non-abstract methods
2. In interface all methods are public	2. In abstract class methods are may/not be public, final & static
3. All variables are public, final & static	3. In abstract class variables are may/not be public, final & static
4. Interface does not support to constructor	4. Abstract class support to constructor
5. Interface can't instantiate	5. Abstract class can't instantiate
6. Keyword- interface	6. Keyword- abstract
7. Interface – Interface : extends	7. Interface- Class : implements Class- Class : extends
8. With in class will implements n no of interfaces by “,” separated	8. With in class will extends only one class at a time.

21. What is Aggregation in java?

Aggregation represents the weak relationship

Example: Car has an indicator (aggregation)

22. What is composition in java?

Composition represents the strong relationship.

Example: Car has an engine (composition)

23. What are access modifiers in Java?

Access modifiers it means scope of visibility

In Java 4 types of access modifiers:

1. Private: Scope with in class
2. Default: Scope with in Package
3. Protected: Scope with in Package and outside package but within subclass

4. Public: Access everywhere

Access Modifier	Witin Class	Within Package	Same Package by subclasses	Outside Package by subclasses	Global
public	Yes	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	Yes	No
default	Yes	Yes	Yes	No	No
private	Yes	No	No	No	No

24. What is static block?

Static blocks are executed before the main() method. Static blocks executed automatically when the class is loaded in memory. A class can have any number of static initialization blocks.

The execution of multiple static blocks will be in the sequence as written in the program.

NOTE 1. If you want to execute code before main method then such code/functionality you can add in static block.
2. Static block it is used for pre-initialization

25. What is static in Java?

Static is a reserved keyword in Java.

The main use of the static keyword in java is used for memory management in java. Whenever we place a static keyword before the initialization of a particular class's methods or variables, these static methods and variables belong to the class instead of their instances or objects.

1. **Static Variable:** Static variables can be used to refer to the common property of all objects (which is not unique for each object).
   Example- company name of employees, CEO of a company, etc. It makes the program memory efficient (i.e. it saves memory).

   The companyName variable is of static type which implies it is shared among all instances of the Employee class.

NOTE The static variables are those variables that are common to all the instances of the class. Only a single copy of the static variable is created and shared among all the instances of the class.

Static variable is a class-level variable, memory allocation of such variables only happens once when the class is loaded in the memory.
If an object modifies the value of a static variable, the change is reflected across all objects.

2. **Static Method:** Once we declared any method with static then not need to create object to access with in main method.

   Static method allows to overload, but it can't allow to override.
3. **Static Class:** Outer class never static, but inner class will be declared as static.
4. **Static Block:** It always execute before main method.

26. Is it possible to override the static method in Java?

No, it is not possible to override the static method in Java. The reason is

1. The static method belongs to the class level, not the object level. In method overriding, it is the object that decides which method is to be called.
2. Also, for class-level methods i.e static methods, the type reference decides which method is called not the object being referred. It concludes the method called is determined at compile time.If a child class defines a static method with the same signature as a static method in the parent class, then the method in the child class hides the method in the parent class.

27. Explain Constructors in Java.

Constructor itself initialize at the time of object creation.

Every time an object is created using the new() keyword, at least one constructor is called.

In Java Constructor have 2 types:

1. Default Constructor: No Parameter

In this Constructor we are not passing any parameters

2. Parameterized Constructor: Passing Parameters

In this Constructor we are going to pass some parameters

NOTE

1. Constructor itself initialize at the time of object creation.
2. Constructor name same as class name.
3. Constructor does not have any return type.
4. Constructor Overload possible in Java.
5. Constructor Override never possible in Java.
6. Java does not support to copy constructor, instead of copy constructor Java support to object cloning concept.

28. What is object cloning?

Object cloning we are able to achieve in Java by using Cloneable interface and override clone() method.

Object cloning in Java is the process of creating an exact copy of an object. It basically means the ability to create an object with a similar state as the original object. To achieve this, Java provides a method clone() to make use of this functionality. This method creates a new instance of the class of the current object and then initializes all its fields with the exact same contents of corresponding fields. To object clone(), the marker interface java.lang.Cloneable must be implemented to avoid any runtime exceptions. One thing you must note is Object clone() is a protected method, thus you need to override it.

29. What is singleton class and how can we make a class singleton?

Singleton class is a class whose only one instance can be created at any given time,in one JVM. A class can be made singleton by making its constructor private.

30. What is Constructor Chaining?

We can call a constructor of a class inside another constructor this is called as constructor chaining.

Constructor chaining can be done in 2 ways:

1. **Within the same class:** For constructors in the same class, the this() keyword can be used.
2. **From the base class:** The super() keyword is used to call the constructor from the base class.

The constructor chaining follows the process of inheritance. The constructor of the sub class first calls the constructor of the super class. Due to this, the creation of sub class's object starts with the initialization of the data members of the super class. The constructor chaining works similarly with any number of classes. Every constructor keeps calling the chain till the top of the chain.

31. What is this keyword in java and its usage?

The this keyword is a reference variable that refers to the current object.

Uses of this keyword-

1. this can be used to refer to the current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this can be used to invoke the current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as an argument in the constructor call.
6. this can be used to return the current class instance from the method.

32. What is super in java?

Super keyword is used for to access member properties(variables and methods) coming from immediate parent class.

Super can be used to invoke immediate parent class constructor.
The super() is called in the class constructor implicitly by the compiler if there is no super or this.

33. Which class is the superclass for all the Java classes?

The Object class is superclass for all the Java classes.

34. Can we overload the main() method?

Yes, we can have any number of main methods in a Java program by using method overloading.

Method overload means same method name but different parameters.

35. What is covariant return type?

Since Java 5, it is possible to override any method by changing the return type of the subclass.
The covariant return type specifies that the return type may vary in the subclass.

Example:

```
class Company {
  Company get() {
    return this;
  }
}

class Product extends Company {
  Product get() {
    return this;
  }

  void show() {
    System.out.println("Welcome to Full Stack Java Developer,
      Pune- Covarient Ex");
  }
}

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

    new Product().get().show();
  }

}
```

Output:

```
Welcome to Full Stack Java Developer, Pune- Covarient Ex
```

36. Explain SOLID Principles?

SOLID principles are an object-oriented approach to software design & development that is used in Java.

1. Single Responsibility
2. Open/Closed
3. Liskov Substitution
4. Interface Segregation
5. Dependency Inversion

- **Single Responsibility Principle:** One class should have one and only one responsibility.

- **Open Closed Principle:** Software components should be open for extension, but closed for modification.
- **Liskov's Substitution Principle:** Derived types must be completely substitutable for their base types.
- **Interface Segregation Principle:** Clients should not be forced to implement unnecessary methods which they will not use.
- **Dependency Inversion Principle:** Depend on abstractions, not on concretions.

Benefits:

1. Clean: SOLID principles make code clean and standard code.
2. Maintainable: with the help of SOLID principles our code becomes more manageable and easier to maintain.
3. Scalable: Easy to refactor or change code.
4. Redundancy: SOLID principles avoid redundant code.
5. Testable: can be easily unit tested.
6. Readable: SOLID principles make the code easy and readable.
7. Independent: code becomes independent by reducing dependencies.
8. Reusable: code becomes reusable.

37. What is Marker Interface & How many marker interface available in Java?

An interface that does not have any methods, an empty interface in java is known as Marker Interface.

In Java 3 Marker Interfaces:

1. **Serializable:** The Serializable interface coming from java.io package. It is used for convert object into byte stream.
2. **Cloneable:** The Cloneable interface coming from java.lang package. It is used for clone the object.
3. **Remote:** The Remote interface coming from java.rmi package. It is used for Network Programming

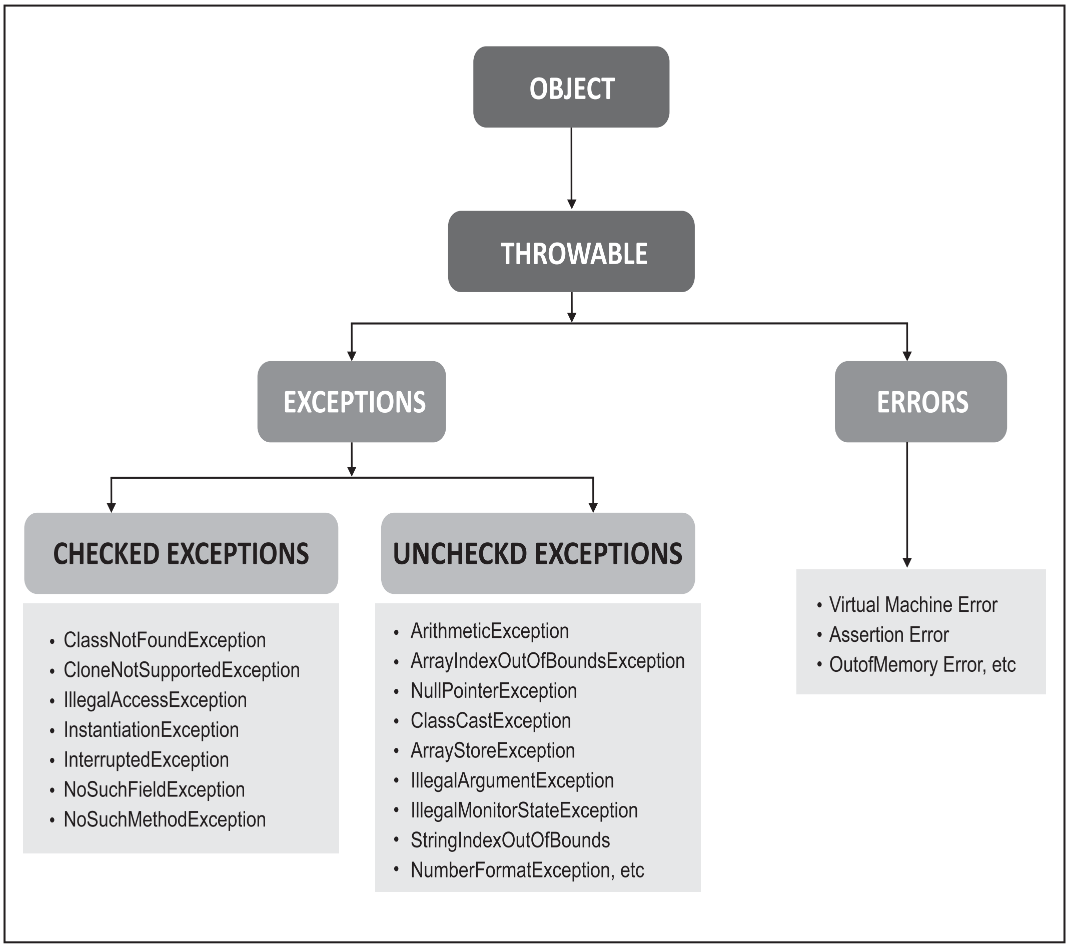
Uses of Marker Interface:

- The main use of the Marker Interface in Java is to convey to the JVM that the class

- implementing some interface of this category has to be granted some special behavior. e.g, when a class implements the Serializable interface, which is a marker interface, then this is an indication to the JVM that the objects of this class can be serialized.
- Similarly, when a class implements Cloneable Interface, then it indicates to the JVM that the objects of this class can be cloned.
- When a class implements Remote Interface, then it indicates to the JVM there should be use for Network- Socket Programming.

38. What is Exception and How to handle it?

Exception is an abnormal condition which is interrupt normal flow of program. Exception is a class coming from java.lang package, and Exception have immediate parent class is Throwable.



In Java 2 types of Exception :

A. Checked (Compile Time) Exception:

1. Those exceptions that are checked at compile-time also known as checked exceptions.
2. The program will not compile if they are not handled.
3. They are child classes of Exception except for RuntimeException.

Example: IOException, SQLException, etc.

B. Unchecked (Run Time) Exception:

1. Those exceptions that are checked at runtime also known as unchecked exceptions.
2. They give runtime errors if not handled explicitly.
3. They are child classes of RuntimeException.

Example: ArithmeticException, NullPointerException, etc.

There are 5 blocks for Exception Handling:

1. Try: Exceptional code will add in try block
2. Catch: It is used for catch the Exception
3. Throw: It is user defined Exception
4. Throws: It is system generated Exception
5. Finally: It always execute.

39. What is the parent class for Exception?

Throwable is a parent class of Exception

40. Can we write multiple catch blocks under single try?

Yes, we can have multiple catch blocks under single try block.

41. Differentiate throw and throws.

Throw	Throws
1. It is used for Custom/User Defined Exception	1.It is System Generated Exception

2. Throw will declare inside method	2.Throws will declare at the time of method declaration
3. In throw will handle only one Exception at a time	3.In throws will handle multiple Exception by “,” separated
4. Will call throw explicitly	4.Throws calls implicitly

42. What will be the output of below code?

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

    try {
      int result = 10 / 0;
    } catch (Exception e) {
      // Here, it will give us compile time error. Parent Exception
         must be declared at last catch block

      e.printStackTrace();
    } catch (ArithmeticException e) {
      e.printStackTrace();
    }
  }

}
```

Output:

```
It will give us compile time error, because Parent class Exception
must be used at last catch block.
```

43. Will the finally block get executed when the return statement is written at the end of try block and catch block as shown below?

Yes, The finally block always gets executed even when the return statement is written at the end of the try block and the catch block. It always executes, whether there is an exception or not.

If you are adding System.exit(0) in try block then finally block does not execute.

What is the output of below code:

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

    System.out.println(get());
  }

  static int get() {
    int result = 10;
    try {
      result = 10 / 0;
      return result;
    } catch (Exception e) {
      return result;
    } finally {
      System.out.println("It's Finally");
    }

  }
}
```

Output:

```
It's Finally
10
```

44. Differentiate final, finally and finalize.

Final- Keyword	Finally- Block	Finalize- Method
Final Variable: Can't reinitialize Final Method: Can't override Final Class: Can't extends	Finally is the block. It always executes. Realtime Use: Will add Database Connection close, Hibernate Session close code in finally block. Note: If you are adding System.exit(0); in try block then finally block does not execute	finalize() is a method it help us to achieve garbage collection in Java. Garbage collection is a process of reclaiming the unused runtime objects. It is performed for memory management. In other words, we can say that It is the process of removing unused objects from the memory to free up space and make this space available for Java Virtual Machine. Java does not supports to destructor, In Java by default System.gc() method available. By referencing object as null and call to System.gc(); method.It will help us to perform clean up.

45.What is try-with-resources?

try-with-resources introduced in Java 7.
It help us to avoid resource leakage and close object automatically. And we are able to use here try without catch and finally.

Example:

```
public class TryWithResoursesEx {
  public static void main(String[] args) {
    try (Scanner scanner = new Scanner(System.in)) {
    }
  }
}
```

46. What is OutOfMemoryError in Java?

OutOfMemoryError is the subclass of java.lang.Error which generally occurs when our JVM runs out of memory.

47. Why String class is Immutable?

String is a class coming from java.lang package and its Immutable.
An immutable string is thread-safe. If one does not make a string immutable, then a change in one reference will inevitably affect the values of other references.

There are **five reasons** String class in Java being immutable are security, caching, synchronization, class loading and performance.

1. Security: parameters are typically represented as String in network connections, database connection URLs, usernames/passwords etc. If it were mutable, these parameters could be easily changed.

2. Caching: when compiler optimizes your String objects, it sees that if two objects have same value (a="Java", and b="Java") and thus you need only one string object (for both a and b, these two will point to the same object).

3. Synchronization and concurrency: making String immutable automatically makes them thread safe thereby solving the synchronization issues.

4. Class loading: String is used as arguments for class loading. If mutable, it could result in wrong class being loaded (because mutable objects change their state).

5. Performance: The hashcode of string is frequently used in Java. For example, in a HashMap. Being immutable guarantees that hashcode will always the same, so that it can be cached without worrying the changes. That means, there is no need to calculate hashcode every time it is used.

48. Differentiate StringBuffer and StringBuilder.String Builder:

StringBuffer	String Builder
1.StringBuffer is mutable	1.StringBuilder is mutable
2.Thread Safe	2.Not Thread Safe
3.Thread Synchronized	3.Not Thread Synchronized
4.Comes in JDK 1.0 version	4.Comes in JDK 1.5 version
5.Performance wise slower than StringBuilder	5.Performance wise faster than StringBuffer

49. What is String Pool?

String pool is a storage space. All string literals/variables are stored in String Pool. It is also known as String Constant Pool or String Intern Pool. It is privately maintained by the Java String class. By default, the String pool is empty.

50. How many ways can we create the string object.

1. String Literal

Java String literal is created by using double quotes.
Example: String s="FullStack";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. String objects are stored in a special memory area known as the string constant pool.

Example:
String s1 = "FullStack";
String s2 = "FullStack"; // It doesn't create a new instance

2. By new keyword

String s = new String("FullStack"); //creates two objects and one reference variable.
Here, 2 objects, one in string constant pool and other in non-pool(heap).

51. Differentiate == and .equals().

==	Equals()
1.This operator is used for comparing addresses (or references), i.e checks if both the objects are pointing to the same memory location.	1.It will compare actual values
2.It is a binary operator in Java.	2.This is a method defined in the Object class.

52. What will be the output of below code?

```
public static void main(String[] args) {
  String s1 = "FullStack";

  String s2 = new String("FullStack");

  System.out.println(s1 == s2);
  System.out.println(s1.equals(s2));
}
```

Output:

```
false
true
```

53. How to make custom immutable class in Java?

Immutable means we can't modify it.

Steps to create Custom Immutable class in Java:

1. Declare class with final
2. All variables are private
3. Don't use setter, use Constructor
4. Make all mutable fields private and final so that its value can be assigned only once.

54. Explain Collection Framework in detail.

Collection itself interface and Collections is a class in Java.Collection it is a group of object.

Java programming language provides a lot of inbuilt data structures and algorithms so that we do not have to design a data structure from scratch every time. These data structures and algorithms are also optimized to a great extent. These optimised data structures and algorithms form the bulk of the Collections framework. They provide a means to perform operations like sorting, insertion, deletion, updating, and searching on data members and objects. The collection framework also provides interfaces and classes with defined methods to perform these functions.

Inside the Collection framework is the Collection interface, Map interface and Iterable interface.

In Collection Framework there are 3 important interfaces:

1. List: It allows duplication of data and follows insertion order

In List there are 2 classes:

1. **ArrayList:** ArrayList Follows DDL(Data Definition Language)
   ArrayList more suitable for Read data.

   Example- In Google Pay check Account Balance.

2. **LinkedList:** LinkedList Follows DML(Data Manipulation Language)
   LinkedList more suitable for Insert/Update/Delete.

   Example- In Google Pay transfer/send money.

2. Set: Uniqueness, does not allow duplication of data.

In Set there are 2 classes:

1. **HashSet:** HashSet does not maintain order
   HashSet allows one null element

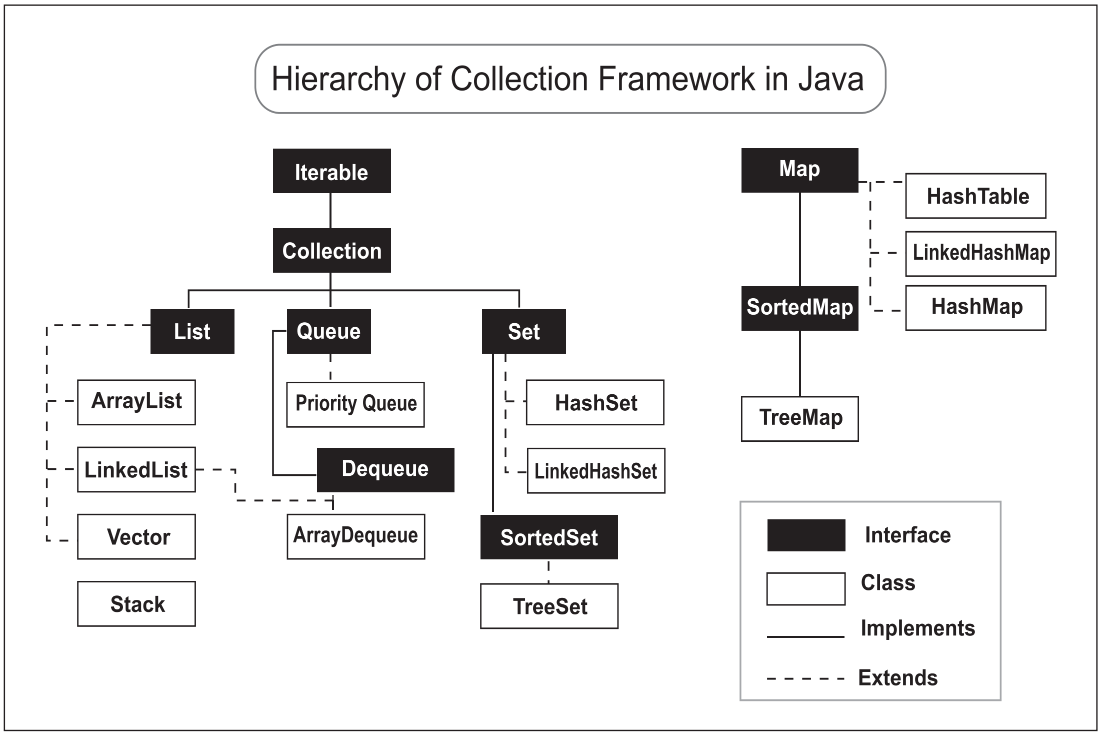
2. **TreeSet:** TreeSet Follows order
   TreeSet does not allow any null element, if any null element added then it will give NullPointerException.

