OOPS and CORE java

1. Keywords: final, static, this & super

static keyword –

- In order to share the same method or variable of any given class, the static keyword is used.
- It can be used for blocks, variables, methods, and nested classes.
- **Static variable –**
  - A static variable is nothing but the class variable.
  - It is common for all objects of a class, or let's say it is shared by all objects of a class, whereas the non-static variable is different for each object of a class.
  - E.g., static String name=" John";
- **Static block –**
  - To initialize static variables, a static block is used.
  - It gets executed only once after class is loaded or an object is created.
  - A class can contain multiple static blocks.
  - E.g. – static String name; static{ name = "Java"; }
- **Static method –**
  - A static method is one that can be invoked without a created object of a class.
  - It can access static variables without using the object of the class.
  - It can access static and non-static methods directly. A non-static method can access the static method, while a static method can access the non-static method only through the object of a class.
  - Eg – public static void main(String args[]) { }
- **Static class –**
  - A class can only be declared as static only if it is a nested class.
  - Non-static members of the Outer class are not accessible by the static class.
  - Eg – class Vehicle{ //Static class static class Car{ } }

this keyword –

- "this" is basically just a reference variable referring to the current object.
- It is used to eliminate the confusion between class attributes and parameters with the same name.
- It can be used to –
  - refer class variable
  - invoke class method or constructor
  - passed as an argument in method or constructor
  - return current class object
- E.g. –

    - this.name = name;
    - this.setName();
    - setName(this);
    - Person p = new Person(this);
    - return this;

super keyword –

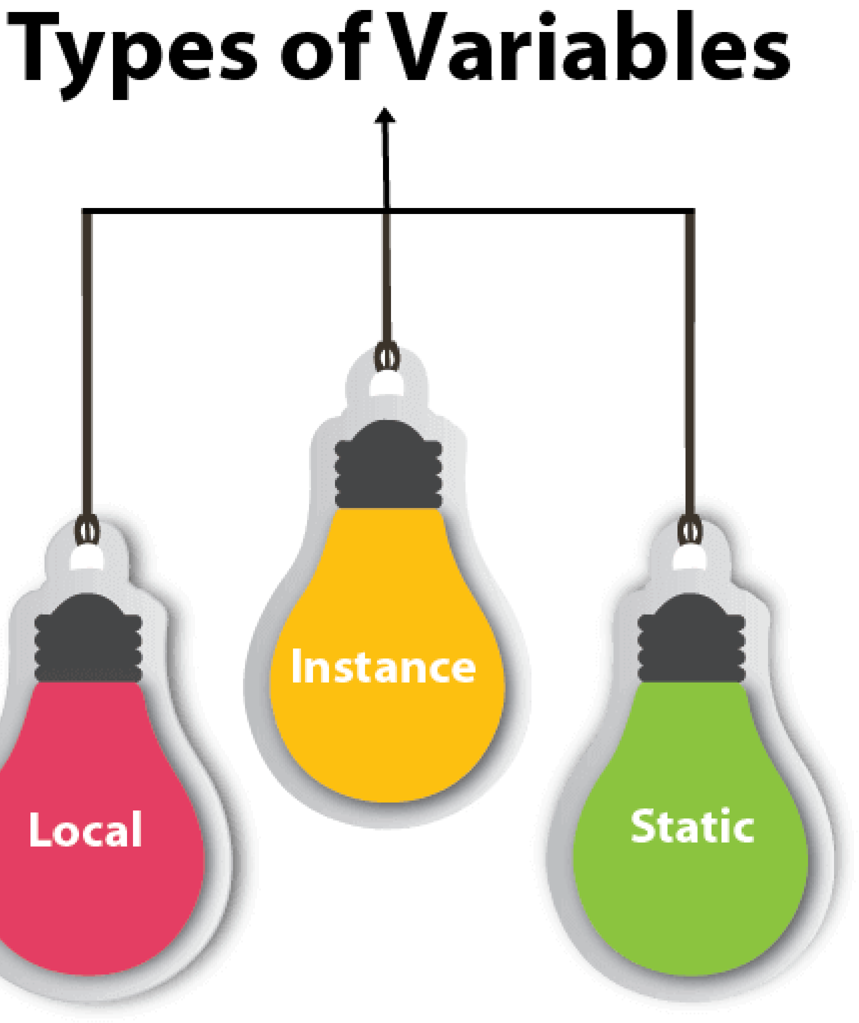
- The “super” keyword is a reference variable in java, used to refer parent class objects.
- The super keyword can be used in three ways –
    - **with variable –**
        - It is used when a derived class and base class has the same data members. There is a possibility of ambiguity in such a case.
        - This keyword can be used to refer to the immediate parent class instance variable.
        - Eg – System.out.println(super.speed);
    - **with method –**
        - It is used when we want to call the parent class method, and parent & child class have same-named methods; then, to resolve ambiguity, we use the super keyword to invoke the immediate parent class method.
        - E.g., super.message();
    - **with constructor –**
        - To invoke the immediate parent class constructor, it is used with a constructor.
        - It can call both parametric as well as non-parametric constructors depending upon the situation.
        - Call to super() must be the first statement in a derived class constructor.
        - Eg – Student() { super(); System.out.println(“constructor”); }

final keyword –

- To apply restrictions on user access, the final keyword is used.
- There are three ways to use the final keyword –
    - **final variable** –
        - Its value cannot be changed once initialized.
        - The compiler cannot assign a default value to a final variable.
        - Eg – final int count = 10;
    - **final method** –
        - A final method cannot be overridden, nor can it be hidden by a subclass.
        - Since the final method cannot be overridden so core functionalities must be declared.
        - Eg – final void manageCount() { //code }
    - **final class** –
        - It cannot be inherited.

- We cannot declare a final class as abstract at the same time as they are antonyms.
- Eg – final class Car { //methods and fields }

2. Local Variable, Static variable and Instance variable? And Differences.



1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

3. Difference between interface and abstract class?

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

4. Difference between creating String with & without new operator. Also check ==&.equals() difference.

With new operator String create the string in heap and put a copy in string const pool so the result of hashcode is same in below case;

```
  String s1 = new String("Test");
   String s2 = new String("Test");
   System.out.println(s1.hashCode() + " "+ s2.hashCode() + " " + s1.equals(s2));
```

But without using new operator its still giving the same hashcode

```
String s1 = new String("Test");
    String s2 = "Test";
    System.out.println(s1.hashCode() + " "+ s2.hashCode() + " " + s1.equals(s2));
```

Then what is the differnce between above two notation of string creation although they are referening to same string in string const. pool

String creation using new()	String creation using String literal
If we create a String using new(), then a new object is created in the heap memory even if that value is already present in the heap memory.	If we create a String using String literal and its value already exists in the string pool, then that String variable also points to that same value in the String pool without the creation of a new String with that value.
It takes more time for the execution and thus has lower performance than using String literal.	It takes less time for the execution and thus has better performance than using new().
Example:	Example:

```
String n1= new String("Java");          String s1="Java";
String n2= new String("Java");          String s2="Java";
String n3= new String("Create");        String s3="Create";
```

Both equals() method and the == operator are used to compare two objects in Java. == is an operator and equals() is method. But == operator compares reference or memory location of objects in a heap, whether they point to the same location or not.
Whenever we create an object using the operator new, it will create a new memory location for that object. So we use the == operator to check memory location or address of two objects are the same or not.

In general, both equals() and "==" operators in Java are used to compare objects to check equality, but here are some of the differences between the two:

1. The main difference between the .equals() method and == operator is that one is a method, and the other is the operator.
2. We can use == operators for reference comparison (**address comparison**) and .equals() method for **content comparison**. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.
3. If a class does not override the equals method, then by default, it uses the equals(Object o) method of the closest parent class that has overridden this method.

Difference between == and .equals() method in Java

S.No.	**== Operator**	**Equals() Method**
1.	== is considered an operator in Java.	Equals() is considered as a method in Java.
2.	It is majorly used to compare the reference values and objects.	It is used to compare the actual content of the object.
3.	We can use the == operator with objects and primitives.	We cannot use the equals method with primitives.
4.	The == operator can't compare conflicting objects, so at that time the compiler surrenders the compile-time error.	The equals() method can compare conflicting objects utilizing the equals() method and returns "false".
5.	== operator cannot be overridden.	Equals() method and can be overridden.

5. Difference between string, string builder and string buffer?

Java String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

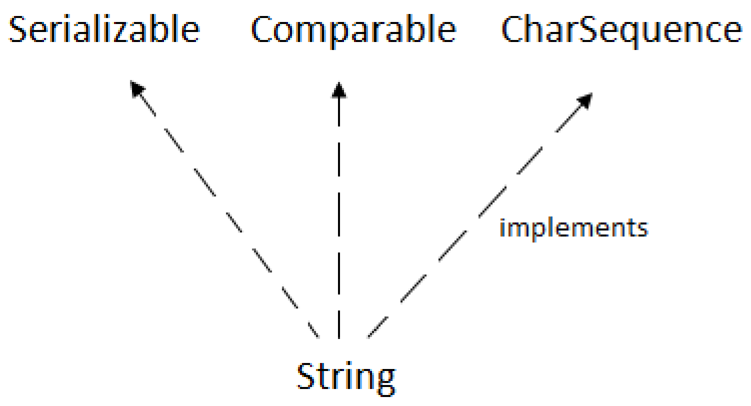
```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

```
1. String s="javatpoint";
```

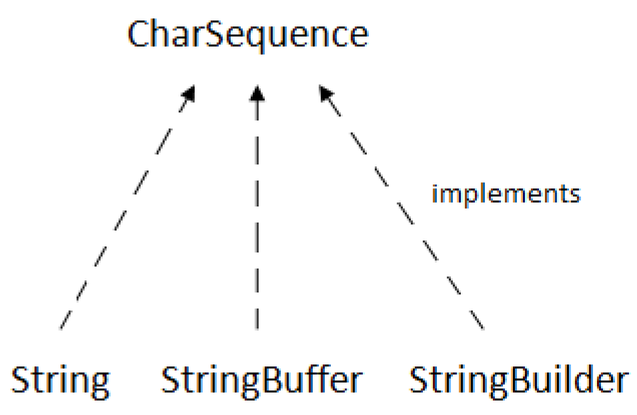
Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in Java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

What is a StringBuffer?

A StringBuffer is a companion class of strings that delivers the functionality of strings. When it comes to the string, it defines immutable character series and fixed-length, while StringBuffer defines growable character sequences.

This is to remember that the StringBuffer can include characters and substrings. To build a string buffer, an object needs to be constructed.

StringBuffer str = new StringBuffer();

What is StringBuilder?

In Java, StringBuilder illustrates a mutable sequence of characters. We know that the String class develops an immutable sequence of characters, so to help in that manner, the StringBuilder class provides an option to the String Class, as it constructs a mutable sequence.

StringBuilder str = new StringBuilder();

Difference between Stringbuffer and Stringbuilder

S.No.	**String Buffer**	**String Builder**
1	StringBuffer was introduced in Java 1.0	StringBuilder was launched in Java 5.

2	StringBuffer is synchronised	StringBuilder is not synchronised
3	Here we cannot access multiple threads at the same time because it is thread-safe.	It is not thread-safe.
4	It is slow as compared to the StringBuilder.	It is faster than StringBuffer.
5	StringBuffer is mutable, hence, we can make changes in a string without creating an object.	StringBuilder is also mutable.
6	Heap memory is used here.	Here also we prefer heap memory.

Exception Handling

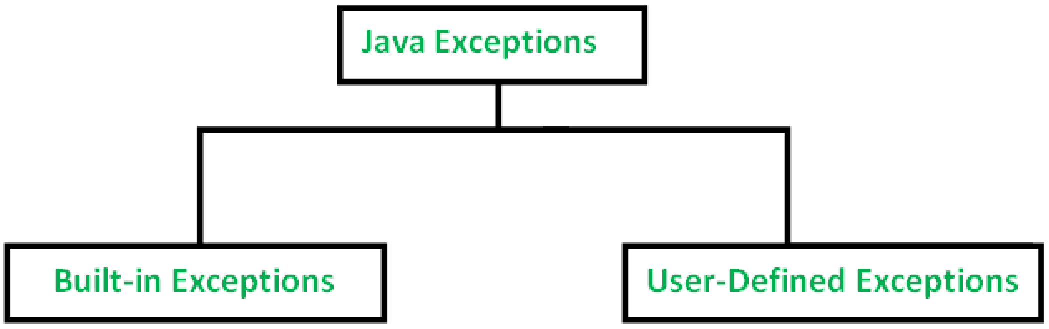
1. What is meant by exception and how to handle it?

What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

The try-catch is the simplest method of handling exceptions. Put the code you want to run in the try block, and any Java exceptions that the code throws are caught by one or more catch blocks. This method will catch any type of Java exceptions that get thrown. This is the simplest mechanism for handling exceptions.

2. What are the types of exceptions?



Built-in Exceptions:

Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1. **ArithmeticException:** It is thrown when an exceptional condition has occurred in an arithmetic operation.
2. **ArrayIndexOutOfBoundsException:** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
3. **ClassNotFoundException:** This Exception is raised when we try to access a class whose definition is not found
4. **FileNotFoundException:** This Exception is raised when a file is not accessible or does not open.
5. **IOException:** It is thrown when an input-output operation failed or interrupted
6. **InterruptedException:** It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.
7. **NoSuchFieldException:** It is thrown when a class does not contain the field (or variable) specified
8. **NoSuchMethodException:** It is thrown when accessing a method that is not found.
9. **NullPointerException:** This exception is raised when referring to the members of a null object. Null represents nothing
10. **NumberFormatException:** This exception is raised when a method could not convert a string into a numeric format.
11. **RuntimeException:** This represents an exception that occurs during runtime.
12. **StringIndexOutOfBoundsException:** It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string
13. **IllegalArgumentException :** This exception will throw the error or error statement when the method receives an argument which is not accurately fit to the given relation or condition. It comes under the unchecked exception.
14. **IllegalStateException :** This exception will throw an error or error message when the method is not accessed for the particular operation in the application. It comes under the unchecked exception.

Examples of Built-in Exception

A. Arithmetic exception

```
// Java program to demonstrate ArithmeticException
class ArithmeticException_Demo
{
   public static void main(String args[])
   {
      try {
         int a = 30, b = 0;
         int c = a/b;  // cannot divide by zero
         System.out.println ("Result = " + c);
      }
      catch(ArithmeticException e) {
         System.out.println ("Can't divide a number by 0");
      }
   }
}
```

Output

```
Can't divide a number by 0
```

User-Defined Exceptions

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, the user can also create exceptions which are called 'user-defined Exceptions'.

The following steps are followed for the creation of a user-defined Exception.

- The user should create an exception class as a subclass of the Exception class. Since all the exceptions are subclasses of the Exception class, the user should also make his class a subclass of it. This is done as:

```
class MyException extends Exception
```

- We can write a default constructor in his own exception class.

```
MyException(){}
```

3. Try, Catch and finally? Difference between Finally block and finalize()?

Try	We specify the block of code that might give rise to the exception in a special block with a "Try" keyword.
Catch	When the exception is raised it needs to be caught by the program. This is done using a "catch" keyword. So a catch block follows the try block that raises an exception. The keyword catch should always be used with a try.

Finally	Sometimes we have an important code in our program that needs to be executed irrespective of whether or not the exception is thrown. This code is placed in a special block starting with the “Finally” keyword. The Finally block follows the Try-catch block.
Throw	The keyword “throw” is used to throw the exception explicitly.
Throws	The keyword “Throws” does not throw an exception but is used to declare exceptions. This keyword is used to indicate that an exception might occur in the program or method.

Try Block In Java
Whenever we are writing a program there could be a code that we suspect might throw an exception. **For example,** we might suspect that there might be a “division by zero” operation in the code that will throw an exception.
This code that might raise an exception is enclosed in a block with the keyword “try”. So the try block contains the code or set of statements that can raise an exception.

The general syntax of the try block is as follows:

```
try{

            //set of statements that can raise exception

      }
```

Catch Block In Java
We use a catch block to handle exceptions. This is the block with the “catch” keyword. The catch block follows the try block.

Whenever an exception occurs in the try block, then the code in the catch block that corresponds to the exception is executed.

The general syntax of the catch block is:

```
catch (Exception e){

                //code to handle exception e

        }
```

Try-Catch Java
The general syntax of the try-catch block is shown below:

```
try{

                //code causing exception
```

```
        }

        catch (exception (exception_type) e (object))

        {

                //exception handling code

        }
```

The try block can have multiple lines of code that can raise multiple exceptions. Each of these exceptions is handled by an independent catch block.

Finally Block In Java

The finally block in Java is usually put after a try or catch block. Note that the finally block cannot exist without a try block. When the finally block is included with try-catch, it becomes a "**try-catch-finally**" block.

The general syntax of the finally block is as follows:

```
try {

</em><em>                //code that might raise exception

</em><em>        }catch {

</em><em>               //code that handles exception

</em><em>        }finally {

</em><em>                //mandatory code to be executed

</em><em>        }
```

Final vs Finally vs Finalize

Sr. no.	Key	final	finally	finalize
1.	Definition	final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.

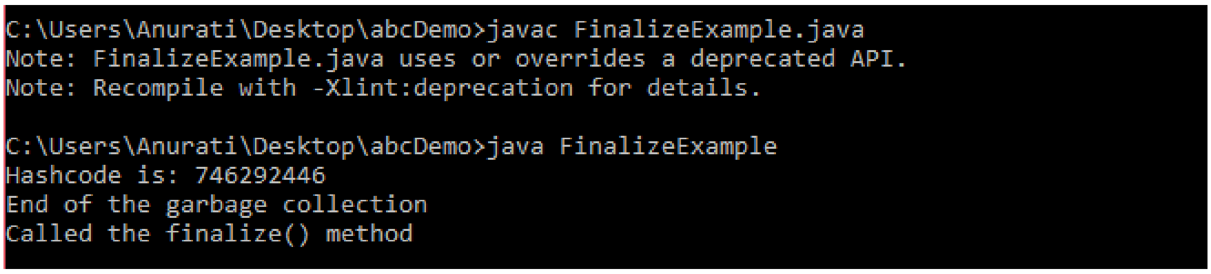
2.	Applicable to	Final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	finalize() method is used with the objects.
3.	Functionality	(1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.
4.	Execution	Final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. It's execution is not dependant on the exception.	finalize method is executed just before the object is destroyed.

Java finalize Example

FinalizeExample.java

```java
public class FinalizeExample {
     public static void main(String[] args)
    {
        FinalizeExample obj = new FinalizeExample();
        // printing the hashcode
        System.out.println("Hashcode is: " + obj.hashCode());
        obj = null;
        // calling the garbage collector using gc()
        System.gc();
        System.out.println("End of the garbage collection");
    }
   // defining the finalize method
    protected void finalize()
    {
        System.out.println("Called the finalize() method");
    }
}
```

Output:


```
C:\Users\Anurati\Desktop\abcDemo>javac FinalizeExample.java
Note: FinalizeExample.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\Anurati\Desktop\abcDemo>java FinalizeExample
Hashcode is: 746292446
End of the garbage collection
Called the finalize() method
```

4. Difference between Runtime and Compiletime exception in java? And custom exception?

Compile time vs Runtime

Compile-time and Runtime are the two programming terms used in the software development. Compile-time is the time at which the source code is converted into an executable code while the run time is the time at which the executable code is started running. Both the compile-time and runtime refer to different types of error.

Compile-time errors

Compile-time errors are the errors that occurred when we write the wrong syntax. If we write the wrong syntax or semantics of any programming language, then the compile-time errors will be thrown by the compiler. The compiler will not allow to run the program until all the errors are removed from the program. When all the errors are removed from the program, then the compiler will generate the executable file.

The compile-time errors can be:

- Syntax errors
- Semantic errors

Runtime errors

The runtime errors are the errors that occur during the execution and after compilation. The examples of runtime errors are division by zero, etc. These errors are not easy to detect as the compiler does not point to these errors.

Let's look at the differences between compile-time and runtime:

Compile-time	**Runtime**
The compile-time errors are the errors which are produced at the compile-time, and they are detected by the compiler.	The runtime errors are the errors which are not generated by the compiler and produce an unpredictable result at the execution time.
In this case, the compiler prevents the code from execution if it detects an error in the program.	In this case, the compiler does not detect the error, so it cannot prevent the code from the execution.
It contains the syntax and semantic errors such as missing semicolon at the end of the statement.	It contains the errors such as division by zero, determining the square root of a negative number

Custom exception:

Java Custom Exception

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

Consider the example 1 in which InvalidAgeException class extends the Exception class.

Using the custom exception, we can have your own exception and message. Here, we have passed a string to the constructor of superclass i.e. Exception class that can be obtained using getMessage() method on the object we have created.

Following are few of the reasons to use custom exceptions:

- To catch and provide specific treatment to a subset of existing Java exceptions.
- Business logic exceptions: These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.

In order to create custom exception, we need to extend Exception class that belongs to java.lang package.

Consider the following example, where we create a custom exception named WrongFileNameException:

```java
public class WrongFileNameException extends Exception {
  public WrongFileNameException(String errorMessage) {
  super(errorMessage);
  }
}

```

5. How exception handling done in recent project?

1. Use try/catch/finally blocks to recover from errors or release resources.
2. Handle common conditions without throwing exceptions.
3. Design classes so that exceptions can be avoided.
4. Throw exceptions instead of returning an error code.

Thread

1. What is thread? Multi-threading and explain thread life cycle in java?

Java Threads

Threads allows a program to operate more efficiently by doing multiple things at the same time.

Threads can be used to perform complicated tasks in the background without interrupting the main program.

Creating a Thread

There are two ways to create a thread.

It can be created by extending the Thread class and overriding its run() method:

Extend Syntax

```
public class Main extends Thread {

  public void run() {

    System.out.println("This code is running in a thread");

  }

}
```

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

1. Extending the Thread class
2. Implementing the Runnable Interface

Thread creation by extending the Thread class
We create a class that extends the **java.lang.Thread** class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

Thread creation by implementing the Runnable Interface
We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

2. How do you make a thread in java? Difference between start() and run() method of thread class?

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Starting a thread:

The **start() method** of Thread class is used to start a newly created thread. It performs the following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

1) Java Thread Example by extending Thread class

FileName: Multi.java

