JAVA DEVELOPER INTERVIEW Q&A

1. Java 100 % pure OOPS Language ?
—> No, it is not. Because it supports primitive data types like int, byte, short etc. which are not objects.

2. What version of Java were you using in the project?
—> Java 11

3. Features of Java
—> 1. Multithreading
    2. Object Oriented Programming
    3. Platform Independent
    4. High Performance

4. What is JIT?
—> The Just-In-Time (JIT) compiler is a component of the runtime environment that improves the performance of Java™ applications by compiling bytecodes to native machine code at run time.

5. Explain JVM, JRE & JDK
—> JVM :- Java Virtual Machine
    JRE :- Java Runtime Environment
    JDK :- Java Development Kit
JDK is for development purpose whereas JRE is for running the java programs.
JDK and JRE both contains JVM so that we can run our java program.
JVM is the heart of java programming language and provides platform independence.

6. Difference between C++ & Java
—>
Java is both a Compiled and Interpreted Language whereas C++ is a Compiled Language.
Java provides built-in support for multithreading whereas C++ doesn't have built-in support for threads, depends on third-party threading libraries.

7. Why is Java a platform independent language?
—> Java is platform independent programming language as Java doesn't require the entire code to be rewritten for all the different platforms. It supports platform independence using Java bytecode and Java Virtual Machine.

8. Difference between Heap & Stack memory in Java
—> The **major difference between Stack memory and heap memory** is that the stack is used to store the order of method execution and local variables while the heap memory stores the objects and it uses dynamic memory allocation and deallocation.

9. Why Strings are immutable?
—> because strings with the same content share storage in a single pool to minimize creating a copy of the same value.

10. Difference between String, String Buffer & String Builder
—> String is immutable whereas StringBuffer and StringBuilder are mutable classes.
StringBuffer is thread-safe and synchronized whereas StringBuilder is not. That's why StringBuilder is faster than StringBuffer.
String concatenation operator (+) internally uses StringBuffer or StringBuilder class.

Feature	String	StringBuilder	StringBuffer
Mutability	Immutable	Mutable	Mutable
Thread Safety	Thread Safe	Not Thread Safe	Thread Safe
Memory Efficiency	High	Efficient	Less Efficient
Performance	High(No-Synchronization)	High(No-Synchronization)	Low(Due to Synchronization)
Usage	This is used when we want immutability	This is used when Thread safety is not required	This is used when Thread safety is required

11. Is String a primitive or derived type in Java ?
—> non-primitive data type

12. Explain String Pool?
—> String Pool in java is a pool of Strings stored in Java Heap Memory.
String literals are stored in it.

13. In Java, how can two strings be compared?
—> To compare strings in Java for equality, you should use String. equals() . If uppercase and lowercase difference isn't important, you can use String. equalsIgnoreCase() .

14. Difference between str1 == str2 and str1.equals(str2)?
—> In Java, str1==str2 will return true if str1 and str2 are strings with the same content. This is because strings with the same content are stored at the same location. str1. equals(str2) will return true if str1 and str 2 are different string objects, but have same content.

15. What are the different string methods in Java?
—> charAt(), concat(), contains(), endsWith(), equals()

16. Is string thread-safe in Java?
—>Immutable objects are by default thread-safe because their state can not be modified once created. Since String is immutable in Java, it's inherently thread-safe. Read-only or final variables in Java are also thread-safe in Java.

17. Why is String used as a HashMap Key in Java?
—> Since String is immutable, its hashcode is cached at the time of creation and it doesn't need to be calculated again. This makes it a great candidate for key in a Map and its processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

18. Why Char Array preferred over String in storing password?
—> So storing a password in a character array clearly mitigates the security risk of stealing a password. With an array, the data can be wiped explicitly data after its work is complete.

19. How to convert String into byte Array?
—>
```
String str = “Gaurab”;
// converting string into byte array by using getBytes()
byte[] arr = str.getBytes();
```

20. How to convert String into Integer and vice versa?
—> We can convert String to an int in java using Integer.parseInt() method. To convert String into Integer, we can use Integer.valueOf() method which returns instance of Integer class.

21. How to convert String into String Builder?
—> To convert a String value to StringBuilder object just append it using the append() method.
```
 String str = “GAURAB”
StringBuilder sb = new StringBuilder();
sb.append(str);
```

22. How to check whether a string is empty?
—> The isEmpty() method checks whether a string is empty or not. This method returns true if the string is empty (length() is 0), and false if not.

23. Features of OOPs in Java
—>
1. Abstraction
2. Polymorphism
3. Inheritance
4. Encapsulation

24. Advantages of using OOPs in Java
—>
1. Locates and Fixes Problems Effortlessly
2. Allows Reuse of Code
3. Offers Security
4. Encourages Scalability

25. What is Class in Java?
—> A class in the context of Java is a template used to create objects and to define object data types and methods

26. What is Object in Java?
—> A Java Object is a member of a class.

27. Difference between local, static & instance variable
—>
Local variables
Temporary variables defined inside methods. They are declared and initialized within that method, and will be made eligible for garbage collection once the method is completed.
For example: in the method “playing with a string,” the string is a local variable.