3. Map: It is Key, Value pair like JSON. Key must be unique, but values allow as duplicate

In Map there are 2 classes:

1. **HashMap:** HashMap does not maintain order
   HashMap allows one null key, and allows multiple null values

2. **TreeMap:** TreeMap Follows order
   TreeMap does not allows any null key, but it allows multiple null values.



55. Differentiate Array List and Linked List.

Both are the classes under java.util package comes under List interface. And List allows to duplication of data.

ArrayList	LinkedList
1. ArrayList follows DDL- Data Definition Language	1. LinkedList follows DML- Data Manipulation Language
2. ArrayList is more suitable for read data	2. LinkedList is more suitable for Insert/Update/Delete data
3. ArrayList follows insertion order	3. LinkedList follows insertion order
4. ArrayList RealTime Example: In PhonePe Check Account Balance, will prefers to use ArrayList	4. LinkedList RealTime Example: In PhonePe Transfer Funds(Send Money), will prefers to use LinkedList

56. When ConcurrentModificationException come?

In iteration(for each, Iterator) if there is any modification happens like add, remove then ConcurrentModificationException comes at Run Time.

57. What is Failfast and Fail safe iterator in Java?

Iterator in Java is used to traverse over a collection's objects. The collections return two types of iterators, either it will be Fail Fast or Fail Safe.

Fail Fast: It means in iteration(for each/Iterator) if we are doing add/remove operations then ConcurrentModificationException occurs. Once Exception occurs it will stopping the whole operation.

These types of iterators do not allow modifying the collection while iterating over it.
No extra memory is required in this case.

Fail Safe: It avoid the Exception- ie. ConcurrentModificationException.
A fail-safe iterator does not throw any exceptions, if the collection is modified during the iteration process.

These types of iterators allow modifying the collection while iterating over it.
Extra memory is required in this case.

Example: CopyOnWriteArrayList, ConcurrentHashMap those are fail safe iterator. Both never throw ConcurrentModificationException in Java

58. How HashMap internally works?

HashMap is a class comes under Map interface.Map is an interface coming from java.util package.In Map no duplicate key but it allows multiple duplicate values

In Map there are 2 classes:

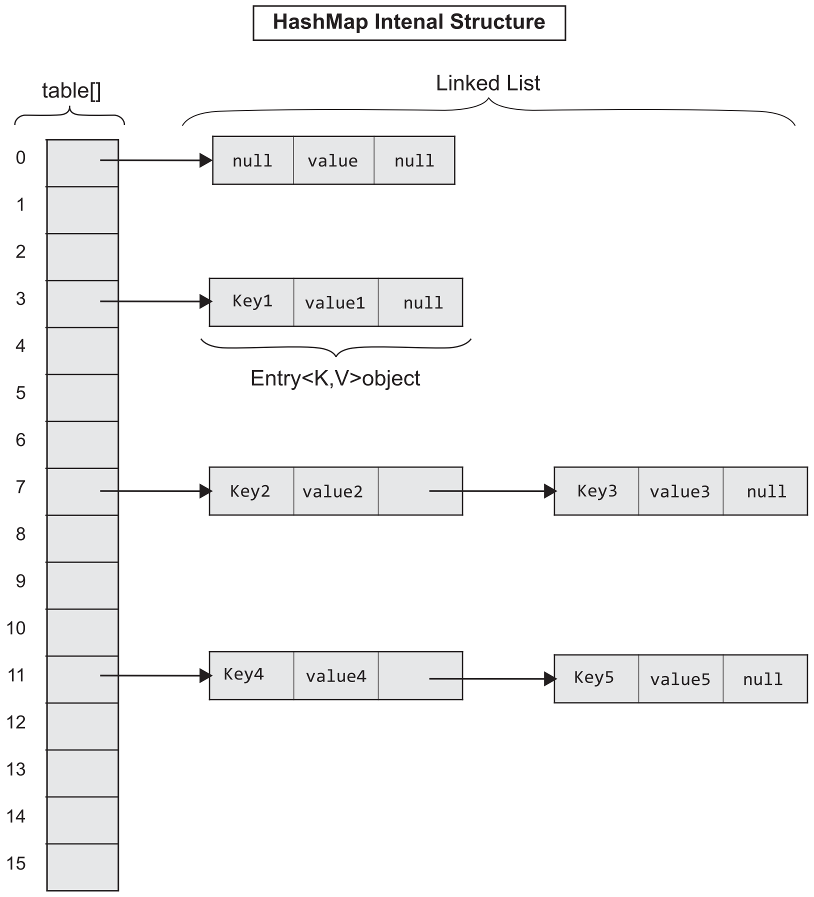
1. **HashMap:** Does not maintain order
   HashMap allows one null key, and multiple null values
2. **TreeMap:** Follows order
   TreeMap does not allow any null key, but it allows multiple null values.

Internal Working of HashMap:

In HashMap there is one bucket and its size is 16 block- 0 to 15

Buckets: Array of the node is called buckets. Each node has a data structure like a LinkedList. More than one node can share the same bucket. It may be different in capacity.
HashMap uses Hashing Principle.
For adding elements in HashMap put() method available.
HashMap internally use LinkedList data structure for storing Key and Value
There are 2 methods- equals() and hashcode()

- **equals():** If two hashmap objects are equals then it must be same hascode.
  The hashCode() method returns a hash code value (an integer number).
- **hashcode():** If two hashcode are same then collision will be arrived. And once collision arrived it will override the same block.



59. What is LinkedHashMap?

The LinkedHashMap class extends the HashMap class to store its entries in a hash table. It internally maintains a doubly-linked list among all of its entries.

- LinkedHashMap follows insertion order.
- LinkedHashMap is a Key, Value pair.
- LinkedHashMap allows one null key and multiple null values.
- LinkedHashMap is not synchronized.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

60. What is ConcurrentHashMap ?

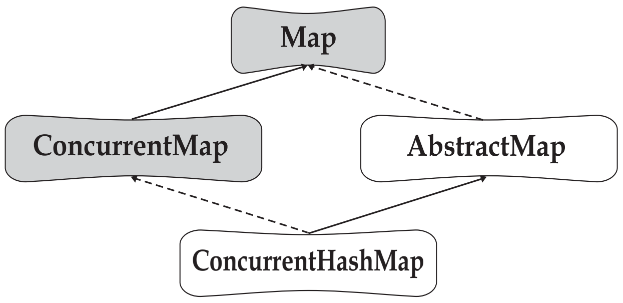
The ConcurrentHashMap class is introduced in JDK 1.5 it comes from java.util.concurrent package, which extends AbstractMap and implements ConcurrentMap as well as to Serializable interface. ConcurrentHashMap is an enhancement of HashMap as we know that while dealing with Threads in our application HashMap is not a good choice because performance-wise HashMap is not up to the mark.

ConcurrentHashMap is a thread-safe implementation of the Map interface in Java, which means multiple threads can access it simultaneously without any synchronization issues.

ConcurrentHashMap is the Map implementation that allows us to modify the Map while iteration. The ConcurrentHashMap operations are thread-safe. ConcurrentHashMap doesn't allow null for keys and values.

Important points of ConcurrentHashMap:

1. ConcurrentHashMap class is thread-safe i.e. multiple threads can operate on a single object without any complications.
2. The underlined data structure for ConcurrentHashMap is HashTable.
3. At a time any number of threads are applicable for a read operation without locking the ConcurrentHashMap object which is not there in HashMap.
4. In ConcurrentHashMap, the Object is divided into a number of segments according to the concurrency level.
5. The default concurrency-level of ConcurrentHashMap is 16.
6. In ConcurrentHashMap, at a time any number of threads can perform retrieval operation but for updated in the object, the thread must lock the particular segment in which the thread wants to operate. This type of locking mechanism is known as Segment locking or bucket locking.



The Hierarchy of ConcurrentHashMap:

class ConcurrentHashMap<K,V> extends AbstractMap<K,V> implements ConcurrentMap<K,V>, Serializable

61. Differentiate HashMap and HashTable.

HashMap	HashTable
1. HashMap allows one null key and multiple null values.	1. Hashtable does not allow any null key or value.
2. HashMap is not Thread Safe and not Thread synchronized.	2. Hashtable is Thread Safe and Thread Synchronized
3. HashMap is fast.	3. Hashtable is slow
4. Iterator in HashMap is fail-fast.	4. Enumerator in Hashtable is not fail-fast.
5. HashMap is a new class introduced in JDK 1.2	5. Hashtable is a legacy class and its from JDK 1.0

62. Differentiate Comparable and Comparator.

Both are interface in Java

Comparable	Comparator
1. Comparable coming from java.lang package	1. Comparator coming from java.util package
2. Comparable have only one method: compareTo()	2. Comparator have two methods: compare() and equals(), but equals() is not mandatory to override.
3. Comparable it is used for single sequence sort	3. Comparator is used for multiple sequence sort
4. The Collections.sort(List) method can be used to sort Comparable type list members.	4. The Collections.sort(List, Comparator) method can be used to sort the list components of the Comparator type.

63. Differentiate between ArrayList and Vector.

ArrayList	Vector
1. ArrayList is not a legacy classand comes in JDK 1.2	1. Vector is a legacy class and comes in JDK 1.0
2. ArrayList is Non-synchronized	2. Vector is Synchronized
3. Increases size by 1/2 of the ArrayList	3. Increases size by double of the ArrayList
4. ArrayList is not thread-safe	4. Vector is thread-safe

64. How to convert Array into ArrayList?

We can convert an Array to ArrayList by using asList() method of Arrays class. asList() method is the static method of Arrays class and accepts the List object.
Example: Arrays.asList(item);

65. What is hashCode() method?

- The hashCode() method returns a hash code value (an integer number).
- The hashCode() method returns the same integer number if two keys (by calling equals() method) are same. However, it is possible that two hash code numbers can have different or the same keys.
- If two objects do not produce an equal result by using the equals() method, then the hashcode() method will provide the different integer result for both the objects.

66. Why we override equals() method?

The equals method is used to check whether two objects are the same or not. It needs to be overridden if we want to check the objects based on their property.

Example: Employee is a class that has 3 data members: empId, empName and empSalary.

However, we want to check the equality of employee object
then, we need to override the equals() method.

NOTE Equals and HashCode Contract

1. If two objects are equal by equals() method then their hash code values must be same.
2. If two objects are not equal by equals() method then thier hash code may be same or different.

```
import java.util.Objects;

class Employee {
  private int empId;
  private String empName;
  private double empSalary;

  public Employee(int empId, String empName, double empSalary) {
    super();
    this.empId = empId;
    this.empName = empName;
    this.empSalary = empSalary;
  }
  @Override
  public int hashCode() {
    return Objects.hash(empId, empName, empSalary);
  }
  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    Employee other = (Employee) obj;
    return empId == other.empId && Objects.equals(empName,
      other.empName) && Double.doubleToLongBits(
        empSalary) == Double.doubleToLongBits(other.empSalary);
  }
}
public class EqualsAndHashCodeContractEx {
  public static void main(String[] args) {
    Employee employee1 = new Employee(121, "SWARA", 96000.22);
    Employee employee2 = new Employee(121, "SWARA", 96000.22);
    System.out.println(employee1.equals(employee2));
  }
}
```

Output:

```
true
```

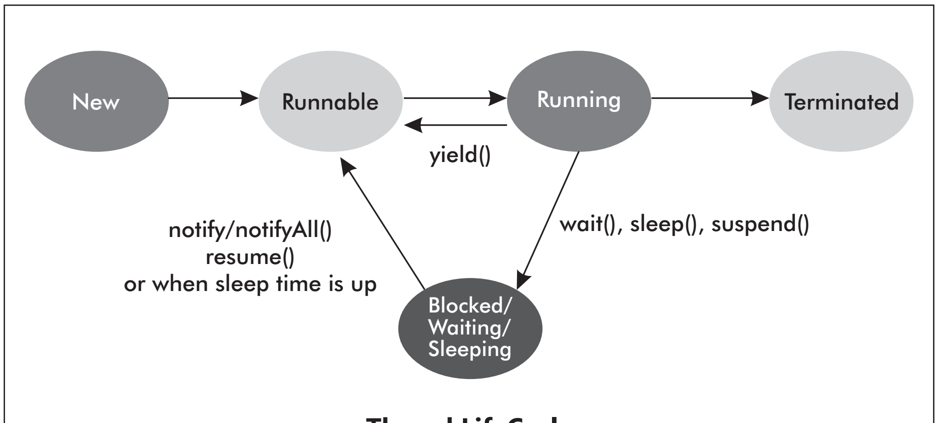
NOTE Here, if we are not followed equals() and hashcode() contracts then it will give us output as false.

67. What is Thread and Explain Thread Lifecycle?

Thread: It is a Lightweight part of process. Thread is a class coming from java.lang package

We are able to create Thread in 2 ways:

1. **extends Thread class**
2. **implements Runnable interface**



Thread LifeCycle

There are 5 states: New, Runnable, Running, Waiting, Terminated

1. **New:** When a thread object is created using new keyword, it is in a state that is called 'New' state.
2. **Runnable:** We use thread.start(); method to start a thread.
   The thread is registered in a pool of threads which is refered by the Thread scheduler to run a particular thread.
3. **Running:** Once Thread is started by using thread.start() method then run() method invokes internally and Thread goes in running state.
4. **Blocked/Waiting/Sleeping:** A thread which is running can be pushed to one of the three states: Blocked/Waiting/Sleeping.

   Blocked state is the one in which a thread is blocked for some resource or a lock which is already acquired by another thread.

   Waiting state is the one in which a thread goes after calling wait() method.

   Sleeping state is the one in which a thread goes after calling sleep() method.

All these 3 states are clubbed as one state because they are more or less similar. In any of these states, the thread is not eligible for running.

5. **Terminated or Dead:**

   A thread goes to Terminated or Dead state when it completes its run() method execution. Once dead, a thread cannot be run again. If someone tries to start a dead thread, then an IllegalThreadStateException is thrown.

68. What is multithreading?

Two or more thread simultaneously comes into execution and they will execute at a same time. It consumes less memory and gives the fast and efficient performance.

69. Which is the best approach for creating thread ?

- The best way for creating threads is to implement runnable interface.
- When we extend Thread class we can't extend any other class.
- When we create thread by implementing runnable interface we can implement Runnable interface and extends class as well. In both ways we have to implement run() method.

70. Can we start same Thread Twice?

No. It will give us Runtime exception- IllegalThreadStateException

71. What is Thread Synchronization?

Two or more Thread simultaneously comes into execution but only one Thread execute at a time that's called Thread Synchronization

Why we use Synchronization:

- Synchronization helps in preventing thread interference.
- Synchronization helps to prevent concurrency problems.

Thread Synchronization can be achieved in the following ways.
1. Synchronized Method
2. Synchronized block
3. Static Synchronization

Lock Concept in Java:

Synchronization Mechanism developed by using the synchronized keyword in java language. It is built on top of the locking mechanism; this locking mechanism is taken care of by Java Virtual Machine (JVM). The synchronized keyword is only applicable for methods and blocks, it can't apply to classes and variables. Synchronized keyword in java creates a block of code is known as a critical section. To enter into the critical section thread needs to obtain the corresponding object's lock.

72. Differentiate wait() and sleep() method.

wait()	sleep()
1. wait() method used for Interthread communication	1. sleep() method used for stops the execution of the current thread for some specified period.
2. It is defined in the object class	2. It is defined in thread class
3. Get wait Thread in execution state then need to call notify or notifyAll() method.	3. In sleep() method after some time interval Thread will comes into execution state
4. The wait() method releases the lock.	4. The sleep() method doesn't release the lock.

73. What is the use of join() method?

join() method waits until current thread dead or completed. Once Current Thread execution is completed then another Thread comes into execution.

It is considered the final method of a thread class.

Joining threads in Java has three functions namely,

1. **join():** Waits for this thread to die
2. **join(long millis):** Waits at most millis milliseconds for this thread to die
3. **join(long millis, int nanos):** Waits at most millis milliseconds plus nano nanoseconds for this thread to die

Example:

```
class MyThread extends Thread {
  public void run() {
    for (int i = 1; i <= 5; i++) {
      try {
        Thread.sleep(1000);
        System.out.println(i); }
catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}

public class JoinEx {
  public static void main(String[] args)
    throws InterruptedException {
    MyThread thread1 = new MyThread();
    MyThread thread2 = new MyThread();
    MyThread thread3 = new MyThread();

    thread1.start();
    thread1.join();
    thread2.start();
    thread3.start();
  }
}
```

Output:

```
1
2
3
4
5
1
1
2
2
3
3
4
4
5
5
```

74. Explain yield() method in thread class?

Yield() method makes the current running thread to move in to runnable state from running state giving chance to remaining threads of equal priority which are in waiting state. yield() makes current thread to sleep for a specified amount of time.There is no guarantee that moving a current running thread from runnable to running state. It all depends on thread scheduler it doesn't guarantee anything.

Calling yield() method on thread does not have any affect if object has a lock. The thread doesn't lose any lock if it has acquired a lock earlier.

75. What is inter-thread communication?

Inter-thread communication is a process or mechanism using which multiple threads can communicate with each other.

It is especially used to avoid thread polling in java.

Using wait(), notify(), and notifyAll() methods we are able to achieve inter-thread communication.

76. Explain thread scheduler in java?

Thread scheduler is a part of JVM use to determine which thread to run at this moment when there are multiple threads. Only threads in runnable state are chosen by scheduler.

Thread scheduler first allocates the processor time to the higher priority threads. To allocate microprocessor time in between the threads of the same priority, thread scheduler follows round robin fashion.

Thread scheduler is only responsible for deciding which thread should be executed. Thread scheduler uses two mechanisms for scheduling the threads: Preemptive and Time Slicing.

Java thread scheduler also works for :

1. It selects the priority of the thread.
2. It determines the waiting time for a thread
3. It checks the Nature of thread

77. List of Object class methods.

1. **clone():** this method is used to create a clone of the object
2. **equals():** this method is used to check whether objects are equal.
3. **hashCode():** this method returns the hash value of the object.
4. **toString():** this method returns the String representation of the given object.
5. **wait():** causes the current method to wait until another thread calls notify() or notifyAll() method.

6. **notify():** wakes up the single thread waiting for this object's monitor.
7. **notifyAll():** wakes up all the threads, waiting for this object's monitor.
8. **finalize():** this method is invoked by garbage collector just before the object is garbage collected.

78. What is daemon thread?

The daemon threads are the low priority threads that provide the background support and services to the user threads.

Daemon thread gets automatically terminated by the JVM if all other user threads are ended/died.

There are 2 methods for daemon thread available in the Thread class:

1. **public void setDaemon(boolean status):** It used to mark the thread daemon thread or a user thread.
2. **public boolean isDaemon():** It checks the thread is daemon or not.

79. What is Volatile keyword in Java?

Volatile keyword is used in multithreaded to achieve the thread safety, as a change in one volatile variable is visible to all other threads so one variable can be used by one thread at a time. The volatile keyword can be used either with primitive type or objects.

When to Use volatile Keyword?

The use of the volatile keyword is important in scenarios where multiple threads are accessing shared variables. Without volatile, there can be issues with memory visibility and reordering.

By declaring a variable as volatile, we ensure that updates to that variable are immediately visible to other threads. It prevents the processor and compiler from reordering instructions involving the volatile variable, maintaining the intended program order.

80. What is Serialization?

- Serialization means convert object into byte stream.
- Transient variable avoids to serialize
- Serialization is used in this case which translates Java object's state to byte-stream to send it over the network or save it in file.

- Serialization in java can be implemented using java.io.Serializable interface
- Serializable itself Marker Interface there is no methods

NOTE If we don't want to serialize some fields during serialization we declare those variables as transient.