```java
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

Output:

```
thread is running...
```

2) Java Thread Example by implementing Runnable interface

FileName: Multi3.java

```java
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

Output:

```
thread is running...
```

3. Difference between creating thread using thread class and runnable? When to use runnable interface vs thread class in java?

When we extend Thread class, each of our thread creates unique object and associate with it. When we implements Runnable, it shares the same object to multiple threads.

Sr. No.	Key	Thread	Runnable
1	Basic	Thread is a class. It is used to create a thread	Runnable is a functional interface which is used to create a thread

Sr. No.	Key	Thread	Runnable
2	Methods	It has multiple methods including start() and run()	It has only abstract method run()
3		Each thread creates a unique object and gets associated with it	Multiple threads share the same objects.
4	Memory	More memory required	Less memory required
5	Limitation	Multiple Inheritance is not allowed in java hence after a class extends Thread class, it can not extend any other class	If a class is implementing the runnable interface then your class can extend another class.

4. Difference between notify() and notifyAll() method in java?

The **notify()** and **notifyAll()** methods with wait() methods are used for communication between the threads. A thread that goes into waiting for state by calling the wait() method will be in waiting for the state until any other thread calls either notify() or notifyAll() method on the same object.

notify(): The notify() method is defined in the Object class, which is Java's top-level class. It's used to wake up only one thread that's waiting for an object, and that thread then begins execution. The thread class notify() method is used to wake up a single thread.

notifyAll(): The notifyAll() wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the wait methods. The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object.

Sr. No.	Key	notify	notifyAll
1	Notification	In case of multiThreading notify() method sends the notification to only one thread among the multiple waiting threads which are waiting for lock.	While notifyAll() methods in the same context sends the notification to all waiting threads instead of single one thread.
2	Thread identification	As in case of notify the notification is sent to single thread among the multiple waiting threads so it is sure that which of those waiting thread is going to receive the lock.	On other hand notifyAll sends notification to all waiting threads hence it is not clear which of the thread is going to receive the lock.
3	Risk factor	In case of notify() method the risk of thread missing is high as notification is sent only single thread and if it misses that than no other thread would get notification and hence the lock.	While in case of notifyAll as notification is to all the waiting threads and hence if any thread misses the notification, there are other threads to do the job.Hence risk is less.
4	Performance	Memory and CPU drain is less as compare to notifyAll as notification is sent to single one thread so performance is better as compare to notifyAll.	On other hand as the cost of no notification is dropped and notification is sent to all waiting threads the memory and CPU drain is more as compare to notify and hence performance of notifyAll is lesser.
5	Interchangeable	In case of the notify() method as only single one thread is in picture hence no concept of thread Interchangeable is possible.	While we should go for notifyAll() if all your waiting threads are interchangeable (the order they wake up doesnâTMt matter).

5. How to stop a thread in java? Explain about sleep () method in a thread?

Whenever we want to stop a thread from running state by calling **stop()** method of **Thread** class in Java.This method stops the execution of a running thread and removes it from the waiting threads pool and garbage collected. A thread will also move to the dead state automatically when it reaches the end of its method. The **stop()** method is **deprecated** in Java due to **thread-safety** issues.

Syntax

```
@Deprecated
public final void stop()
```

Thread Class is a class that is basically a thread of execution of the programs. It is present in Java.lang package. Thread class contains the **Sleep()** method. There are two overloaded methods of Sleep() method present in Thread Class, one is with one argument and another one is with two arguments. The sleep() method is used to stop the execution of the current thread(whichever might be executing in the system) for a specific duration of the time and after that time duration gets over, the thread which is executing earlier starts to execute again.

Important Point Regarding Thread.sleep() Method:

- Method Whenever Thread.sleep() functions to execute, it always pauses the current thread execution.
- If any other thread interrupts when the thread is sleeping, then InterruptedException will be thrown.
- If the system is busy, then the actual time the thread will sleep will be more as compared to that passed while calling the sleep method and if the system has less load, then the actual sleep time of the thread will be close to that passed while calling sleep() method.

Syntax of Sleep() Method

1. **public static void** sleep(**long** mls) **throws** InterruptedException
2. **public static void** sleep(**long** mls, **int** n) **throws** InterruptedException

6. What is Synchronization? What is the disadvantage of Synchronization?

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Advantage: Using Synchronized keyword it reduces problem of data inconsistency.
Disadvantage: **It increase waiting time of thread**. At a time only one thread can operate on object so other threads have to wait. So it creates performance problem.

7. Difference between Synchronized block and Synchronized Method?

Synchronized Method & Synchronized Block

Synchronization is the ability to control the access of multiple threads to share resources. Without synchronization, it is possible for one thread to modify a shared resource while another thread is in the process of using or updating that resource.

There are two synchronization syntax in Java Language. The practical differences are in controlling scope and the monitor. With a synchronized method, the lock is obtained for the duration of the entire method. With synchronized blocks you can specify exactly when the lock is needed

The **main difference between synchronized block and synchronized method is that synchronized block locks the code within the block whereas synchronized method locks the entire object.**

8. What is deadlock and how to avoid it.?

Deadlock is a scenario where a set of processes is blocked because each process has acquired a lock on a particular resource and is waiting for another resource locked by some other process.

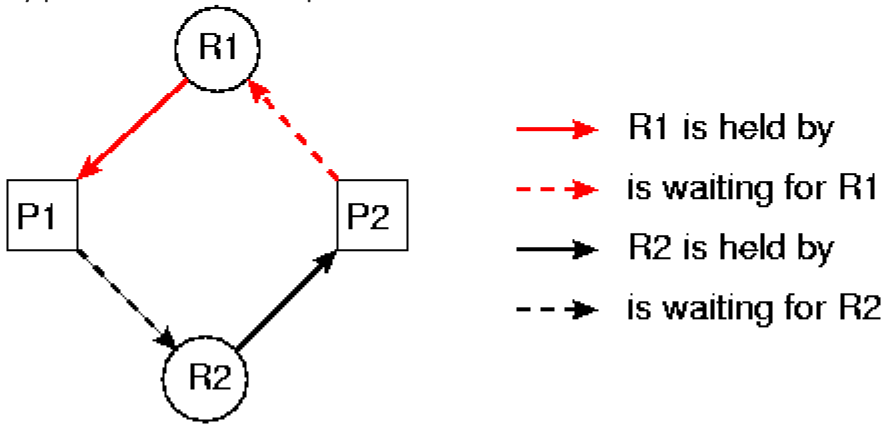
Let me give you an example:

Suppose process1 is making a transaction from one account (R1) to another account (R2), and for that, it will try to acquire a lock on resource R1 and R2.

At the same time, process2 is trying to transfer funds from one account (R2) to another account (R1), and for that, it will try to acquire a lock on resource R2 and R1.

In this scenario, process1 will acquire the lock on resource R1, and process2 will acquire the lock on resource R2, and they will keep on waiting for each other to release the resource.

E.g. Process1 will keep on waiting to acquire a lock on resource R2, which is locked by process2 and process2 will keep on waiting to acquire a lock on resource R1 which is locked by process1 and ends up in deadlock.

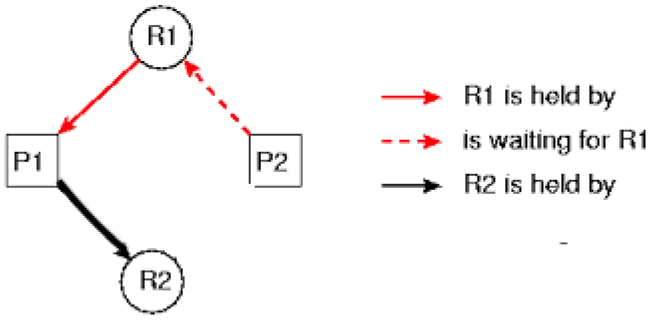


How to avoid Deadlock?

In order to avoid deadlock, you have to acquire a lock in the fixed order.

Let me explain by resolving the above deadlock.

If process1 gets the lock on resource R1 and then R2, at the same time, process2 also tries to get the lock on resources in the same order as process1, i.e. On resource R1 and then R2 instead of R2 and then R1.



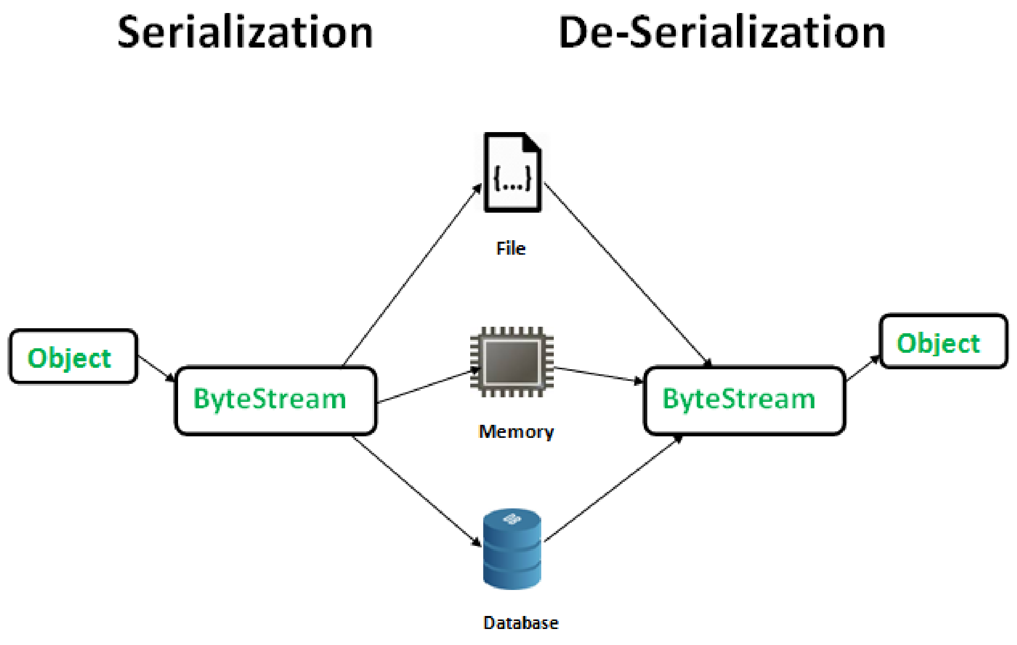
In this case, process2 has to wait for process1 to finish the transaction.

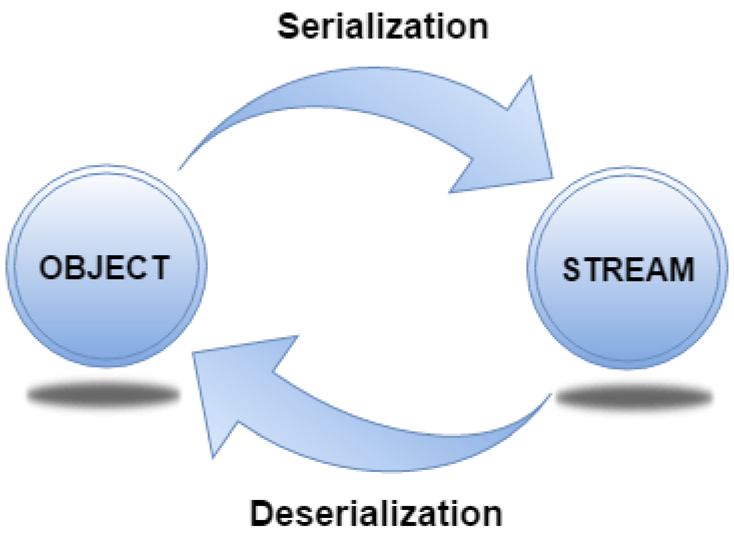
Once process1 commits the transaction successfully, it will release the locks on the resources; therefore process 2 will get the required resources in order to complete the transaction successfully without getting into the deadlock.

Therefore, please make sure you acquire the lock in the fixed order throughout your application.

9. What is meant by Serialization?Difference between Serialization and Deserialization in Java.

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.





java.io.Serializable interface

Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The **Cloneable** and **Remote** are also marker interfaces.

The **Serializable** interface must be implemented by the class whose object needs to be persisted.

The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

Let's see the example given below:

Student.java

```
import java.io.Serializable;
public class Student implements Serializable{
 int id;
 String name;
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }
}
```

In the above example, ***Student*** class implements Serializable interface. Now its objects can be converted into stream. The main class implementation of is showed in the next code.

Example of Java Serialization

In this example, we are going to serialize the object of ***Student*** class from above code. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.

Persist.java

```java
import java.io.*;
class Persist{
 public static void main(String args[]){
  try{
  //Creating the object
  Student s1 =new Student(211,"ravi");
  //Creating stream and writing the object
  FileOutputStream fout=new FileOutputStream("f.txt");
  ObjectOutputStream out=new ObjectOutputStream(fout);
  out.writeObject(s1);
  out.flush();
  //closing the stream
  out.close();
  System.out.println("success");
  }catch(Exception e){System.out.println(e);}
 }
}
```

10. What is the purpose of a transient variable?

Transient variable in Java

A **transient** variable is a special type of variable which we create by using the **transient** keyword. It is a special type of variable which have a non-serialized value at the time of serialization. A variable that is initialized by its default value during de-serialization is known as a transient variable.

A transient variable plays an important role in preventing an object from being serialized. We can make any variable transient by using the **transient** keyword.

Usually, the interviewer asks the difference between the **volatile** and the **transient** variable. So, the **volatile** and the **transient** variables are both different from each other. The transient keyword is mainly used at the time of serializing an object, while the **volatile** keyword is related to the visibility of variables that are modified by multiple threads at the time of concurrent programming.

Both the volatile and transient keywords are rarely used by the programmer and not so much popular as compared to **public, static** or **final**.

11. Which methods are used during Serialization and Deserialization process?

The ObjectOutputStream class contains **writeObject() method for serializing an Object. The ObjectInputStream class contains readObject() method for deserializing an object**.

Method	**Description**
1) public final void writeObject(Object obj) throws IOException {}	It writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	It flushes the current output stream.
3) public void close() throws IOException {}	It closes the current output stream.

Method	**Description**
1) public final Object readObject() throws IOException, ClassNotFoundException{}	It reads an object from the input stream.
2) public void close() throws IOException {}	It closes ObjectInputStream.

12. What is the purpose of a Volatile Variable?

Volatile Keyword in Java

Volatile keyword is used to modify the value of a variable by different threads. It is also used to make classes thread safe. It means that multiple threads can use a method and instance of the classes at the same time without any problem. The volatile keyword can be used either with primitive type or objects.

The volatile keyword does not cache the value of the variable and always read the variable from the main memory. The volatile keyword cannot be used with classes or methods. However, it is used with variables. It also guarantees visibility and ordering. It prevents the compiler from the reordering of code.

The contents of the particular device register could change at any time, so you need the volatile keyword to ensure that such accesses are not optimized away by the compiler.

Example

```
class Test
{
static int var=5;
}
```

When to use it?

- You can use a volatile variable if you want to read and write long and double variable automatically.
- It can be used as an alternative way of achieving synchronization in Java.
- All reader threads will see the updated value of the volatile variable after completing the write operation. If you are not using the volatile keyword, different reader thread may see different values.
- It is used to inform the compiler that multiple threads will access a particular statement. It prevents the compiler from doing any reordering or any optimization.
- If you do not use volatile variable compiler can reorder the code, free to write in cache value of volatile variable instead of reading from the main memory.

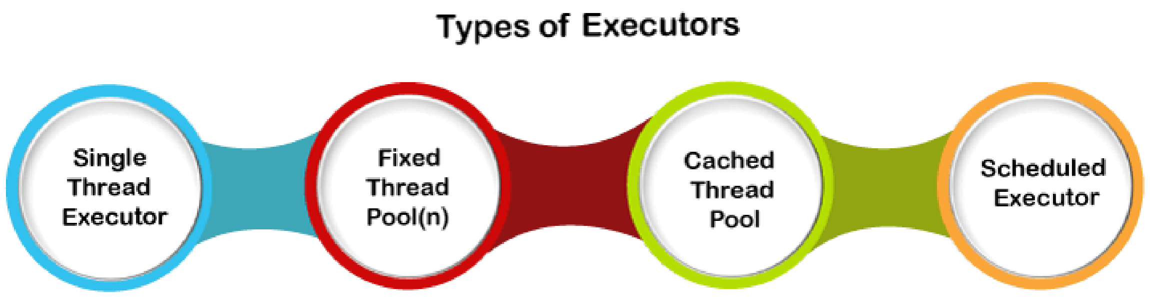
13. Thread Executor framwork If candidate knows else ignore.

Executor Framework Java

A framework having a bunch of components that are used for managing worker threads efficiently is referred to as **Executor Framework**. The Executor API reduces the execution of the task from the actual task to be executed through the **Executors**. The executor framework is an implementation of the **Producer-Consumer** pattern.

The **java.util.concurrent.Executors** class provides a set of methods for creating **ThreadPools** of worker threads.

In order to use the executor framework, we have to create a thread pool for executing the task by submitting that task to that thread pool.



1) SingleThreadExecutor

The **SingleThreadExecutor** is a special type of executor that has only a single thread. It is used when we need to execute tasks in sequential order. In case when a thread dies due to some error or exception at the time of executing a task, a new thread is created, and all the subsequent tasks execute in that new one.

1. ExecutorService executor = Executors.newSingleThreadExecutor()

2) FixedThreadPool(n)

FixedThreadPool is another special type of executor that is a thread pool having a fixed number of threads. By this executor, the submitted task is executed by the n thread. In case when we need to execute more tasks after submitting previous tasks, they store in the **LinkedBlockingQueue** until previous tasks are not completed. The n denotes the total number of thread which are supported by the underlying processor.

1. ExecutorService executor = Executors.newFixedThreadPool(4);

3) CachedThreadPool

The **CachedThreadPool** is a special type of thread pool that is used to execute short-lived parallel tasks. The cached thread pool doesn't have a fixed number of threads. When a new task comes at a time when all the threads are busy in executing some other tasks, a new thread creates by the pool and add to the executor. When a thread becomes free, it carries out the execution of the new tasks. Threads are terminated and removed from the cached when they remain idle for sixty seconds.

1. ExecutorService executor = Executors.newCachedThreadPool();

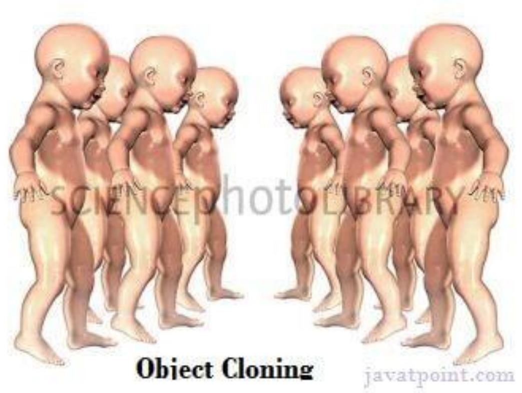
4) ScheduledExecutor

The **ScheduledExecutor** is another special type of executor which we use to run a certain task at regular intervals. It is also used when we need to delay a certain task.

1. ScheduledExecutorService scheduledExecService = Executors.newScheduledThreadPool(1);

14. what is Cloning and How to implement?

Object Cloning in Java



The **object cloning** is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.

The **clone() method** is defined in the Object class. Syntax of the clone() method is as follows:

1. **protected** Object clone() **throws** CloneNotSupportedException

Why use clone() method ?

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we use object cloning.

Advantage of Object cloning

Although Object.clone() has some design issues but it is still a popular and easy way of copying objects. Following is a list of advantages of using clone() method:

- You don't need to write lengthy and repetitive codes. Just use an abstract class with a 4- or 5-line long clone() method.
- It is the easiest and most efficient way for copying objects, especially if we are applying it to an already developed or an old project. Just define a parent class, implement Cloneable in it, provide the definition of the clone() method and the task will be done.

- Clone() is the fastest way to copy array.

15. Multi threading scenario based quesations

https://www.interviewbit.com/multithreading-interview-questions/

16. Difference between sleep and wait?

Sleep(): This Method is used to pause the execution of current thread for a specified time in Milliseconds. Here, Thread does not lose its ownership of the monitor and resume's it's execution
Wait(): This method is defined in object class. It tells the calling thread (a.k.a Current Thread) to wait until another thread invoke's the notify() or notifyAll() method for this object, The thread waits until it reobtains the ownership of the monitor and Resume's Execution.

Wait()	Sleep()
Wait() method belongs to Object class.	Sleep() method belongs to Thread class.
Wait() method releases lock during Synchronization.	Sleep() method does not release the lock on object during Synchronization.
Wait() should be called only from Synchronized context.	There is no need to call sleep() from Synchronized context.
Wait() is not a static method.	Sleep() is a static method.
Wait() Has Three Overloaded Methods: • wait() • wait(long timeout) • wait(long timeout, int nanos)	Sleep() Has Two Overloaded Methods:

17. What is Daemon thread?

Daemon thread in Java is a low-priority thread that runs in the background to perform tasks such as garbage collection. Daemon thread in Java is also a service provider thread that provides services to the user thread. Its life depends on the mercy of user threads i.e. when all the user threads die, JVM terminates this thread automatically.

In simple words, we can say that it provides services to user threads for background supporting tasks. It has no role in life other than to serve user threads.

Example of Daemon Thread in Java: Garbage collection in Java (gc), finalizer, etc.
Properties of Java Daemon Thread

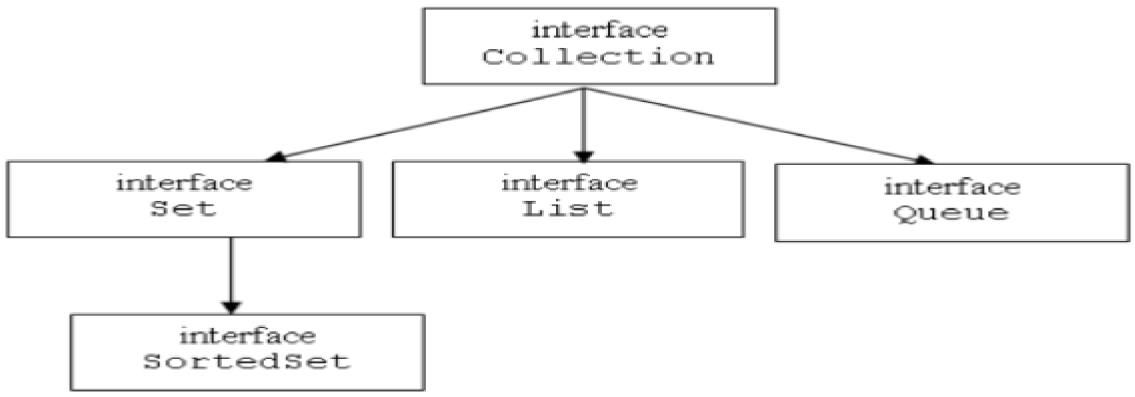
- They can not prevent the JVM from exiting when all the user threads finish their execution.
- JVM terminates itself when all user threads finish their execution.
- If JVM finds a running daemon thread, it terminates the thread and, after that, shutdown it. JVM does not care whether the Daemon thread is running or not.
- It is an utmost low priority thread.

18. How two thread shares common data ?

All static and controlled data is shared between threads. All other data can also be shared through arguments/parameters and through based references, as long as the data is allocated and is not freed until all of the threads have finished using the data.

Collections

1. List,Set & Map differes? And their purpose?



List

A List in java extends the collection interface and represent an sequenced or ordered group of elements. It can contain duplicate elements. It also defines some additional methods which it inherits from Collection interface.

Note: Elements in List can be inserted, updated, or retrieved by their position or index. Index or position value starts from 0.

Example