Instance variables
They are variables that are inherent to an object and that can be accessed from inside any method, constructor or block.
They are destroyed when the object is destroyed.

Class variables (static variable)
Class variables or static variables are declared with the static keyword in a class. They are similar to instance variables, but they are created when the program starts and destroyed when the program stops.The main difference with instance variables is in what scope they are available. A class variable is accessible from an object instance, while an instance variable is not accessible from a class method.

28. What is encapsulation?
—> **Encapsulation in Java** is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.
We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

29. What is polymorphism?
—> **Polymorphism in Java** is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

30. Difference between Compile time & Runtime Polymorphism

Compile Time Polymorphism	Runtime Polymorphism
• It is also known as static polymorphism, early binding, or overloading	• It is also known as dynamic polymorphism, late binding, or overriding
• It is executed at the compile time	• It is executed at the run time
• The method is called with the help of a compiler	• The method is not called with the help of a compiler
• The program's execution is fast as it involves the use of a compiler	• The program's execution is slow as it does not involve a compiler
• It is achieved by function overloading and operator overloading	• It is achieved by virtual overloading and pointers
• Compile-time polymorphism tends to be less flexible as all commands are operated at the compile time • Example: Method Overloading	• Run time polymorphism tends to be more flexible as all commands are executed at the run time

31. What does constructor do?
—> Constructors are methods that are automatically executed every time you create an object. The purpose of a constructor is to construct an object and assign values to the object's members. A constructor takes the same name as the class to which it belongs, and does not return any values.

32. What is inheritance?
—> **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

33. What is Abstraction in Java?
—> Abstraction in Java refers to hiding the implementation details of a code and exposing only the necessary information to the user. Abstraction in Java can be achieved using the following tools it provides :
Abstract classes
Interfaces

34. What is Data hiding?
—> It is hiding internal object details from outside users.

35. Difference between Data hiding & Abstraction in Java

Abstraction	Data Hiding
Focuses on hiding the complexity of the system.	Focuses on protecting the data by achieving encapsulation (a mechanism to wrap up variables and methods
This is usually achieved using abstract class concept, or by implementing interfaces.	This can be achieved using access specifiers, such as private, and protected.
It helps to secure the software as it hides the internal implementation and exposes only required functions.	This acts as a security layer. It keeps the data and code safe from external inheritance as we use setter and getter methods to set and get

It can be implemented by creating a class that only represents important attributes without including	Getters and setters can be used to access the data or to modify it.

36. How much memory does class occupy?
—> Classes don’t occupy memory space.

37. Is it always necessary to create object from class?
—> In Java, it's not always necessary to create an object from a class, but it's a fundamental concept of object-oriented programming (OOP) that's heavily used.There are situations where you might interact with a class without explicitly creating an object, such as accessing static members (variables or methods) of the class. For example:

```
public class MyClass {
    public static int myStaticVariable = 10;

    public static void myStaticMethod() {
        System.out.println("This is a static method.");
    }
}

// Accessing static members without creating an object
int value = MyClass.myStaticVariable;
MyClass.myStaticMethod();
```

Here, you're interacting with the MyClass without creating an instance of it.
Another scenario is when you have nested classes, and you access them through the enclosing class without creating instances of the nested classes.
However, in most cases, you create objects from classes to utilize their attributes and methods.
For example:

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public void greet() {
        System.out.println("Hello, my name is " + name);
    }
}