During deserialization transient variables are initialized with default values for primitives and null for object references

The main uses of serialization are :

1. **Persistence:** We can write data to a file or database and can be used later by deserializing it.
2. **Communication :** To pass an object over network by making remote procedure call.
3. **Copying :** We can create duplicates of original object by using byte array.
4. **To distribute objects across different JVMs.**

81. What is the transient keyword?

Once declared any variable with transient then it will avoid serialization.

During serialization, we do not want the confidential data to be saved on the hard disk. In such cases, the Transient keyword in Java is used along with the variable to prevent the serialization of that particular variable.

This informs the JVM that these transient marked variables need to be excluded while serializing the object.

82. What are the different ways to create object in Java?

There are 5 ways to create objects in Java:

1. Using new keyword
2. Using clone() method
3. Using Deserialization
4. Using newInstance() method of Class
5. Using newInstance() method of Constructor class

Using the new keyword in java is the most basic way to create an object. This is the most common way to create an object in java.

83. What are Built-In Java Annotations?

1. **@Override:** @Override annotation assures that the subclass method is overriding the parent class method.
2. **@SuppressWarnings:** @SuppressWarnings annotation: is used to suppress warnings issued by the compiler.
3. **@Deprecated:** @Deprecated annotation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

84. Differentiate NoClassDefFoundError& ClassNotFoundException.

ClassNotFoundException is an exception that occurs when you try to load a class at run time using Class.forName() or loadClass() methods and mentioned classes are not found in the classpath.

NoClassDefFoundError is an error that occurs when a particular class is present at compile time, but was missing at run time.

ClassNotFoundException	NoClassDefFoundError
1. It is an exception. It is type of java.lang.Exception.	1. It is an error. It is type of java.lang.Error.
2. It occurs when an application tries to load a class at run time which is not updated in the classpath.	2. It occurs when java runtime system doesn't find a class definition, which is present at compile time, but missing at run time.
3. It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemClass().	3. It is thrown by the Java Runtime System.
4. It occurs when classpath is not updated with required JAR files.	4. It occurs when required class definition is missing at runtime.

85. List out JDK 1.8 Features.

It is a bigger release of Java and was released by Oracle on 18 March 2014. Java provided support for functional programming, new Java 8 APIs, new Java 8 streaming API, functional interfaces, default methods, date-time API changes, etc.

1. **Lambada Expression**: A function that can be shared or referred to as an object. Symbol: `->`

- It enables Functional programming.
- Less Coding. It help us to reduce line of code.

2. **Functional Interface**: It have Single Abstract Method
   - It enables Functional programming.
   - Less Coding. It help us to reduce line of code.
3. **Static Method in Interface**: It allows implement body with in interface.
4. **Default Method in Interface:** It allows implement with in interface.
5. **Date Time API:** there are 2 classes Local Date & LocalDateTime.

   Both import from java.time package and time format is yyyy-MM-dd
6. **For Each:** It help us to iterate records
7. **Stream API:** It is used with collection framework to sort, filter, collect data, etc
8. **Collectors:** It help us to count the records
9. **String Joiner:** It will help us to add delimiter with in String
10. **Parallel Sort:** It help us to sort integer type of data
11. **Optional Class:** It help us to avoid null pointer exception
12. **Method Reference:** Uses function as a parameter to invoke a method.
    Symbol- : :
    - It enables Functional programming
    - Less Coding. It help us to reduce line of code

86. List out JDK 11 Features.

Java 11 is the second LTS(Long Term Support) release after Java 8. Since Java 11, Oracle JDK would no longer be free for commercial use. You can use it in developing stages but to use it commercially, you need to buy a license.

1. **Running Java File with single command**

   From Java 11 you don't need to compile the java source file with javac tool first. You can directly run the file with java command and it implicitly compiles. This feature comes under JEP 330.
2. **New utility methods in String class**
   - isBlank() - This instance method returns a boolean value. Empty Strings and Strings with only white spaces are treated as blank.
   - lines() This method returns a stream of strings, which is a collection of all

substrings split by lines

- strip(), stripLeading(), stripTrailing() strip() - Removes the white space from both, beginning and the end of string.

3. Local-Variable Syntax for Lambda Parameters

- JEP 323, Local-Variable Syntax for Lambda Parameters is the only language
- feature release in Java 11. In Java 10, Local Variable Type Inference was introduced
- .Thus we could infer the type of the variable from the RHS - var list = new ArrayList<String>(); JEP 323 allows var to be used to declare the formal parameters of an implicitly typed lambda expression.
- We can now define :
  (var s1, var s2) -> s1 + s2

4. Nested Based Access Control

Previous Java versions allowed access of private members to nested classes(nestmates), but we cannot use them with the Reflection API. Java 11 no longer uses bridge methods and provides the getNestHost(), getNestMembers(), and isNestmatOf() methods for the Reflection API.

5. JEP 321: HTTP Client

Java 11 standardizes the Http CLient API. The new API supports both HTTP/1.1 and HTTP/2. It is designed to improve the overall performance of sending requests by a client and receiving responses from the server. It also natively supports WebSockets.

6. Reading/Writing Strings to and from the Files

Java 11 makes it a lot easier to read and write strings. The readString() and writeString() static methods are added to the Files class for this purpose.

7. JEP 328: Flight Recorder

Flight Recorder which earlier used to be a commercial add-on in Oracle JDK is now open-sourced since Oracle JDK is itself not free anymore. JFR is a profiling tool used to gather diagnostics and profiling data from a running Java application. Its performance overhead is negligible and that's usually below 1%. Hence it can be used in production applications.

87. What is Lambda Expression?

Lambda Expression is an anonymous function which accepts a set of input parameters and returns results. Lambda Expression is a block of code without any name, with or without parameters and with or without results. This block of code is

executed on demand.
Lambda Expression Symbol: ->

Lambda Expression use:

1. It enables functional programming
2. Less Coding, it will help us to reduce line of code
3. Lambda expressions can be passed as a parameter to another method.
4. Lambda expressions can be standalone without belonging to any class.

Example:

```
interface Department {
  void get();
}

public class LambdaExpressionEx {
  public static void main(String[] args) {
    Department department = () ->System.out.println("FULL STACK
      JAVA DEVELOPER, PUNE");
      department.get();
  }

}
```

Output:

```
FULL STACK JAVA DEVELOPER, PUNE
```

88. What is Stream API? What are Intermediate & Terminate Operations?

- The Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.
- We can use Stream to sort, filter, collect, print, map, reduce and convert from one data structure to another, etc.
- Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.
- Stream is functional in nature. Operations performed on a stream do 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.

There are 2 types of Operations in Streams:

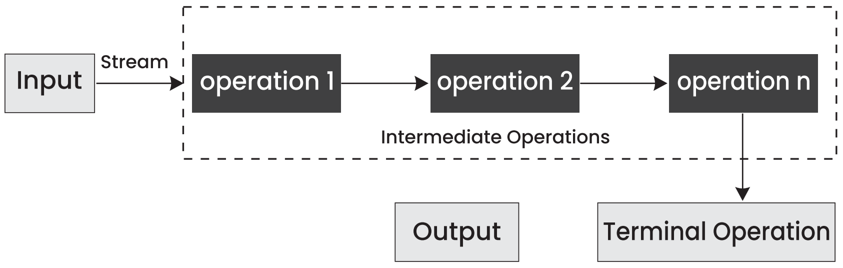
1. Intermediate Operations- Intermediate Operations are the types of operations in which multiple methods are chained in a row.

Operations: map(), filter(), sorted(), distinct(), limit(), skip().

2. Terminate Operations- Terminal Operations are the type of Operations that return the result. These Operations are not processed further just return a final result value.

Operations: forEach(), collect(), reduce(), min(), max(), count(), toArray(), allMatch(), findFirst(), anyMatch(), findAny().

Intermediate Operations	Terminal Operations
1. They return stream.	1. They return non-stream values.
2.They can be chained together to form a pipeline of operations.	2. They can't be chained together.
3. Pipeline of operations may contain any number of intermediate operations.	3. Pipeline of operations can have maximum one terminal operation, that too at the end.
4. Intermediate operations are lazily loaded.	4. Terminal operations are eagerly loaded.
5. They don't produce end result.	5. They produce end result.
5.Examples : filter(), map(), distinct(), sorted(), limit(), skip()	5. Examples : forEach(), toArray(), reduce(), collect(), min(), max(), count(), anyMatch(), allMatch(), noneMatch(), findFirst(), findAny()



89. What are Functional interfaces?

Functional Interfaces are an interface with only one abstract method. Due to which it is also known as the Single Abstract Method (SAM) interface. It is known as a functional interface because it wraps a function as an interface or in other words a function is represented by a single abstract method of the interface.

Functional interfaces can have any number of default, static, and overridden methods. For declaring Functional Interfaces @FunctionalInterface annotation is optional to use. If this annotation is used for interfaces with more than one abstract method, it will generate a compiler error.

Functional interfaces from previous Java versions(JDK 7)

Runnable, Collable, ActionListener, Comparator, and Comparable.

Some of the functional interfaces in the standard library(JDK 8):

There are a lot of functional interfaces in the java.util.function package, the more common ones include but not limited to:

1. **Function** : it takes one argument and returns a result
2. **Consumer** : it takes one argument and returns no result (represents a side effect)
3. **Supplier** : it takes not argument and returns a result
4. **Predicate** : it takes one argument and returns a boolean
5. **BiFunction** : it takes two arguments and returns a result
6. **BinaryOperator** : it is similar to a BiFunction, taking two arguments and returning a result. The two arguments and the result are all of the same types
7. **UnaryOperator** : it is similar to a Function, taking a single argument and returning a result of the same type

Rules to define a Functional Interface:

1. Define an interface with one and only one abstract method.
2. We cannot define more than one abstract method.
3. Use @FunctionalInterface annotation in the interface definition.
4. We can define any number of other methods like Default methods, Static methods.
5. If we override java.lang.Object class's method as an abstract method, which does not count as an abstract method.

90. What is an Optional class?

Optional class comes in java 8, It help us to avoid NullPointerException

Optional is a container type which may or may not contain value i.e. zero(null) or one(not-null) value. It is part of java.util package. There are pre-defined methods like isPresent(), which returns true if the value is present or else false.

Before Java 8, developers depended on if-else blocks to act on variables that might hold a null reference. Often, developers forget the check resulting in the so-called NullPointerException and the application crashes unwantedly during runtime.

Example:

```
package com.csi.core;

import java.util.Optional;


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

    String name = null;

    Optional<String> compnayName = Optional.ofNullable(name);
if (compnayName.isPresent()) {
      System.out.println(name.length());
    } else {
      System.out.println("Null");
    }
  }
}
```

Output:

```
Null
```

91. Differentiate map() and flatMap().

map()	flatMap()
1. It processes stream of values.	1. It processes stream of stream of values.
2. One-to-one mapping occurs.	2. One-to-many mapping occurs.
3. It only performs the mapping.	3. It performs mapping as well as flattening.
4. map() is used only for transformation.	4. flatMap() is used both for transformation and mapping.
5. The function passed to map() operation returns a single value for a single input.	5. The function you pass to flatmap() operation returns an arbitrary number of values (zero or more) as output for a single value.

Example:

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

class Customer {
  private int custId;
  private String custName;
  private List<Long> custContactNumber;

  public Customer(int custId, String custName, List<Long>
  custContactNumber) {
    super();
    this.custId = custId;
    this.custName = custName;
    this.custContactNumber = custContactNumber;
  }

  public int getCustId() {
    return custId;
  }

  public void setCustId(int custId) {
    this.custId = custId;
  }

  public String getCustName() {
    return custName;
  }

  public void setCustName(String custName) {
    this.custName = custName;
  }

  public List<Long> getCustContactNumber() {
    return custContactNumber;
  }

  public void setCustContactNumber(List<Long> custContactNumber) {
    this.custContactNumber = custContactNumber;
  }

  @Override
  public String toString() {
    return "Customer [custId=" + custId + ", custName=" + custName
+ ", custContactNumber=" + custContactNumber + "]";
```

```
 }

}

public class MapAndFlatMapEx {

  public static void main(String[] args) {

    List<Customer> customerList = Stream
      .of(new Customer(121, "SWARA", Arrays.asList(7887575991L,
9623639693L)),
        new Customer(122, "SWARAJ", Arrays.asList(9356943970L,
9767186443L)))
      .collect(Collectors.toList());

    List<List<Long>> contactNumbers =
customerList.stream().map(customer ->
customer.getCustContactNumber())
      .collect(Collectors.toList());

    System.out.println("\n map: " + contactNumbers);
List<Long> contactNumber = customerList.stream().flatMap(customer -
> customer.getCustContactNumber().stream())
      .collect(Collectors.toList());
    System.out.println("\n flatmap: " + contactNumber);
  }

}
```

Output:

```
map: [[7887575991, 9623639693], [9356943970, 9767186443]]

flatmap: [7887575991, 9623639693, 9356943970, 9767186443]
```

92. Filter employees having salary equals or more than 50000.00 using stream API

```
import java.util.stream.Stream;

class Employee {
private int empId;
private String empName;
private int empAge;
private double empSalary;
```

```
public Employee(int empId, String empName, int empAge, double
empSalary) {
  super();
  this.empId = empId;
  this.empName = empName;
  this.empAge = empAge;
  this.empSalary = empSalary;
 }

 public int getEmpId() {
  return empId;
 }

 public void setEmpId(int empId) {
  this.empId = empId;
 }

 public String getEmpName() {
  return empName;
 }

 public void setEmpName(String empName) {
  this.empName = empName;
 }

 public int getEmpAge() {
  return empAge;
 }

 public void setEmpAge(int empAge) {
  this.empAge = empAge;
 }

 public double getEmpSalary() {
  return empSalary;
 }

 public void setEmpSalary(double empSalary) {
  this.empSalary = empSalary;
 }

 @Override
 public String toString() {
  return "Employee [empId=" + empId + ", empName=" + empName + ",
 empAge=" + empAge + ", empSalary=" + empSalary
    + "]";
 }
```

```
public class HRMApplication {

 public static void main(String[] args) {

  Stream.of(new Employee(121, "SWARA", 22, 96000.00), new
  Employee(122, "RAM", 27, 76000.00),
    new Employee(123, "LAKSHMI", 18, 50000.00), new Employee(127,
   "APARNA", 25, 45000.22),
    new Employee(129, "VENKAT", 29, 56000.00)).filter(emp ->
    emp.getEmpSalary() >= 50000.00)
    .forEach(System.out::println);
  }
}

Or

employeeList.stream().filter(emp -> emp.getEmpSalary() >=
50000.00).forEach(System.out::println);
```

93. Filter employees having age equals or more than 30 using stream API.

```
employeeList.stream().filter(emp -> emp.getEmpAge() >=
30).forEach(System.out::println);
```

94. Sort Employee List by Name using Stream API.

```
employeeList.stream().sorted(Comparator.comparing(Employee::
getEmpName)).forEach(System.out::println);
```

95. Sort Employee List by Age using Stream API.

```
employeeList.stream().sorted(Comparator.comparingInt(Employee::
getEmpAge)).forEach(System.out::println);
```

96. Sort Employee List by Salary using Stream API.

```
employees.stream().sorted(Comparator.comparingDouble(
Employee::getEmpSalary)).forEach(System.out::println);
```

97. Fetch Second Largest Salary Employee Datafrom Employee List.

```
Employee employee =
employeeList.stream().sorted(Comparator.comparingDouble(Employee::
getEmpSalary).reversed())
.collect(Collectors.toList()).get(1);

System.out.println(employee);
```

98. Fetch Second Largest Number from Integer List using Stream API.

```
List<Integer> integerList = Arrays.asList(5, 3, 9, 8, 4, 7);
int secondLargestNumber =
integerList.stream().sorted(Comparator.comparingInt(Integer::
intValue).reversed())
.collect(Collectors.toList()).get(1);

System.out.println(secondLargestNumber);
```

99. Summation of Integer List using Stream API.

```
List<Integer> integerList = Arrays.asList(5, 3, 9, 8, 4, 7);
int sum =
integerList.stream().collect(Collectors.summingInt(Integer::int
Value));

System.out.println(sum);
```

100. Duplicate number and its count using Stream API.

```
List<Integer> integerList = Arrays.asList(5, 3, 7, 9, 8, 4, 7);

Map<Integer, Long> counterMap = integerList.stream()
.collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));

counterMap.forEach((k, v) -> {
  if (v > 1) {
    System.out.println(k + ": " + v);
  }
});
```

101. Duplicate character and its count using Stream API.

```
String name = "Swara";

Map<Character, Long> counterMap =
name.toLowerCase().chars().mapToObj(c -> (char) c)
.collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));

counterMap.forEach((k, v) -> {
  if (v > 1) {
    System.out.println(k + ": " + v);
  }
});
```

102. Duplicate word from String and its count.

```
String name = "Java Python Java React";
Map<String, Long> counterMap = Arrays.asList(name.
  toLowerCase().split(" ")).stream()
.collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));

counterMap.forEach((k, v) -> {
  if (v > 1) {
    System.out.println(k + ": " + v);
  }
});
```

103. Duplicate word and its count using Stream API.

```
List<String> list = Arrays.asList("Java", "Python", "Java", "React");
Map<String, Long> counterMap = list.stream()
.collect(Collectors.groupingBy(String::toLowerCase,
Collectors.counting()));

counterMap.forEach((k, v) -> {
  if (v > 1) {
    System.out.println(k + ": " + v);
  }
});
```

104. Count words from String.

```
public static void main(String args[]) {
  String name = "Full Stack Java Developer Pune";

  long count = Arrays.asList(name.split(" ")).stream().collect
(Collectors.counting());

  System.out.println(count);
}
```

105. Distinct word using Stream API.

```
List<String> list = Arrays.asList("Java", "Python", "Java", "React");
list.stream().distinct().forEach(System.out::println);
```

106. Distinct characters using JDK 8.

```
String name = "Swara";
Stream.of(name).map(w ->
w.split("")).flatMap(Arrays::stream).distinct().forEach(System.out:
:print);
```

107. Write a java program to find repeated character with number of occurrences without Stream API.

```
public static void main(String[] args) {

  String name = "Swara";
  char str[] = name.toCharArray();
  Arrays.sort(str);

  for (int i = 0; i < str.length; i++) {
    int count = 1;
    char temp = str[i];
    for (int j = i + 1; j < str.length; j++) {
      if (str[i] == str[j]) {
        count++;
        i++;
      }
    }
    if (count > 1) {
      System.out.println(temp + ": " + count);
    }
  }
```

108. Write a Java Code for HashMap sort by Key.

```
Map<Integer, String> sortedData =
 hashMap.entrySet().stream().sorted(Map.Entry.comparingByKey())
.collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,
(v1, v2) -> v2, LinkedHashMap::new));

sortedData.forEach((k, v) -> {
  System.out.println(k + ": " + v);
});
```

109. Write a Java Code for HashMap sort by Value.

```
Map<Integer, String> sortedData =
 hashMap.entrySet().stream().sorted(Map.Entry.comparingByValue())
.collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,
(v1, v2) -> v2, LinkedHashMap::new));

sortedData.forEach((k, v) -> {
  System.out.println(k + ": " + v);
});
```

110. HashMap program sort by Employee Age. Key as empId, and Value as Employee object- empId, empName, empAge, empSalary.

```
List<Employee> employeeList = Stream.of(new Employee(121, "SWARA",
22, 96000.00),
  new Employee(122, "RAM", 27, 76000.00), new Employee(123,
"LAKSHMI", 18, 50000.00),
  new Employee(127, "APARNA", 25, 45000.22), new Employee(129,
"VENKAT", 29, 56000.00))
.collect(Collectors.toList());

Map<Integer, Employee> employeeMap = new HashMap<Integer,
Employee>();

for (Employee employee : employeeList) {
  employeeMap.put(employee.getEmpId(), employee);
}

employeeMap.entrySet().stream().map(emp ->
emp.getValue()).sorted(Comparator.comparingInt(Employee::getEmpAge)
)
.forEach(System.out::println);
```

111. Write a program for reverse String.

```
public static void main(String[] args) {

  String name = "Swara";

  for (int i = name.length() - 1; i >= 0; i--) {
    System.out.print(name.charAt(i));
  }
}
```

112. Write a program for palindrome string.

```
public static void main(String[] args) {

  String orgString = "madam";

  String revString = "";
  for (int i = orgString.length() - 1; i >= 0; i--) {
    revString = revString + orgString.charAt(i);
  }

  if (orgString.equals(revString))
    System.out.println("Yes, Palindrom");
  else
    System.out.println("No, Not Palindrom");
}
```

113. First Non-repeated character in String.

```
public static void main(String[] args) {

  String string = "aaccbdee";

  for (char i : string.toCharArray()) {
    if (string.indexOf(i) == string.lastIndexOf(i)) {
      System.out.println(i);
      break;
    }
  }

}
```

114. Employee Department wise salary calculation.