```
import java.util.*;
public class Main {
  public static void main(String args[]){
    //Create List object
    List<String> mySubjects = new ArrayList<>();

    //Add elements to list
    mySubjects.add("Java");
    mySubjects.add("Spring");
    mySubjects.add("Hibernate");


    //Add elements at specified position
    mySubjects.add(1,"SQL");
    mySubjects.add(2,"Oracle");

    System.out.println("My Subjects:");
    //Print all subjects
    for(String subject : mySubjects){
     System.out.println(subject);
    }

    //Print element on 2nd index
    System.out.println("Element at 2nd index: "+mySubjects.get(2));
  }
}
```

Output

```
My Subjects:
Java
SQL
Oracle
Spring
Hibernate
Element at 2nd index: Oracle
```

Set

A set represents a group or collection of items. Set has a special property that is unique items, it can not contain a duplicate item or element. It extends the collection interface.

Note: Set interface does not have any additional method other than methods inherited from Collection interface. With all collection interface methods it adds the restriction that it can not contain a duplicate elements.

Example

```
import java.util.*;
public class Main {

    public static void main(String args[]){
        //Create Set object
        Set<String> mySubjects = new HashSet<>();

        //Add elements to Set
        mySubjects.add("Java");
        mySubjects.add("Spring");
        mySubjects.add("Hibernate");

        System.out.println("My Subjects:");
        //Print all subjects
        for(String subject : mySubjects){
         System.out.println(subject);
        }

    }
}
```

Output

```
My Subjects:
Java
Hibernate
Spring
```

Map

A map in java, not extends the Collection interface. It represents a group of special elements or objects. Every map element or object contains key and value pair. A map can’t contain duplicate keys and one key can refer to at most one value.

Example

```
import java.util.*;
public class Main {

    public static void main(String args[]){
        Map<Integer,String> mysubjects = new HashMap<Integer,String>();

        //Add elements to map
        mysubjects.put(1,"Java");
        mysubjects.put(2,"Spring");
        mysubjects.put(3,"Oracle");
```

```
      //Print map elements in key value form
      for(Map.Entry subject : mysubjects.entrySet())
        System.out.println(subject.getKey()+" - "+subject.getValue());

   }
}
```
Output

```
1 - Java
2 - Spring
3 - Oracle
```
When to use List, Set and Map in Java?

Use of a data structure or collection is depends upon the requirement.

- **Use Set:** If you need group of unique elements.
- **Use List:** If get operations are higher than any other operation.
- **Use Map:** If objects contains the key and value pair.

2. Equals() & Hashcode() uses?how to overriede?

Equals() and Hashcode() in Java

The equals() and hashcode() are the two important methods provided by the **Object** class for comparing objects. Since the Object class is the parent class for all Java objects, hence all objects inherit the default implementation of these two methods. In this topic, we will see the detailed description of equals() and hashcode() methods, how they are related to each other, and how we can implement these two methods in Java.

Java equals()

- The java equals() is a method of *lang.Object* class, and it is used to compare two objects.
- To compare two objects that whether they are the same, it compares the values of both the object's attributes.
- By default, two objects will be the same only if stored in the same memory location.

Syntax:

```
1. public boolean equals(Object obj)
```

Parameter:

obj: It takes the reference object as the parameter, with which we need to make the comparison.

Returns:

It returns the true if both the objects are the same, else returns false.

General Contract of equals() method

There are some general principles defined by Java SE that must be followed while implementing the equals() method in Java. The equals() method must be:

- ***reflexive***: An object x must be equal to itself, which means, for object x, **equals(x)** should return true.
- ***symmetric***: for two given objects x and y, *x.equals(y)* must return true if and only if *equals(x)* returns true.
- ***transitive***: for any objects x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
- ***consistent***: for any objects x and y, the value of x.equals(y) should change, only if the property in equals() changes.
- For any object x, the *equals(null)* must return false.

Java hashcode()

- A **hashcode** is an integer value associated with every object in Java, facilitating the hashing in hash tables.
- To get this hashcode value for an object, we can use the hashcode() method in Java. It is the means ***hashcode() method that returns the integer hashcode value of the given object***.
- Since this method is defined in the Object class, hence it is inherited by user-defined classes also.
- The hashcode() method returns the same hash value when called on two objects, which are equal according to the equals() method. And if the objects are unequal, it usually returns different hash values.

Syntax:

```
1. public int hashCode()
```

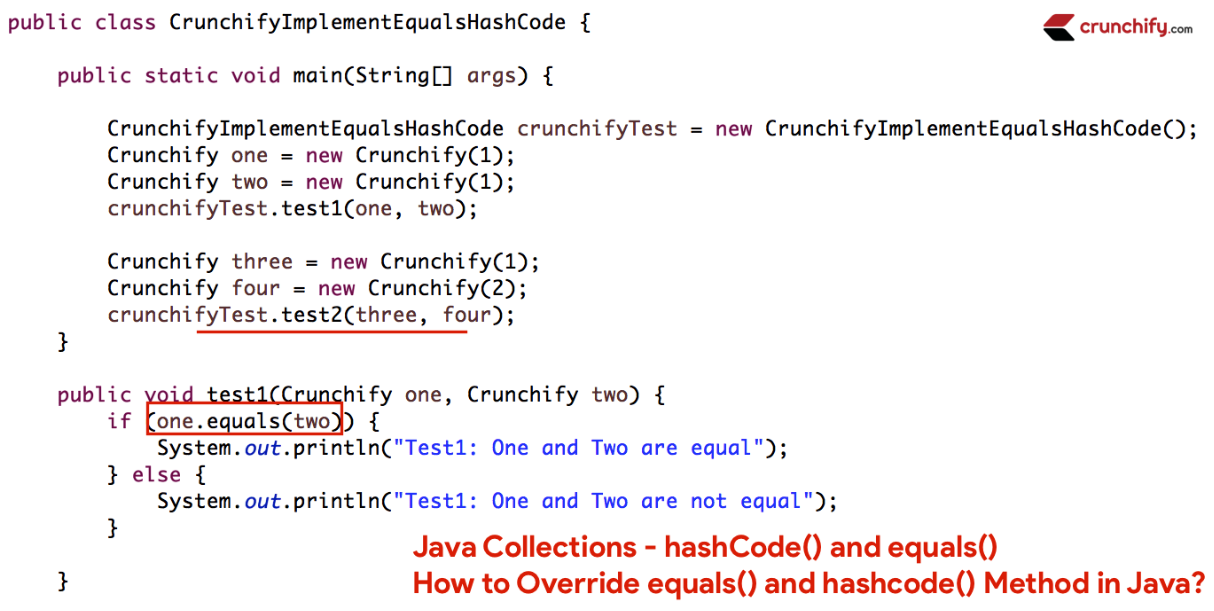
Returns:

It returns the hash code value for the given objects.

Contract for hashcode() method in Java

- If two objects are the same as per the equals(Object) method, then if we call the hashCode() method on each of the two objects, it must provide the same integer result.

Why to Override equals(Object) and hashCode() method ?
if a class overrides equals, it must override hashCode. **when they are both overridden, equals and hashCode must use the same set of fields**. if two objects are equal, then their hashCode values must be equal as well. if the object is immutable, then hashCode is a candidate for caching and lazy initialization


crunchify.com
Java Collections - hashCode() and equals()
How to Override equals() and hashcode() Method in Java?

```
public class CrunchifyImplementEqualsHashCode {

    public static void main(String[] args) {

        CrunchifyImplementEqualsHashCode crunchifyTest = new CrunchifyImplementEqualsHashCode();
        Crunchify one = new Crunchify(1);
        Crunchify two = new Crunchify(1);
        crunchifyTest.test1(one, two);

        Crunchify three = new Crunchify(1);
        Crunchify four = new Crunchify(2);
        crunchifyTest.test2(three, four);
    }

    public void test1(Crunchify one, Crunchify two) {
        if (one.equals(two)) {
            System.out.println("Test1: One and Two are equal");
        } else {
            System.out.println("Test1: One and Two are not equal");
        }

    }
```

3. What is the benefit of Generics in Collections Framework?

The generic collections are type-safe and checked at compile-time. These generic collections **allow the datatypes to pass as parameters to classes**. The Compiler is responsible for checking the compatibility of the types

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.

Before generics, we can store any type of objects in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety: We can hold only a single type of objects in generics. It doesn?t allow to store other objects.

Without Generics, we can store any type of objects.

```
List list = new ArrayList();
list.add(10);
list.add("10");
```

With Generics, it is required to specify the type of object we need to store.

```
List<Integer> list = new ArrayList<Integer>();
list.add(10);
list.add("10");// compile-time error
```

2) Type casting is not required: There is no need to typecast the object.

Before Generics, we need to type cast.

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);//typecasting
```

After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

3) Compile-Time Checking: It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32);//Compile Time Error
```

Syntax to use generic collection

```
ClassOrInterface<Type>
```

Example to use Generics in java

```
ArrayList<String>
```

4. Difference between Array and Array List?

In Computer Science, an array is a group of the same data types having contiguous memory locations. Array in java is an object that includes elements of a sole data type.

An ArrayList class in Java is known as a resizable array. Unlink Array, ArrayList can change the length after being created.

Basis of Comparison	Array	Array List
Definition	A straightforward data structure with a continuous memory location, an array stores its contents with the same name but distinct index numbers for each element of the array it contains. It is imperative that all of the data stored in an array be of the same type. After an array has been declared, its size cannot be changed.	The Java collection framework contains a data structure known as an ArrayList, which is dynamic in nature. Additionally, it has components that are of the same type. In this case, it is not necessary for us to specify the length of the list.
Static/Dynamic	Arrays are static	ArrayList is dynamic
Resizable	Fixed Length	Can be Resizable
Initialization	When performing initialization for an array, it is required to specify the size of the array.	It is not necessary to mention the size of an ArrayList.
Performance	Arrays are faster	ArrayList is slower
Generic Type	An array can store primitive data as well as objects, but it cannot store generics.	ArrayList is able to store generics as well as objects, but it cannot store data of primitive types.
Iteration	Only loops are permitted in this area.	It is acceptable to use loops and iterators.
Type Safety	It is not type-safe.	It is type-safe.
Length	Makes use of the length object	Make use of size() function
Adding Elements	The additions are done with the assignment operator.	ArrayList uses add() method for performing additions
Single/Multi-Dimensional	Single and multiple dimensions are also a possibility.	You are only permitted to use single dimension.

5. Difference between HashSet and TreeSet?

Hash set and tree set both belong to the collection framework. HashSet is the implementation of the Set interface whereas Tree set implements sorted set. Tree set is backed by TreeMap while HashSet is backed by a hashmap.

Sr. No.	Key	Hash Set	Tree Set
1	Implementation	Hash set is implemented using HashTable	The tree set is implemented using a tree structure.
2	Null Object	HashSet allows a null object	The tree set does not allow the null object. It throws the null pointer exception.
3	Methods	Hash set use equals method to compare two objects	Tree set use compare method for comparing two objects.
4	Heterogeneous object	Hash set doesn't now allow a heterogeneous object	Tree set allows a heterogeneous object
5	Ordering	HashSet does not maintain any order	TreeSet maintains an object in sorted order

6. Difference between HashMap,Concurrent Hashmap and HashTable?

The HashMap class of the Java collections framework **provides the functionality of the hash table data structure**. It stores elements in key/value pairs. Here, keys are unique identifiers used to associate each value on a map. The HashMap class implements the Map interface.

The **ConcurrentHashMap** class is introduced in JDK 1.5 belongs to **java.util.concurrent** package, which implements ConcurrentMap as well as to Serializable interface also. 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.

A concurrent hash table (concurrent hash map) is **an implementation of hash tables allowing concurrent access by multiple threads using a hash function**.

Synchronized HashMap Vs HashTable Vs ConcurrentHashMap In Java :

	Synchronized HashMap	HashTable	ConcurrentHashMap
Locking Level	Object Level	Object Level	Segment Level
Synchronized operations	All operations are synchronized.	All operations are synchronized.	Only update operations are synchronized.
How many threads can enter into a map at a time?	Only one thread	Only one thread	By default, 16 threads can perform update operations and any number of threads can perform read operations at a time.
Null Keys And Null Values	Allows one null key and any number of null values.	Doesn't allow null keys and null values.	Doesn't allow null keys and null values.
Nature Of Iterators	Fail-Fast	Fail-Safe	Fail-Safe
Introduced In?	JDK 1.2	JDK 1.1	JDK 1.5
When To Use?	Use only when high level of data consistency is required in multi threaded environment.	Don't Use. Not recommended as it is a legacy class.	Use in all multi threaded environment except where high level of data consistency is required.

7. Difference between LinkedList and Array List?

ArrayList and LinkedList both implement the List interface and maintain insertion order. Both are non-synchronized classes.

However, there are many differences between the ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.
5) The memory location for the elements of an ArrayList is contiguous.	The location for the elements of a linked list is not contagious.
6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.	There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized.
7) To be precise, an ArrayList is a resizable array.	LinkedList implements the doubly linked list of the list interface.

8. Difference between Treeset and Hashset?

Hash set and tree set both belong to the collection framework. HashSet is the implementation of the Set interface whereas Tree set implements sorted set. Tree set is backed by TreeMap while HashSet is backed by a hashmap.

Sr. No.	Key	Hash Set	Tree Set
1	Implementation	Hash set is implemented using HashTable	The tree set is implemented using a tree structure.

Sr. No.	Key	Hash Set	Tree Set
2	Null Object	HashSet allows a null object	The tree set does not allow the null object. It throws the null pointer exception.
3	Methods	Hash set use equals method to compare two objects	Tree set use compare method for comparing two objects.
4	Heterogeneous object	Hash set doesn't now allow a heterogeneous object	Tree set allows a heterogeneous object
5	Ordering	HashSet does not maintain any order	TreeSet maintains an object in sorted order

Hash set and tree set both belong to the collection framework. HashSet is the implementation of the Set interface whereas Tree set implements sorted set. Tree set is backed by TreeMap while HashSet is backed by a hashmap.

9. How put() & get() works in Hashmap?

The java.util.HashMap.put() method of HashMap is used to insert a mapping into a map. This means we can insert a specific key and the value it is mapping to into a particular map. If an existing key is passed then the previous value gets replaced by the new value. If a new pair is passed, then the pair gets inserted as a whole.

Syntax:

Hash_Map.put(*key, value*)
Parameters: The method takes two parameters, both are of the Object type of the HashMap.

- *key:* This refers to the key element that needs to be inserted into the Map for mapping.
- *value:* This refers to the value that the above key would map into.

The java.util.HashMap.get() method of HashMap class is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter. It returns NULL when the map contains no such mapping for the key.

Syntax:
Hash_Map.get(*Object key_element*)
Parameter: The method takes one parameter *key_element* of object type and refers to the key whose associated value is supposed to be fetched.
Return Value: The method returns the value associated with the *key_element* in the parameter

10. Explain the Priority Queue & Blocking queue

The interface Queue is available in the java.util package and does extend the Collection interface. It is used to keep the elements that are processed in the First In First Out (FIFO) manner. It is an ordered list of objects, where insertion of elements occurs at the end of the list, and removal of elements occur at the beginning of the list.

PriorityQueue Class

PriorityQueue is also class that is defined in the collection framework that gives us a way for processing the objects on the basis of priority. It is already described that the insertion and deletion of objects follows FIFO pattern in the Java queue. However, sometimes the elements of the queue are needed to be processed according to the priority, that's where a PriorityQueue comes into action.

PriorityQueue Class Declaration

Let's see the declaration for java.util.PriorityQueue class.

1. **public class** PriorityQueue<E> **extends** AbstractQueue<E> **implements** Serializable

BlockingQueue in Java

Before directly jumping to the topic 'Blocking Queue' let us first understand Queue in brief. Queue is an ordered list of objects where insertions take place at the rear end of the list and deletion of elements takes place from the front end. Therefore, it is also said that Queue is based on FIFO (First-In-First-Out) principle.

Some important points with respect to the Blocking Queue :

- A BlockingQueue may have a remainingCapacity beyond which we cannot insert any element without blocking.
- All the implementations related to the BlockingQueue are thread-safe. All the methods achieve their events using internal locks or other forms of concurrency control.
- A BlockingQueue does not accept null elements. If we try to add a null value the implementation throws a NullPointerException.

11. What is difference between fail-fast and fail-safe ?

Iterators in Java are part of the Java Collection framework. They are used to retrieve elements one by one. The Java Collection supports two types of iterators; Fail Fast and Fail Safe. These iterators are very useful in exception handling.

The Fail fast iterator aborts the operation as soon it exposes failures and stops the entire operation. Comparatively, Fail Safe iterator doesn't abort the operation in case of a failure. Instead, it tries to avoid failures as much as possible.

Fail-fast Iterator

When we use the Fail-fast iterator, it immediately throws **ConcurrentModificationException** when an element is added or removed from the collection while the thread is iterating over the collection. Examples: Iterator in HashMap, Iterator on ArrayList, etc.

Fail-safe Iterator

The Java SE specification doesn't use the **Fail-safe** term. It is better to call it a **Non-Fail-fast** Iterator. The Fail-safe iterator doesn't throw the **ConcurrentModificationException,** and it tries to avoid raising the exception. The Fail-safe iterator creates a copy of the original collection or object array and iterates over that copied collection. Any structural modification made in the iterator affects the copied collection, not the original collection. Therefore, the original collection remains structurally unchanged.

Sr. No.	Key	Fail-Fast	Fail-Safe
1	Exception	Any changes in the collection, such as adding, removing and updating collection during a thread are iterating collection then Fail fast throw concurrent modification exception.	The fail-safe collection doesn't throw exception.
2.	Type of collection	ArrayList and hashmap collection are the examples of fail-fast iterator	CopyOnWrite and concurrent modification are the examples of a fail-safe iterator
3.	Performance and Memory	It's work on actual collection instead. So, this iterator doesn't require extra memory and time	It's working on a clone of the collection instead of actual collection. It is overhead in terms of time and memory
4.	Modifications	Iterators don't allow modifications of a collection while iterating over it.	Fail-Safe iterators allow modifications of a collection while iterating over it.

12. What is Comparable and Comparator interface ? Which is good?

Comparable and Comparator both are interfaces and can be used to sort collection elements.

However, there are many differences between Comparable and Comparator interfaces that are given below.

Comparable	Comparator
1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable **affects the original class**, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.

13. What is the difference between Mutable map key and unmutable map key?

Mutable and Immutable in Java

Java is an object-oriented programming language. As it is an object-oriented programming language, it's all methods and mechanism revolves around the objects. One object-based concept is mutable and immutable in Java. Objects in Java are either mutable or immutable; it depends on how the object can be iterated.

In this section, we will discuss mutable and immutable objects in Java. Further, we will see the difference between them.

What are Mutable Objects

The mutable objects are objects whose value can be changed after initialization. We can change the object's values, such as field and states, after the object is created. For example, **Java.util.Date**, **StringBuilder**, **StringBuffer**, etc.

When we made a change in existing mutable objects, no new object will be created; instead, it will alter the value of the existing object. These object's classes provide methods to make changes in it.

In Scala, there are two kinds of maps: Mutable and Immutable. **A mutable map object's value can be changed, while immutable maps do not allow the values associated with keys to be changed**.

Design Concepts

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

In Java, Singleton is a design pattern that ensures that a class can only have one object.

To create a singleton class, a class must implement the following properties:

- Create a `private` constructor of the class to restrict object creation outside of the class.
- Create a `private` attribute of the class type that refers to the single object.
- Create a `public static` method that allows us to create and access the object we created. Inside the method, we will create a condition that restricts us from creating more than one object.

Example: Java Singleton Class Syntax