// Creating an object of Person class
Person person = new Person("John");
person.greet();
```

Here, person is an object of the Person class, and you're calling the greet() method on it.
This is a typical usage pattern in Java programming.

38. What is Constructor?
—> A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

39. Rules for creating Java constructor
There are two rules defined for the constructor.
- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized

39. Types of constructor
—> There are two types of constructors in Java:
- Default constructor (no-arg constructor)
- Parameterized constructor

40. What is copy constructor?
—> A copy constructor in a Java class is a constructor that creates an object using another object of the same Java class.

41. What is destructor?
—> The **destructor** is the opposite of the constructor. The constructor is used to initialize objects while the destructor is used to delete or destroy the object that releases the resource occupied by the object. —Finalize()—

42. Types of inheritance
—> Java supports the following four types of inheritance:
- Single Inheritance
- Multi-level Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

43. What is Sub-Class?
—> A class that is derived from another class is called Sub-Class.

44. What is Interface?
—> An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

45. Difference between Overloading & Overriding

Method Overloading	Method Overriding
Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
It occurs within the class.	It is performed in two classes with inheritance relationships.
Method overloading may or may not require inheritance.	Method overriding always needs inheritance.
In method overloading, methods must have the same name and different signatures.	In method overriding, methods must have the same name and same signature.

46. What is abstract class?
—> A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

47. Difference between Abstract Class & Interface

The difference between abstract class and interface are as follows:

Criteria	Abstract Class	Interface
Final Variables	An abstract class comes with non-final variables.	A Java interface comes with variables declared final by default.
Implementation	An abstract class in Java includes final, non-final, static, and non-static variables.	An interface in Java comes with static and final variables.
Default Implementation	Abstract class helps with interface implementation.	Interface cannot help with the implementation of an abstract class.
Inheritance vs Abstraction	The keyword "implements" is useful for implementing a Java interface.	The keyword "extends" is useful for implementing an abstract class.
Accessibility of Data Members	A Java abstract class comes with private, protected, and other class members.	Variables or members of a Java interface are by default final.
Speed	Faster than a Java Interface	Slower than a Java abstract class
Structure	Abstract method only	Abstract and concrete methods
Data Fields	Includes data fields	Don't include data fields
Inheritance	An abstract class in Java supports multiple inheritance.	An interface in Java does not support multiple inheritance.
Abstract Keyword	Abstract keywords are used for declaring an abstract class.	Inheritance keywords are used for declaring a Java inheritance.
Defining Fields	Users can define fields and constants	Fields cannot be defined.

48. Access modifiers in Java
—> Default, Private, Protected & Public

49. What is Garbage Collection in Java?
—> Garbage collection in Java is the automated process of deleting code that's no longer needed or used. This automatically frees up memory space and ideally makes coding Java apps easier for developers.

50. Islands of isolated objects
—> When two objects 'a', and 'b' reference each other, and they are not referenced by any other object, it is known as island of isolation.
It is a group of objects which reference each other but they are not referenced but other objects of other applications at all.
Example:

```
public class Demo{
  Demo i;
  public static void main(String[] args){
    Demo my_ob_1 = new Demo();
    System.out.println("Demo object one has been created");
    Demo my_ob_2 = new Demo();
    System.out.println("Demo object two has been created");
    my_ob_1.i = my_ob_2;
    my_ob_2.i = my_ob_1;
    my_ob_1 = null;
    my_ob_2 = null;
    System.gc();
  }
  @Override
  protected void finalize() throws Throwable{
    System.out.println("The finalize method has been called on the object");
  }
}
```

51. Finalize in GC
—> The finalize method will be called after the GC detects that the object is no longer reachable, and before it actually reclaims the memory used by the object.
- If an object never becomes unreachable, `finalize()` will never be called on it.
- If the GC doesn't run then `finalize()` may never be called.

52. What is Exception?
—> In Java, Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions.

53. Types of Exception
—>
- Checked Exception :- ClassNotFoundException, IOException, SQLException, FileNotFoundException etc.

- Unchecked Exception :- ArithmeticException, ClassCastException, NullPointerException etc.

54. What is meant by Exception Handling?
—> Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

55. Throwable
—> The Throwable class is the superclass of all errors and exceptions in the Java language

56. Throw and throws
—> The main difference between Throw and Throws keywords in Java is that you can use the Throw keyword to throw an exception explicitly in the code. In contrast, you can use the Throws keyword to declare that a method might throw an exception in the code.

```
if (balance < withdrawAmount) {
   throw new InsufficientBalanceException("You have insufficient balance");
}
```

```
public void withdraw(double amount) throws InsufficientBalanceException {
  if (balance < withdrawAmount) {
      throw new InsufficientBalanceException("You have insufficient balance");
  }
  // ...
}
```

57. How are exception handled in Java?
—> Java provides two different options to handle an exception. You can either use the try-catch-finally approach to handle all kinds of exceptions. Or you can use the try-with-resource approach which allows an easier cleanup process for resources.

58. How do you handle checked exception?
—> These exceptions can be handled by the try-catch block or by using throws keyword otherwise the program will give a compilation error.

59. Difference between Checked & Unchecked Exception

S. No.	Checked Exception	Unchecked Exception
1	Checked exceptions happen at compile time when the source code is transformed into an executable code.	Unchecked exceptions happen at runtime when the executable program starts running.

2	The checked exception is checked by the compiler.	These types of exceptions are not checked by the compiler.
3	Checked exceptions can be created manually.	They can also be created manually.
4	This exception is counted as a sub-class of the class.	This exception happens in runtime, and hence it is not included in the exception class.
5	Java Virtual Machine requires the exception to to be caught or handled.	Java Virtual Machine does not need the exception to be caught or handled.

60. Can you catch and handle multiple exceptions in Java? How?
—> Yes, we can