```
public static void main(String[] args) {

  List<Employee> empList = Stream.of(new Employee(121, "Swara",
"IT", 96000),
    new Employee(122, "Lakshmi", "HR", 45000), new Employee(129,
"Ram", "IT", 50000))
    .collect(Collectors.toList());
  System.out.println(empList.stream().collect(
    Collectors.groupingBy(Employee::getDeptName,
Collectors.summingDouble(Employee::getEmpSalary))));
}
```

115. Swap number without 3rd variable.

```
public static void main(String[] args) {
  int num1 = 10, num2 = 20;

  num1 = num1 + num2;
  num2 = num1 - num2;
  num1 = num1 - num2;

  System.out.println("\n After Swap: \n num1: " + num1 + "\n num2:
" + num2);
}
```

116. Swap String without 3rd variable.

```
public static void main(String[] args) {

  String s1 = "FullStack";
  String s2 = "Java";
s1 = s1 + s2;
  s2 = s1.substring(0, s1.length() - s2.length());
  s1 = s1.substring(s2.length());

  System.out.println("\n s1: " + s1 + "\n s2: " + s2);
}
```

117. Factorial Number

```
public static void main(String[] args) {
  int i, fact = 1, n = 5;

  for (i = 1; i <= n; I++) {
    fact = fact * I;
  }
  System.out.println("Factorial Result: " + fact);
}
```

118. Fibonacci Series

```
public static void main(String[] args) {
  int t1 = 0, t2 = 1;

  for (int i = 1; i <= 10; ++I) {
    System.out.print(t1 + " ");
    int sum = t1 + t2;
    t1 = t2;
    t2 = sum;
  }
}
```

119. Prime Number using stream API

```
public static void main(String[] args) {
  int number = 12;

  System.out.println(IntStream.rangeClosed(2, number /
2).noneMatch(i -> number % i == 0));
}
```

120. Find even number form Integer List.

```
public static void main(String[] args) {
  List<Integer> list = Arrays.asList(22, 12, 32, 61, 45, 65, 85);

  list.stream().filter(i -> i % 2==0).forEach(System.out::println);
}
```

121. Program for Convert List into Map.

```
public static void main(String[] args) {
 List<Employee> empList = new ArrayList<Employee>();
empList.add(new Employee(121, "SWARA", 65000));
  empList.add(new Employee(122, "LAKSHMI", 95000));

  empList.stream().collect(Collectors.toMap(Employee::getEmpId,
Employee::getEmpName)).forEach((k, v) -> {
    System.out.println(k + ": " + v);
  });
}
```

122. What is SDLC and List out its phases?

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

Software Development Life Cycle consists of the following phases :

1. **Requirements gathering and analysis:** This phase involves gathering information about the software requirements from stakeholders, such as customers, end-users, and business analysts.

   It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry.

2. **Design:** In this phase, the software design is created, which includes the overall architecture of the software, data structures, and interfaces.

   It has two steps:

   High-level design (HLD): It gives the architecture of software products.

   Low-level design (LLD): It describes how each and every feature in the product should work and every component.

3. **Implementation or coding:** In this phase of SDLC the actual development starts and the product is built.

   The code reviews are done to ensure software follows code standards and security controls are implemented. Security vulnerability tests like penetration testing are also done to identify potential issues.

   In Implementation phase always we need to take care of requirements coming from BRD(Business Requirements Document).

4. **Testing:** The software is thoroughly tested to ensure that it meets the requirements and works correctly.

However, this phase refers to the testing of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS(Software Requirements Specification).

5. **Deployment:** Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization.

   The product may first be released in a limited segment and tested in the real business environment (**UAT**- User acceptance testing). After successful testing, the software is deployed to a production environment and made available to end-users.

6. **Maintenance:** This phase includes ongoing support, bug fixes, and updates to the software.

Following are the most important and popular **SDLC models** followed in the industry :

1. Waterfall Model
2. Iterative Model
3. Spiral Model
4. V-Model
5. Big Bang Model

123. What are Design patterns in Java.

Design Patterns are well documented and understood by software architects, designers and developers for their application within a specific solution.

Design Pattern Advantages:

1. Design Patterns provide the solutions that help to define the system architecture.
2. Design Patterns capture the software engineering experiences.
3. Design Patterns provide transparency to the design of an application.
4. Design Patterns are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
5. Design patterns don't guarantee an absolute solution to a problem, they provide clarity to the system architecture and the possibility of building a better system.

Java design patterns are divided into three categories: creational, structural, and behavioral design patterns.

Creational Design Pattern	Structural Design Pattern	Behavioral Design Pattern
1. Singleton Pattern	1. Adapter Pattern	1.Chain Of Responsi_ bility Pattern
2. Factory Pattern	2. Bridge Pattern	2. Command Pattern
3. Abstract Factory Pattern	3. Composite Pattern	3. Interpreter Pattern
4. Prototype Pattern	4. Decorator Pattern	4. Iterator Pattern
5. Builder Pattern.	5. Facade Pattern	5. Mediator Pattern
6.Object Pool Pattern	6. Flyweight Pattern	6. Memento Pattern
	7. Proxy Pattern	7. Observer Pattern
		8. State Pattern
		9. Strategy Pattern
		10. Template Pattern
		11. Visitor Pattern

Here, Added **3 Creational Design Pattern** & **Microservices SAGA Design Pattern**

1. What is Singleton Design Pattern?

Its creational design pattern.

1. Singleton pattern involves a single class which is responsible to create an object while making sure that only single object gets created.
2. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

When Will We Need Singleton Pattern?

Consider we want to create a LoggerService class which we can use across our project to log. Almost all the classes will use the LoggerService to log info, warn, error, etc. If every class creates a new LoggerService instance, it will occupy more memory, leading to an out-of-memory error. Singleton pattern, in this case, ensures that only one instance of this class is created and accessible globally throughout the project context.

Steps:

1. Create private Default Constructor
2. private static final Singleton singletonInstance = new Singleton();
3. Create private constructor
4. Create getter | no setter
5. Only one instance of the class
6. Real time usage- Used in Logging, Cache, Session, Driver, etc

Example:

```
class Singleton {
  private Singleton() {

  }

private static final Singleton singletonInstance = new Singleton();

  public static Singleton getInstance() {
    return singletonInstance;
  }
}

public class SingletonDesignPatternEx {

  public static void main(String[] args) {
    Singleton singleton1 = Singleton.getInstance();
    Singleton singleton2 = Singleton.getInstance();

    System.out.println(singleton1.hashCode());
    System.out.println(singleton2.hashCode());

  }
}
```

Output: It will print same hashcode

```
468121027
468121027
```

2. What is Factory Pattern?

Its creational design pattern.

1. In the Factory pattern, we don't expose the creation logic to the client and refer the created object using a standard interface. The Factory Pattern is also known as Virtual Constructor.
2. The factory design pattern is used when we have a superclass with multiple sub-classes and based on input, we need to return one of the sub-class.
3. This pattern takes out the responsibility of the instantiation of a class from the client program to the factory class.
4. In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

NOTE

- ☑ Factory pattern used when we have multiple sub-classes of a Super class and based on input we want to return one class instance.
- ☑ Factory pattern provides abstraction between implementation & client classes.
- ☑ Factory pattern used for remove the instantiation of client classes from client code.

Steps:

1. Define a factory method inside an interface.
2. Let the subclass implements the above factory method and decides which object to create.
3. Factory class returns required class instance

Example-

```
interface Category {
  void print();
}

class Sports implements Category {

  @Override
  public void print() {
    // TODO Auto-generated method stub
    System.out.println("Sports Items");
  }

}

class Electronics implements Category {

  @Override
  public void print() {
    // TODO Auto-generated method stub
    System.out.println("Electronics Items");
  }

}

class Healthcare implements Category {

  @Override
  public void print() {
    // TODO Auto-generated method stub
System.out.println("Healthcare Items");
   }

 }

 class CategoryFactory {

   public Category getCategory(String category) {

     if (category == null) {
       return null;
     }
     if (category.equals("Sports")) {
       return new Sports();
     }
     if (category.equals("Electronics")) {
       return new Electronics();
     }
     if (category.equals("Healthcare")) {
       return new Healthcare();
     }
     return null;

   }
 }

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

     CategoryFactory factory = new CategoryFactory();

     Category category = factory.getCategory("Electronics");

     category.print();

   }
 }
```

Output:

```
Electronics Items
```

3. What s Builder Design Pattern?

Builder pattern was introduced to solve some of the problems with Factory and Abstract Factory design patterns when the Object contains a lot of attributes.

1. Java Builder class should have a public constructor with all the required attributes as parameters.
2. Java Builder class should have methods to set the optional parameters and it should return the same Builder object after setting the optional attribute.
3. The final step is to provide a build() method in the builder class that will return the Object needed by client program. For this we need to have a private constructor in the Class with Builder class as argument.

4. What is SAGA Design Pattern :

- SAGA is a Microservices Design Pattern
- SAGA is a Database per Service pattern

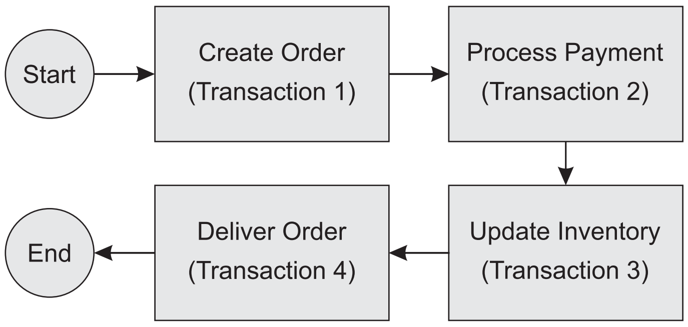
Benefits:

1. One of the benefits of microservice architecture is that we can choose the technology stack per service. For instance, we can decide to use a relational database for service A and a NoSQL database for service B.
2. This model lets the service manage domain data independently on a data store that best suites its data types and schema. Further, it also lets the service scale its data stores on demand and insulates it from the failures of other services.

To demonstrate the use of distributed transactions, we'll take an example of an e-commerce application that processes online orders and is implemented with microservice architecture.

There is a microservice to create the orders, one that processes the payment, another that updates the inventory and the last one that delivers the order.

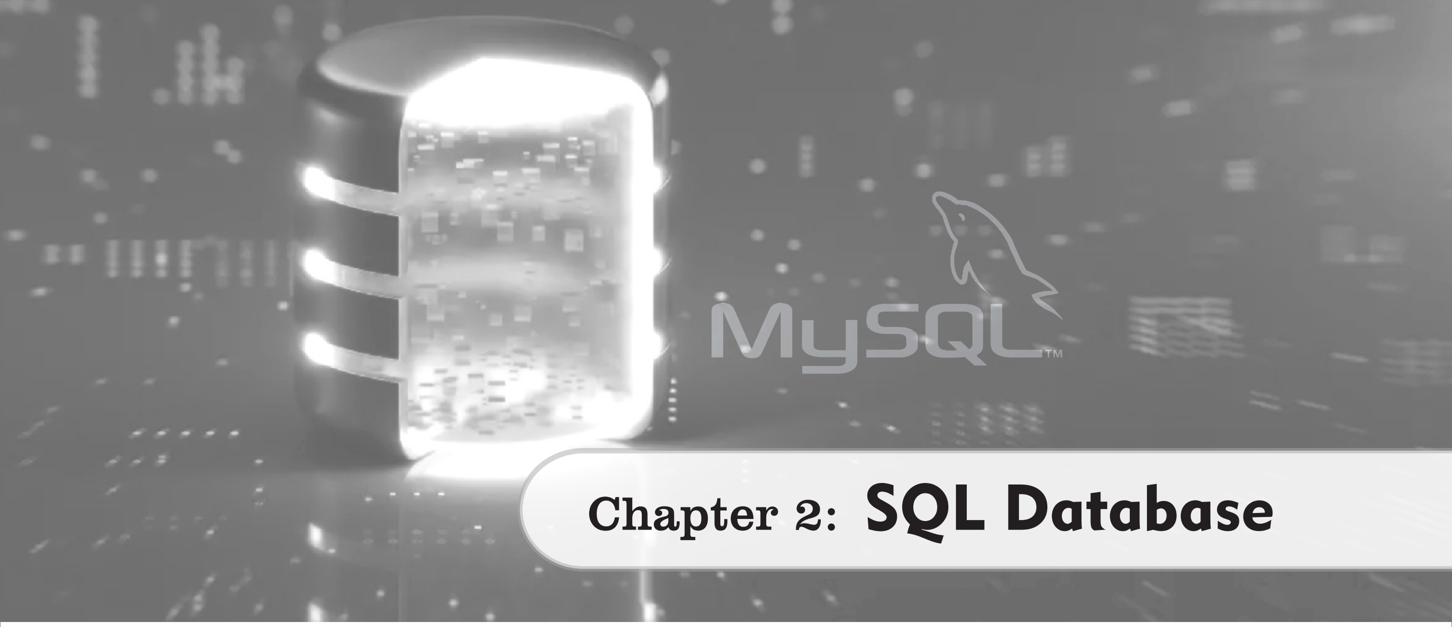
Each of these microservices performs a local transaction to implement the individual functionalities:



This is an example of a distributed transaction as the transaction boundary crosses multiple services and databases.

To ensure a successful order processing service, all four microservices must complete the individual local transactions. If any of the microservices fail to complete its local transaction, all the completed preceding transactions should roll back to ensure data integrity.

NOTES



Chapter 2: SQL Database

1. What are the steps to connect with JDBC?

1. Load Driver class:
   Class.forName("com.mysql.jdbc.Driver");
2. Establish Connection
   Connection
   connection=DriverManager.getConnection("jdbc:mysql://localhost:3306/

   db_name", "user_name", "password");
3. Create Statement
4. Execute Query
5. Close Database connection (Optional)

2. Differentiate Primary Key, Unique Key .

PRI: For PRIMARY:	UNI: For UNIQUE:
1. Primary Key Uniquely identifies each record in a table. Primary key are generally used as Foreign keys in the other table.	1. Unique Key enforces the uniqueness of values in a column or set of columns, but does not necessarily identify each record.
2. Primary key cannot accept NULL values.	2. Unique key can accept NULL values.
3. Only one Primary Key per table.	3. Multiple Unique Keys can exist in a table.
4. Primary key supports auto-increment value.	4. Unique key does not support auto-increment value.
5. We cannot change or delete values stored in primary key.	5. We can change unique key values.

3. Explain Joins in SQL.

The SQL Join clause is used to combine records (rows) from two or more tables

In SQL there are 4 important Joins:

1. Inner Join: Common data from both table
2. Left Outer Join: All Data from Left and common data from Right table
3. Right Outer Join: All data from Right table and common data from Left table
4. Full Outer Join: All Data from both tables, But MySQL does not support to Full Outer Join. In MySQL we are using Left Outer & Right Outer Union to achieve Full Outer Join.

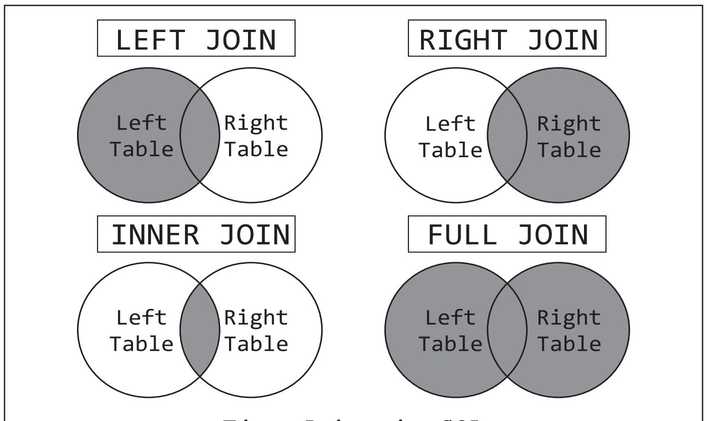


Fig. Joins in SQL

4. How many Aggregate function in SQL?

In SQL 5 aggregate functions

MIN, MAX, AVG, SUM & COUNT

5. Write a SQL query to find second largest salary?

Second Largest Salary:

```
SELECT MAX(empsalary) FROM employee WHERE
empsalary NOT IN (SELECT MAX(empsalary) FROM employee)
```

Second Largest Salary with Employee Name:

```
SELECT empname,empsalary FROM employee
WHERE empsalary = (SELECT MAX(empsalary) FROM employee
WHERE empsalary < (SELECT MAX(empsalary) FROM employee));
```

Third Largest Salary using LIMIT:

```
SELECT DISTINCT(empsalary) FROM employee ORDER BY empsalary DESC
LIMIT 2, 1;
```

Nth Largest Salary using LIMIT:

```
SELECT DISTINCT(empsalary) FROM employee ORDER BY empsalary DESC
LIMIT n-1, 1;
```

For Example: for 5th largest salary : LIMIT 4, 1;

6. Differentiate Delete and Truncate.

Delete	Truncate
1.The DELETE command deletes one or more existing records from the table.	1.The TRUNCATE Command deletes all the rows from the existing table, leaving the row with the column names.
2.We can restore any deleted row or multiple rows from the database using the ROLLBACK command.	2.We cannot restore all the deleted rows from the database using the ROLLBACK command.
3.The DELETE command performs slower	3.The TRUNCATE command works faster
4.The DELETE command is Data Manipulation Language Command.	4.The TRUNCATE command is a Data Definition Language Command.

7. What is Stored Procedure?

We are giving input parameter and Stored Procedure it will give us output parameter. **Example-** For age calculation we are giving input parameter as DOB and Stored Procedure gives us output parameter as age.

A stored procedure is a subroutine available to applications that access a relational database management system (RDBMS). Such procedures are stored in the database data dictionary. The sole disadvantage of stored procedure is that it can be executed nowhere except in the database and occupies more memory in the database server. It also provides a sense of security and functionality as users who can't access the data directly can be granted access via stored procedures.

Syntax:
Creating a Procedure :

```
CREATE PROCEDURE procedure_name
(parameter1 data_type, parameter2 data_type)
AS
BEGIN
– SQL statements to be executed
END
```

To Execute the procedure:

```
EXEC procedure_name parameter1_value, parameter2_value
```

8. What is views?

A view in SQL is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
To create the view, we can select the fields from one or more tables present in the database.

Example:

Creating View from a single table