```
class SingletonExample {

  // private field that refers to the object
  private static SingletonExample singleObject;

  private SingletonExample() {
    // constructor of the SingletonExample class
  }

  public static SingletonExample getInstance() {
    // write code that allows us to create only one object
    // access the object as per our need
  }
}
```

In the above example,

- `private static SingletonExample singleObject` - a reference to the object of the class.
- `private SingletonExample()` - a private constructor that restricts creating objects outside of the class.
- `public static SingletonExample getInstance()` - this method returns the reference to the only object of the class. Since the method `static`, it can be accessed using the class name.

2. What is Immutable class and How to write it? What is mutable class?

The mutable class examples are StringBuffer, Java. util. Date, StringBuilder, etc. Whereas the immutable objects are legacy classes, wrapper classes, String class, etc.

Immutable class in java means that **once an object is created, we cannot change its content**. In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable

Immutable class in java means that once an object is created, we cannot change its content. In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable. We can create our own immutable class as well. Prior to going ahead do go through characteristics of immutability in order to have a good understanding while implementing the same. Following are the requirements:

- The class must be declared as final so that child classes can't be created.
- Data members in the class must be declared private so that direct access is not allowed.
- Data members in the class must be declared as final so that we can't change the value of it after object creation.
- A parameterized constructor should initialize all the fields performing a deep copy so that data members can't be modified with an object reference.
- Deep Copy of objects should be performed in the getter methods to return a copy rather than returning the actual object reference)

3. Design patterns : Factory, Abstract Factory & DAO, MVC etc..

Factory Method Pattern

A Factory Pattern or Factory Method Pattern says that just **define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate.** In other words, subclasses are responsible to create the instance of the class.

The Factory Method Pattern is also known as **Virtual Constructor.**

Advantage of Factory Design Pattern

- Factory Method Pattern allows the sub-classes to choose the type of objects to create.
- It promotes the **loose-coupling** by eliminating the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class, so that it will work with any classes that implement that interface or that extends that abstract class.

Usage of Factory Design Pattern

- When a class doesn't know what sub-classes will be required to create
- When a class wants that its sub-classes specify the objects to be created.
- When the parent classes choose the creation of objects to its sub-classes.

Abstract Factory Pattern

Abstract Factory Pattern says that just **define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete sub-classes.**That means Abstract Factory lets a class returns a factory of classes. So, this is the reason that Abstract Factory Pattern is one level higher than the Factory Pattern.

An Abstract Factory Pattern is also known as **Kit.**

Advantage of Abstract Factory Pattern

- Abstract Factory Pattern isolates the client code from concrete (implementation) classes.
- It eases the exchanging of object families.
- It promotes consistency among objects.

Usage of Abstract Factory Pattern

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

- When the family of related objects has to be used together, then this constraint needs to be enforced.
- When you want to provide a library of objects that does not show implementations and only reveals interfaces.
- When the system needs to be configured with one of a multiple family of objects.

DAO Class in Java

Data Access Object patterns, often known as **DAO** patterns, are used to divide high level business services from low level data accessing APIs or actions. The members of the Data Access Object Pattern are listed below.

Data Access Object Interface: The Data Access Object Interface specifies the common operations to be carried out on a model object (s).

Concrete Data Access Object class: This class implements the aforementioned interface. This class is in charge of obtaining data from a data source, which could be a database, XML, or another type of storage system.

Model or Value Object: This object is a straightforward POJO with get/set methods for storing data obtained using the DAO class.

Next →← Prev

MVC Architecture in Java

The Model-View-Controller (MVC) is a well-known design pattern in the web development field. It is way to organize our code. It specifies that a program or application shall consist of data model, presentation information and control information. The MVC pattern needs all these components to be separated as different objects.

In this section, we will discuss the MVC Architecture in Java, alongwith its advantages and disadvantages and examples to understand the implementation of MVC in Java.

What is MVC architecture in Java?

The model designs based on the MVC architecture follow MVC design pattern. The application logic is separated from the user interface while designing the software using model designs.

The MVC pattern architecture consists of three layers:

- **Model:** It represents the business layer of application. It is an object to carry the data that can also contain the logic to update controller if data is changed.

- **View:** It represents the presentation layer of application. It is used to visualize the data that the model contains.
- **Controller:** It works on both the model and view. It is used to manage the flow of application, i.e. data flow in the model object and to update the view whenever data is changed.

SQL Questions

1. What is Primary Key? Foreign Key?Unique Key?

Primary Key

- Primary key cannot have a NULL value.
- Each table can have only one primary key.
- By default, Primary key is clustered index, and the data in database table is physically organized in the sequence of clustered index.
- Primary key can be related to another tables as a Foreign Key.
- We can generate ID automatically with the help of Auto Increment field. Primary key supports Auto Increment value.
- We can define Primary key constraint on temporary table and table variable.
- We can't delete primary key value from the parent table which is used as a foreign key in child table. To delete we first need to delete that primary key value from the child table.

Unique Key

- Unique Constraint may have a NULL value.
- Each table can have more than one Unique Constraint.
- By default, Unique key is a unique non-clustered index.
- Unique Constraint can not be related with another table's as a Foreign Key.

Foreign Key

- Foreign key is a field in the table that is Primary key in another table.

- Foreign key can accept multiple null value.
- Foreign key do not automatically create an index, clustered or non-clustered. You can manually create an index on foreign key.
- We can have more than one foreign key in a table.
- There are actual advantages to having a foreign key be supported with a clustered index, but you get only one per table. What's the advantage? If you are selecting the parent plus all child records, you want the child records next to each other. This is easy to accomplish using a clustered index.
- Having a null foreign key is usually a bad idea instead of NULL referred to as "orphan record".
- We can’t define foreign key constraint on temporary table or table variable.
- We can delete the foreign key value from the child table even though that refers to the primary key of the parent table.

2. What is a join?What are the types of join and explain each with differences?

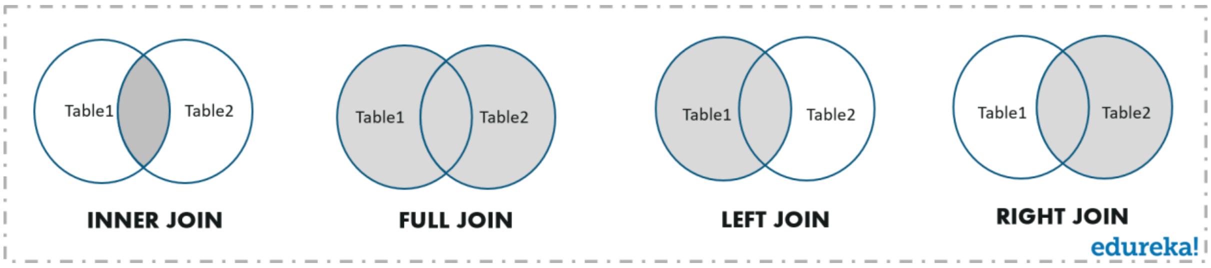
JOINS in SQL are commands which are used to combine rows from two or more tables, based on a related column between those tables. There are predominantly used when a user is trying to extract data from tables which have one-to-many or many-to-many relationships between them.

Now, that you know what joins mean, let us next learn the different types of joins.

How many types of Joins are there in SQL?
There are mainly four types of joins that you need to understand. They are:

- INNER JOIN
- FULL JOIN
- LEFT JOIN
- RIGHT JOIN

You can refer to the below image.



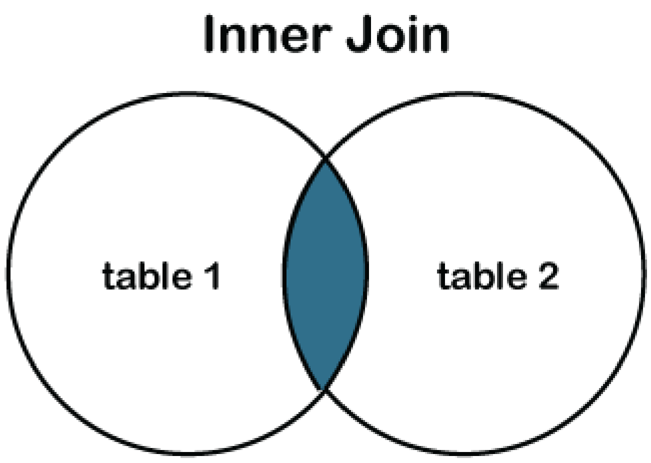
Inner Join

The inner join is used to select all matching rows or columns in both tables or as long as the defined condition is valid in SQL.

Syntax:

1. Select column_1, column_2, column_3 FROM table_1 INNER JOIN table_2 ON table_1.column = table_2.column;

We can represent the inner join through the Venn diagram, as follows:



Natural Join

It is a type of inner type that joins two or more tables based on the same column name and has the same data type present on both tables.

Syntax:

1. Select * from tablename1 Natural JOIN tablename_2;

We have two tables: **Students** and the **Teachers** Tables. Let's write the SQL Queries to join the table using the **Natural JOIN** as follows:

1. Select * from Students Natural JOIN Teachers;

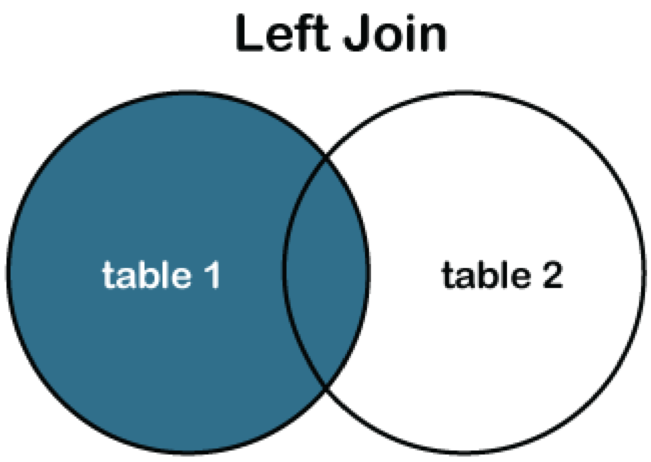
LEFT JOIN

The **LEFT JOIN** is used to retrieve all records from the left table (table1) and the matched rows or columns from the right table (table2). If both tables do not contain any matched rows or columns, it returns the NULL.

Syntax:

1. Select column_1, column_2, column(s) FROM table_1 LEFT JOIN table_2 ON table_1. column_name = table_2.column_name;

We can also represent the left join through the Venn diagram, as follows:



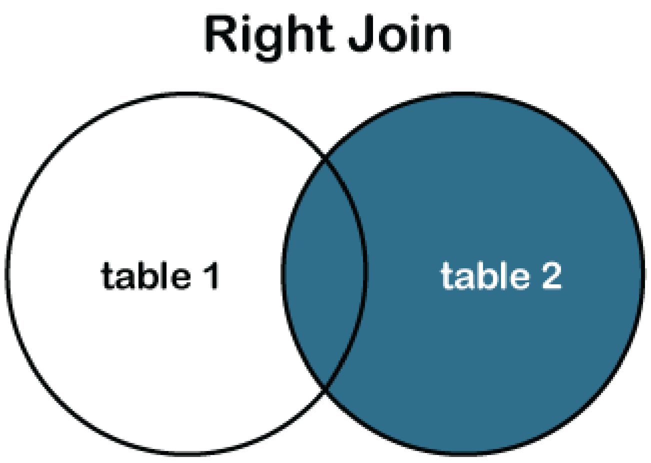
RIGHT JOIN or RIGHT Outer JOIN:

The **RIGHT JOIN** is used to retrieve all records from the right table (table2) and the matched rows or columns from the left table (table1). If both tables do not contain any matched rows or columns, it returns the NULL.

Syntax:

1. Select column_1, column_2, column(s) FROM table_1 RIGHT JOIN table_2 ON table_ 1.column_name = table_2.column_name;

We can also represent the right join through the Venn diagram, as follows:



FULL JOIN or FULL Outer JOIN:

It is a combination result set of both **LEFT JOIN** and **RIGHT JOIN**. The joined tables return all records from both the tables and if no matches are found in the table, it places NULL. It is also called a **FULL OUTER JOIN**.

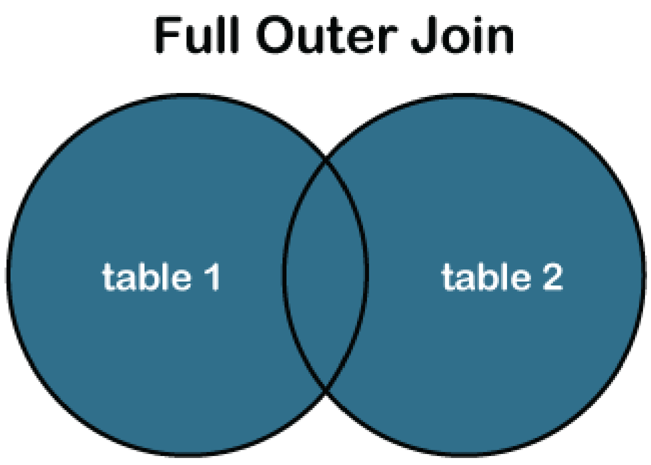
Syntax:

1. Select column_1, column_2, column(s) FROM table_1 FULL JOIN table_2 ON table_1. column_name = table_2.column_name;

Or, **FULL OUTER JOIN**

1. Select column_1, column_2, column(s) FROM table_1 FULL OUTER JOIN table_2 ON table_1.column_name = table_2.column_name;

We can also represent the full outer join through the Venn diagram, as follows:



3. What is normalization?What are all the different normalizations?

Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is Normalization?

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

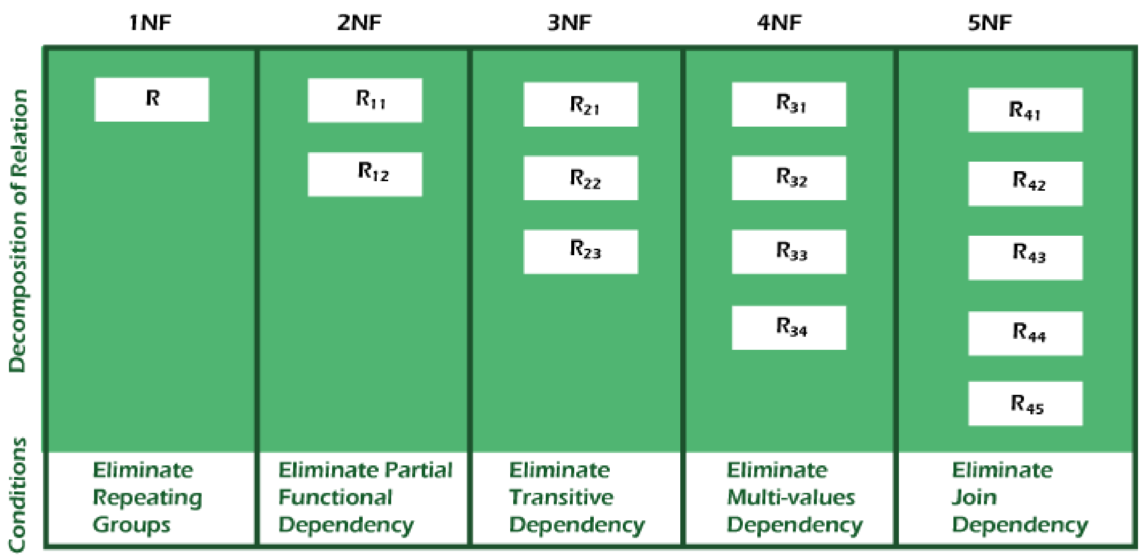
Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Following are the various types of Normal forms:


1NF
2NF
3NF
4NF
5NF
Decomposition of Relation
Conditions
R
R11
R12
R21
R22
R23
R31
R32
R33
R34
R41
R42
R43
R44
R45
Eliminate Repeating Groups
Eliminate Partial Functional Dependency
Eliminate Transitive Dependency
Eliminate Multi-values Dependency
Eliminate Join Dependency

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.

4. Ask Group by, having… also SQL query like Find Max of Salary, Second max salary etc,How to select unique records from a table?

The GROUP BY clause is a SQL command that is used to **group rows that have the same values**. The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.

That's what it does, **summarizing data** from the database.

The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

Having clause is **used to filter data according to the conditions provided**. Having clause is generally used in reports of large data. Having clause is only used with the SELECT clause. The expression in the syntax can only have constants.

5. Transaction handling?

SQL Server Transaction

A transaction in SQL Server is a **sequential group of statements or queries** to perform single or multiple tasks in a database. Each transaction may have single read, write, update, or delete operations or a combination of all these operations. Each transaction must happen two things in SQL Server:

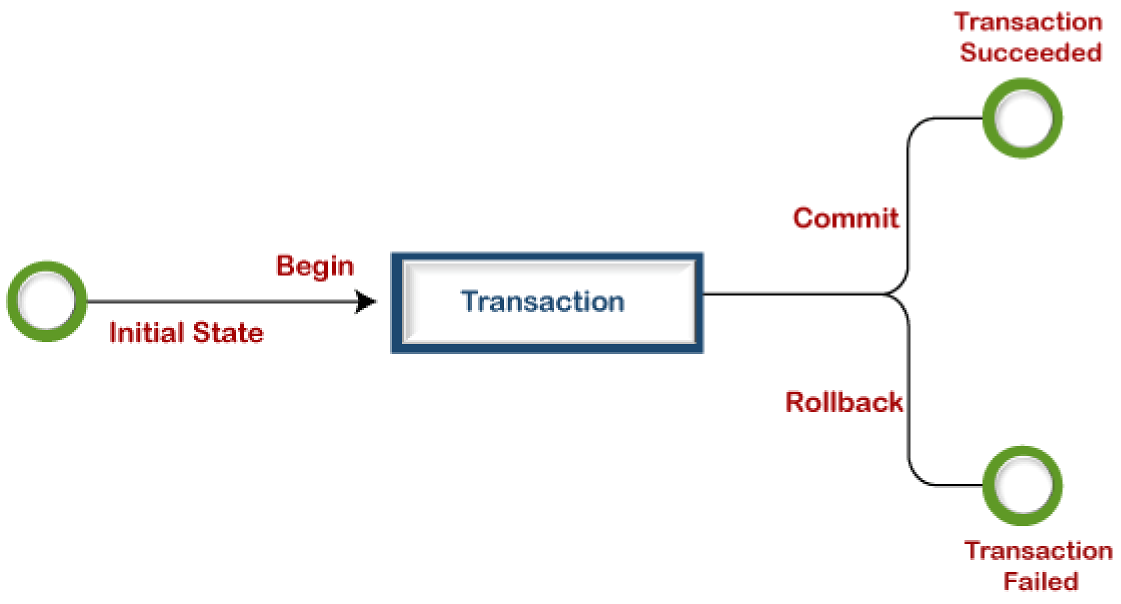
- Either all modification is successful when the transaction is committed.
- Or, all modifications are undone when the transaction is rollback.

A transaction cannot be successful until all of the operations in the set are completed. It means that if any argument fails, the transaction operation will fail. Each transaction begins with the first executable SQL statement and ends when it finds a commit or rollback, either explicitly or implicitly. It uses the **COMMIT** or **ROLLBACK** statements explicitly, as well as implicitly when a DDL statement is used.

The below pictorial representation explains the transaction process:

Transaction Modes in SQL Server

There are three different transaction modes that SQL Server can use:



Transaction Modes in SQL Server

There are three different transaction modes that SQL Server can use:

Auto-commit Transaction Mode: It is the SQL Server's default transaction mode. It will evaluate each SQL statement as a transaction, and the results are committed or rolled back accordingly. Thus the successful statements are immediately committed, while the failed statements are immediately rolled back.

Implicit Transaction Mode. This mode allows SQL Server to begin the implicit transaction for each DML statement, but it explicitly requires the use of commit or rollback commands at the end of the statements.

Explicit Transaction Mode: This mode is defined by the user that allows us to identify a transaction's beginning and ending points exactly. It will automatically abort in case of a fatal error.

Transaction Control

The following are the commands used to control transactions:

- **BEGIN TRANSACTION:** It is a command that indicates the beginning of each transaction.
- **COMMIT:** It is a command used to save the changes permanently in the database.
- **ROLLBACK:** It is a command used to cancel all modifications and goes into their previous state.

- **SAVEPOINT:** This command creates points within groups of transactions that allow us to roll back only a portion of a transaction rather than the entire transaction.
- **RELEASE SAVEPOINT:** It is used to remove an already existing SAVEPOINT.
- **SET TRANSACTION:** This command gives a transaction a name, which can be used to make it read-only or read/write or assign it to a specific rollback segment.

6. What is a View? Index? What are all the different types of indexes?

SQL CREATE VIEW Statement

In SQL, a view 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.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The Index in SQL is a special table used to speed up the searching of the data in the database tables. It also retrieves a vast amount of data from the tables frequently. The INDEX requires its own space in the hard disk.

Different Types of Indexes in SQL Server

- Clustered Index.
- Non-Clustered Index.
- Column Store Index.
- Filtered Index.
- Hash Index.
- Unique Index.

7. SQL optional questions

https://www.interviewbit.com/sql-interview-questions/

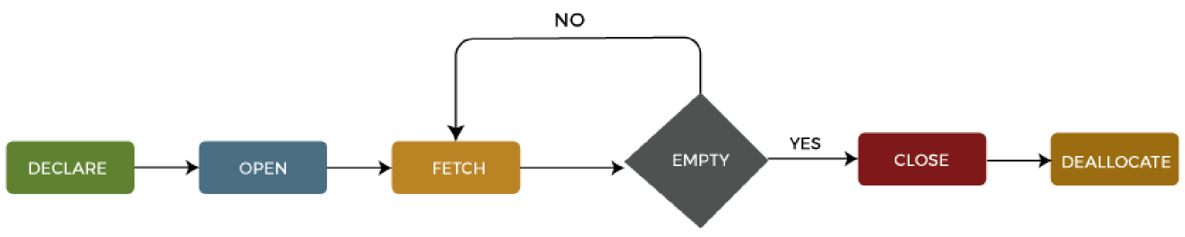
8. What is a Cursor?Stored Procedure?What is a trigger? Advantages and Disadvantages of Stored Procedure?

A cursor in SQL Server is a d**atabase object that allows us to retrieve each row at a time and manipulate its data**. A cursor is nothing more than a pointer to a row. It's always used in conjunction with a SELECT statement. It is usually a collection of SQL logic that loops through a predetermined number of rows one by one. A simple illustration of the cursor is when we have an extensive database of worker's records and want to calculate each worker's salary after deducting taxes and leaves.

The SQL Server **cursor's purpose is to update the data row by row, change it, or perform calculations that are not possible when we retrieve all records at once**. It's also useful for performing administrative tasks like SQL Server database backups in sequential order. Cursors are mainly used in the development, DBA, and ETL processes.

Life Cycle of the cursor

We can describe the life cycle of a cursor into the **five different sections** as follows:



A trigger is a set of SQL statements that reside in system memory with unique names. It is a specialized category of stored procedure that is called automatically when a database server event occurs. Each trigger is always associated with a table.

A **trigger is called a special procedure** because it cannot be called directly like a stored procedure. The key distinction between the trigger and procedure is that a trigger is called automatically when a data modification event occurs against a table. A stored procedure, on the other hand, must be invoked directly.

The following are the main characteristics that distinguish triggers from stored procedures:

- We cannot manually execute/invoked triggers.
- Triggers have no chance of receiving parameters.
- A transaction cannot be committed or rolled back inside a trigger.

Syntax of Trigger

We can create a trigger in SQL Server by using the **CREATE TRIGGER** statement as follows:

```
CREATE TRIGGER schema.trigger_name
ON table_name
AFTER  {INSERT, UPDATE, DELETE}
[NOT FOR REPLICATION]
AS
{SQL_Statements}
```

A stored procedure is a **group of one or more pre-compiled SQL statements** into a logical unit. It is stored as an object inside the database server. It is a subroutine or a subprogram in the common computing language that has been created and stored in the database. Each procedure in SQL Server always **contains a name, parameter lists, and Transact-SQL statements**. The SQL Database Server stores the stored procedures as **named objects**. We can invoke the procedures by using triggers, other procedures, and applications such as Java, Python, PHP, etc. It can support almost all relational database systems.

Advantages :
The main advantages of stored procedure are given below:

1. **Better Performance –**
   The procedure calls are quick and efficient as stored procedures are compiled once and stored in executable form.Hence the response is quick. The executable code is automatically cached, hence lowers the memory requirements.
2. **Higher Productivity –**
   Since the same piece of code is used again and again so, it results in higher productivity.
3. **Ease of Use –**
   To create a stored procedure, one can use any Java Integrated Development Environment (IDE). Then, they can be deployed on any tier of network architecture.
4. **Scalability –**
   Stored procedures increase scalability by isolating application processing on the server.

5. **Maintainability –**
   Maintaining a procedure on a server is much easier then maintaining copies on various client machines, this is because scripts are in one location.
6. **Security –**
   Access to the Oracle data can be restricted by allowing users to manipulate the data only through stored procedures that execute with their definer's privileges.

Disadvantages :
The main disadvantages of stored procedures are given below:

1. **Testing –**
   Testing of a logic which is encapsulated inside a stored procedure is very difficult. Any data errors in handling stored procedures are not generated until runtime.
2. **Debugging –**
   Depending on the database technology, debugging stored procedures will either be very difficult or not possible at all. Some relational databases such as SQL Server have some debugging capabilities.
3. **Versioning –**
   Version control is not supported by the stored procedure.
4. **Cost –**
   An extra developer in the form of DBA is required to access the SQL and write a better stored procedure. This will automatically incur added cost.
5. **Portability –**
   Complex stored procedures will not always port to upgraded versions of the same database. This is specially true in case of moving from one database type(Oracle) to another database type(MS SQL Server).

A trigger is a set of SQL statements that reside in system memory with unique names. It is a specialized category of stored procedure that is called automatically when a database server event occurs. Each trigger is always associated with a table.

A **trigger is called a special procedure** because it cannot be called directly like a stored procedure. The key distinction between the trigger and procedure is that a trigger is called automatically when a data modification event occurs against a table. A stored procedure, on the other hand, must be invoked directly.

The following are the main characteristics that distinguish triggers from stored procedures:

- We cannot manually execute/invoked triggers.
- Triggers have no chance of receiving parameters.
- A transaction cannot be committed or rolled back inside a trigger.

Syntax of Trigger

We can create a trigger in SQL Server by using the **CREATE TRIGGER** statement as follows:

1. **CREATE TRIGGER schema**.trigger_name
2. **ON** table_name
3. **AFTER** {**INSERT**, **UPDATE**, **DELETE**}
4. [NOT **FOR** REPLICATION]
5. **AS**
6. {SQL_Statements}

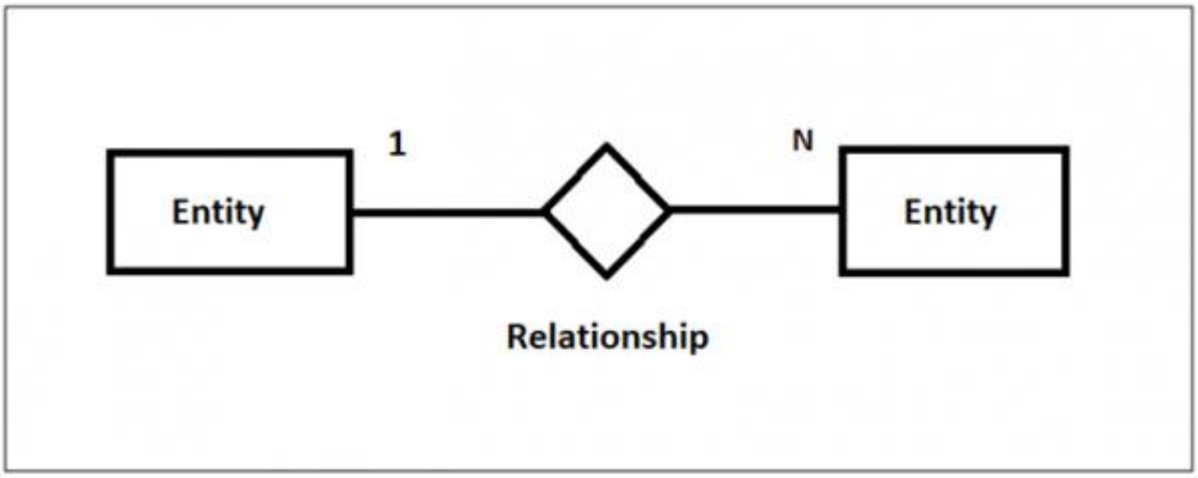
9. What is a relationship and what are they? Like one to many, one to One etc

A relationship can be One-to-Many or Many-to-One in DBMS. Let us see what that means with examples −

One-to-Many Relationship

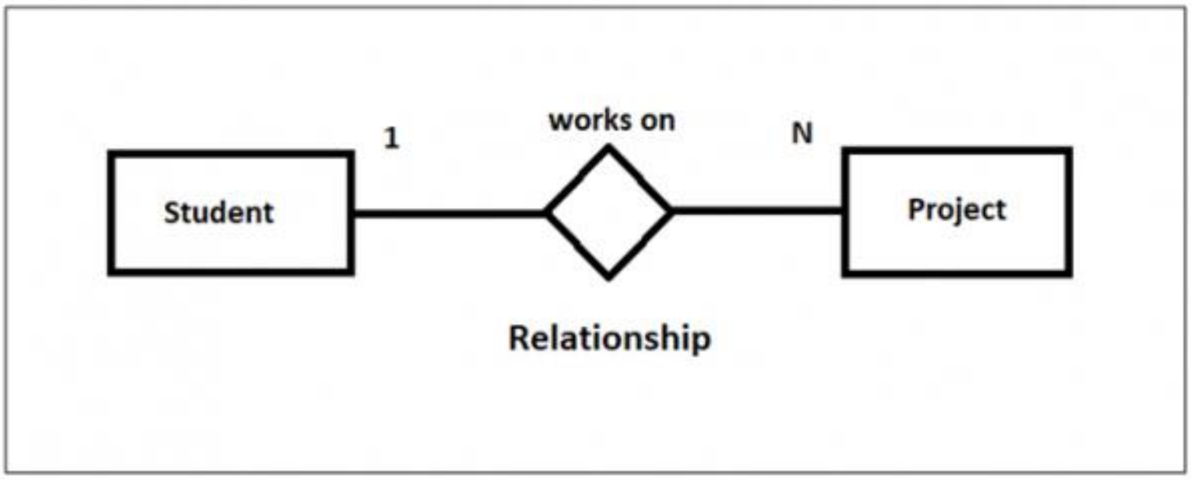
One-to-Many relationship in DBMS is a relationship between instances of an entity with more than one instance of another entity.

The relation can be shown as −



Let us see an example −

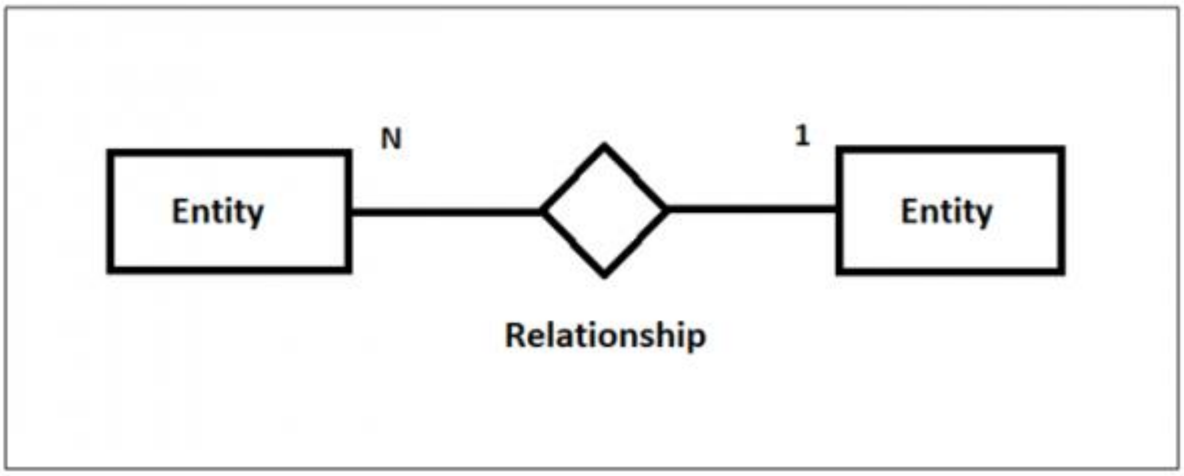
A student can work on more than one project. **Student** and **Project** are entities here. An individual student working on 2 projects at a time would be considered as One-to-Many relationship in DBMS as shown below:



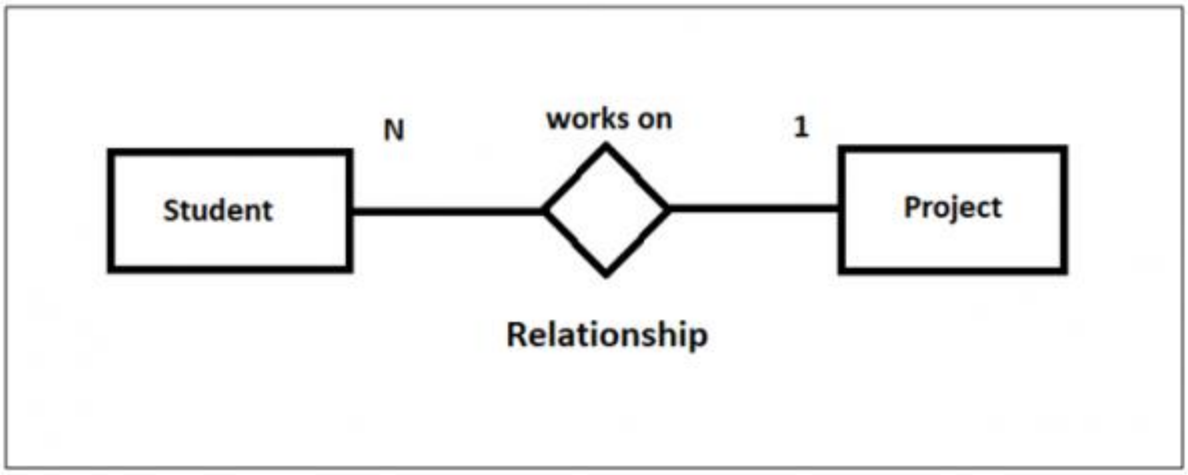
Many to One Relationship

Many-to-One relationship in DBMS is a relationship between more than one instances of an entity with one instance of another entity.

The relation can be stated as −



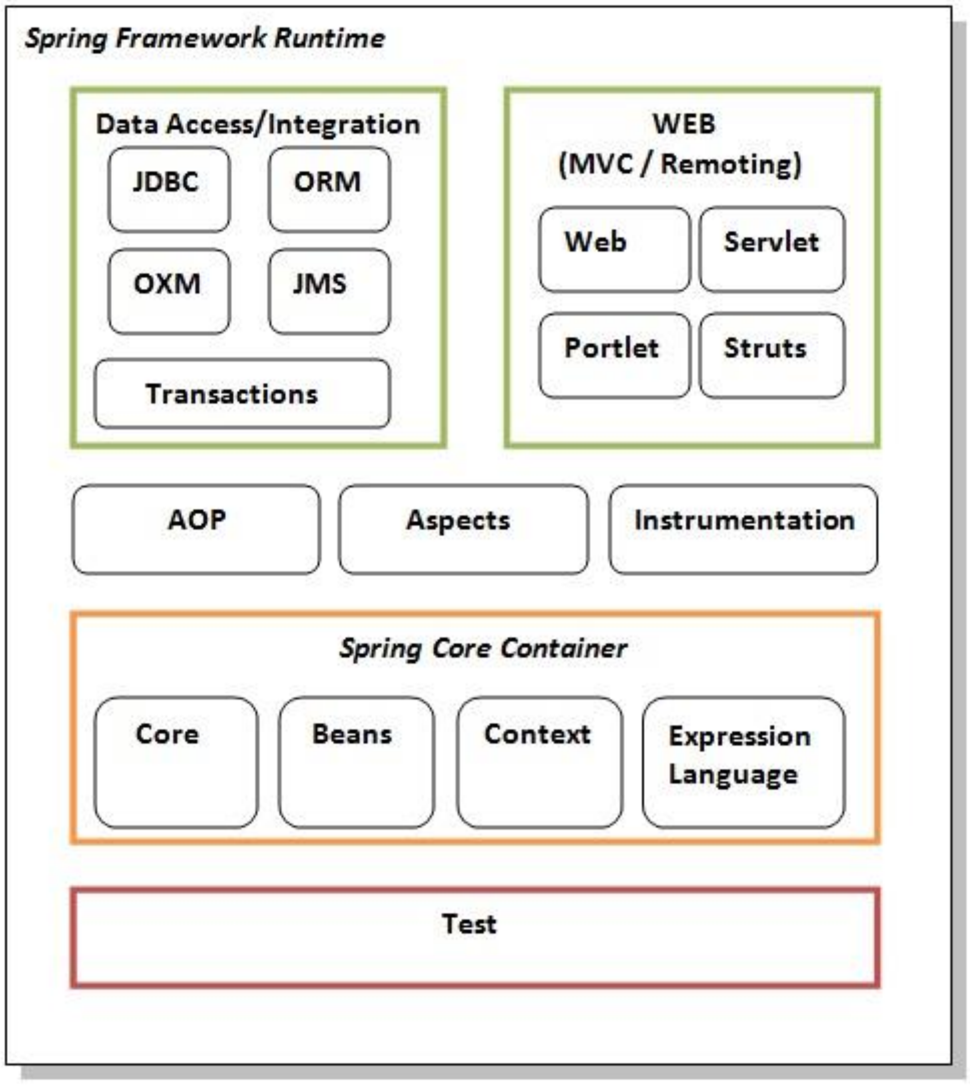
A project can have more than one student working on it. A team of five students in a college in assigned a project that they need to complete in let us say one month. This states a relationship between two entities **Student** and **Project**.



Spring

1. What is Spring? What are features of Spring?Which are the Spring framework modules?

The Spring framework comprises of many modules such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc. These modules are grouped into Test, Core Container, AOP, Aspects, Instrumentation, Data Access / Integration, Web (MVC / Remoting) as displayed in the following diagram.



Features of the Spring Framework

The features of the Spring framework such as IoC, AOP, and transaction management, make it unique among the list of frameworks. Some of the most important features of the Spring framework are as follows:

- **IoC container:**
  Refers to the core container that uses the DI or IoC pattern to implicitly provide an object reference in a class during runtime. This pattern acts as an alternative to the service locator pattern. The IoC container contains assembler code that handles the configuration management of application objects.
  The Spring framework provides two packages, namely org.springframework.beans and org.springframework.context which helps in providing the functionality of the IoC container.
- **Data access framework:**
  Allows the developers to use persistence APIs, such as JDBC and Hibernate, for storing persistence data in database. It helps in solving various problems of the developer, such as how to interact with a database connection, how to make sure that the connection is closed, how to deal with exceptions, and how to implement transaction management It also enables the developers to easily write code to access the persistence data throughout the application.
- **Spring MVC framework:**
  Allows you to build Web applications based on MVC architecture. All the requests made by a user first go through the controller and are then dispatched to different views, that is, to different JSP pages or Servlets. The form handling and form validating features of the Spring MVC framework can be easily integrated with all popular view technologies such as ISP, Jasper Report, FreeMarker, and Velocity.
- **Transaction management:**
  Helps in handling transaction management of an application without affecting its code. This framework provides Java Transaction API (JTA) for global transactions managed by an application server and local transactions managed by using the JDBC Hibernate, Java Data Objects (JDO), or other data access APIs. It enables the developer to model a wide range of transactions on the basis of Spring's declarative and programmatic transaction management.
- **Spring Web Service:**
  Generates Web service endpoints and definitions based on Java classes, but it is difficult to manage them in an application. To solve this problem, Spring Web Service provides layered-based approaches that are separately managed by Extensible Markup Language (XML) parsing (the technique of reading and manipulating XML). Spring provides effective mapping for transmitting incoming XML message request to an object and the developer to easily distribute XML message (object) between two machines.
- **JDBC abstraction layer:**
  Helps the users in handling errors in an easy and efficient manner. The JDBC programming code can be reduced when this abstraction layer is implemented in a Web application. This layer handles exceptions such as DriverNotFound. All SQLExceptions are translated into the DataAccessException class. Spring's data access exception is not JDBC specific and hence Data Access Objects (DAO) are not bound to JDBC only.
- **Spring TestContext framework:**
  Provides facilities of unit and integration testing for the Spring applications. Moreover, the Spring TestContext framework provides specific integration

testing functionalities such as context management and caching DI of test fixtures, and transactional test management with default rollback semantics.

2. What is Spring IOC container?Spring beans?

Spring IoC Container

Spring IoC is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, objects dependencies are injected by other assembler objects. Spring IoC container is the program that **injects** dependencies into an object and make it ready for our use. We have already looked how we can use Spring Dependency Injection to implement IoC in our applications. Spring IoC container classes are part of org.springframework.beans and org.springframework.context packages. Spring IoC container provides us different ways to decouple the object dependencies. BeanFactory is the root interface of Spring IoC container. ApplicationContext is the child interface of BeanFactory interface that provide Spring AOP features, i18n etc. Some of the useful child-interfaces of ApplicationContext are ConfigurableApplicationContext and WebApplicationContext. Spring Framework provides a number of useful ApplicationContext implementation classes that we can use to get the spring context and then the Spring Bean. Some of the useful ApplicationContext implementations that we use are;

- **AnnotationConfigApplicationContext**: If we are using Spring in standalone java applications and using annotations for Configuration, then we can use this to initialize the container and get the bean objects.
- **ClassPathXmlApplicationContext**: If we have spring bean configuration xml file in standalone application, then we can use this class to load the file and get the container object.
- **FileSystemXmlApplicationContext**: This is similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.
- **AnnotationConfigWebApplicationContext** and **XmlWebApplicationContext** for web applications.

Usually, if you are working on Spring MVC application and your application is configured to use Spring Framework, Spring IoC container gets initialized when the application starts and when a bean is requested, the dependencies are injected automatically. However, for a standalone application, you need to initialize the container somewhere in the application and then use it to get the spring beans.

Spring Bean

Spring Bean is nothing special, any object in the Spring framework that we initialize through Spring container is called Spring Bean. Any normal Java POJO class can be a Spring Bean if it’s configured to be initialized via container by providing configuration metadata information.

Spring Bean Scopes

There are five scopes defined for Spring Beans.

1. **singleton** - Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues.
2. **prototype** - A new instance will be created every time the bean is requested.
3. **request** - This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
4. **session** - A new bean will be created for each HTTP session by the container.
5. **global-session** - This is used to create global session beans for Portlet applications.

Spring Framework is extendable and we can create our own scopes too. However, most of the times we are good with the scopes provided by the framework.

Spring Bean Configuration

Spring Framework provides three ways to configure beans to be used in the application.

1. **Annotation Based Configuration** - By using @Service or @Component annotations. Scope details can be provided with @Scope annotation.
2. **XML Based Configuration** - By creating Spring Configuration XML file to configure the beans. If you are using Spring MVC framework, the xml based configuration can be loaded automatically by writing some boiler plate code in web.xml file.
3. **Java Based Configuration** - Starting from Spring 3.0, we can configure Spring beans using java programs. Some important annotations used for java based configuration are @Configuration, @ComponentScan and @Bean.

3. Dependency Injection? IOC?

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 Container uses Dependency Injection(DI) to manage the components that make up the application. It gets the information about the objects from a configuration file(XML) or Java Code or Java Annotations and Java POJO class. These objects are called Beans. Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion Of Control. The followings are some of the main features of Spring IoC,

- Creating Object for us,
- Managing our objects,
- Helping our application to be configurable,
- Managing dependencies

Spring Dependency Injection

Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The design principle of Inversion of Control emphasizes keeping the Java classes independent of each other and the container frees them from object creation and maintenance. These classes, managed by Spring, must adhere to the standard definition of Java-Bean. Dependency Injection in Spring also ensures loose coupling between the classes. There are two types of Spring Dependency Injection.

1. Setter Dependency Injection (SDI)
2. Constructor Dependency Injection (CDI)

A. Setter Dependency Injection (SDI)

Setter Injection is the simpler of the two Dependency Injection methods. In this, the Dependency Injection will be injected with the help of setter and/or getter methods. Now to set the Dependency Injection as Setter Injection in the bean, it is done through the bean-configuration file For this, the property to be set with the Setter Injection is declared under the **<property>** tag in the bean-config file.

B. Constructor Dependency Injection (CDI)

In Constructor Injection, the Dependency Injection will be injected with the help of constructors. Now to set the Dependency Injection as Constructor Dependency Injection in bean, it is done through the bean-configuration file. For this, the property to be set with the CDI is declared under the **<constructor-arg>** tag in the bean-config file.

Let us finally come up with cut-throat differences between them depicted via the table given below to get a better understanding as there persists always a dilemma if not understood to great depth.

Spring IoC (Inversion of Control)	Spring Dependency Injection
Spring IoC Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle.	Spring Dependency injection is a way to inject the dependency of a framework component by the following ways of spring: Constructor Injection and Setter Injection
Spring helps in creating objects, managing objects, configurations, etc. because of IoC (Inversion of Control).	Spring framework helps in the creation of loosely-coupled applications because of Dependency Injection.
Spring IoC is achieved through Dependency Injection.	Dependency Injection is the method of providing the dependencies and Inversion of Control is the end result of Dependency Injection.

Spring IoC (Inversion of Control)	Spring Dependency Injection
IoC is a design principle where the control flow of the program is inverted.	Dependency Injection is one of the subtypes of the IOC principle.
Aspect-Oriented Programing is one way to implement Inversion of Control.	In case of any changes in business requirements, no code change is required.

4. Types of Dependency Injection? Which is good?

Setter Injection is the preferred choice when a number of dependencies to be injected is a lot more than normal, if some of those arguments are optional than using a Builder design pattern is also a good option. In Summary, both Setter Injection and Constructor Injection have their own advantages and disadvantages.

5. What bean scopes does Spring support? Explain them.

When defining a <bean> you have the option of declaring a scope for that bean. For example, to force Spring to produce a new bean instance each time one is needed, you should declare the bean's scope attribute to be **prototype**. Similarly, if you want Spring to return the same bean instance each time one is needed, you should declare the bean's scope attribute to be **singleton**.

The Spring Framework supports the following five scopes, three of which are available only if you use a web-aware ApplicationContext.

Sr.No.	Scope & Description
1	**singleton** This scopes the bean definition to a single instance per Spring IoC container (default).
2	**prototype** This scopes a single bean definition to have any number of object instances.

3	**request** This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
4	**session** This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
5	**global-session** This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

6. Explain Bean lifecycle in Spring framework?

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.

Note: We can choose custom method name instead of **init()** and **destroy()**. Here, we will use init() method to execute all its code as the spring container starts up and the bean is instantiated, and destroy() method to execute all its code on closing the container.
Ways to implement the life cycle of a bean
Spring provides three ways to implement the life cycle of a bean. In order to understand these three ways, let's take an example. In this example, we will write and activate init() and destroy() method for our bean (HelloWorld.java) to print some message on start and close of Spring container. Therefore, the three ways to implement this are:
1. By XML: In this approach, in order to avail custom init() and destroy() method for a bean we have to register these two methods inside 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 annotate **init()** method by **@PostConstruct** annotation and **destroy()** method by **@PreDestroy** annotation.
Note: To invoke the **destroy()** method we have to call the **close()** method of ConfigurableApplicationContext.
Therefore, the following steps are followed:

- Firstly, we need to create a bean HelloWorld.java in this case and annotate the custom init() method with @PostConstruct and destroy() method with @PreDestroy.

7.What is AOP?

Aspect Oriented Programming (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class.

AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by **cross-cutting concerns**.

A **cross-cutting concern** is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

Where use AOP?

AOP is mostly used in following cases:

- to provide declarative enterprise services such as declarative transaction management.
- It allows users to implement custom aspects.

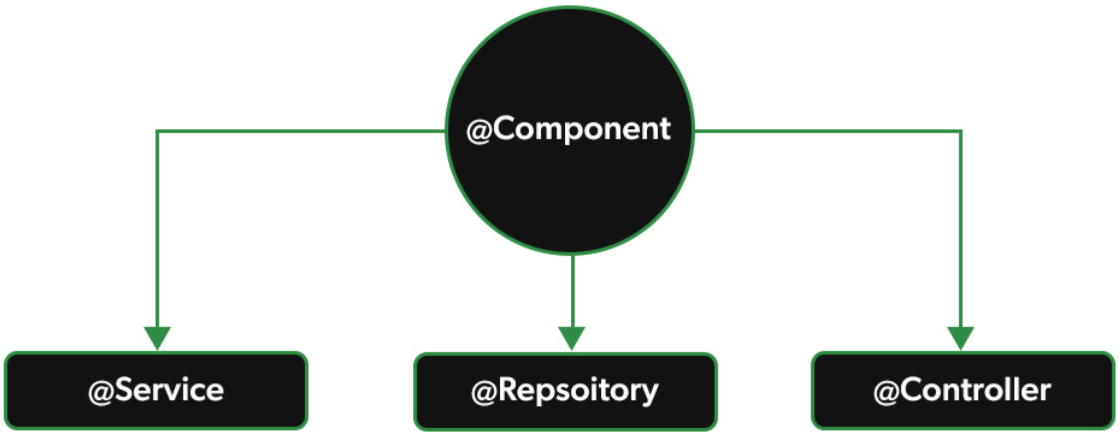
9. What's the difference between @Component, @Controller, @Repository & @Service annotations in Spring?

Spring Annotations are a form of metadata that provides data about a program. Annotations are used to provide supplemental information about a program. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program. Here, we are going to discuss the difference between the 4 most important annotations in Spring, @Component, @Repository, @Service, and @Controller.

@Component 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. A component is responsible for some operations. Spring framework provides three other specific annotations to be used when marking a class as a Component.

1. @Service
2. @Repository
3. @Controller



A. @Service Annotation

In an application, the business logic resides within the service layer so we use the **@Service Annotation** to indicate that a class belongs to that layer. It is a specialization of **@Component Annotation**. 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. Spring context will autodetect these classes when annotation-based configuration and classpath scanning is used.

B. @Repository Annotation

@Repository Annotation is also 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. Though it is a specialization of @Component annotation, so Spring Repository classes are autodetected by the spring framework through classpath scanning. This annotation is a general-purpose stereotype annotation which very close to the DAO pattern where DAO classes are responsible for providing CRUD operations on database tables.

C. @Controller Annotation

Spring @Controller annotation is also a specialization of **@Component** annotation. 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. It's mostly used with Spring MVC applications. This annotation acts as a stereotype for the annotated class, indicating its role. The dispatcher scans such annotated classes for mapped methods and detects **@RequestMapping** annotations.

@Service Annotation	**@Repository Annotation**	**@Controller Annotation**
@Service annotation is used with classes that provide some business functionalities.	@Repository Annotation is used to indicate that the class provides the mechanism for storage, retrieval, update, delete and search operation on objects.	@Controller annotation indicates that a particular class serves the role of a controller.
@Service Annotation is a specialization of @Component Annotation.	@Repository Annotation is also a specialization of @Component Annotation.	@Controller annotation is also a specialization of @Component annotation.
It is used to mark the class as a service provider.	It is used to mark the interface as DAO (Data Access Object) provider.	It's used to mark a class as a web request handler.
It is a stereotype for the service layer.	It is also a stereotype for the DAO layer.	It is also a stereotype for the presentation layer (spring-MVC).
Switch can be possible. But it is not recommended.	Switch can be possible. But it is not recommended.	We cannot switch this annotation with any other like @Service or @Repository.
It is a Stereotype Annotation.	It is also a Stereotype Annotation.	It is also a Stereotype Annotation.

Hibernate

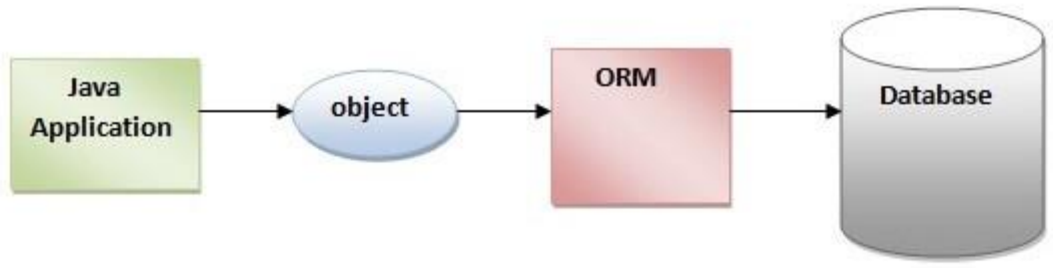
1. What is hibernate? ORM?Mention some of the advantages of using ORM over JDBC.

Hibernate Framework

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

ORM Tool

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

What is JPA?

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

Advantages of Hibernate Framework

Following are the advantages of hibernate framework:

1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

2) Fast Performance

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status.

Why is ORM preferred over JDBC?

- It allows business code access the objects rather than Database tables.
- It hides the details of SQL queries from OO logic.
- It is based on JDBC “under hood”.
- Dealing with database implementation is not required.
- Entities are based on business concepts rather than database structures.

2. What are the core interfaces of Hibernate?

1. Session Interface : The basic interface for all hibernate applications. The instances are light weighted and can be created and destroyed without expensive process.

2. SessionFactory interface : The delivery of session objects to hibernate applications is done by this interface. For the whole application, there will be generally one SessionFactory and can be shared by all the application threads.

3. Configuration Interface : Hibernate bootstrap action is configured by this interface. The location specification is specified by specific mapping documents, is done by the instance of this interface.

4. Transaction Interface : This is an optional interface. This interface is used to abstract the code from a transaction that is implemented such as a JDBC / JTA transaction.

5. Query and Criteria interface : The queries from the user are allowed by this interface apart from controlling the flow of the query execution.

3. What is SessionFactory? Session?

SessionFactory In Hibernate.

SessionFactory is an interface available in org.hibernate package which extends Referenceable and Serializable interface and provides factory methods to get session object. Let's see some points about SessionFactory.

- SessionFactory is thread safe so multiple threads can access the SessionFactory at the same time.
- SessionFactory is Immutable. Once we create SessionFactory we can not modify it(If you look SessionFactory methods we don't have any setter kind of method)

Session In Hibernate.

Session is an interface available in org.hibernate package which provides different API to communicate java application to hibernate. Let's see some points related to Session.

- The session is not thread-safe.
- The main use of the Session object to perform create, get, and delete operations for java classes(entity).
- The first level cache belongs to the session object. See an example of the first-level cache in detail.
- We can have multiple sessions for a SessionFactory.

4. Is Session a thread-safe object?

No, Session is not a thread-safe object, many threads can access it simultaneously. In other words, you can share it between threads.

5. What's the difference between load() and get() method in hibernate? Save or SaveOrUpdate?

In hibernate, get() and load() are two methods which is used to fetch data for the given identifier. They both belong to Hibernate session class. Get() method return null, If no row is available in the session cache or the database for the given identifier whereas load() method throws object not found exception.

Sr. No.	Key	Get()	Load()
1	Basic	It is used to fetch data from the database for the given identifier	It is also used to fetch data from the database for the given identifier
2	Null Object	It object not found for the given identifier then it will return null object	It will throw object not found exception
3	Lazy or Eager loading	It returns fully initialized object so this method eager load the object	It always returns proxy object so this method is lazy load the object
4	Performance	It is slower than load() because it return fully initialized object which impact the performance of the application	It is slightly faster.
5.	Use Case	If you are not sure that object exist then use get() method	If you are sure that object exist then use load() method

Hibernate provides the save method on the session interface to **save a record** into the database. So the session.save **inserts a record** into the database. It **returns the id**, that is the primary key that will be assigned to the database record.

What is saveOrUpdate?

In addition to the save method, Hibernate also provides a method called saveOrUpdateon the session interface. Just as the name suggests, this method either **performs a save operation or an update operation**. If there is no record in the database corresponding to the object passed in, it inserts a record, but if there is a record corresponding to the object passed in, it just updates the existing record. This is generally useful in applications where a UI is involved. In such cases the back end might not be sure if the UI has sent a new object or an existing object with some updated fields. So instead of writing explicit code in the back end to check if a record exists, the saveOrUpdate method can be used.

How are they similar?

Both the save and the saveOrUpdatemethods can be **used to insert a record** to the database. Also, both methods are **Hibernate's proprietary methods** and are not part of the JPA specification.

save	saveOrUpdate
Inserts a record	Inserts or Updates a record
Returns the id of the inserted record	Does not return anything, returns a void
Moves an entity from transient state to persistent state	Moves an entity from transient or detached state to persistent state

6. What is the difference between session.save() and session.persist() method?

Save() and persist() both methods are used for saving object in the database.

Save() – Persist the given transient instance, first assigning a generated identifier. (Or using the current value of the identifier property if the assigned generator is used.) This operation cascades to associated instances if the association is mapped with cascade="save-update".

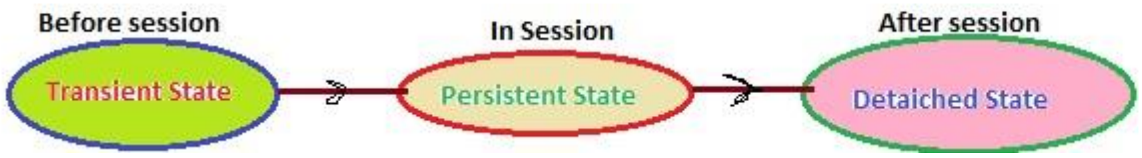
As per docs –

persist() – Make a transient instance persistent. This operation cascades to associated instances if the association is mapped with cascade="persist".

Sr. No.	Key	save()	persist()
1	Basic	It stores object in database	It also stores object in database

Sr. No.	Key	save()	persist()
2	Return Type	It return generated id and return type is serializable	It does not return anything. Its void return type.
3	Transaction Boundaries	It can save object within boundaries and outside boundaries	It can only save object within the transaction boundaries
4.	Detached Object	It will create a new row in the table for detached object	It will throw persistence exception for detached object
5.	Supported by	It is only supported by Hibernate	It is also supported by JPA

7. Different states in hibernates?How can we reattach any detached objects in Hibernate?



1. Transient State:

A New instance of a persistent class which is not associated with a ***Session***, has no representation in the ***database*** and no identifier value is considered ***transient*** by Hibernate:

```
UserDetail user = new UserDetail();

user.setUserName("Dinesh Rajput");

// user is in a transient state
```

2. Persistent State:
A persistent instance has a representation in the ***database*** , an identifier value and is associated with a ***Session***. You can make a transient instance persistent by associating it with a ***Session***:

```
Long id = (Long) session.save(user);
```

```
// user is now in a persistent state
```

3. Detached State:
Now, if we close the ***Hibernate Session***, the ***persistent*** instance will become a ***detached*** instance: it isn't attached to a ***Session*** anymore (but can still be modified and reattached to a new Session later though).

```
session.close();

//user in detached state
```

proper way to re-attach detached objects in Hibernate?

1. getHibernateTemplate(). update(obj) This works if and only if an object doesn't already exist in the hibernate session. ...
2. getHibernateTemplate(). merge(obj) This works if and only if an object exists in the hibernate session.

8. What is lazy loading in hibernate?

Lazy loading is a fetching technique used for all the entities in Hibernate. It decides whether to load a child class object while loading the parent class object. When we use association mapping in Hibernate, it is required to define the fetching technique. The main purpose of lazy loading is to fetch the needed objects from the database.

Lazy loading can be used with all types of Hibernate mapping, i.e., one-to-one, one-to-many, many-to-one, and many-to-many.

Syntax of Lazy Loading

To enable Lazy loading, we use the following annotation parameter:

```
fetch= FetchType.LAZY
```

Following code is the syntax of lazy loading:

```
@OneToOne(fetch= FetchType.LAZY)
```

9. Transaction handling, exception handling

Transaction Interface in Hibernate

In hibernate framework, we have **Transaction** interface that defines the unit of work. It maintains abstraction from the transaction implementation (JTA,JDBC).

A transaction is associated with Session and instantiated by calling **session.beginTransaction()**.

The methods of Transaction interface are as follows:

1. **void begin()** starts a new transaction.
2. **void commit()** ends the unit of work unless we are in FlushMode.NEVER.
3. **void rollback()** forces this transaction to rollback.
4. **void setTimeout(int seconds)** it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
5. **boolean isAlive()** checks if the transaction is still alive.
6. **void registerSynchronization(Synchronization s)** registers a user synchronization callback for this transaction.
7. **boolean wasCommited()** checks if the transaction is commited successfully.
8. **boolean wasRolledBack()** checks if the transaction is rolledback successfully

Hibernate Exception Overview Many conditions can cause exceptions to be thrown while using Hibernate. These can be mapping errors, infrastructure problems, SQL errors, data integrity violations, session problems, and transaction errors.

Hibernate provides better handle than the JDBCException . Developers can **use the try and catch block** to handle the exceptions. Put the line of code that may cause an exception in a try block and according to that exception put the exception handler in the catch block.

10. Spring security

Spring Security is a framework which provides various security features like: authentication, authorization to create secure Java Enterprise Applications.

It overcomes all the problems that come during creating non spring security applications and manage new server environment for the application.

This framework targets two major areas of application are authentication and authorization. Authentication is the process of knowing and identifying the user that wants to access.

Advantages

Spring Security has numerous advantages. Some of that are given below.

- Comprehensive support for authentication and authorization.
- Protection against common tasks

- Servlet API integration
- Integration with Spring MVC
- Portability
- CSRF protection
- Java Configuration suppor

SpringBoot

1. What is Actuator? How it's work?

Spring Boot Actuator is a sub-project of the Spring Boot Framework. It includes a number of additional features that help us to monitor and manage the Spring Boot application. It contains the actuator endpoints (the place where the resources live). We can use **HTTP** and **JMX** endpoints to manage and monitor the Spring Boot application. If we want to get production-ready features in an application, we should use the S**pring Boot actuator.**

Spring Boot Actuator Features

There are **three** main features of Spring Boot Actuator:

- **Endpoints**
- **Metrics**
- **Audit**

Enabling Spring Boot Actuator

We can enable actuator by injecting the dependency **spring-boot-starter-actuator** in the pom.xml file.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
```

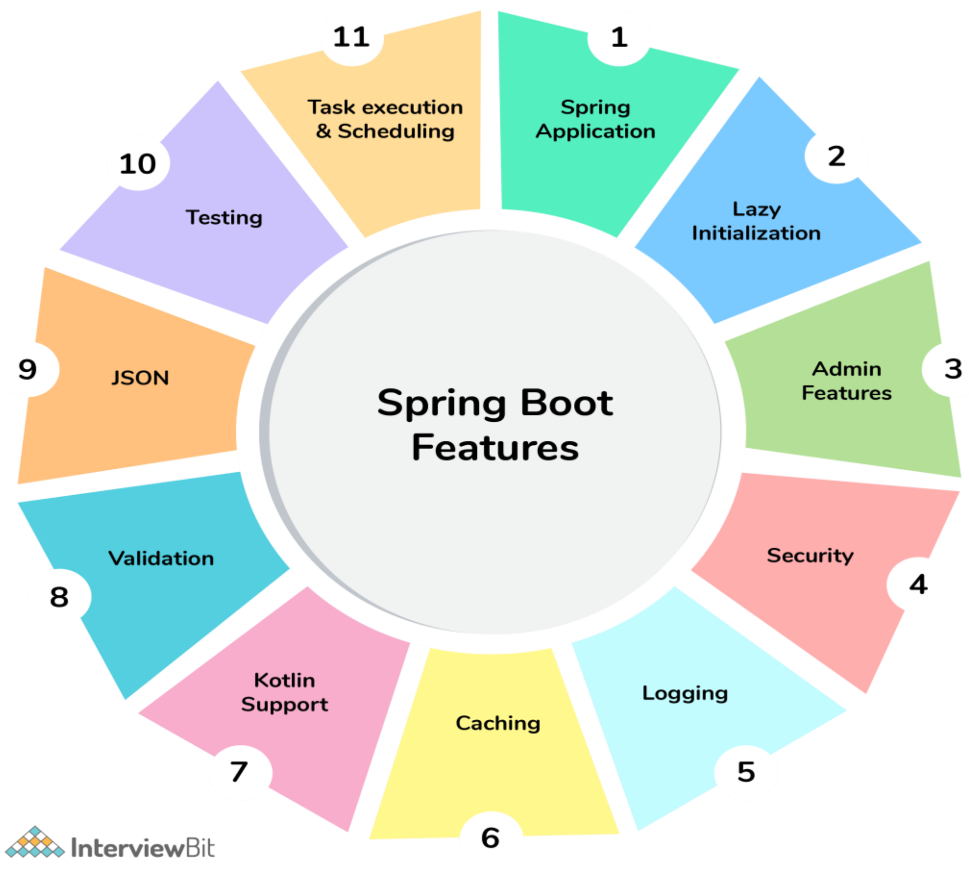
```
    <version>2.2.2.RELEASE</version>
</dependency>
```

2. What is Spring boot?

Sprint boot is a Java-based spring framework used for Rapid Application Development (to build stand-alone microservices). It has extra support of auto-configuration and embedded application server like tomcat, jetty, etc.

Features of Spring Boot that make it different?

- Creates stand-alone spring application with minimal configuration needed.
- It has embedded tomcat, jetty which makes it just code and run the application.
- Provide production-ready features such as metrics, health checks, and externalized configuration.
- Absolutely no requirement for XML configuration.



3. What are the advantages of using Spring Boot?

The advantages of Spring Boot are listed below:

- Easy to understand and develop spring applications.
- Spring Boot is nothing but an existing framework with the addition of an embedded HTTP server and annotation configuration which makes it easier to understand and faster the process of development.
- Increases productivity and reduces development time.
- Minimum configuration.
- We don't need to write any XML configuration, only a few annotations are required to do the configuration.

4. What are the Spring Boot key components?

Below are the four key components of spring-boot:

- Spring Boot auto-configuration.
- Spring Boot CLI.
- Spring Boot starter POMs.
- Spring Boot Actuators.

5. Why Spring Boot over Spring?

Below are some key points which spring boot offers but spring doesn't:

- Starter POM.
- Version Management.
- Auto Configuration.
- Component Scanning.
- Embedded server.
- InMemory DB.
- Actuators

6. What is the starter dependency of the Spring boot module?

Spring boot provides numbers of starter dependency, here are the most commonly used -

- Data JPA starter.
- Test Starter.
- Security starter.
- Web starter.
- Mail starter.
- Thymeleaf starter.

7. How does Spring Boot works?

Spring Boot automatically configures your application based on the dependencies you have added to the project by using annotation. The entry point of the spring boot application is the class that contains @SpringBootApplication annotation and the main method.

Spring Boot automatically scans all the components included in the project by using @ComponentScan annotation.

8. What does the @SpringBootApplication annotation do internally?

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan with their default attributes. Spring Boot enables the developer to use a single annotation instead of using multiple. But, as we know, Spring provided loosely coupled features that we can use for each annotation as per our project needs.

9. What is the purpose of using @ComponentScan in the class files?

Spring Boot application scans all the beans and package declarations when the application initializes. You need to add the @ComponentScan annotation for your class file to scan **your components added to your project.**

10. How does a spring boot application get started?

Just like any other Java program, a Spring Boot application must have a main method. This method serves as an entry point, which invokes the SpringApplication#run method to bootstrap the application.

```
@SpringBootApplication
public class MyApplication {

    public static void main(String[] args) {

        SpringApplication.run(MyApplication.class);
         // other statements
    }
}
```

11. What are starter dependencies?

Spring boot starter is a maven template that contains a collection of all the relevant transitive dependencies that are needed to start a particular functionality.
Like we need to import spring-boot-starter-web dependency for creating a web application.

```
<dependency>
```

```
<groupId> org.springframework.boot</groupId>
<artifactId> spring-boot-starter-web </artifactId>
</dependency>
```

11. What is Spring Initializer?

Spring Initializer is a web application that helps you to create an initial spring boot project structure and provides a maven or gradle file to build your code. It solves the problem of setting up a framework when you are starting a project from scratch.