```
try
{
  statements;
}
catch(ExceptionType1 e1)
{
  statements;
}
catch(ExceptionType2 e2)
{
  statements;
}
```

61. What is Stack trace and how is it related to exception handling?
—> In Java, **stack trace** is nothing but location of the exceptions.

62. What is exception chaining?
—> In Java, a chained exception is an exception that is caused by another exception

63. Can we have statement between try, catch & finally block?
—> We can write statements like try with catch block, try with multiple catch blocks, try with finally block and try with catch and finally blocks and cannot write any code or statements between these combinations. If we try to put any statements between these blocks, it will throw a compile-time error.

64. Different between final, finally & finalize

Final	**Finally**	**Finalize**
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.
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.

65. Difference between ClassNotFoundException & NoClassDefFoundError
—> Java runtime throws ClassNotFoundException while trying to load a class at runtime only and the name was provided during runtime. In the case of NoClassDefFoundError, the class was present at compile time, but Java runtime could not find it in Java classpath during runtime.

66. Unreachable catch block error
—> When we are keeping multiple catch blocks, the order of catch blocks must be from most specific to most general ones. i.e subclasses of Exception must come first and superclasses later. If we keep superclasses first and subclasses later, the compiler will throw an unreachable catch block error.

67. What does JVM do when exception occurs?
—> The JVM starts the search from the method where the exception occurred and then goes up the call stack, checking each method in turn for a catch block that can handle the exception. If a catch block is found, the JVM transfers control to that block and the exception is considered to be handled.

68. What happens when an exception is thrown by main method?
—> Java Runtime terminates the program and print the exception message and stack trace in system console.

69. What happens to the exception object when exception is handled?
—> Exception object will get garbage-collected when it is no longer reachable from any live thread.

70. Different scenario when exception in thread main type of error could occur
—> In real-world scenarios, a program could run into an exception if it tries to access a corrupted/non-existent file or if a network error happens or if the JVM runs out of memory, or if the code is trying to access an index of an array which does not exist and so on.

71. Does finally block always gets executed?
—> The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs.

72. Why is it always recommended to keep clean up activities like closing I/O or DB Connection inside finally?
—> The finally block will always run if an uncaught exception is thrown, but the rest of the code in the method will be skipped.So if you put clean-up code after the finally block, it won't get called if there is an exception.

73. Are we allowed to only use try block without catch and finally?
—> No. Try block must be followed by catch block or finally block or both.

74. Is it possible to throw exception inside lambda expression?
—> Yes, we can handle unchecked exceptions
Example;

```
List<Integer> integers = Arrays.asList(3, 9, 7, 0, 10, 20);
integers.forEach(i -> {
   try {
      System.out.println(50 / i);
   } catch (ArithmeticException e) {
      System.err.println(
        "Arithmetic Exception occured : " + e.getMessage());
   }
});
```

we can handle checked exceptions

75. Rules we should follow when overriding a method throwing an exception
- If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
- If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

76. Difference between Collection & Collections
- Collection is called interface in java whereas Collections is called a utility class in java and both of them can be found in java.util.package.
- Since java 8, collection is an interface with static as well as abstract and default methods whereas collections operate only with static methods.

77. Difference between ArrayList & LinkedList
- ArrayList internally uses dynamic array to store element whereas LinkedList uses doubly linked list to store element.
- ArrayList can act as a List only because it implements List only whereas LinkedList can act as both List and Queue because it implements List & Deque Interface.
- ArrayList is better for storing and accessing data whereas LinkedList is better for manipulating the data.

78. Difference between ArrayList & Vector

ArrayList	**Vector**
ArrayList is not synchronized.	Vector is synchronized.