```
CREATE VIEW employees AS
SELECT empname, empaddress
FROM employee
WHERE empid < 5;
```

9. What is index in SQL?

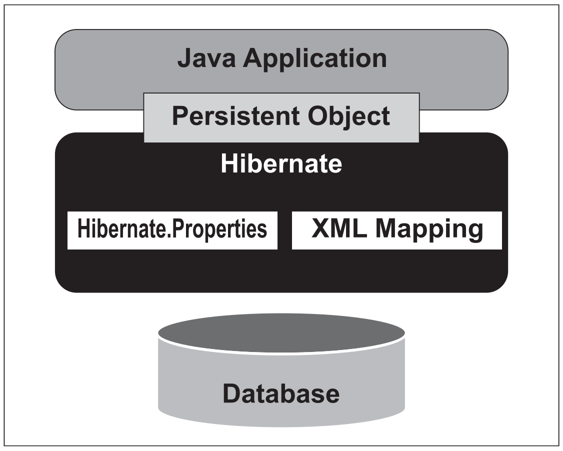
It is used to retrieve data from the database very fast. Indexes are special lookup tables.

An Index is used to speed up select queries and where clauses. But it shows down the data input with insert and update statements. Indexes can be created or dropped without affecting the data.

Chapter 3: FRAMEWORK

1. What is Hibernate and List out its features?

Hibernate is an ORM tool [Object Relational Mapping].
Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight ORM tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.



Why use Hibernate?

Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects.
It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost.

Hibernate Query Language- HQL is very simple, efficient, and very flexible for performing complex operations on relational databases without writing complicated queries. HQL is the object-oriented representation of query language, i.e instead of using table name, we make use of the class name which makes this language independent of any database. This makes use of the Query interface provided by Hibernate. The Query object is obtained by calling the createQuery() method of the hibernate Session interface.

Hibernate Features:

1. Database Migration Easy.
2. Hibernate will take care of creating table.
3. Hibernate will take care of SQL queries.
4. HQL [Hibernate Query Language]- Database Independent.
5. High Performance.
6. Open Source and Lightweight.
7. Faster Development

2. What is a Session in Hibernate?

A session is an Interface, it is a First Level Cache that maintains the connection between Java object application and database. Session also has methods for storing, retrieving, modifying or deleting data from database using methods like save(), persist(), load(), get(), update(), saveOrUpdate(), merge(), delete(), etc. Additionally, It has factory methods to return Query, Criteria, and Transaction objects.

3. What is a SessionFactory?

SessionFactory is an interface, it is a Second Level Cache provides an instance of Session. It is a factory class that gives the Session objects based on the configuration parameters in order to establish the connection to the database.With in application there will be single instance of SessionFactory.

4. Differentiate first level cache and second level cache in Hibernate.

First Level Cache: Session	Second Level Cache: SessionFactory
1. This cache is enabled by default and there is no way to disable it.	1. This is disabled by default, but we can enable it through configuration.
2. For each transaction we creating new Session	2. For whole application we have one SessionFactory
3. The first level cache is available only until the session is open, once the session is closed, the first level cache is destroyed.	3. The second-level cache is available through the application's life cycle, it is only destroyed and recreated when an application is restarted.

5. Differentiate session.get() and session.load() in Hibernate.

session.get()	session.load()
1. get returns null if the object is not found.	1. load throws ObjectNotFound Exception if the object is not found.
2. get hit every time the method is called.	2. load hit only when it is really needed and this is called Lazy Loading which makes the method better.
3. This method gets the data from the database as soon as it is called.	3. load returns a proxy object and loads the data only when it is required.
4. get method should be used if we are unsure about the existence of data in the database	4. load method is to be used when we are sure that the data is present in the database.

6. What is the actual use of @Entity, @Id, @GeneratedValue, @Temporal, @Transient, @Casecade annotation.

1. **@Entity:** It is used on the model classes by using "@Entity" and tells that the classes are entity beans.
   And if we are not using @Table annotation then it will consider POJO class name as table name.

2. **@Table:** It is used on the model classes by using "@Table" and tells that the class maps to the table name in the database.

3. **@Column:** It is used for defining the column name in the database table

4. **@Id:** It is used on the attribute in a class to indicate that attribute is the primary key in the bean entity.

5. **@GeneratedValue:** It is used for defining the strategy used for primary key auto generation/increment. This annotation is used along with javax.persistence.GenerationType enum.

6. **@Temporal:** @Temporal annotation has the single parameter value of type TemporalType.
   It can be either DATE, TIME or TIMESTAMP, depending on which we want to use for the mapping.

7. **@Transient:** It will avoid persist data into database table.

8. **@Casecade:** It will help us to perform same operations with entity relationship mapping table. It is used for defining the cascading action between two bean entities. It is used with org.hibernate.annotations.CascadeType enum to define the type of cascading.

7. What is Spring Framework and Why it so popular?

Spring is a powerful open-source, loosely coupled, lightweight java framework for reducing the complexity of developing enterprise-level applications. This framework is also called the "framework of frameworks" as spring provides support to various other important frameworks like JSF, Hibernate, Structs, EJB, etc.

Features of Spring Framework:

- Lightweight
- Simple
- Loosely Coupled
- Testability
- Development Faster

8. What is IOC?

IOC: Inversion of Control

Entire ApplicationContext (container) beans that called IOC.

General Life Example- Coffee Machine

Spring IOC (Inversion of Control) Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle.

The Spring container uses Dependency Injection (DI) for managing the application components by creating objects, wiring them together along with configuring and managing their overall life cycles. The instructions for the spring container to do the tasks can be provided either by XML configuration, Java annotations, or Java code.

9. What is DI?

DI: Dependency Injection

In ApplicationContext there is available bean id that called DI. With in ApplicationContext there will be unique bean id.

General life example- With in Coffee machine unique Espresso, Cappuccino coffee.

The main idea in Dependency Injection is that you don't have to create your objects but you just have to describe how they should be created.
In Java, the 2 major ways of achieving dependency injection are:

Constructor injection: Here, the IoC container invokes the class constructor with a number of arguments where each argument represents a dependency on the other class.

Setter injection: Here, the spring container calls the setter methods on the beans after invoking a no-argument static factory method or default constructor to instantiate the bean

10. Differentiate constructor and setter injection DI.

Constructor	Setter
1. In constructor *injection*, partial injection is not allowed	1. Partial injection allowed in Setter.
2. The constructor injection doesn't override the setter property.	2. Setter overrides the Constructor DI.
3. Constructor injection creates a new instance if any modification is done.	3. The creation of a new instance is not possible in setter injection.
4. In case the bean has many properties, then constructor injection is preferred	4. If few properties, then setter injection is preferred

11. How many Beans Scope available in Spring Framework?

When you create a bean definition what you are actually creating is a recipe for creating actual instances of the class defined by that bean definition. The idea that a bean definition is a recipe is important, because it means that, just like a class, you can potentially have many object instances created from a single recipe.

You can control not only the various dependencies and configuration values that are to be plugged into an object that is created from a particular bean definition, but also the scope of the objects created from a particular bean definition. This approach is very powerful and gives you the flexibility to choose the scope of the objects you create through configuration instead of having to 'bake in' the scope of an object at the Java class level. Beans can be defined to be deployed in one of a number of scopes:

out of the box, the Spring Framework supports exactly five scopes (of which three are available only if you are using a web-aware ApplicationContext).

The Spring Framework has 5 Beans scope:

1. ***Singleton:*** The scope of bean definition while using this would be a single instance per IOC container. It is available by default.
   By default, singleton beans scope available.
2. ***Prototype:*** Each time it will create new instance
3. ***Request:*** It is a HTTP Request used in Web Application
4. ***Session:*** It is a HTTP Session used in Web Application
5. ***Global:*** session: It is a HTTP Global-session used in Web Application

12. Explain Spring Beans Life Cycle.

Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed. Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom init() method and the destroy() method.

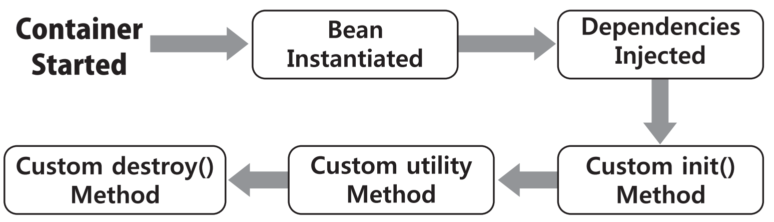


Fig. Spring Beans Life Cycle

In Java Spring there are 3 ways to implement Bean Life Cycle-

1. **By XML:** In this approach, in order to avail custom init() and destroy() methods for a bean we have to register these two methods inside the Spring XML configuration file while defining a bean.
2. **By Programmatic Approach:** To provide the facility to the created bean to invoke custom init() method on the startup of a spring container and to invoke the custom destroy() method on closing the container, we need to implement our bean with two interfaces namely InitializingBean, DisposableBean and will have to override afterPropertiesSet() and destroy() method. afterPropertiesSet() method is invoked as the container starts and the bean is instantiated whereas, the destroy() method is

invoked just after the container is closed.

NOTE To invoke destroy method we have to call a close() method of ConfigurableApplicationContext.

3. **Using Annotation:** To provide the facility to the created bean to invoke custom init() method on the startup of a spring container and to invoke the custom destroy() method on closing the container, we need to annotate init() method by @PostConstruct annotation and destroy() method by @PreDestroy annotation.

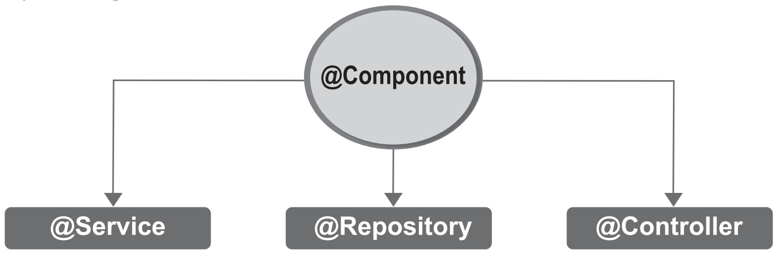
13. How many Autowiring mode available in Spring?

The Spring Framework has 5 Autowiring mode:

1. **no:** This means no autowiring and is the default setting. An explicit bean reference should be used for wiring.
2. **byName:** The bean dependency is injected according to the name of the bean. This matches and wires its properties with the beans defined by the same names as per the configuration.
3. **byType:** This injects the bean dependency based on type.
4. **constructor:** Here, it injects the bean dependency by calling the constructor of the class. It has a large number of parameters.
5. **autodetect:** First the container tries to wire using autowire by the constructor, if it isn't possible then it tries to autowire by byType.

14. What are stereotype annotations in spring?

Spring Framework provides us with some special annotations. These annotations are used to create Spring beans automatically in the application context.
@Component annotation is the main Stereotype Annotation. @Component is a class-level annotation. It is used to denote a class as a Component. We can use @Component across the application to mark the beans as Spring's managed components. So, the stereotype annotations in spring are **@Component, @Service, @Repository**, and **@Controller.**



There are some Stereotype meta-annotations which is derived from @Component those are:

1. **@Service:** We specify a class with @Service to indicate that they're holding the business. One most important thing about the @Service Annotation is it can be applied only to classes. It is used to mark the class as a service provider. So overall @Service annotation is used with classes that provide some business functionalities.

2. **@Repository:** We specify a interface with @Repository to indicate that they're dealing with DAO database tables. @Repository Annotation is a specialization of @Component annotation which is used to indicate that the class provides the mechanism for storage, retrieval, update, delete and search operation on objects.

3. **@Controller-** We specify a class with @Controller to indicate that they're front controllers and responsible to handle user requests and return the appropriate response. The @Controller annotation is typically used in combination with annotated handler methods based on the @RequestMapping annotation. It can be applied to classes only. It's used to mark a class as a web request handler. It's mostly used with Spring MVC applications.

15. What is Spring Boot and List out the Features?

Spring Boot is an open-source, java-based framework that provides support for Rapid Application Development and gives a platform for developing stand-alone and production-ready applications within minutes.

It has extra support of auto-configuration and embedded server like Tomcat, Jetty and Undertow.

Features of Spring Boot

1. Inbuild Tomcat Server available
2. We are able to create Production Ready application with in minute
3. No XML Configuration required
4. DevTool Depedency: Not need to restart application after changes. DevTool help us for Live Reload application.
5. Development Faster
6. Spring Boot also offers pinpointed 'started' POMs to Maven configuration.
7. Spring Initializer : This is a web application that helps a developer in creating an

internal project structure. The developer does not have to manually set up the structure of the project while making use of this feature.

8. Auto-Configuration : This helps in loading the default configurations according to the project you are working on. In this way, unnecessary WAR files can be avoided.
9. Spring Actuator: Spring boot uses actuator to provide "Management Endpoints" which helps the developer in going through the Application Internals, Metrics etc.
10. Logging and Security : This ensures that all the applications made using Spring Boot are properly secured without any hassle.

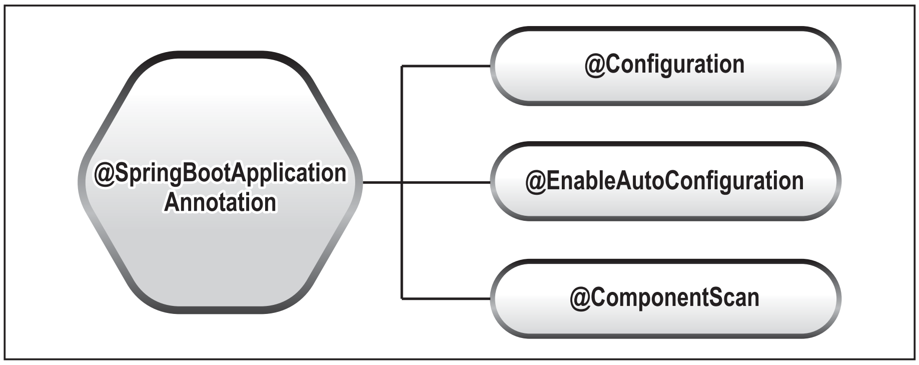
Spring Boot Application Workflow:

The entry point of a spring boot application is @SpringBootApplication and the main method. It scans all the components automatically that are present in the project by using @ComponentScan annotation. It also configures the application automatically based on the dependencies added to the project by using @EnableAutoConfiguration annotation.

1. The client makes the HTTP requests.
2. This request goes to the controller, and the controller maps and handles it.
3. The request is passed on to the service layer where all the business logic is performed on the data that is mapped to JPA with model classes.
4. As a result, a web page is returned to the user if no error occurred.

16. What does @SpringBootApplication annotation do internally?

@SpringBootApplication annotation is one point replacement for using @SpringBootConfiguration, @EnableAutoConfiguration and @ComponentScan annotations alongside their default attributes.



This enables the developer to use a single annotation instead of using multiple annotations thus lessening the lines of code. However, Spring provides loosely coupled features which is why we can use these annotations as per our project needs.
If you are not using @SpringBootApplication annotation then application goes to failure in startup only.

1. **@SpringBootConfiguration:** Designates this class as a configuration class. Configuration classes are the heart of Java-based application configuration in Spring. They can use @Bean annotated methods to specify bean definitions.
2. **@EnableAutoConfiguration:** Enables Spring Boot's auto-configuration feature, which attempts to automatically configure your application based on the dependencies in its classpath. For example, if Spring MVC is on the classpath, this annotation flags the application to be web-applicable and activates key behaviours like setting up a DispatcherServlet.
3. **@ComponentScan:** Enables component scanning. This allows Spring to automatically discover other components, configurations and services in the same package as the one where the @SpringBootApplication is placed, allowing it to automatically manage them (i.e., create bean instances for your classes at application startup)

17. Can we change the default port of the embedded Tomcat server in Spring boot?

Yes, by using server.port=2025 in application.properties file.
Here, you can set any 4-digit port number.

18. What are HTTP Request Methods?

- GET- Read data
- POST- Save fresh data
- PUT- Update/modify existing data
- PATCH- For partial update data
- DELETE- Delete data

19. What is the HTTP Status Code and list out HTTP Status code?

The Server issues an HTTP Status Code in response to a request from the client. Status code is a 3-digit integer.
The first digit defines the status code's category, which begins with a number between one and five.

For example, 1xx Informational, 2xx Success, 3xx Redirection, 4xx Client Error, 5xx Server Error

Important HTTP Status Code:

1. 200 OK
2. 201 Created
3. 400 Bad Request
4. 401 Unauthorized
5. 402 Payment Required
6. 403 Forbidden
7. 404 Not Found
8. 405 Method Not Allowed
9. 408 Request Timeout
10. 409 Conflict
11. 415 Unsupported Media Type
12. 500 Internal Server Error
13. 501 Not Implemented
14. 502 Bad Gateway
15. 503 Service Unavailable
16. 504 Gateway Timeout
17. 505 HTTP Version Not Supported

20. What are the uses of below annotations?

Core Spring Framework Annotations:

1. **@Required:** The @Required annotation is applied on the setter method of the bean file. Use of this annotation indicates that the annotated bean must be populated at configuration time with the required property. Otherwise, it will throw a **BeanInitializationException**.

2. **@Autowired:** Spring's dependency injection wires an appropriate bean into the marked class member. In Spring framework, spring itself take care of creating object, developer not need to create object by using new keyword. The @Autowired annotation is applied on fields, instance variables, setter methods,

and constructors. It provides auto wiring to bean properties. When we apply the @autowire to a field, Spring contain will automatically assign the fields with assigned values. We can also use this annotation on private fields.

3. **@Configuration:** It mark a class as a source of bean definitions. It is configured using the Java class. A Java class that is annotated with this annotation is a configuration itself and will import the methods to instantiate and configure the dependencies.

4. **@ComponentScan:** The @ComponentScan annotation is used to scan a package for beans. It is used with @Configuration annotation. We can also define the base package for scanning. If we do not specify the base package, it will scan from the package of the class that declares this annotation.

5. **@Bean:** It indicates that a method produces a bean to be managed by the Spring container. The @Bean annotation is an alternative of XML <bean> tag. It is a method level annotation, which tells the method to produce a bean to be managed by the Spring Container. It also works with @Configuration annotation to create Spring beans. As the @Configuration holds methods to instantiate and configure dependencies. The methods, which are annotated with @Bean annotation acts as bean ID and it creates and returns the actual bean.

6. **@Qualifier:** It filters what beans should be used to @Autowire a field or parameter.
   The @Qualifier annotation is used along with @Autowired annotation. It provides more control over the dependency injection process. It can be specified on individual method parameters or constructors' arguments. It is useful for reducing the duplicity while creating more bean files of the same type and we want to use only one of them with a property.

7. **@Scope-** A bean's scope is set using the @Scope annotation. By default, the Spring framework creates exactly one instance for each bean declared in the IoC container. This instance is shared in the scope of the entire IoC container and is returned for all subsequent getBean() calls and bean references. Singleton is the default scope of all the beans.

   The Spring Framework has 5 Beans scope- **singleton, prototype, request, session, globalSession.**

   Realtime Example: In shopping applications, customers expect to get different lists to be created. To achieve this behaviour, the scope of the Java class bean needs to be set to prototype. Then Spring framework creates a new bean instance for each getBean() method call. So, we need to add the @Scope("prototype").

8. **@Lazy:** It makes @Bean or @Component be initialized on demand rather than eagerly.**@Value:** The @value annotation is used at filed, constructor parameter, and method parameter level. It indicates a default value for the field or parameter to initialize the property with. Its use is quite similar to @Autowired annotation; it also injects values from a property file into a bean's attribute. It supports both ${.....} and #{....} placeholders.

***Spring Framework Stereotype Annotations*:**

1. **@Component:** The @Component annotation is used at the class level to mark a Java class as a bean. When we mark a class with @Component annotation, it will be found during the classpath. The Spring framework will configure the annotated class in the application context as a Spring Bean.

2. **@Service:** We specify a class with @Service to indicate that they're holding the business.One most important thing about the @Service Annotation is it can be applied only to classes. It is used to mark the class as a service provider. So overall @Service annotation is used with classes that provide some business functionalities.

3. **@Repository:** The @Repository annotation is used on java classes which directly access the database. It acts as a marker for any class that is used as a repository or DAO (Data Access Object). This annotation has a built-in translation feature means when the class found an exception, it will automatically handle with that; there is no need to add a try-catch block manually.
4. **@Controller-** We specify a class with @Controller to indicate that they're front controllers and responsible to handle user requests and return the appropriate response.

   The @Controller annotation is typically used in combination with annotated handler methods based on the @RequestMapping annotation. It can be applied to classes only. It's used to mark a class as a web request handler. It's mostly used with Spring MVC applications.

Spring Boot Annotations

@SpringBootApplication: @SpringBootApplication is the combination of three annotations @EnableAutoConfiguration, @ComponentScan, and @Configuration. The @SpringBootApplication annotation is used on the application class while setting up a new Spring Boot project. The class that is annotated with this annotation must be placed in the base package. The main task of this annotation is to scan the projects; it scans its sub-packages only. For example, if we annotate a class that is available in a sub-package then it will not scan the main package.

Rest API Annotations:

1. **@Controller:** It marks the class as web controller, capable of handling the requests.
   We specify a class with @Controller to indicate that they're front controllers and responsible to handle user requests and return the appropriate response.
   The @Controller annotation is typically used in combination with annotated handler methods based on the @RequestMapping annotation. It can be applied to classes only. It's mostly used with Spring MVC applications.

2. **@RestController:** @RestController annotation is a combination of @Controller and @ResponseBody annotations. Itself annotated with the @ResponseBody annotation. If the @RestController is used, then there is no need for annotating each method with @ResponseBody.

3. **@RequestMapping:** The @RequestMapping annotation is used to map/route the web requests. It can hold several optional components such as consumes, header, method, name, params, path, value, etc. This annotation can be used with classes as well as methods.

4. **@GetMapping:** @GetMapping annotation for mapping HTTP GET requests onto specific handler methods. It is used for fetch data.
   Specifically, @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET)

5. **@PostMapping:** @PostMapping annotation for mapping HTTP POST requests onto specific handler methods. It is used for save fresh data.
   Specifically, @PostMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST)

6. **@PutMapping:** @PutMapping annotation for mapping HTTP PUT requests onto specific handler methods. It is used for update existing data.

Specifically, @PutMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PUT).

7. **@PatchMapping:** @PatchMapping annotation for mapping HTTP PATCH requests onto specific handler methods. It is used for partial updation.
   Specifically, @PatchMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PATCH).

8. **@DeleteMapping-** @DeleteMapping annotation for mapping HTTP DELETE requests onto specific handler methods. It is used for delete data.
   Specifically, @DeleteMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE).

9. **@RequestBody:** @RequestBody annotation indicating a method parameter should be bound to the body of the web request. The body of the request is passed through an HttpMessageConverter to resolve the method argument depending on the content type of the request. Optionally, automatic validation can be applied by annotating the argument with @Valid annotation.

10. **@ResponseBody:** It is used to bind the method return value to the response body. It acknowledges the Spring Boot Framework to serialize a return an object into JSON and XML format.

11. **@PathVariable:** The @PathVariable annotation is used to extract the values from the URI. It is the most suitable annotation for building the RESTful web service, where the URL contains a path variable. The multiple @PathVariable can be defined in a method.

12. **@RequestParam:** It is used to extract the query parameters from the URL. It is most suitable for developing web applications. It can define the default values if the query parameter is not present in the URL.

13. **@RequestHeader:** It is used to get the details about the HTTP request headers. It can be used as a method parameter. The optional elements of this annotation are name, value, required, default Value.

14. **@RequestAttribute-** It is used to wrap a method parameter to the request attribute. It gives convenient access to the request attributes from a controller method. Using this annotation, we can access objects that are populated on the server-side.

15. **@CrossOrigin:** CORS (Cross-Origin Resource Sharing) enables cross-domain communication for the annotated request handler methods. It help us to communicate with Front End Application.
   It is used to enable cross-origin requests. It useful for the cases, if a host serving JavaScript is different from the data serving host. In order to enable this communication, we just need to add the @CrossOrigin annotation.

16. **@ControllerAdvice:** Controller advises working as a global exception handler for all API exceptions. Controller advice helps keep the controller code clean and declutter from error related code. @ControllerAdvice annotation is a specialization of @Component. The classes annotated with @ControllerAdvice are auto-detected by class path scanning.

17. **@ExceptionHandler:** @ExceptionHandler annotation for handling exceptions in specific handler classes and/or handler methods. Handler methods which are annotated with this annotation are allowed to have very flexible signatures. Spring calls this method when a request handler method throws any of the specified exceptions. The caught exception can be passed to the method as an argument.

18. **@ResponseStatus:** We can specify the desired HTTP status of the response if we annotate a request handler method with this annotation. We can declare the status code with the code argument, or its alias, the value argument. Also, we can provide a reason using the reason argument.

21. Differentiate @Controller and @RestController.

Controller	RestController
1.@Controller does not contain @ResponseBody annotation	1.@RestController itself contains @ResponseBody annotation
2.Mostly will use Controller in Spring MVC	2.In Spring Boot always will prefers to use @RestController
3.In @Controller we can return a view in Spring Web MVC	3.In @RestController we can return object as a response
4.The @Controller is a specialization of @Component annotation	4.The @RestController is a specialization of @Controller annotation

5.The @Controller annotation indicates that a particular class serves the role of a controller. Spring Controller annotation is typically used in combination with annotated handler methods based on the @RequestMapping annotation. It can be applied to classes only. It's used to mark a class as a web request handler	5.It is used at the class level and allows the class to handle the requests made by the client
6.It was added to Spring 2.5 version	6.It was added to Spring 4.0 version

22. Explain Spring MVC Workflow.

Spring MVC:
Spring MVC is a Java framework used to build web-based applications. The MVC term signifies that it follows the Model View Controller design pattern.

Model contains the core data of the application. In general, they consist of POJO (Plain Old Java Object).

View is responsible for rendering the model's data, and it generates HTML output that the client's browser can interpret.

Controller contains the business logic of the application.

Spring MVC Workflow:

1. The client system sends an HTTP request to a specific URL.
   Example: www.amazon.com
2. DispatcherServlet, configured in the web.xml file, receives the request.
3. It then passes the request to a specific controller by finding it using Handler Mapping depending on the URL requested using @RequestMapping and @Controller annotations.
4. Spring MVC Controller executes the business logic and then returns a logical view name and model to DispatcherServlet.
5. DispatcherServlet consults view resolvers until the actual View is determined for rendering the output.
6. DispatcherServlet contacts the chosen view (like JSP) with model data, and it renders the output depending on the model data.
7. The provided output is returned to the client in the form of a response.

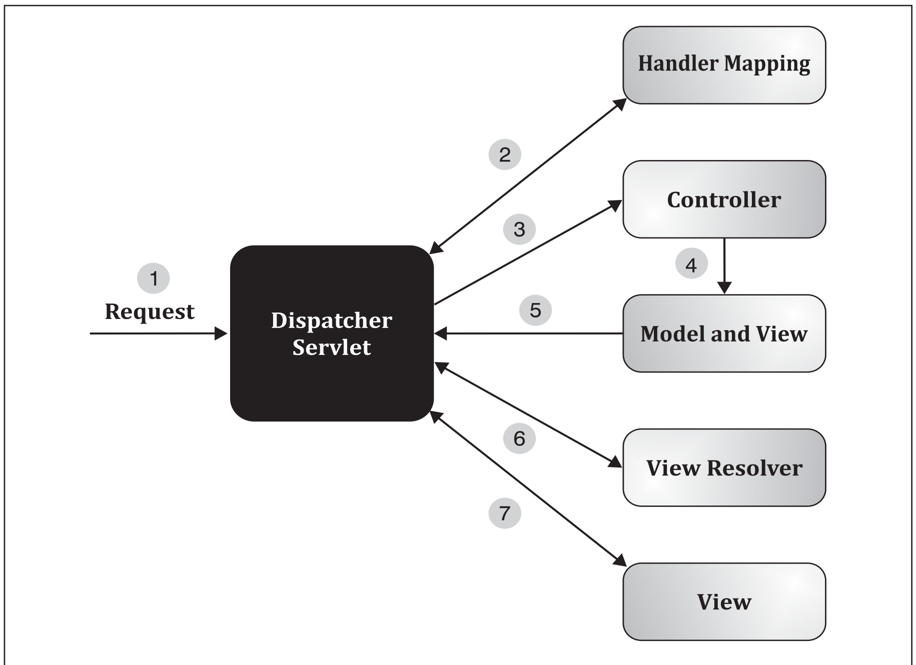


Fig. Spring MVC Workflow

23. Differentiate @RequestParam and @PathVariable.

@RequestParam	@ PathVariable
1.It is used to extract query parameters that is anything after “?” in the URL	1.It is used to extract the data present as part of the URI itself.
2. @RequestParam supports optional parameters, as the absence of a parameter will result in a default value or null.	2.@PathVariable requires the parameter to be present in the URL path. It is not optional.

3.Example:
`@DeleteMapping("/deletedatabyid/")`
`public ResponseEntity<String>`
`deleteDataById(@RequestParam int empId){`
`employeeServiceImpl.deleteDataById(empId);`
`return ResponseEntity.ok("Data Deleted`
`Successfully"); }`
	3.Example:
	`@DeleteMapping("/deletedatabyid/{empId}")`
	`public ResponseEntity<String>`
	`deleteDataById(@PathVariable int empId){`
	`employeeServiceImpl.deleteDataById(empId);`
	`return ResponseEntity.ok("Data Deleted`
	`Successfully");    }`

4.@RequestParam is more useful on a traditional web application where data is passed in the query parameters	4. @PathVariable is more suitable for RESTful web services where URL contains values.
5.Mostly @RequestParam used in Spring MVC	5. Mostly @PathVariable used in Spring Boot

24. How to create custom exception in Spring Boot.

- Create package as exception then inside package we are going to create Custom Exception class- Ex. RecordNotFoundException then extends Exception/ RunTimeException
- Use Parameterized Constructor and pass String type of parameter.
- Use super() keyword pass this String type parameter.
- In class level use @ResponseStatus(value= HttpStatus.NOT_FOUND)

Example:

```
public class RecordNotFoundException extends RuntimeException {
  public RecordNotFoundException(String msg) {
    super(msg);
  }
}
```

25. What are Profiles(Environment) in Spring Boot?

Profiles in the Spring framework enables users to map components and beans to specific profiles(environment), such as the **Development (DEV) profile, Production (PROD) profile, the Test(QA) profile.**

In Spring Boot, the annotation @Profile is used to map components and beans to a certain profile.

Once you are done with the profile-specific configuration, you have to set the active profile in an environment. To do that, either you can
Use Dspring.profiles.active=prod in arguments.
Use spring.profiles.active=prod in application.properties file.

26. Differentiate SOAP & REST API?

SOAP (Simple Object Access Protocol)	**REST** (Representational state transfer)
1.It only supports XML format.	1.It supports various formats like HTML, XML and JSON.
2.SOAP is more secure	2.REST is less secure than SOAP
3.SOAP cannot use REST because SOAP is a protocol.	3.REST can use SOAP as a protocol for web services.
4.SOAP defines standards to be strictly followed.	4.REST does not define too much standards like SOAP.
5.SOAP is less preferred than REST.	5.REST more preferred than SOAP.

27. What is actuator in Spring Boot?

It help us monitor systems health and provides endpoint security
By default, spring security is enabled for all actuator endpoints if it available in the classpath.

Spring Boot Actuator Features:

There are 3 main features of Spring Boot Actuator:

1. **Endpoints**: The actuator endpoints allows us to monitor and interact with the application. Spring Boot provides a number of built-in endpoints. We can also create our own endpoint. We can enable and disable each endpoint individually.
2. **Metrics:** Spring Boot Actuator provides dimensional metrics by integrating with the micrometer. The micrometer is integrated into Spring Boot. It is the instrumentation library powering the delivery of application metrics from Spring. It provides vendor-neutral interfaces for timers, gauges, counters, distribution summaries, and long task timers with a dimensional data model.
3. **Audit:** Spring Boot provides a flexible audit framework that publishes events to an AuditEventRepository. It automatically publishes the authentication events if spring-security is in execution.

28. Give some brief about application.properties file.

In application.propeties file we are adding mostly db configuration. In a Spring Boot Application, 'application.properties' is an input file to set the application up. Unlike

the standard Spring framework configuration, this file is auto detected in a Spring Boot Application. It is placed inside “src/main/resources” directory. Therefore, we don't need to specially register a PropertySource, or even provide a path to a property file.

An application.properties file stores various properties in key=value format. These properties are used to provide input to Spring container object, which behaves like one time input or static data.

29. What is Data JPA & Why it's more powerful?

Its lightweight framework, And Data JPA uses Hibernate as internal implementation and Data JPA have inbuilt methods ex- save(), saveAll(), findById(), findAll(), deleteById(), deleteAll(), etc.

Spring Data JPA is part of Spring Data family. Spring Data makes it easier to create Spring driven applications that use new ways to access data, such as non-relational databases, map-reduction frameworks, cloud services, as well as well-advanced relational database support.

Features:

- Reduce code size for CRUD operations by using JPARepository
- Support QueryDSL and JPA queries
- Audit of domain classes
- Support for batch loading, sorting, dynamical queries
- We are able to create custom methods as per requirements

30. What is Junit?

JUnit is an open-source, Java-based unit testing framework that plays a crucial role in achieving the culture of TDD (Test Driven Development). The TDD culture lays strong emphasis on setting up the test data for testing a logic that would be implemented once the testing is successful. JUnit helps to increase the software stability as it helps in identifying the bug in the code logic at an early stage without requiring the software to go to production. This helps in reducing the time required to debug any issues later on.

31. What is Mockito and list out some annotations?

Mockito is an open-source Java mocking framework that allows the creation of test objects that simulate the behaviour (mock) of real-world objects. This helps in achieving test-driven development.

Mockito framework allows developers to verify system behaviours without establishing expectations.

Here, we are giving dummy input to test actual methods and not going to persist unnecessary data into database.

Annotations

1. **@SpringBootTest:** It is a class level annotation.
   Spring Boot provides @SpringBootTest annotation for Integration testing. This annotation creates an application context and loads the full application context.
   @SpringBootTest will bootstrap the full application context, which means we can @Autowire any bean that's picked up by component scanning into our test.

2. @RunWith: It is a class level annotation.
It is used to keep the test clean and improves debugging. It also detects the unused stubs available in the test and initialize mocks annotated with @Mock annotation. The @RunWith annotation is available in the org.mockito.junit package.

3. @Mock: Spring Boot provides @SpringBootTest annotation for Integration testing. This annotation creates an application context and loads the full application context.@SpringBootTest will bootstrap the full application context, which means we can @Autowire any bean that's picked up by component scanning into our test.

4. @Test: It is a method level annotation.
@Test annotation is used for marking the method as a test method and it should be executed when the user runs the class.

5. @InjectMocks: The '@InjectMocks' annotation automatically injects mock objects into the class that is under test. It simplifies the process of setting up the test environment by automatically wiring the mock objects into the tested class.

6. @Spy: The 'Mockito.spy()' method allows you to create a partial mock object by wrapping an existing object. It retains the original behavior of the object while allowing you to stub or verify specific methods.

7. @Captor: It allows the creation of a field-level argument captor. It is used with the Mockito's verify() method to get the values passed when a method is called. Like other annotations, @Captor annotation is also available in the org.mockito package.

32. List out some Mockito methods.

1. when() Method:
The 'when()' method in Mockito enables the stubbing of methods. It allows us to specify the behavior of a mock object when certain methods are called. It follows the pattern: "When a specific method is called, then return a specific value."

2. **thenReturn Method:**
The 'thenReturn()' method is used to stub a method and specify the value it should return when invoked.

3. **assertEquals():**
It is used for validating whether the expected and actual values are equal.

4. **verify() Method:**
   The 'verify()' method in Mockito verifies whether specific methods have been called during a test. It allows us to validate the occurrence of expected behaviours in our code. This method is typically used at the end of a test to ensure that the defined methods have been invoked as intended.

5. **times() Method:**
   The 'times()' method in Mockito is utilized to verify a method's exact number of invocations. It allows us to declare the number of times a method should be invoked during testing. This method helps ensure that the expected behaviour is correctly executed and the desired interactions occur as intended.

6. **spy() Method:**
   Mockito provides a method called 'spy()' that allows us to mock an object partially. When using the 'spy()' method, we create a spy or stub of a real object.

7. **doReturn() Method:**
   The 'doReturn()' method in Mockito is used in special cases when the 'Mockito.when(object)' method cannot be used. It provides an alternative way to stub method invocations on mock objects. However, it is generally recommended to use the 'Mockito.when(object)' method for stubbing, as it is type-safe and more readable.

8. **only() Method:**
   The 'only()' method in Mockito is used to verify that a specific method is the only one invoked during a test. It ensures that no other methods are called on the given mock object.

33. What is mocking and stubbing?

- **Mocking**: Mocking is a phenomenon where an object mimics a real object. An object on which you set expectations.
- **Stubbing**: Stubbing are the codes responsible for taking place of another component. An object that provides predefined answers to method calls.

34. How to ignore tests in Junit?

We need to ignore test cases when we have certain functionalities that are not certain or they are under development. To wait for the completion of the code, we can avoid running these test cases. In JUnit 4, we can achieve this by using @Ignore annotation over the test methods. In JUnit 5, we can do it using @Disabled annotation over the test methods.

35. What are the difference between @Before and @After.

@Before	@After
Methods annotated with @Before are executed before each test case.This is used for preparing the test environment like to read inputs, initialise the variables etc.	Methods annotated with @After are executed after each test case. This is used for cleaning up the test environment like deleting the temporary data, restoring the value of variables to default. This helps in saving memory.

36. When and why should we use spy?

Spy is a partial mock object supported by the Mockito framework.

Uses of spy objects:

1. Whenever mock is not set up, interaction on spy results in real method calls. Spy objects allow verifying interactions such as whether the method was called and how many times it was called, etc.
2. They provide flexibility for setting up partial mocks.
   Example- If we have 2 methods, and we want one method to be mocked and the other to be called actually, then we can use spy.

The main difference between mock and spy is that in mock, we create a complete fake object whereas, in spy, we have a hybrid of a real object.

37. What is Microservices and Explain its architecture?

2 or more Spring Boot module communicate with each other that's called Microservices. Microservices or more appropriately Microservices Architecture is an SDLC approach based on which large applications are built as a collection of small functional modules. These functional modules are independently deployable, scalable, target specific business goals, and communicate with each other over standard protocols. Such modules can also be implemented using different programming languages, have their databases, and deployed on different software environments. Each module here is minimal and complete.

Benefits:

1. Self-contained, and independent deployment module.

2. Independently managed services.
3. In order to improve performance, the demand service can be deployed on multiple servers.
4. It is easier to test and has fewer dependencies.
5. A greater degree of scalability and agility.
6. Simplicity in debugging & maintenance.
7. Better communication between developers and business users.
8. Development teams of a smaller size.

Drawbacks:

1. Due to the complexity of the architecture, testing and monitoring are more difficult.
2. Lacks the proper corporate culture for it to work.
3. Complex development.
4. Requires a cultural shift.
5. Expensive compared to monoliths.
6. Security implications.
7. Maintaining the network is more difficult.

Architecture of Microservices:

The main idea behind microservices is that some types of applications are easier to build and maintain when they are broken down into many small pieces that work together.

API Gateway: End user request always comes on API Gateway and API Gateway will take care of traversing this request to particular services. Basically, the API Gateway is a reverse proxy to microservices and acts as a single-entry point into the system. It is similar to a Facade pattern from object-oriented design and similar to the notion of an "Anti-Corruption Layer" in Domain Driven Design. It makes the processes of API design, implementation, and management considerably simpler and more consistent. The API gateway also helps by recording data for analysis and auditing purposes, load balancing, caching, and static response handling. The diagram below shows how the gateway typically fits into the overall microservice architecture.

Eureka Server: Register all services as instances in Eureka Server.

It, also referred to as Netflix Service Discovery Server, is an application that keeps track of all client-service applications. As every Microservice registers to Eureka Server, Eureka Server knows all the client applications running on the different ports and IP addresses. It generally uses Spring Cloud and is not heavy on the application development process.

Hystrix Dashboard: Monitor System Health

Netflix has provided a bunch of libraries to form an ecosystem in microservices. Like Eureka, Hystrix is also an open source library provided by Netflix in the Microservices space. Hystrix implements the Circuit Breaker pattern. You don't have to write the network or thread programming to handle fault tolerance in the Microservices. You need to use Hystrix library just by giving parameters and that's it. Hystrix is going to do all the work for you internally. The best part of it is that it works amazingly with Spring Boot. If you pick any Microservice based project, there are pretty good chances that they are using Hystrix.

Zipkin: It will help us for distributed log tracing.

Zipkin is very efficient tool for distributed tracing in microservices ecosystem. Distributed tracing, in general, is latency measurement of each component in a distributed transaction where multiple microservices are invoked to serve a single business usecase.

Zipkin was originally developed at Twitter, based on a concept of a Google paper that described Google's internally-built distributed app debugger – dapper. It manages both the collection and lookup of this data. To use Zipkin, applications are instrumented to report timing data to it.

Sleuth: It identify which service is depends with another services.

Sleuth introduces unique IDs to your logging which are consistent between microservice calls which makes it possible to find how a single request travels from one microservice to the next.

Spring Cloud Sleuth adds two types of IDs to your logging, one called a trace ID and the other called a span ID. The span ID represents a basic unit of work, for example sending an HTTP request. The trace ID contains a set of span IDs, forming a tree-like structure. The trace ID will remain the same as one microservice calls the next.

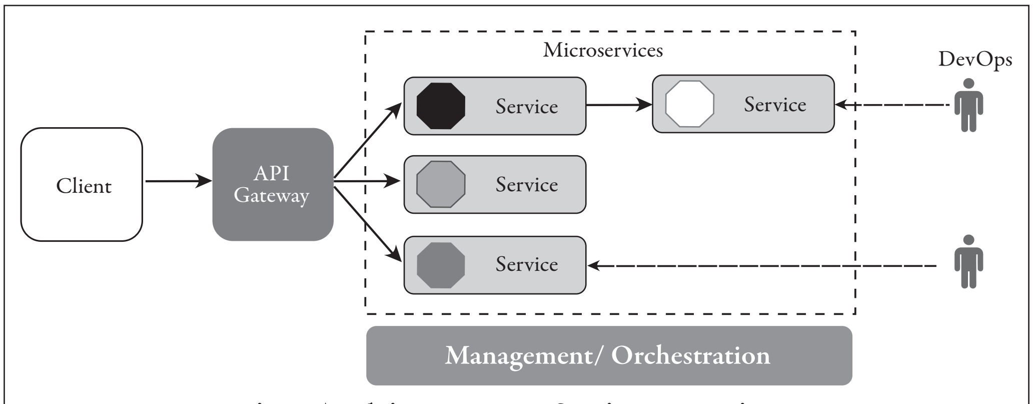


Fig. Architecture of Microservices

38. What is Circuit Breaker in Microservices?

In a microservice based application, Circuit Breaker is a technique, where we stop executing an erroneous method and redirect every request to a custom method (Fallback method). Generally, we stop execution of a particular method if it is continuously throwing an exception. When we break the circuit, we also avoid any cascading failures to the downstream services. Its basic function is to interrupt current flow after a fault is detected. A circuit breaker can be reset to resume normal operation either manually or automatically.Depending on the state, Circuit Breaker changes its behavior:

A) Closed

If Client request is sent to the actual service method only, then it is called as CLOSED CIRCUIT. Hence, this state represents that the service is running properly and providing the expected functionality.

B) Open

If Client request is redirected to Fallback method, then such case is an OPEN CIRCUIT. Hence, this state represents that service is unavailable or faulty and error rate is beyond the threshold.

C) Half-Open

Once the state becomes OPEN, we wait for some time in the OPEN state. After a certain period of time, the state becomes HALF_OPEN.

During this period, we do send some requests to Service to check if we still get the proper response.

If the failure rate is below the threshold, the state would become CLOSED. If the failure rate is above the threshold, then the state becomes OPEN once again. This cycle continues till the service becomes stable.

39. Explain TDD & DDD.

TDD- Test Driven Development
First come tests and then the code. The minimal piece of code is written in order to pass the designed test. In other words, it is the process of testing the code before its accrual writing. If the code passes the test, then we can proceed to its refactoring.

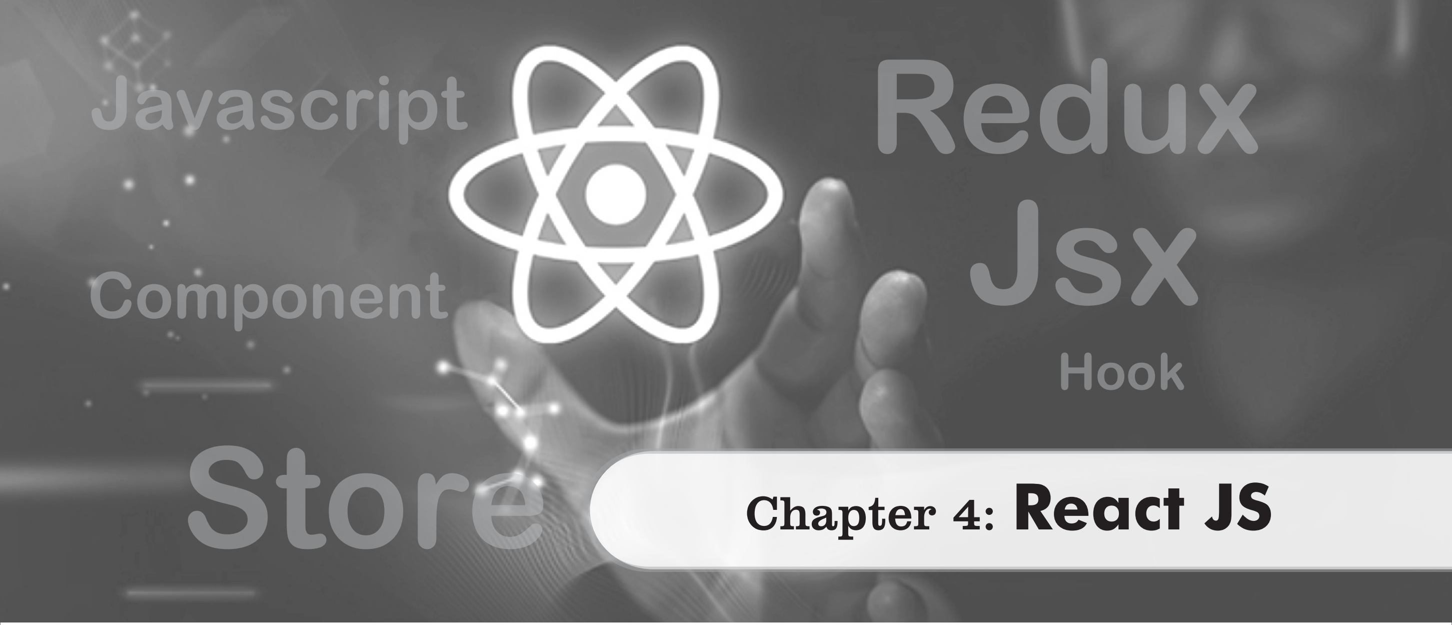
DDD- Domain Driven Development
The way of creating complex systems by developing the separate parts of it. At first, the domain (a set of functionality) is defined and described as before creating something it is necessary to understand what exactly it will be. In other words, it is the process of being informed about the domain before code writing.

40.Difference between Monolithic, SOA and Microservice Architecture

Monolithic Architecture: It is "like a big container" where all the software components of an application are bundled together tightly. It is usually built as one large system and is one code-base.

SOA (Service-Oriented Architecture): It is a group of services interacting or communicating with each other. Depending on the nature of the communication, it can be simple data exchange or it could involve several services coordinating some activity.

Microservice Architecture: It involves structuring an application in the form of a cluster of small, autonomous services modeled around a business domain. The functional modules can be deployed independently, are scalable, are aimed at achieving specific business goals, and communicate with each other over standard protocols.



Chapter 4: React JS

1. What is React JS and What are the features of ReactJS?

React is a JavaScript library for building user interfaces. React allows us to create reusable UI components. React is a widely used JavaScript library that was launched in 2011. It was created by developers at Facebook, and it is primarily used for front-end development. React uses the component-based approach, which ensures that you build components that possess high reusability.

React JS Features:

1. Performance: ReactJS use JSX, which is faster compared to normal JavaScript and HTML.
2. Virtual DOM: Boosting Performance and Efficiency. Virtual DOM is a less time taking procedure to update webpages content.
3. Components: Building Blocks for Reusable and Maintainable Code

   Unidirectional Data Flow: Streamlining Data Management
4. JSX: Bridging the Gap Between JavaScript and HTML React Hooks
5. Context API: Managing Global State
6. Concurrent Mode: Optimizing User Experience

2. What is Virtual DOM?

DOM stands for 'Document Object Model'. In simple terms, it is a structured representation of the HTML elements that are present in a webpage or web app. DOM represents the entire UI of your application.

A virtual DOM is a simple JavaScript object that is the exact copy of the corresponding real DOM. It can be considered a node tree that consists of elements, their attributes, and other properties. Using the render() in React, it creates a node tree and updates it based on the changes that occur in the data model.

3. What is ES6 and What are the features of Es6?

Es6 stands for ECMAScript 6.

ECMAScript was created to standardize JavaScript, and ES6 is the 6th version of ECMAScript, it was published in 2015, and is also known as ECMAScript 2015.

ES6 Features:

- const and let
- Arrow Functions
- Classes
- Rest and Spread Operations
- Template Strings
- Promises -with async and await
- Import and Export

4. What are the different phases of ReactJS component lifecycle ?

In ReactJS, the development of each component involves the use of different lifecycle methods. All the lifecycle methods used to create such components, together constitute the component's lifecycle.

There are 4 phases :

1. Initializing: This is the phase that involves the beginning of the journey of the component to the DOM.