12. What is Spring Boot CLI and what are its benefits?

Spring Boot CLI is a command-line interface that allows you to create a spring-based java application using Groovy.

Advanced Spring Boot Questions

13. What Are the Basic Annotations that Spring Boot Offers?

The primary annotations that Spring Boot offers reside in its org.springframework.boot.autoconfigure and its sub-packages. Here are a couple of basic ones:

@EnableAutoConfiguration – to make Spring Boot look for auto-configuration beans on its classpath and automatically apply them.

@SpringBootApplication – used to denote the main class of a Boot Application. This annotation combines @Configuration, @EnableAutoConfiguration, and @ComponentScan annotations with their default attributes.

14. What is Spring Boot dependency management?

Spring Boot dependency management is used to manage dependencies and configuration automatically without you specifying the version for any of that dependencies.

15. Can we create a non-web application in Spring Boot?

Yes, we can create a non-web application by removing the web dependencies from the classpath along with changing the way Spring Boot creates the application context.

16. Is it possible to change the port of the embedded Tomcat server in Spring Boot?

Yes, it is possible. By using the **server.port** in the **application.properties**.

17. What is the default port of tomcat in spring boot?

The default port of the tomcat server-id 8080. It can be changed by adding **sever.port** properties in the **application.property** file.

18. Can we override or replace the Embedded tomcat server in Spring Boot?

Yes, we can replace the Embedded Tomcat server with any server by using the Starter dependency in the **pom.xml** file. Like you can use spring-boot-starter-jetty as a dependency for using a jetty server in your project.

19. Can we disable the default web server in the Spring boot application?

Yes, we can use **application.properties** to configure the web application type **i.e** spring.main.web-application-type=none.

20. How to disable a specific auto-configuration class?

You can use exclude attribute of @EnableAutoConfiguration if you want auto-configuration not to apply to any specific class.

```
//use of exclude
@EnableAutoConfiguration(exclude={className})
```

21. Explain @RestController annotation in Sprint boot?

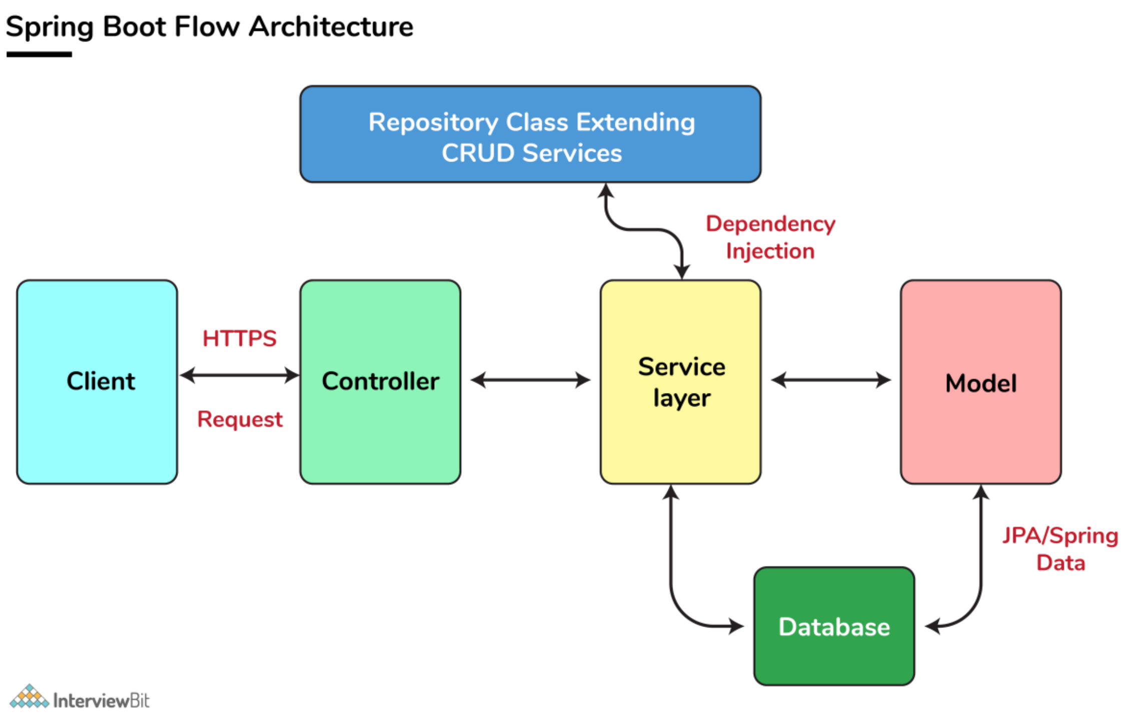
It is a combination of @Controller and @ResponseBody, used for creating a restful controller. It converts the response to JSON or XML. It ensures that data returned by each method will be written straight into the response body instead of returning a template.

22. What is the difference between @RestController and @Controller in Spring Boot?

@Controller Map of the model object to view or template and make it human readable but @RestController simply returns the object and object data is directly written in HTTP response as JSON or XML.

23. Describe the flow of HTTPS requests through the Spring Boot application?

On a high-level spring boot application follow the MVC pattern which is depicted in the below flow diagram.



24. What is the difference between RequestMapping and GetMapping?

RequestMapping can be used with GET, POST, PUT, and many other request methods using the method attribute on the annotation. Whereas getMapping is only an extension of RequestMapping which helps you to improve on clarity on request.

25. What is the use of Profiles in spring boot?

While developing the application we deal with multiple environments such as dev, QA, Prod, and each environment requires a different configuration. For eg., we might be using an embedded H2 database for dev but for prod, we might have proprietary Oracle or DB2. Even if DBMS is the same across the environment, the URLs will be different.

To make this easy and clean, Spring has the provision of Profiles to keep the separate configuration of environments.

26. What is Spring Actuator? What are its advantages?

An actuator is an additional feature of Spring that helps you to monitor and manage your application when you push it to production. These actuators include auditing, health, CPU usage, HTTP hits, and metric gathering, and many more that are automatically applied to your application.

27. How to enable Actuator in Spring boot application?

To enable the spring actuator feature, we need to add the dependency of “spring-boot-starter-actuator” in pom.xml.

```
<dependency>
<groupId> org.springframework.boot</groupId>
<artifactId> spring-boot-starter-actuator </artifactId>
</dependency>
```

28. What are the actuator-provided endpoints used for monitoring the Spring boot application?

Actuators provide below pre-defined endpoints to monitor our application -

- Health
- Info
- Beans
- Mappings
- Configprops
- Httptrace
- Heapdump
- Threaddump
- Shutdown

29. How to get the list of all the beans in your Spring boot application?

Spring Boot actuator “/Beans” is used to get the list of all the spring beans in your application.

30. How to check the environment properties in your Spring boot application?

Spring Boot actuator “/env” returns the list of all the environment properties of running the spring boot application.

31. How to enable debugging log in the spring boot application?

Debugging logs can be enabled in three ways -

- We can start the application with --debug switch.
- We can set the logging.level.root=debug property in application.property file.
- We can set the logging level of the root logger to debug in the supplied logging configuration file.

32. Where do we define properties in the Spring Boot application?

You can define both application and Spring boot-related properties into a file called application.properties. You can create this file manually or use Spring Initializer to create this file. You don't need to do any special configuration to instruct Spring Boot to load this file, If it exists in classpath then spring boot automatically loads it and configure itself and the application code accordingly.

33. What is dependency Injection?

The process of injecting dependent bean objects into target bean objects is called dependency injection.

- Setter Injection: The IOC container will inject the dependent bean object into the target bean object by calling the setter method.
- Constructor Injection: The IOC container will inject the dependent bean object into the target bean object by calling the target bean constructor.
- Field Injection: The IOC container will inject the dependent bean object into the target bean object by Reflection API.

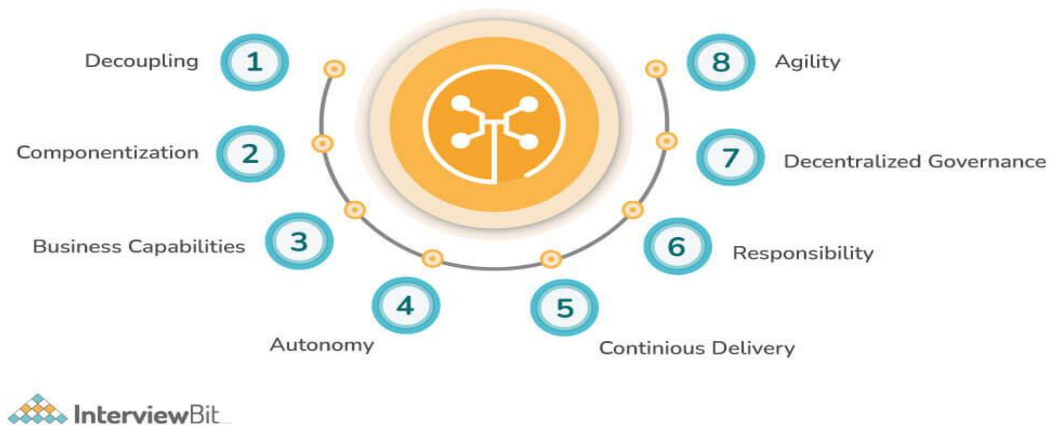
34. What is an IOC container?

IoC Container is a framework for implementing automatic dependency injection. It manages object creation and its life-time and also injects dependencies into the class.

MicroServices

1. Write main features of Microservices.

Some of the main features of Microservices include:



- **Decoupling**: Within a system, services are largely decoupled. The application as a whole can therefore be easily constructed, altered, and scalable
- **Componentization**: Microservices are viewed as independent components that can easily be exchanged or upgraded
- **Business Capabilities**: Microservices are relatively simple and only focus on one service
- **Team autonomy**: Each developer works independently of each other, allowing for a faster project timeline
- **Continuous Delivery**: Enables frequent software releases through systematic automation of software development, testing, and approval
- **Responsibility:** Microservices are not focused on applications as projects. Rather, they see applications as products they are responsible for
- **Decentralized Governance:** Choosing the right tool according to the job is the goal. Developers can choose the best tools to solve their problems
- **Agility:** Microservices facilitate agile development. It is possible to create new features quickly and discard them again at any time.

2. Write main components of Microservices.

Some of the main components of microservices include:

- Containers, Clustering, and Orchestration
- IaC [Infrastructure as Code Conception]
- Cloud Infrastructure
- API Gateway
- Enterprise Service Bus
- Service Delivery

3. What are the benefits and drawbacks of Microservices?

Benefits:

- Self-contained, and independent deployment module.
- Independently managed services.

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

Drawbacks:

- Due to the complexity of the architecture, testing and monitoring are more difficult.
- Lacks the proper corporate culture for it to work.
- Pre-planning is essential.
- Complex development.
- Requires a cultural shift.
- Expensive compared to monoliths.
- Security implications.
- Maintaining the network is more difficult.

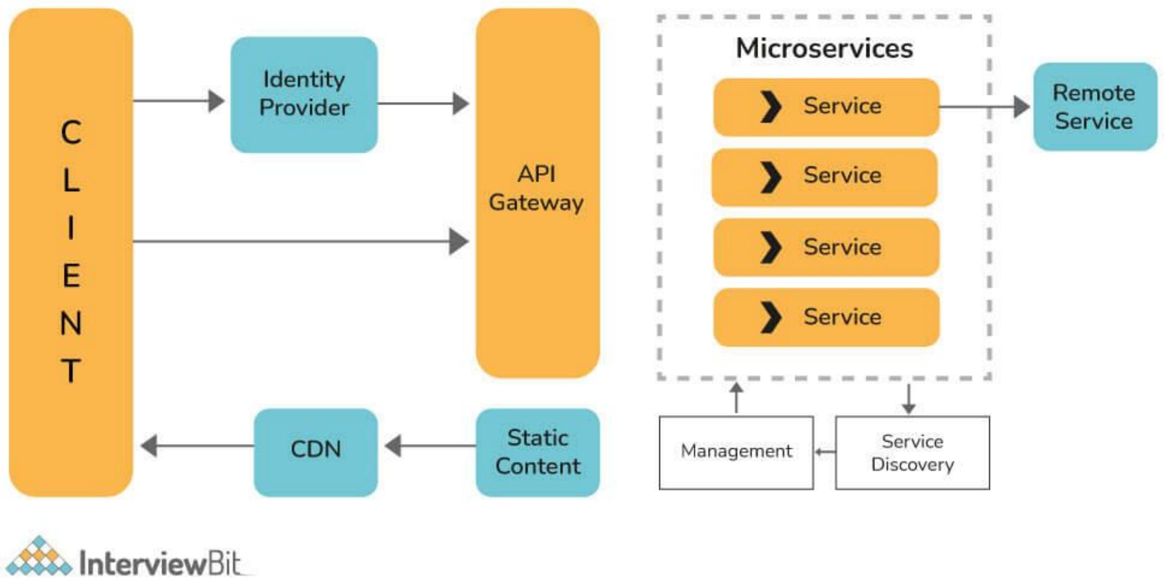
4. Name three common tools mostly used for microservices.

Three common tools used for microservices include:

- Wiremock
- Docker
- Hystrix

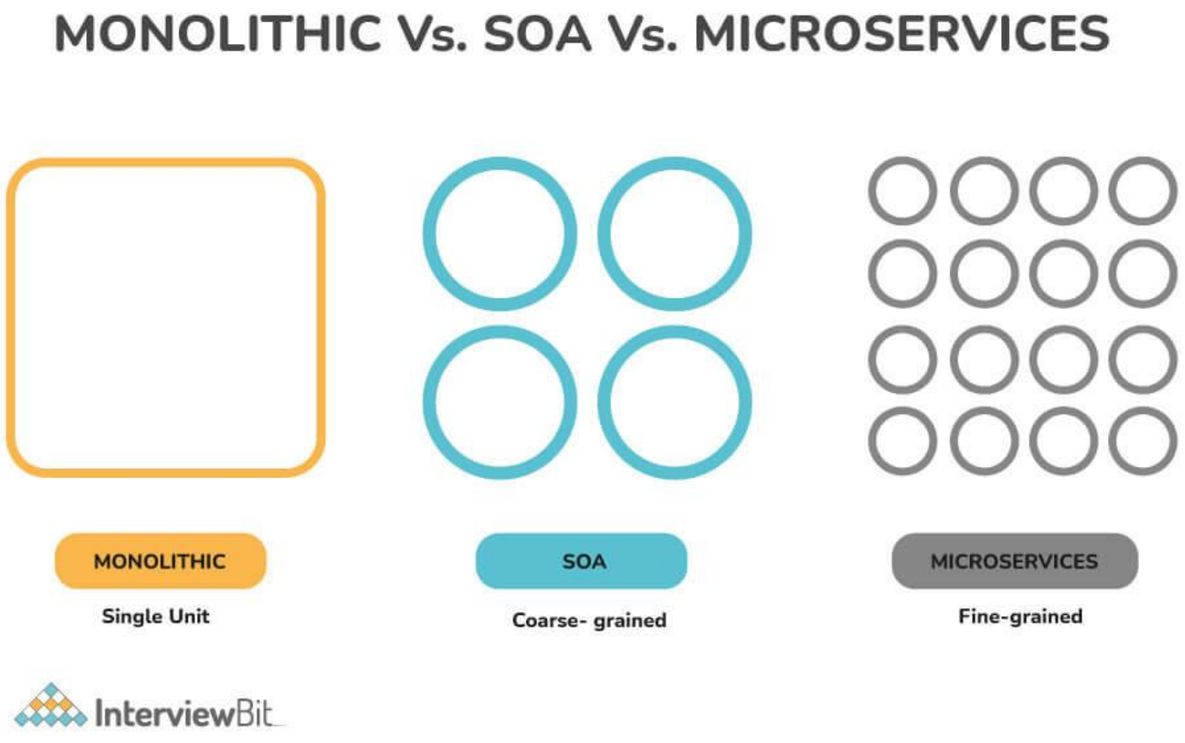
5. Explain the working of Microservice Architecture.

Microservice architectures consist of the following components:



- **Clients**: Different users send requests from various devices.
- **Identity Provider**: Validate a user's or client's identity and issue security tokens.
- **API Gateway:** Handles the requests from clients.
- **Static Content:** Contains all of the system's content.
- **Management:** Services are balanced on nodes and failures are identified.
- **Service Discovery:** A guide to discovering the routes of communication between microservices.
- **Content Delivery Network:** Includes distributed network of proxy servers and their data centers.
- **Remote Service:** Provides remote access to data or information that resides on networked computers and devices.

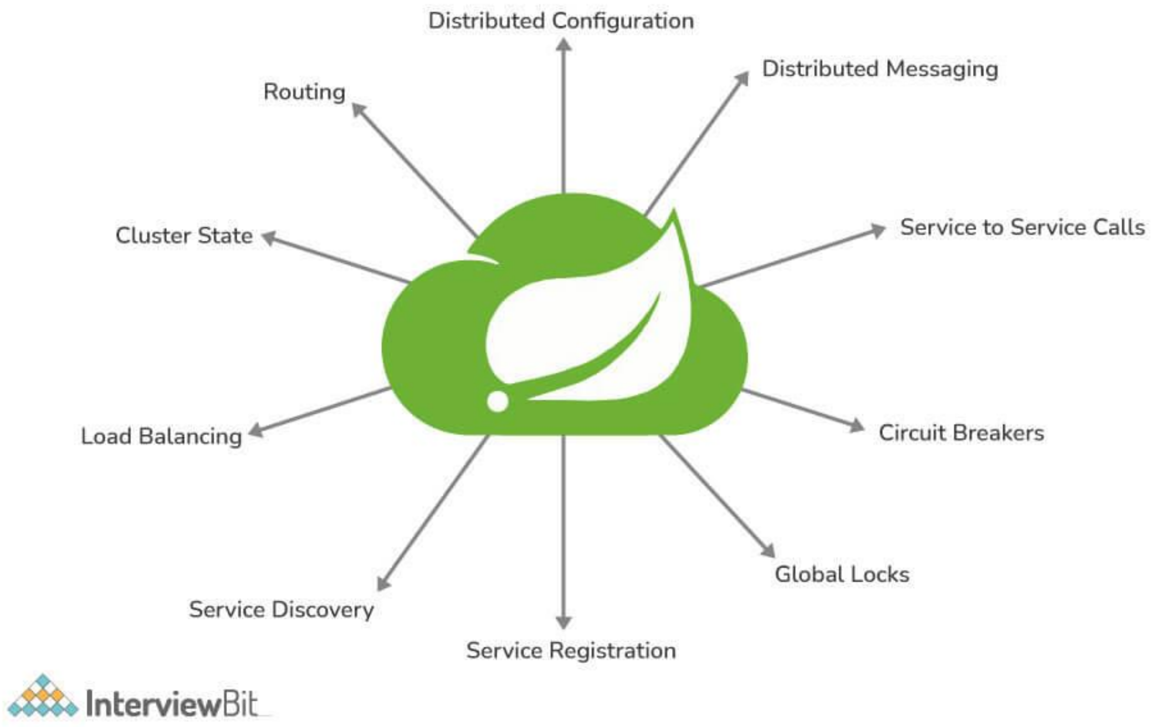
6. Write difference between Monolithic, SOA and Microservices Architecture.



- **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.

7. Explain spring cloud and spring boot.

Spring Cloud: In Microservices, the Spring cloud is a system that integrates with external systems. This is a short-lived framework designed to build applications quickly. It contributes significantly to microservice architecture due to its association with finite amounts of data processing. Some of the features of spring cloud are shown below:



Spring Boot: Spring Boot is an open-sourced, Java-based framework that provides its developers with a platform on which they can create stand-alone, production-grade Spring applications. In addition to reducing development time and increasing productivity, it is easily understood.

Spring boot
3
Used to create stand-alone Java applications.
2
Takes an opinionated view of the Spring platform.
1
Makes it easy to create Spring-powered, production-grade applications.
InterviewBit

8. What is the role of actuator in spring boot?

A spring boot actuator is a project that provides restful web services to access the current state of an application that is running in production. In addition, you can monitor and manage application usage in a production environment without having to code or configure any of the applications.

9. Explain how you can override the default properties of Spring boot projects.

By specifying properties in the application.properties file, it is possible to override the default properties of a spring boot project.

Example:
In Spring MVC applications, you need to specify a suffix and prefix. You can do this by adding the properties listed below in the application.properties file.

- For suffix – spring.mvc.view.suffix: .jsp
- For prefix – spring.mvc.view.prefix: /WEB-INF/

10. What issues are generally solved by spring clouds?

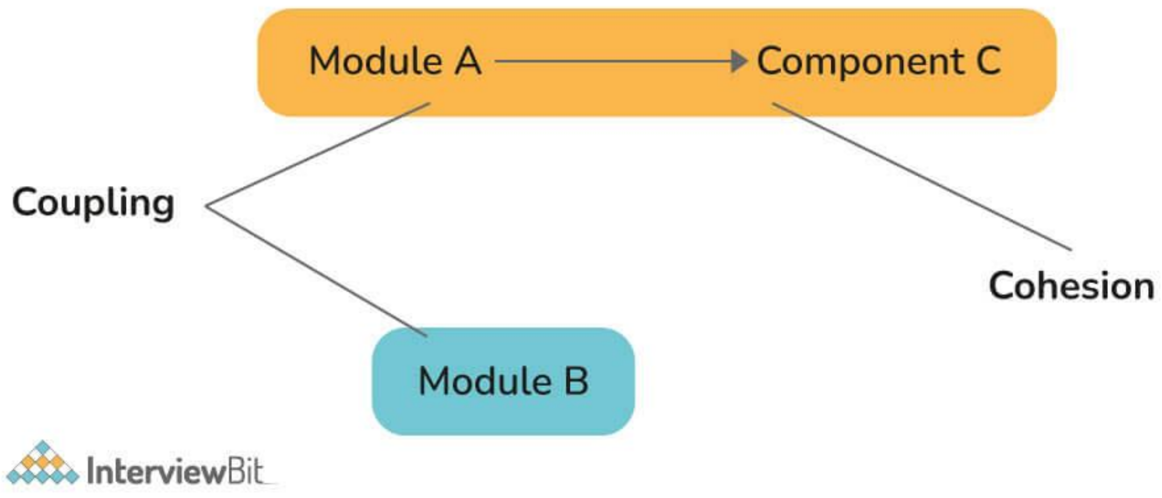
The following problems can be solved with spring cloud:

- **Complicated issues caused by distributed systems:** This includes network issues, latency problems, bandwidth problems, and security issues.
- **Service Discovery issues:** Service discovery allows processes and services to communicate and locate each other within a cluster.
- **Redundancy issues:** Distributed systems can often have redundancy issues.
- **Load balancing issues:** Optimize the distribution of workloads among multiple computing resources, including computer clusters, central processing units, and network links.
- **Reduces performance issues:** Reduces performance issues caused by various operational overheads.