ArrayList is **not a legacy** class. It is introduced in JDK 1.2.	Vector is a **legacy** class.
ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized

79. Difference between List & Set

List	**Set**	**Map**
The elements can be duplicated in the list interface.	Duplicate elements are not permitted in a set.	Duplicate elements are not allowed on the map.
The list preserves the order of inclusion.	There is no insertion order maintained by sets.	Furthermore, there is no insertion order maintained by the map.
LinkedList and Array List are classes used for list implementation.	The classes used to implement sets include HashSet, LinkedHashSet, and TreeSet.	HashMap, HashTable, ConcurrentHashMap, and LinkedHashMap are the different map implementation classes.

80. Difference between HashSet & TreeSet & LinkedHashSet

HashSet	TreeSet	LinkedHashSet
HashSet uses HashMap internally to store it's elements.	TreeSet uses TreeMap internally to store it's elements.	LinkedHashSet uses LinkedHashMap internally to store it's elements.
HashSet gives better performance than the LinkedHashSet and TreeSet.	TreeSet gives less performance than the HashSet and LinkedHashSet as it has to sort the elements after each insertion and removal operations.	The performance of LinkedHashSet is between HashSet and TreeSet. It's performance is almost similar to HashSet. But slightly in the slower side as it also maintains LinkedList internally to maintain the insertion order of elements.
HashSet allows maximum one null element.	TreeSet doesn't allow even a single null element. If you try to insert null element into TreeSet, it throws NullPointerException.	LinkedHashSet also allows maximum one null element.

HashSet	TreeSet	LinkedHashSet
Use HashSet if you don't want to maintain any order of elements.	Use TreeSet if you want to sort the elements according to some Comparator.	Use LinkedHashSet if you want to maintain insertion order of elements.

81. Can you add null element in TreeSet or HashSet?
- HashSet allows maximum one null element.
- TreeSet doesn't allow even a single null element. If you try to insert null element into TreeSet, it throws NullPointerException.

82. What is Priority Queue?
—> A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a Queue follows the First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that's when the PriorityQueue comes into play.

83. Difference between HashSet & HashMap

HashMap	**HashSet**
In HashMap we store a **key-value pair**. It maintains the mapping of key and value.	In HashSet, we store **objects**.
It does not allow **duplicate keys**, but **duplicate values** are **allowed**.	It does not allow **duplicate values**.
It can contain a single null key and multiple null values.	It can contain a single null value.
HashMap uses the **put()** method to add the elements in the HashMap.	HashSet uses the **add()** method to add elements in the HashSet.

84. Internal working of HashMap?
Here is a step-by-step explanation of how a hashmap works internally:
- When a key-value pair is inserted into the hashmap, the hashmap computes a hash code for the key using the hash function.
- The hash code is used to determine the index in the array where the key-value pair should be stored. The hash code is typically used to compute the remainder of the hash code divided by the size of the array, which ensures that the index is within the bounds of the array.
- If the index is already occupied by another key-value pair, a collision has occurred. There are different ways to handle collisions, but a common approach is to use a linked list or a similar data structure to store multiple key-value pairs at the same index.
- When retrieving a value based on a key, the hashmap computes the hash code for the key and uses it to find the index in the array where the value is stored.
- If the index contains a linked list or a similar data structure, the hashmap searches through the list to find the key-value pair with the matching key.
- If the key is not found, the hashmap returns null or throws an exception, depending on the implementation.
- To ensure efficient performance, the hashmap typically uses a load factor to determine when to resize the array. When the number of key-value pairs exceeds the product of the load factor and the size of the array, the hashmap creates a new, larger array and rehashes all the key-value pairs into the new array.

85. Difference between ArrayList & Array
—> An array is a fixed-length data structure. ArrayList is a variable-length data structure. It can be resized itself when needed.

86. How can you make ArrayList read-only?
—> Here we have converted the existing list fruitList to unmodifiable List. If we alter the "unmodifiableList", it will cause UnsupportedOperationException.