2. Mounting : Mounting means putting elements into the DOM.
3. Updating: Here, the component can be updated and rendered again if required after it gets added to the DOM.
4. Unmounting : The final phase involves the destruction of the component and its eventual removal from the DOM.

Phase 1: Initializing:

This is the initial phase of the React component lifecycle. Initializing phase involves all the declarations, definitions, and initialization of properties, default props as well as the initial state of the component required by the developer. In a class-based component, this is implemented in the constructor of the component. This phaseoccurs only once throughout the entire lifecycle.

The methods included in this phase are:

getDefaultProps(): The method invoked immediately before the component is created or any props from the parent component are passed into the said (child) component. It is used to specify the default value of the props.

getInitialState(): The method invoked immediately before the component is created and used to specify the default value of the state.

Phase 2: Mounting:

The second phase of the React component lifecycle. It commences when the component is positioned over the DOM container(meaning, an instance of the component is created and inserted into the DOM) and rendered on a webpage.

It consists of 2 methods:

componentWillMount(): The method invoked immediately before the component is positioned on the DOM, i.e. right before the component is rendered on the screen for the very first time.

componentDidMount(): The method invoked immediately after the component is positioned on the DOM, i.e. right after the component is rendered on the screen for the very first time.

Phase 3: Updating:

The third phase of the ReactJS Component Lifecycle is the Updation phase. Followed by the mounting phase, it updates the states and properties that were declared and initialized during the initialization phase (if at all any changes are required). It is also responsible for handling user interaction and passing data within the component hierarchy. Unlike the initialization phase, this phase can be repeated multiple times. Some of the lifecycle methods falling into this category are as follows:

componentWillReceiveProps(): The method invoked immediately before the props of a mounted component get reassigned. It accepts new props which may/may not be the same as the original props.

shouldComponentUpdate(): The method invoked before deciding whether the newly rendered props are required to be displayed on the webpage or not. It is useful in those scenarios when there is such a requirement not to display the new props on the screen.

componentWillUpdate(): The method invoked immediately before the component is re-rendered after updating the states and/or properties.

componentDidUpdate(): The method invoked immediately after the component is re-rendered after updating the states and/or properties.

Phase 4: Unmounting:

Unmounting is the last phase of the ReactJS component lifecycle.

This phase includes those lifecycle methods which are used when a component is getting detached from the DOM container. It is also responsible for performing the required cleanup tasks. Once unmounted, a component can not be re-mounted again.

componentWillUnmount(): The method invoked immediately before the component is removed from the DOM at last, i.e. right when the component is completely removed from the page and this shows the end of its lifecycle.

5. What are the components in React?

Components are the building blocks of any React application, and a single app usually consists of multiple components. A component is essentially a piece of the user interface. It splits the user interface into independent, reusable parts that can be processed separately.

There are two types of components in React:

1. Class Component:

These types of components can hold and manage their own state and have a separate render method to return JSX on the screen. They are also called Stateful components as they can have a state.

2. Functional Component:

These types of components have no state of their own and only contain render methods, and therefore are also called stateless components. They may derive data from other components as props (properties).

6. What are props in React?

Props are the shorthand name given to properties in React. Props allows to pass data from one component to other components as an argument. Props are read-only components that are immutable in nature. In an application, props follow a hierarchy that is passed down from parent to child components. However, the reverse is not supported. This is done to ensure that there is only a single directional flow of data at all times.

7. What are states in React?

State Holds information about the components. States is considered a source of objects that control aspects such as component behaviour and rendering. In React, states are mutable and used to easily create dynamic and interactive components.
They are accessed via this.state().

8. What is arrow function in React?

An arrow function is used to write an expression in React. It allows users to manually bind components easily. The functionality of arrow functions can be very useful when you are working with higher-order functions.
They are also called 'fat arrow' (=>) the functions.

9. What are events in React?

Whenever there are actions performed in React, such as hovering the mouse or pressing a key on the keyboard, these actions trigger events. Events then perform set activities as a response to these triggers. Handling an event in React is very similar to that in the DOM architecture.

10.What is useEffect Hook in React?

- Hooks is a new feature that has been added to React since version 16.8. Numerous common built-in hooks are available in React to manage states, isolate side effects, create references, boost performance, etc.
- The useEffect hook is a function in React that allows developers to perform side effects in a functional component. This can include things like data fetching, setting up subscriptions, responding to the component's lifecycle events, or updating the DOM in response to changes in state or props.
- The useEffect react hook is called after every render and takes a callback function as an argument, which contains the code for the side effect. This allows for a cleaner and more declarative approach to managing side effects in functional components.

11. What is Redux?

Redux is used to store the state of the application in a single entity. This simple entity is usually a JavaScript object. It is a predictable state container for JavaScript applications and is used for the entire applications state management. Applications developed with Redux are easy to test and can run in different environments showing consistent behaviour.

12. What are Hooks in React?

Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don't work inside classes — they let you use React without classes. Hooks were added to the React version, v16.8. The stateful logic can be easily extracted from a component, along with testing and reusing it. All of this is done without making any changes to the component hierarchy.

There are 3 rules for hooks:

1. Hooks can only be called inside React function components.
2. Hooks can only be called at the top level of a component.
3. Hooks cannot be conditional

13. How React JS communicate with Spring Boot?

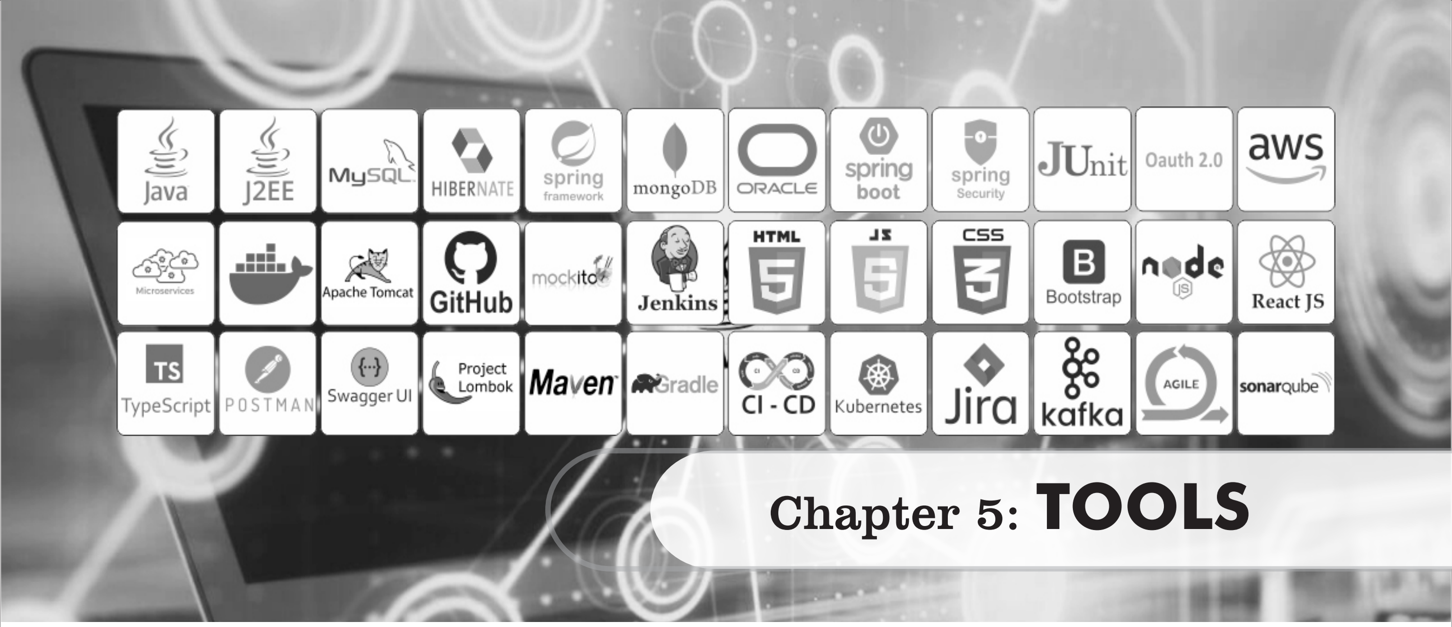
React uses Axios as HTTP third party library for communicate with the backend [REST API].
From Back End Spring Boot controller class need to add @CrossOrigin annotation.

14. What is TypeScript?

TypeScript is a superset of JavaScript, which means that any valid JavaScript code is also valid TypeScript code. However, TypeScript extends JavaScript by providing additional features that help developers catch and fix errors early in the development process.

The main feature of TypeScript is its support for static typing. In JavaScript, variables can hold values of any type, and this can sometimes lead to errors that are only discovered at runtime. TypeScript introduces a type system, which allows developers to specify the types of variables, function parameters, and return values. This helps catch type-related errors during the development process, which can lead to more robust and maintainable code.

TypeScript files use a .ts extension, in contrast to the .js extension used by the JavaScript files.



Chapter 5: TOOLS

1. What is Maven and Explain its actual use?

- Maven is a Local Build Management Tool
- It will help us to create new Fresh JAR/WAR File.
- Maven helps the developer to create a java-based project more easily. Accessibility of new feature created or added in Maven can be easily added to a project in Maven configuration. It increases the performance of project and building process.

The main feature of Maven is that it can download the project dependency libraries automatically.

2. What is code coverage and Why use Code Coverage?

Code coverage measures the amount of source code of a software application when it is executed. It is calculated by comparing the number of lines, statements, branches, or functions executed during testing to the total number of elements present in the code.

Example: An application consists of 100 lines of code, and during testing, assume that 70 lines of code are executed, then code coverage is 70%.

Why use Code Coverage Tools:

It is a trusty companion that helps you ensure your code is functioning exactly as it should.

1. **Code quality:** Developers can acquire insights into the quality of their code by measuring the code coverage. Elevated code coverage implies that more code paths have been tested, reducing the possibility of undiscovered errors. Code coverage tools facilitate developers to write testable code, improving code quality

and maintainability.

2. **Untested code:** Code coverage tools help determine areas of code that have not been executed during testing. This helps developers to identify portions that may contain undetected bugs. By identifying which parts of the code are untested, developers can create targeted test cases to ensure wide coverage.

3. **Measure Effectiveness:** Tools for measuring code coverage offer metrics for gauging the efficiency of test suites. They can display the proportion of code covered by tests, which shows how effectively the tests exercise the codebase. With the aid of this data, developers may evaluate how thorough their testing was and pinpoint areas that need more testing.

Code Coverage Tools:

- JaCoCo
- SonarQube
- JMockit
- Serenity

3. Which tool you have used for Code/Version Control Management?

GitHub

4. What is Docker and what are its advantages?

Docker is an open-source centralized platform designed to create, deploy, and run applications.

Docker uses container on the host's operating system to run applications.
Before Docker, many users face the problem that a particular code is running in the developer's system but not in the user's system.

So, the main reason to develop docker is to help developers to develop applications easily, ship them into containers, and can be deployed anywhere.

Advantages

1. **Lighter weight:** Unlike VMs, containers don't carry the payload of an entire OS instance and hypervisor. They include only the OS processes and dependencies necessary to execute the code.

2. **Improved developer productivity:** Containerized applications can be written once and run anywhere. And compared to VMs, containers are faster and easier to

deploy, provision and restart. This makes them ideal for use in continuous integration and continuous delivery (CI/CD) pipelines and a better fit for development teams adopting Agile and DevOps practices.

3. Greater resource efficiency: With containers, developers can run several times as many copies of an application on the same hardware as they can using VMs. This can reduce cloud spending.

5. What is Kubernetes(K8s) and its uses?

Kubernetes is a distributed open-source technology that helps us in scheduling and executing application containers within and across clusters. **K8s** is another term for Kubernetes.

A Kubernetes cluster consists of two types of resources:

1. **The Master:** Coordinates all activities in the cluster, for Example: scheduling applications, maintaining applications' state, scaling applications, and rolling out new updates
2. **Nodes:** A node is an instance of an OS that serves as a worker machine in a Kubernetes cluster.

   Also, Node will have two components

   1. **Kubelet :** Agent for managing and communicating with the master
   2. **Tool** (Docker/containers) **:** Tools for running container operations.

Pods are the smallest unit of objects that can be deployed on Kubernetes, Kubernetes packages one or more containers into a higher-level structure called a pod. Pod runs one level higher than the container.

A POD always runs on a Node but they share a few resources which can be Shared Volumes, Cluster Unique IP, and Info about how to run each container. All containers in the pod are going to be scheduled on an equivalent node.

Services are the unified way of accessing the workloads on the pods, The Control plane which is the core of Kubernetes is an API server that lets you query, and manipulate the state of an object in Kubernetes.

Uses:

- Kubernetes places control for the user where the server will host the container. It will control how to launch. So, Kubernetes automates various manual processes.
- Kubernetes manages various clusters at the same time.
- It provides various additional services like management of containers, security,

networking, and storage.

- Kubernetes self-monitors the health of nodes and containers.
- With Kubernetes, users can scale resources not only vertically but also horizontally that too easily and quickly.

6. What is Apache Kafka and list out its features?

Apache Kafka is a streaming platform that is free and open-source used for Java messaging services. Kafka was first built at LinkedIn as a messaging queue, but it has evolved into much more. It's a versatile tool for working with data streams that may be applied to a variety of scenarios.
Kafka is a distributed system, which means it can scale up as needed. All you have to do now is add new Kafka nodes (servers) to the cluster.

Kafka can process a large amount of data in a short amount of time. It also has low latency, making it possible to process data in real-time. It is used for fault-tolerant storage as well as publishing and subscribing to a stream of records. The programs are intended to process timing and consumption records. Kafka replicates log partitions from many hosts. Developers and users contribute coding updates, which it keeps, reads, and analyses in real-time. For messaging, website activity tracking, log aggregation, and commit logs, Kafka is employed. Although Kafka can be used as a database, it lacks a data schema and indexes.

features of Kafka:

- Kafka is a messaging system built for high throughput and fault tolerance.
- Kafka has a built-in patriation system known as a Topic.
- Kafka Includes a replication feature as well.
- Kafka provides a queue that can handle large amounts of data and move messages from one sender to another.
- Kafka can also save the messages to storage and replicate them across the cluster.
- For coordination and synchronization with other services, Kafka collaborates with Zookeeper.
- Apache Spark is well supported by Kafka

Following are the major components of Kafka:-

1. Topic:

A Topic is a category or feed in which records are saved and published.
Topics are used to organize all of Kafka's records. Consumer apps read data from topics, whereas producer applications write data to them. Records published to the

cluster remain in the cluster for the duration of a configurable retention period.
Kafka keeps records in the log, and it's up to the consumers to keep track of where they are in the log (the "offset"). As messages are read, a consumer typically advances the offset in a linear fashion. The consumer, on the other hand, is in charge of the position, as he or she can consume messages in any order. When reprocessing records, for example, a consumer can reset to an older offset.

2. Producer:

A Kafka producer is a data source for one or more Kafka topics that optimizes, writes, and publishes messages. Partitioning allows Kafka producers to serialize, compress, and load balance data among brokers.

3. Consumer:

Data is read by consumers by reading messages from topics to which they have subscribed. Consumers will be divided into groups. Each consumer in a consumer group will be responsible for reading a subset of the partitions of each subject to which they have subscribed.

4. Broker:

A Kafka broker is a server that works as part of a Kafka cluster (in other words, a Kafka cluster is made up of a number of brokers). Multiple brokers typically work together to build a Kafka cluster, which provides load balancing, reliable redundancy, and failover. The cluster is managed and coordinated by brokers using Apache ZooKeeper. Without sacrificing performance, each broker instance can handle read and write volumes of hundreds of thousands per second (and gigabytes of messages). It is not possible to connect directly to the Kafka Server by bypassing ZooKeeper. Any client request cannot be serviced if ZooKeeper is down.

7. Which tool you have used for API Testing?

- Postman API: Postman is better for API testing and has good data security features.
- Swagger UI : Swagger is better for API documentation and design management.

8. What is splunk?

Splunk is the perfect tool for trace error log.
Splunk is a tool designed to search, monitor, analyze, and visualize machine-generated data in real-time. It is widely used for log management, security information and event management and business intelligence.

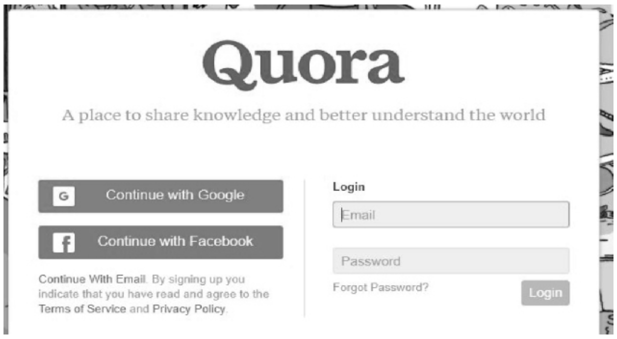
Key aspects of Splunk:

- **Log Management and Analysis:** Splunk is primarily used for ingesting, indexing, and searching through large volumes of log data generated by applications, servers, network devices, and other sources.
- **Real-time Data Processing:** It excels at real-time data processing, allowing organizations to gain insights into their operational data as events occur.
- **Search and Query Language:** Splunk employs its own powerful search and query language known as SPL (Splunk Processing Language). This language allows users to perform complex searches and data transformations.
- **Dashboards and Visualization:** Splunk offers visualization tools to create dashboards and reports, allowing users to create graphical representations of their data for easier analysis.
- **Alerting and Monitoring:** It provides alerting capabilities to notify users of specific events or patterns in the data. This is crucial for real-time incident detection and response.

9. What is Oauth?

Oauth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones.

Generally speaking, OAuth (Open Authorization Protocol) enables users to authenticate themselves with third-party service providers. With this protocol, you can access client applications on HTTP for third-party providers such as Gmail, LinkedIn, GitHub, Facebook, etc. Using it, you can also share resources on one site with another site without requiring their credentials.
Example: In Quora there is no signup option, you can go ahead with third party Google, Facebook credentials for login and access Quora.



Continue With Email. By signing up you indicate that you have read and agree to the Terms of Service and Privacy Policy.

10. What is JWT and Explain JWT structure?

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

When should you use JSON Web Tokens-

1. Authorization: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

2. Information Exchange: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

JSON Web Token structure:

JSON Web Tokens consist of three parts separated by dots (.), which are:

- Header
- Payload
- SignatureTherefore, a JWT typically looks like the following.

xxxxx.yyyyy.zzzzz

1. Header:

The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

Example:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

this JSON is Base64Url encoded to form the first part of the JWT.

2. Payload:

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims.

Example:

```
{
  "sub": "1234567890",
  "name": "Swara Jadhav",
  "admin": true
}
```

The payload is Base64Url encoded to form the second part of the JSON Web Token.

3. Signature:

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header and sign that.

Example:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

Putting all together:

```
eyJhbGci0iJIUZI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjMONTY30DkwIiwibmFtZSI6IkpvaG4
gRG91IiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD0901PSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

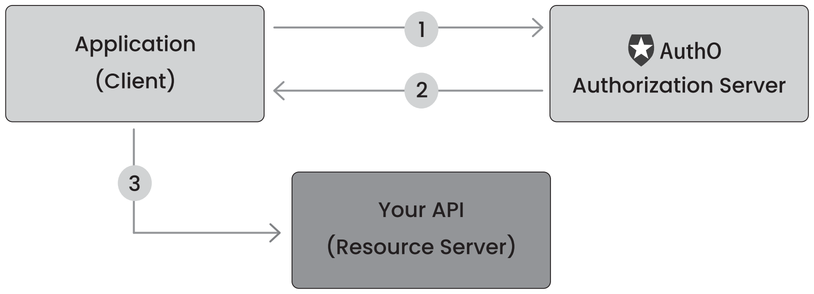
How JWT works:

In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned. Since tokens are credentials, great care must be taken to prevent security issues. In general, you should not keep tokens longer than required.

Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema.

Authorization: Bearer <token>

How JWT is obtained and used to access APIs:



- The application or client requests authorization to the authorization server. This is performed through one of the different authorization flows. For example, a typical OpenID Connect compliant web application will go through the /oauth/authorize endpoint using the authorization code flow.
- When the authorization is granted, the authorization server returns an access token to the application.
- The application uses the access token to access a protected resource (like an API).

11. What is Jenkins ?

Jenkins is Centralize Build Management Tool.

Jenkins is an open-source continuous integration/continuous delivery and deployment (CI/CD) automation software DevOps tool written in the Java programming language. It is used to implement CI/CD workflows, called pipelines. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.

With Jenkins, organizations can accelerate the software development process through automation. Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis, and much more.

Jenkins achieves Continuous Integration with the help of plugins. Plugins allow the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example- Git, Maven project, Amazon EC2.

12. What is AWS and what are services you have used in your project?

AWS (Amazon Web Services) is a platform to provide secure cloud services, database storage, offerings for computing power, content delivery, and other services to help the business scale and develop.

Services:

1. Elastic Beanstalk:
Application Deployment Simplified: Elastic Beanstalk making deployment easier for developers to deploy applications. Developers can upload their code, and Elastic Beanstalk automatically handles the deployment, capacity provisioning, load balancing, etc.

Automatic Scaling: Elastic Beanstalk can automatically scale the application environment up or down based on traffic demands. Developers can also manually adjust the capacity settings if needed.

Monitoring and Logging: Elastic Beanstalk includes built-in monitoring and logging capabilities through integration with AWS CloudWatch. This allows developers to monitor performance, track resource usage, and set up alarms for critical metrics.

2. EC2: Elastic Compute Cloud (Amazon Ec2)

Virtual Servers (Instances): EC2 provides virtual servers, known as instances, that you can use to run applications. These instances are resizable, meaning you can easily scale up or down based on your needs.

Elastic Load Balancing: EC2 can be used in conjunction with Elastic Load Balancing to distribute incoming traffic across multiple instances. This helps improve the availability and fault tolerance of your applications.

IAM Roles: You can assign IAM roles to EC2 instances, allowing them to securely access other AWS resources without needing to store access credentials on the instance.

3.RDS: Relational Database Service, is a managed database service provided by Amazon Web Services (AWS). It's designed to make it easier to set up, operate, and scale a relational database in the cloud.
Amazon RDS supports several popular database engines, including MySQL, PostgreSQL, MariaDB, Oracle, Microsoft SQL Server, and Amazon Aurora, etc.

4.Lambda: AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). It allows you to run code without provisioning or managing servers, paying only for the compute time consumed.

With Lambda, you don't need to worry about provisioning or managing servers. AWS handles the infrastructure, including server maintenance, patching, and scaling, allowing you to focus solely on writing code.

Pay-per-Use Pricing: You are charged based on number of requests for your functions and the time your code executes. There is no charge when your code is not running.

13. What is Agile and explain various Agile Ceremony?

Agile help us to give Productivity from each team mates.
The Agile methodology is a way to manage a project by breaking it up into several phases. It involves constant collaboration with stakeholders and continuous improvement at every stage. Once the work begins, teams cycle through a process of planning, executing, and evaluating. Continuous collaboration is vital, both with team members and project stakeholders.

Agile Process have 4 Ceremony:

1. **Sprint Planning-** Here will discuss about sprint task estimation and development-story point efforts. Sprint Cycle should be 2 week- 10 Days.

   Purpose: Sprint planning sets up the entire team for success throughout the sprint. Coming into the meeting, the product owner will have a prioritized product backlog. They discuss each item with the development team, and the group collectively estimates the effort involved. The development team will then make a sprint forecast outlining how much work the team can complete from the product backlog.

2. **Daily Scrum-** Here each team mates need to give below updates:
   - What you did yesterday
   - What are you doing now
   - Which Problems you are facing

   Daily Scrum call not be more than 30 Minutes

   Purpose: Stand-up is designed to quickly inform everyone of what's going on across the team. It's not a detailed status meeting. The tone should be light and fun, but informative.

3. **Backlog Refinements-** Here will discuss about Last Sprint Backlog story

   Purpose: Priority Backlog will get completed using skill full resources.

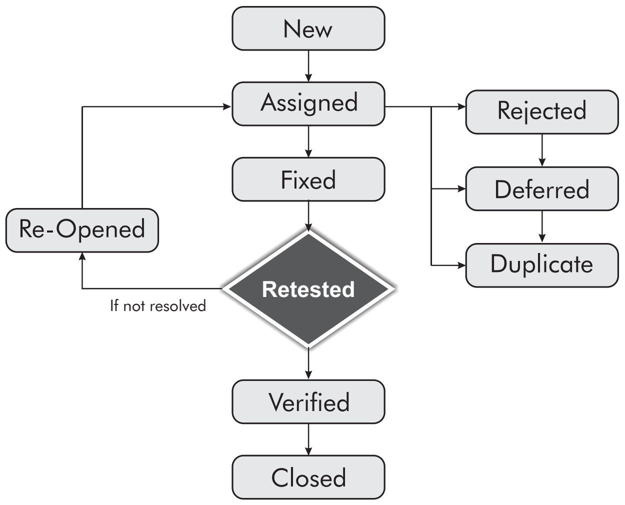
4. **Sprint Retrospective-** It will help us to create positive energy. This is the right time to say thanks whoever help us to complete story in current sprint.

Purpose: Agile is about getting rapid feedback to make the product and development culture better. Retrospectives help the team understand what worked well–and what didn't. Retrospectives aren't just a time for complaints without action. Use retrospectives to find out what's working so the team can continue to focus on those areas. Also, find out what's not working and use the time to find creative solutions and develop an action plan. Continuous improvement is what sustains and drives development within an agile team, and retrospectives are a key part of that.

14. What is JIRA?

Jira is a software tool known for its Story/Issue/Defect Tracking and Project Management capabilities. It is a Cloud-based technology that helps teams plan, execute, and report their work. This software finds applications in Bug Tracking, Issue Tracking related to Software, and Project Management. Projects based on Agile Methodology be it Kanban, Scrum or any other Agile view utilize Jira for managing software projects. Jira provides you the flexibility of managing, tracking, and scheduling all your agile software development projects together.
Each developer needs to update task status in JIRA by EOD. Scrum Master able to see those update, if required more input they will ask information in scrum call also.

Defect/Bug Life Cycle



15. What is Git and list out some useful commands?

GitHub is a version control tool it is used for code control management.
Git is a version control system that allows you to track changes in your codebase, collaborate with other developers, and manage different versions of your project. Code will be available in centralize repository in cloud platform. Developer needs to

clone repo in local, and create feature branch. Once code changes are done after code review Developer commit the code in feature branch. And Lead people merge this feature branch into main branch.

Git Commands:

1. git init: Initializes a new Git repository in the current directory.
2. git clone [repository_url]: Creates a copy of an existing Git repository from a remote server.
3. git add [file(s)]: Adds files to the staging area to be included in the next commit.
4. git commit -m "[commit_message]": Records changes in the repository with a message describing the changes.
5. git status: Shows the current state of the repository, including which files have been modified and which are staged for commit.
6. git diff: Shows the changes made to the code since the last commit.
7. git pull: Fetches and merges changes from the remote repository to your local repository.
8. git push: Uploads local changes to the remote repository.
9. git branch: Lists all existing branches in the repository.
10. git branch [branch_name]: Creates a new branch with the specified name.
11. git checkout [branch_name]: Switches to the specified branch.
12. git checkout -b [new_branch_name]: Creates a new branch and switch to that branch
13. git merge [branch_name]: Combines changes from the specified branch into the current branch.
14. git remote -v: Lists all remote repositories associated with the local repository.
15. git remote add [name] [url]: Adds a new remote repository.
16. git log: Displays a chronological list of commits.
17. git reset [commit]: Resets the current branch to a specific commit.
18. git revert [commit]: Creates a new commit that undoes the changes made in a previous commit.
19. git stash: Temporarily stores changes that haven't been committed so you can switch branches.
20. git tag [tag_name] [commit_sha]: Creates a new tag at a specific commit.
21. git push --tags: Pushes all tags to the remote repository.
22. git gui: it open Graphical User Interface

NOTES

Full Stack Developer Life

This Full Stack Developer Life career thoughts is based on my past 10+ years of experience in Software Development Industry. I have gone through different stages in my career starting from Software Trainee Developer till Solution Architect. I am not going to dictate any of the points, but all the practices listed here contributed a lot in my software development career, so if you think they make some sense for you then try to adopt it. If you have any suggestions then kindly feel free to write me back at **contact@fullstackjavadeveloper.in**

Rule 1- What is Practice?

I would say:
1. Practice is a habit.
2. Practice does not need to remember.
3. Practice comes by practicing.
4. Practice needs lot of dedication and commitment.

To become a successful software developer, you need lot of practice, dedication and commitment.

Rule 2- Do you read Software Source Code?

Only few software developers will have positive answer because reading and understanding an existing software source code is the most boring task. If you are one of them who feels reading software source code is a boring task, then you are missing one of the most important best practices, which a software developer should have in his/her life.
There are many attributes of software codes (indentation, comments, history header, function structure, etc.), which you will learn by reading existing code, specially, a code written by well-experienced software developers.

Rule 3- Follow the defined standards, don't create it

Most of the standard software organizations maintain their coding standards. These standards would have been set up by well-experienced software developers after spending years with software development. This is equivalent to following footsteps of great people left behind them.
1. File Naming convention
2. Function & Module Naming convention
3. Variable Naming convention
4. History, Indentation, Comments
5. Readability guidelines.

Rule 4 - Code should be written to be reviewed

While writing your software code, keep in mind that someone is going to review your code and you will have to face criticism about one or more of the following points but not limited to:
1. Not following standard
2. Not keeping performance in mind
3. History, Indentation, Comments are not appropriate.
4. Open files are not closed
5. Allocated memory has not been released
6. Too much hard coding.
7. Repeated code.

Rule 5- Testing to be followed like an Important thing

Testing is mandatory after every small or big change no matter how tight schedule you have or you just changed a small comment inside the code, you have testing due for the changed code.
1. Tight schedule, no compromise.
2. Changed just a comment, still you have to test it.
3. Changed just a variable name, testing has to be done.
4. If you feel lazy...it's too dangerous.

Rule 6 - Keep your Code and Documents Safe

A smart developer keeps habit of taking daily backup of the produced artifacts, otherwise machine crash can crash you as well. You should keep your artifacts at your local machine as well as another secure machine, so that in case of machine crash, you can continue with the saved copy of the source code or documents.

Rule 7- New technologies are coming everyday

If you want to sustain in the market, then you would have to keep yourself updated with latest IT tools, and technologies. Following are the few sources:
1. Technical Forums over the internet.
2. Technical magazines on various IT subjects.
3. Technical Bulletin Boards
4. Conferences, Trainings and Workshops

Career Planning:

Today's professional life is very dynamic and to move along with it we need a proper career planning. When you start your career as a software developer, you really do not know how exactly you will perform in the industry, though you have confidence that whatever you do, will be done in the best way.
1. Do you want to continue as software developer forever, which could be a very good option and there are many people, who love coding forever.
2. If you are very good in designing software components and your past designs have been appreciated a lot, then you can think to go in technical side and become software architect.
3. If you are very good in managing things, have good command over people and have great convincing abilities, then you can think of going towards management role, which will start with leading a small team.

Career Advice For Software Professional & Students:

Key message for you :

“Work very hard! Those who work very hard for the first 3 years of their life enjoy the next 50 years of relaxation and those who spend the first 3 years in relaxation spend remaining years working hard.”

Looking forward to meeting you and having wonderful interactions. The Full Stack Java Developer, Pune Team is delighted to have you Welcome!

All the BEST & Good Luck with your interview!

FULL STACK JAVA DEVELOPER

Your Dream Job Closer Than You Think

Hello Mates,

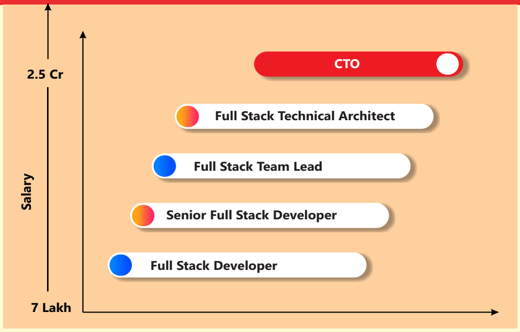
If you want to learn Java Realtime Live Projects then

JOIN US

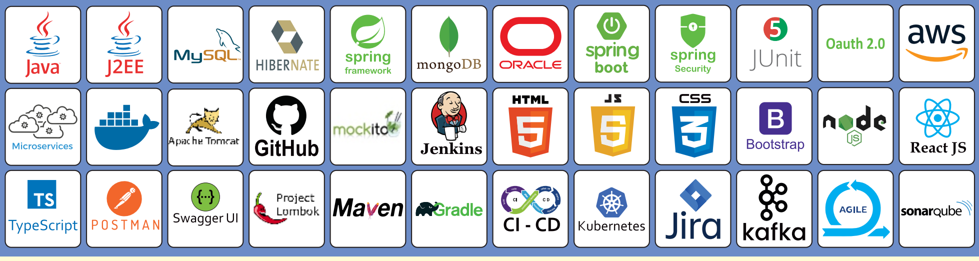
Start Building Rich, Functional Applications And Land a Full Stack Developer Jobs in Good MNC

Best Suited For Software Professionals & Students

We are delivering all sessions with real-time hands-on scenarios to help you understand in a better way.



Technologies & Tools Covered



Program highlights

- 20+ Industrial Projects
- Build real-world applications like Amazon, Walmart, Zomato
- Industrial Training by Industry Expert & Assistance with job Placement and Career Guidance
- Live Doubt Resolutions
- Interview Preparation & Mentoring
- Practical Experience through Real-World Projects
- Industry-relevant projects to develop a portfolio and gain practical experience
- 400% Salary Hike with Structured Program & dedicated support
- Access to an active community network of Full Stack Java Developers
- Industry relevant learning, designed for Fresh Graduates
- Guidance on best practices for Code Quality, Security & Scalability
- Personalized feedback from Experienced Instructor and Industry Experts
- Industry recognized certificate offered by Full Stack Java Developer
- Our Institute offers a fully-equipped software lab where you can get hands-on practice with the latest tools and technologies.
- 100% Placement Records with 5* Rated in Google

"Whether you're a beginner looking to get started with programming or an experienced developer looking to Upskill, our program has everything you need to succeed."

Please visit us at Pune Office: **Full Stack Java Developer Pune, Inspiria Mall, 4rth floor, Near Bhakti Shakti Chowk, Nigdi, Pimpri Chinchwad, Pune, MH, India- 411044.**

Contact: **+91 7887575991 | +91 9623639693 | +91 9356943970 | +91 9767186443 | +91 9689162909**

www.fullstackjavadeveloper.in