11. What do you mean by Cohesion and Coupling?

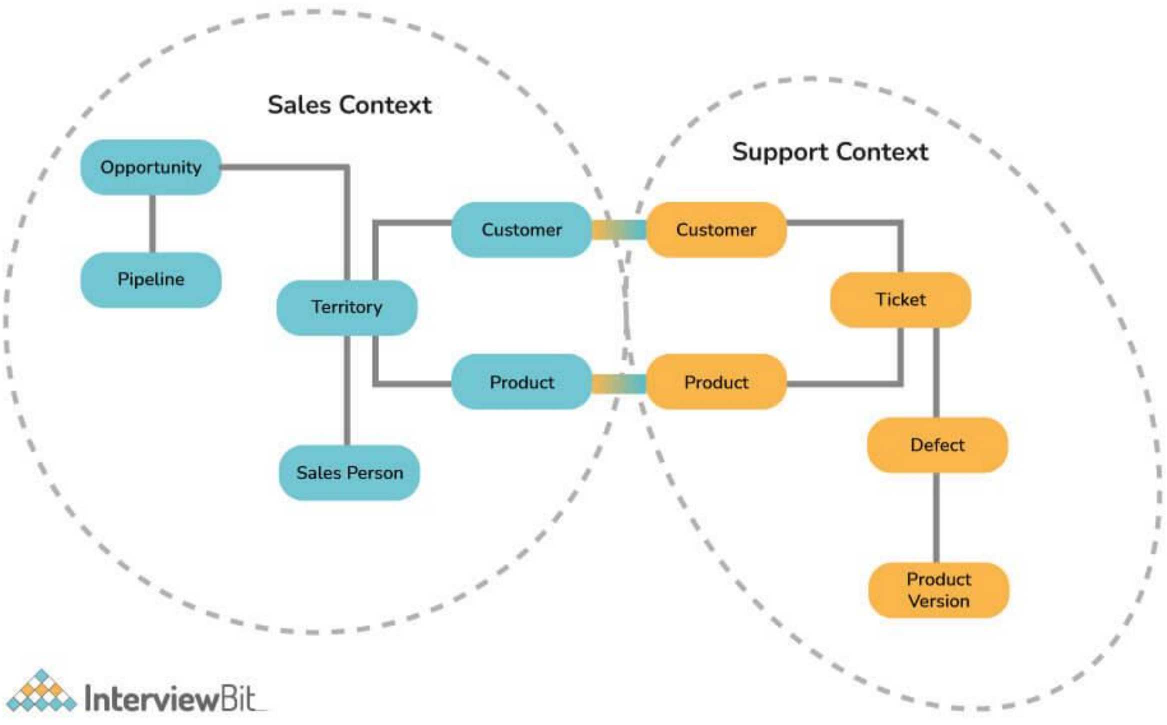
Coupling: It is defined as a relationship between software modules A and B, and how much one module depends or interacts with another one. Couplings fall into three major categories. Modules can be highly coupled (highly dependent), loosely coupled, and uncoupled from each other. The best kind of coupling is loose coupling, which is achieved through interfaces.

Cohesion: It is defined as a relationship between two or more parts/elements of a module that serves the same purpose. Generally, a module with high cohesion can perform a specific function efficiently without needing communication with any other modules. High cohesion enhances the functionality of the module.



12. What do you mean by Bounded Context?

A Bounded Context is a central pattern in DDD (Domain-Driven Design), which deals with collaboration across large models and teams. DDD breaks large models down into multiple contexts to make them more manageable. Additionally, it explains their relationship explicitly. The concept promotes an object-oriented approach to developing services bound to a data model and is also responsible for ensuring the integrity and mutability of said data model.



13. Write the fundamental characteristics of Microservice Design.

- **Based on Business Capabilities**: Services are divided and organized around business capabilities.

- **Products not projects:** A product should belong to the team that handles it.
- **Essential messaging frameworks:** Rely on functional messaging frameworks: Eliminate centralized service buses by embracing the concept of decentralization.
- **Decentralized Governance:** The development teams are accountable for all aspects of the software they produce.
- **Decentralized data management:** Microservices allow each service to manage its data separately.
- **Automated infrastructure:** These systems are complete and can be deployed independently.
- **Design for failure:** Increase the tolerance for failure of services by focusing on continuous monitoring of the applications.
-

14. What are the challenges that one has to face while using Microservices?

The challenges that one has to face while using microservices can be both functional and technical as given below:

Functional Challenges:

- Require heavy infrastructure setup.
- Need Heavy investment.
- Require excessive planning to handle or manage operations overhead.

Technical Challenges:

- Microservices are always interdependent. Therefore, they must communicate with each other.
- It is a heavily involved model because it is a distributed system.
- You need to be prepared for operations overhead if you are using Microservice architecture.
- To support heterogeneously distributed microservices, you need skilled professionals.
- It is difficult to automate because of the number of smaller components. For that reason, each component must be built, deployed, and monitored separately.
- It is difficult to manage configurations across different environments for all components.
- Challenges associated with deployment, debugging, and testing.

15. Explain PACT in microservices.

PACT is defined as an open-source tool that allows service providers and consumers to test interactions in isolation against contracts that have been made to increase the reliability of microservice integration. It also offers support for numerous languages, such as Ruby, Java, Scala, .NET, JavaScript, Swift/Objective-C.

16. Explain how independent microservices communicate with each other.

Communication between microservices can take place through:

- HTTP/REST with JSON or binary protocol for request-response
- Websockets for streaming.
- A broker or server program that uses advanced routing algorithms.

RabbitMQ, Nats, Kafka, etc., can be used as message brokers; each is built to handle a particular message semantic. You can also use Backend as a Service like Space Cloud to automate your entire backend.

17. What do you mean by client certificates?

The client certificate is a type of digital certificate that generally allows client systems to authenticate their requests to remote servers. In many mutual authentication designs, it plays a key role in providing strong assurance of the requestor's identity.

18. Explain CDC.

As the name implies, CDC (Consumer-Driven Contract) basically ensures service communication compatibility by establishing an agreement between consumers and service providers regarding the format of the data exchanged between them. An agreement like this is called a contract. Basically, it is a pattern used to develop Microservices so that they can be efficiently used by external systems.

19. Name some famous companies that use Microservice architecture.

Microservices architecture has replaced monolithic architecture for most large-scale websites like:

- Twitter
- Netflix
- Amazon, etc.

Microservices Interview Questions for Experienced

20. What do you mean by Semantic Monitoring?

The semantic monitoring method, also called synthetic monitoring, uses automated tests and monitoring of the application to identify errors in business processes. This technology provides a deeper look into the transaction performance, service availability, and overall application performance to identify performance issues of microservices, catch bugs in transactions and provide an overall higher level of performance.

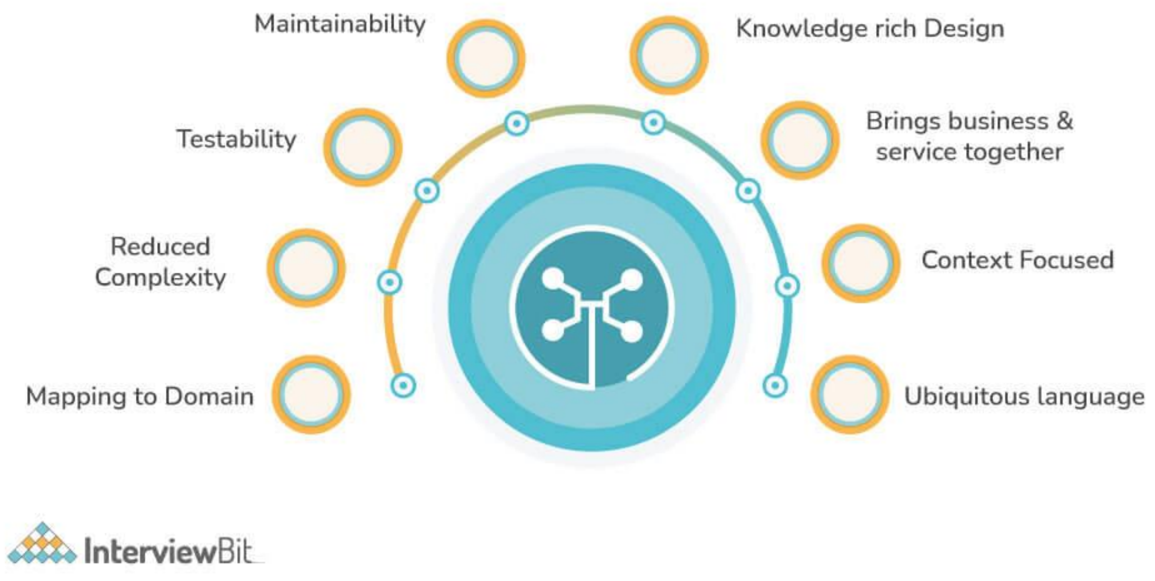
21. Explain continuous monitoring.

Continuous monitoring involves identifying compliance and risk issues in a company's financial and operational environment. It consists of people, processes, and working systems that support efficient and effective operations.

22. What do you mean by Domain driven design?

DDD (Domain-Driven-Design) is basically an architectural style that is based on Object-Oriented Analysis Design approaches and principles. In this approach, the business domain is modeled carefully in software, without regard to how the system actually works. By interconnecting related components of the software system into a continuously evolving system, it facilitates the development of complex systems. There are three fundamental principles underlying it as shown below:

- Concentrate on the core domain and domain logic.
- Analyze domain models to find complex designs.
- Engage in regular collaboration with the domain experts to improve the application model and address emerging domain issues.



23. Explain OAuth.

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 GitHub, Facebook, etc. Using it, you can also share resources on one site with another site without requiring their credentials.

24. What do you mean by Distributed Transaction?

Distributed transactions are an outdated approach in today's microservice architecture that leaves the developer with severe scalability issues. Transactions are distributed to several services that are called to complete the transaction in sequence. With so many moving parts, it is very complex and prone to failure.

25. Explain Idempotence and its usage.

The term 'idempotence' refers to the repeated performance of a task despite the same outcome. In other words, it is a situation in which a task is performed repeatedly with the end result remaining the same.

Usage: When the remote service or data source receives instructions more than once, Idempotence ensures that it will process each request once.

26. What do you mean by end-to-end microservices testing?

Usually, end-to-end (E2E) microservice testing is an uncoordinated, high-cost technique that is used to ensure that all components work together for a complete user journey. Usually, it is done through the user interface, mimicking how it appears to the user. It also ensures all processes in the workflow are working properly.

27. Explain the term Eureka in Microservices.

Eureka Server, 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.

28. Explain the way to implement service discovery in microservices architecture.

There are many ways to set up service discovery, but Netflix's Eureka is the most efficient. This is a hassle-free procedure that doesn't add much weight to the application. It also supports a wide range of web applications. A number of annotations are provided by Spring Cloud to make its use as simple as possible and to hide complex concepts.

29. Explain the importance of reports and dashboards in microservices.

Monitoring a system usually involves the use of reports and dashboards. Using reports and dashboards for microservices can help you:

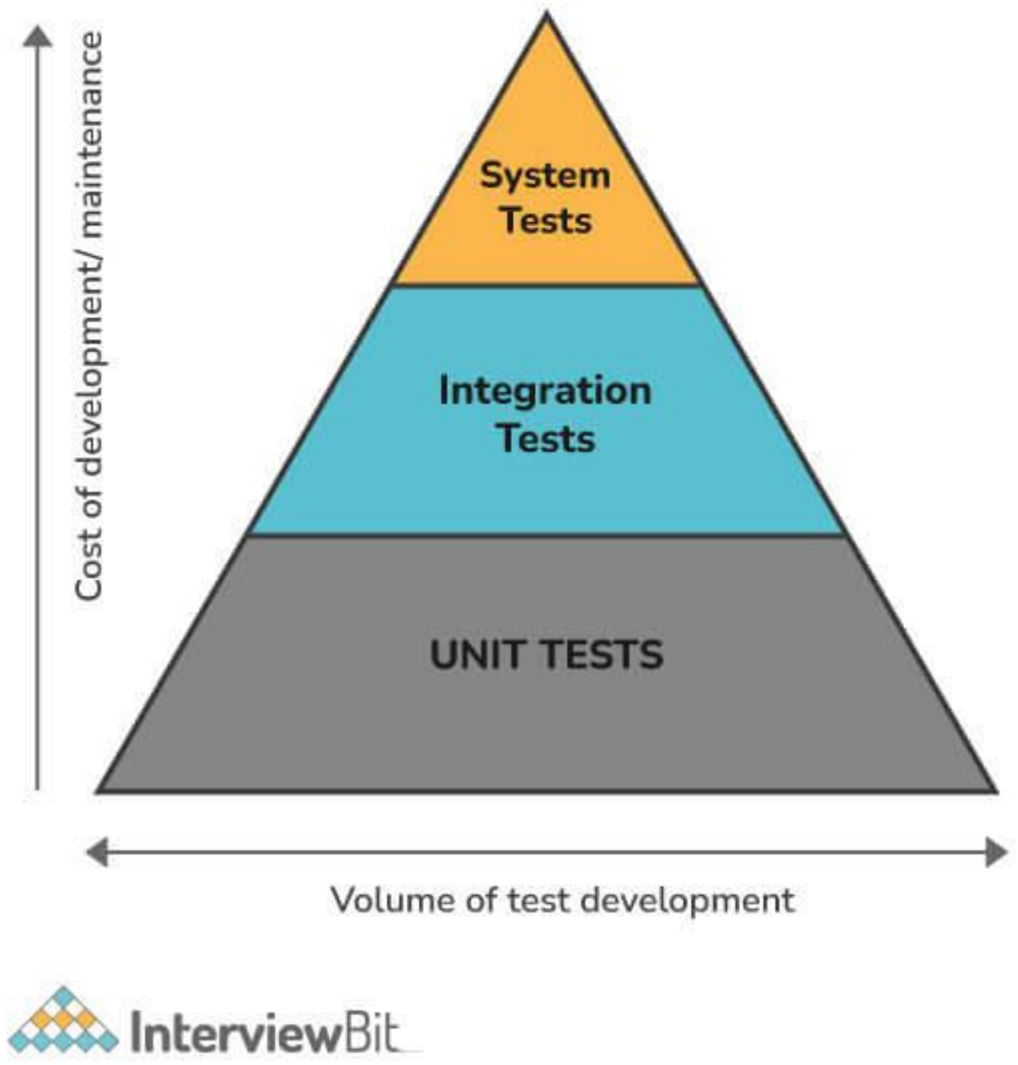
- Determine which microservices support which resources.
- Determine which services are impacted whenever changes are made or occur to components.
- Make documentation easy to access whenever needed.
- Review deployed component versions.
- Determine the level of maturity and compliance from the components.

30. What are Reactive Extensions in Microservices?

A reactive extension, also known as Rx, is basically a design approach that calls multiple services and then generates a single response by combining the results. The calls can either be blocking or not blocking, synchronous or asynchronous. A popular tool in distributed systems, Rx works exactly opposite to legacy flows.

31. Explain type of tests mostly used in Microservices.

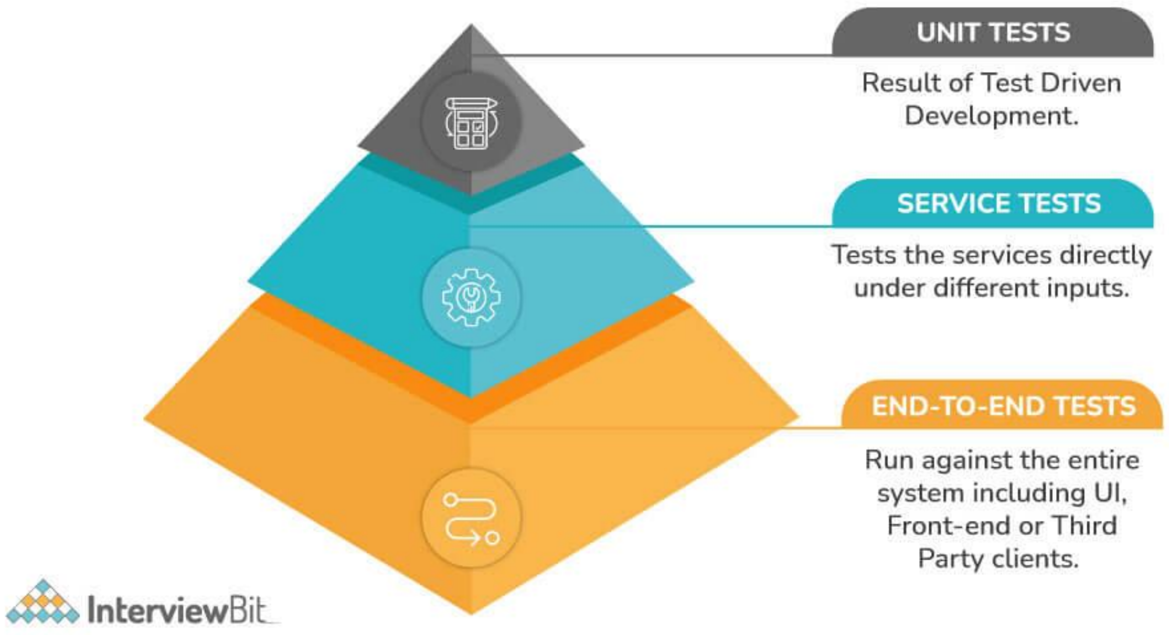
As there are multiple microservices working together, microservice testing becomes quite complex when working with microservices. Consequently, tests are categorized according to their level:



Botton-level tests: The bottom-level tests are those that deal with technology, such as unit tests and performance tests. This is a completely automated process.
Middle-level tests: In the middle, we have exploratory tests such as stress tests and usability tests.
Top-level tests: In the top-level testing, we have a limited number of acceptance tests. The acceptance tests help stakeholders understand and verify the software features.

32. What do you mean by Mike Cohn’s Test Pyramid?

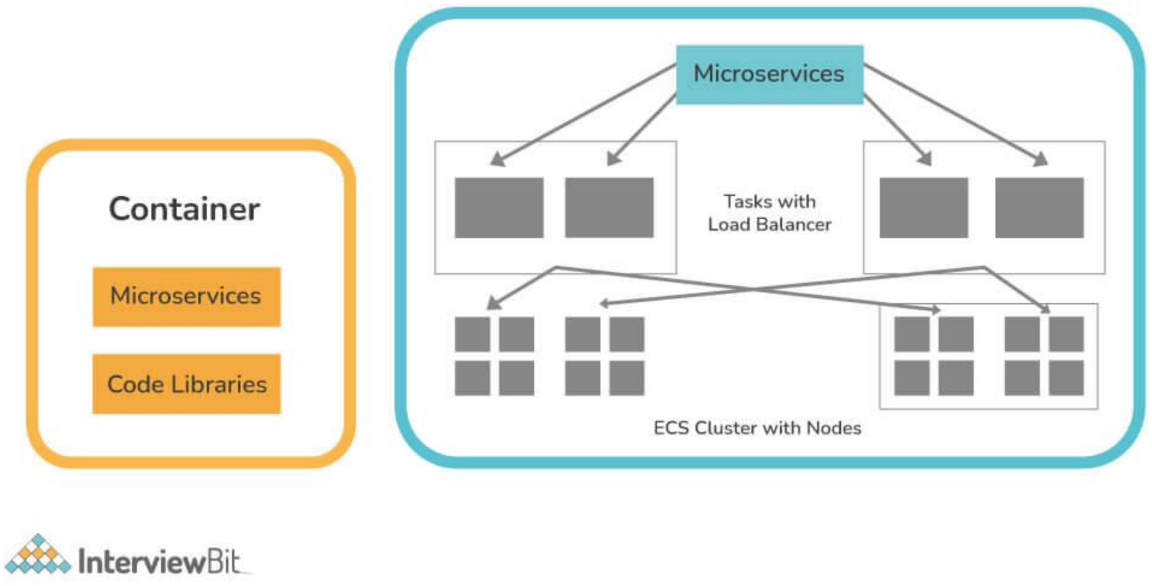
Mike Cohn's Test Pyramid explains the different types of automated tests needed for software development. The test pyramid is basically used to maximize automation at all levels of testing, including unit testing, service level testing, UI testing, etc. The pyramid also states that unit tests are faster and more isolated, while UI tests, which are at the top, are more time-consuming and are centered around integration.


UNIT TESTS
Result of Test Driven Development.
SERVICE TESTS
Tests the services directly under different inputs.
END-TO-END TESTS
Run against the entire system including UI, Front-end or Third Party clients.
InterviewBit

In accordance with the pyramid, the number of tests should be highest at the first layer. At the service layer, fewer tests should be performed than at the unit test level, but greater than that at the end-to-end level.

33. Explain Container in Microservices.

Containers are useful technologies for allocating and sharing resources. It is considered the most effective and easiest method for managing microservice-based applications to develop and deploy them individually. Using Docker, you may also encapsulate a microservice along with its dependencies in a container image, which can then be used to roll on-demand instances of the microservice without any additional work.



34. What is the main role of docker in microservices?

Docker generally provides a container environment, in which any application can be hosted. This is accomplished by tightly packaging both the application and the dependencies required to support it. These packaged products are referred to as Containers, and since Docker is used to doing that, they are called Docker containers. Docker, in essence, allows you to containerize your microservices and manage these microservices more easily.