```
public class UnmodifiableArrayList {
public static void main(String[] args) {
List<String>fruitList = new ArrayList<String>();
fruitList.add("Mango");
fruitList.add("Banana");
fruitList.add("Apple");
fruitList.add("Strawberry");
fruitList.add("Pineapple");
List<String>unmodifiableList= Collections.unmodifiableList(fruitList);
unmodifiableList.add("INDIA");
System.out.println(fruitList);
}
}
```

87. Difference between Comparable & Comparator

Comparable	Comparator
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.
Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.

88. Explain BlockingQueue.
—> BlockingQueue is a java Queue that support operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.

89. Difference between fail-fast & fail-safe iterator
—> 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.
- The Fail Fast system is a system that shuts down immediately after an error is reported. All the operations will be aborted instantly in it.
- The Fail Safe is a system that continues to operate even after an error or fail has occurred. These systems do not abort the operations instantly; instead, they will try to hide the errors and will try to avoid failures as much as possible.

90. RandomAccess Interface? Which collection type implements this?
—> ArrayList implements RandomAccess interface.
The RandomAccess interface is a marker interface in Java, which means it doesn't declare any methods of its own. Instead, it serves as a marker to indicate that a class implementing it supports efficient random-access operations.

91. Difference between Iterator & Enumeration

Iterator	**Enumeration**
Iterator has the remove() method.	Enumeration does not have the remove() method.
Iterator is not a legacy interface. Iterator can be used for the traversal of HashMap, LinkedList, ArrayList, HashSet, TreeMap, TreeSet .	Enumeration is a legacy interface which is used for traversing Vector, Hashtable.

92. Difference between HashMap & HashTable

HashMap	**HashTable**
Non-synchronized	Synchronized
Allows one null key and multiple null values.	Does not permit null keys or values. Inserting null can lead to a NullPointerException.
The iterator is fail-fast. Throws ConcurrentModificationException if modified by another thread during iteration.	Enumerator is not fail-fast. Due to internal synchronization, concurrent modification risks are minimized during enumeration.

93. Why does HashMap allows null where as HashTable doesn’t allow null?
—> A HashTable stores the object in the key, value pair. In order to store and retrieve the object successfully, the object which is used as a key must implement the hashCode method and the equals method. Since null is not an object, it cannot implement these hashCode method and the equals method. So if we store null inside the HashTable, it will not work and throw a null pointer exception error. However, HashMap is a modern version of HashTable and it was created later. It will allow one null key and any number of null values.

94. How can you synchronise ArrayList?
—> We can use Collections. synchronizedList(List<T>) method to synchronize collections in java. The synchronizedList(List<T>) method is used to return a synchronized (thread-safe) list backed by the specified list.

95. Why do we need synchronised ArrayList when we do have vector which is synchronised?
—> ArrayList is faster. Since it is non-synchronized, while vector operations give slower performance since they are synchronized (thread-safe)

96. Difference between HashMap & TreeMap

HashMap	**TreeMap**
Java **HashMap** is a hashtable based implementation of Map interface.	Java **TreeMap** is a Tree structure-based implementation of Map interface.
HashMap allows a **single** null key and **multiple** null values.	TreeMap does not allow **null** keys but can have **multiple** null values.
HashMap is **faster** than TreeMap because it provides constant-time performance that is O(1) for the basic operations like get() and put().	TreeMap is **slow** in comparison to HashMap because it provides the performance of O(log(n)) for most operations like add(), remove() and contains().
HashMap does not maintain any order.	The elements are sorted in **natural order** (ascending).

97. Concurrent HashMap
—> ConcurrentHashMap is a thread-safe implementation of the Map interface in Java, which means multiple threads can access it simultaneously without any synchronization issues.One of the key features of the ConcurrentHashMap is that it provides fine-grained locking, meaning that it locks only the portion of the map being modified, rather than the entire map. This makes it highly scalable and efficient for concurrent operations.

98. What do you mean by Multithreading? Why is it important?
—> Multithreading in Java is **a process of executing multiple threads simultaneously**. A thread is a lightweight sub-process, the smallest unit of processing. You can perform many operations together, so it saves time.

99. Benefits of using Multithreading
- You can perform many operations together, so it saves time.
- Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.
- It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

100. What is Thread in Java?
—> A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution. Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

101. What are the two ways of implementing thread?
—> Two ways of creating thread :
- By extending Thread class
- By implementing Runnable interface.

102. Life cycle of Thread
—> In Java, a thread always exists in any one of the following states. These states are:
- New
- Active
- Blocked / Waiting
- Timed Waiting
- Terminated

New: Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

Active: When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

Runnable: A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state. A program implementing multithreading acquires a fixed slice of time to each individual thread. Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time. Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. In the runnable state, there is a queue where the threads lie.

Running: When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

Blocked or Waiting: Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

Timed Waiting: Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B. Thus, thread lies in the waiting state for a specific span of time, and not forever. A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state. After the time runs out, the thread wakes up and start its execution from when it has left earlier.

Terminated: A thread reaches the termination state because of the following reasons:
- When a thread has finished its job, then it exists or terminates normally.
- **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respawn (active after kill) the dead thread.

103. Difference between Thread & Process

Process	**Thread**
Process means any program is in execution.	Thread means a segment of a process.
It takes more time for creation.	It takes less time for creation.

104. Difference between Class Lock & Object Lock

Class Lock	**Object Lock**
This lock is used to make static data thread-safe.	This lock is used to make non-static data thread-safe.

We can get a class level lock as follows:
public class GFG {
  public void m1( ) {
     synchronized (GFG.class) {
    // some line of code
  }
}
	We can get object level lock as follows:
	public class GFG {
	  public void m1( ) {
	     synchronized (this) {
	        // some line of code
	     }
	  }
	}

105. Difference between User Thread & Daemon Thread

User Thread	**Daemon Thread**
user threads are high priority threads which always run in foreground	It is low priority threads which always run in background
JVM wait for user thread to finish its task.	JVM doesn't wait for daemon thread to finish its task.
User thread is usually created by the application for executing some task concurrently	daemon thread is mostly created by JVM like for some garbage collection job.

106. How to create Daemon Thread?
—> Daemon threads are created using the `setDaemon(true)` method of the `Thread` class.

```
// Java program to demonstrate the usage of
// setDaemon() and isDaemon() method.

public class DaemonThread extends Thread
{
        public DaemonThread(String name){
                super(name);
        }

        public void run()
        {
                // Checking whether the thread is Daemon or not
                if(Thread.currentThread().isDaemon())
                {
                        System.out.println(getName() + " is Daemon thread");
                }

                else
                {
                        System.out.println(getName() + " is User thread");
                }
        }

        public static void main(String[] args)
        {

                DaemonThread t1 = new DaemonThread("t1");
                DaemonThread t2 = new DaemonThread("t2");
                DaemonThread t3 = new DaemonThread("t3");

                // Setting user thread t1 to Daemon
                t1.setDaemon(true);

                // starting first 2 threads
                t1.start();
                t2.start();

                // Setting user thread t3 to Daemon
                t3.setDaemon(true);
                t3.start();
        }
}
```

107. What are wait() and sleep() method?

wait()	**sleep()**
It is used for thread synchronization.	It is used for thread timing
Time independent	Time-dependent

wait()	sleep()
It must be called within a synchronized block	It can be called anywhere in the code

108. Difference between notify() & notifyAll()

notify()	notifyAll()
In the case of the multiThreading, notify() method sends the notification to only one thread among the multiple waiting threads which are waiting for the send lock.	While notifyAll() methods in the same context send notifications to all waiting threads instead of a single thread.
In the case of notify() method, the risk of thread missing is high as notification is sent only a single thread, and if it misses that, then no other thread would get a notification and hence the lock.	While in the case of notifyAll(), it sends a notification to all the waiting threads, and hence if any thread misses the notification, there are other threads to do the job. Hence the risk is less.

109. Why wait(), notify() and notifyAll() methods are present in Object class?
—> Wait(), notify() and notifyAll() method being in Object class allows all the threads created on that object to communicate with other. [As multiple threads may exist on same object].

110. Difference between Runnable & Callable Interface

Runnable	Callable
It cannot throw a checked Exception.	It can throw a checked Exception.
In a runnable interface, one needs to override the run() method in Java.	In order to use Callable, you need to override the call()

111. What is start() & run() method of Thread?
—> When a program calls the start() method, a new thread is created and then the run() method is executed. But if we directly call the run() method then no new thread will be created and run() method will be executed as a normal method call on the current calling thread itself and no multi-threading will take place.

```
class MyThread extends Thread {
        public void run(){
                System.out.println("Current thread name: "
                                                + Thread.currentThread().getName());
                System.out.println("run() method called");
        }}

class GeeksforGeeks {
        public static void main(String[] args){
                MyThread t = new MyThread();
                t.start();
}}
```

```
class MyThread extends Thread {
        public void run()
        {
                System.out.println("Current thread name: "
                                                + Thread.currentThread().getName());

                System.out.println("run() method called");
        }
}

class GeeksforGeeks {
        public static void main(String[] args)
        {
                MyThread t = new MyThread();
                t.run();
        }
}
```

112. Explain Thread Pool?
—> Thread pools can help to improve the performance of your applications by reusing threads and avoiding the overhead of creating new threads each time a task is executed.

113. What is the purpose of join() method?
—> The join() method in Java is provided by the java.lang.Thread class that permits one thread to wait until the other thread to finish its execution. Suppose *th* be the object the class Thread whose thread is doing its execution currently, then the *th.join();* statement ensures that *th* is finished before the program does the execution of the next statement.

114. What is deadlock and when it occurs?
—> Deadlock in java is a programming situation where two or more threads are blocked forever. Java deadlock situation arises with at least two threads and two or more resources.

115. What is Volatile Keyword?
—> The volatile keyword in Java is used to indicate that a variable's value may be modified by multiple threads concurrently.

116. What is Transient Keyword?
—> Transient variables are special types of variables created by using the transient keyword.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**.

117. How do threads communicate with each other?
—> Inter-thread communication involves the communication of threads with each other. The three methods that are used to implement inter-thread communication in Java.

- wait() : This method causes the current thread to release the lock. This is done until a specific amount of time has passed or another thread calls the notify() or notifyAll() method for this object.

- notify() : This method wakes a single thread out of multiple threads on the current object’s monitor. The choice of thread is arbitrary.

- notifyAll() : This method wakes up all the threads that are on the current object’s monitor.

```
class BankClient {
  int balAmount = 5000;
  synchronized void withdrawMoney(int amount) {
    System.out.println("Withdrawing money");
    balAmount -= amount;
    System.out.println("The balance amount is: " + balAmount);
  }
  synchronized void depositMoney(int amount) {
    System.out.println("Depositing money");
    balAmount += amount;
    System.out.println("The balance amount is: " + balAmount);
    notify();
  }
}
public class ThreadCommunicationTest {
  public static void main(String args[]) {
    final BankClient client = new BankClient();
    new Thread() {
      public void run() {
        client.withdrawMoney(3000);
      }
    }.start();
    new Thread() {
      public void run() {
       client.depositMoney(2000);
      }
    }.start();
  }
}
```

OUTPUT:
```
Withdrawing money
The balance amount is: 2000
Depositing money
The balance amount is: 4000
```

118. Can two threads execute two methods (static & non-static concurrently)?
—> Yes, We can access both the methods at a single point of time. One method (static one) associated to Class object and other one (non static) associated to the instance object of the class. So we can actually access both at a time as there are two different objects and two different locks associated to the process.
e.g. A class ABC is there having two methods .
```
a). public static synchronized void check() {
}
b.). public synchronized void process(){
}
```

SO two threads A and B simultaneously can access the individual methods at a single point of time.
As threads concept is on locking of the objects. One keep a lock on the class and another one on the instance object.

119. What is synchronisation Process ? Why use it?
—> Synchronization in Java is the process that allows only one thread at a particular time to complete a given task entirely.

120. What is synchronised method and synchronised block ? Which is preferred?
—> Synchronized block is used to prevent multiple threads from executing a portion of a code in a method at the same point in time. On the other hand, synchronized method will prevent multiple threads from executing the entire method at the same point in time.

121. What is thread starvation?
—> Thread starvation is a situation where a thread is unable to access the CPU or other resources it needs to perform its task, because other threads have higher priority or hold locks on those resources.

122. What is LiveLock? What happens when it occurs?
—> Livelock is another concurrency problem and is similar to deadlock. In livelock, two or more threads keep on transferring states between one another instead of waiting infinitely.

123. Can you start a thread twice?
—> No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown.

124. Explain context switching
—> The process of context switching involves the storage of the context/state of a given process in a way that it can be reloaded whenever required, and its execution can be then resumed from the very same point as earlier.

125. What is cyclicBarrier & countDownLatch?
—> CountDownLatch: A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads is complete.

CyclicBarrier: A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point.

126. What is Thread Scheduler & Time Slicing?
—> A component of Java that decides which thread to run or execute and which thread to wait is called a **thread scheduler in Java**.
Priority of each thread lies between 1 to 10. If a thread has a higher priority, it means that thread has got a better chance of getting picked up by the thread scheduler.

127. What is shutdown hook?
—> A shutdown hook is simply an initialized but unstarted thread.

128. What is busy spinning?
—> Busy Spinning is a wait strategy in which one thread waits for some condition to happen which is to be set by some other thread.

129. What is ConcurrentHashMap & HashTable? Why is ConcurrentHashMap faster than HashTable?
—> A ConcurrentHashMap is one implementation of the ConcurrentMap interface, and it's one of the thread-safe collections that Java provides.

The Hashtable class in Java is one of the oldest members of the Java Collection Framework. A hash table is an unordered collection of key-value pairs, with a unique key for each value

ConcurrentHashMap uses multiple locks on segment level while Hashtable uses single locks for whole data. ConcurrentHashMap belongs to the Executor framework which are generally faster in single threaded environment while Hashtable is belongs to the Collection framework.

130. Explain thread priority
—> Java thread priority is an integer property of threads that determines the order in which threads are scheduled for execution.

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

131. What do you mean by ThreadLocal Variable?
—> ThreadLocal is a class in Java that allows you to create variables that can only be read and written by the same thread. This can be useful in situations where you have multiple threads accessing the same variable, but you want to ensure that each thread has its own isolated copy of the variable.

132. What is semaphore?
—> In Java's concurrency API, a semaphore is another synchronization tool that simultaneously controls the number of threads accessing a particular resource or section of code. It manages a set of permits; threads must acquire a permit before proceeding.

133. Explain Thread Group? Why we shouldn’t use it?
—> A ThreadGroup represents a set of threads.
We shouldn’t use it because many of the methods of the ThreadGroup class (for example, allowThreadSuspension() , resume() , stop() , and suspend()) are deprecated.

134. What is ExecutorService Interface?
—> ExecutorService is a JDK API that simplifies running tasks in asynchronous mode. Generally speaking, ExecutorService automatically provides a pool of threads and an API for assigning tasks to it.

135. What will happen if we don’t override thread class run() method ?
—> When we call start() method on thread, it internally calls run() method with newly created thread. So, if we don't override run() method newly created thread won't be called and nothing will happen.

136. What is lock interface? Why is it better to use lock interface rather than synchronised block?
—> The lock interface ensures the sequence in which the waiting thread will be given access. Synchronized block doesn't handle it. It provides options of timeout when the lock is not granted.

137. Is it possible to call run() method directly to start a new thread?
—> No, it will not start a new thread, It is not possible to start a new thread by calling run() method because JVM will not create a new thread until you call the start method. If you call the run() method directly than it will be called on the same thread.

138. Is it possible that each thread can have its stack in multithreading programming?
—> Yes. Its true

139. Features of Java 8
—> They are:
- Lambda expression
- Stream API
- Default method
- Optional Class
- Functional Interface

140. What is Functional Interface? (SAM —> Single Abstract Method)
—> A functional interface is an interface that contains only one abstract method. It can have any number of default, static methods but can contain only one abstract method.

141. Can Functional Interface extend/inherit another interface?
—> A functional interface can't extend another interface which has an abstract method, because it will void the fact that a functional interface allows only one abstract method, however functional interface can inherit another interface if it contains only static and default methods in it.

142. What is default method? Why is it used?
—> Default methods are methods that can have a body. The most common use of interface default methods is to incrementally provide additional functionality to a given type without breaking down the implementing classes.

143. What are static methods in Interface?
—> Static methods were introduced in Java 8 to allow interfaces to define methods that belong to the interface itself, not to any instance of the interface

144. Name some Functional Interface?
—> Runnable, Callable, Comparator, Comparable

145. What is Lambda expression?How is it related to functional interface?
—> Lambda expressions are nothing but the implementation of the functional interface. A functional interface is nothing but an interface that has only one abstract method. To declare an interface as a functional interface, Java provides an annotation @FunctionalInterface.

```
// this is functional interface
@FunctionalInterface
interface MyInterface{

   // abstract method
   double getPiValue();
}
```

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

    // declare a reference to MyInterface
    MyInterface ref;

    // lambda expression
    ref = () -> 3.1415;

    System.out.println("Value of Pi = " + ref.getPiValue());
    }
}
```

146. What is the syntax of Lambda Expression?

```
() -> {
// Add some functionality like print some statements
}
```

147. What is Method Reference?
—> Java came up with a new feature called method reference in Java 8, which is basically a compact and easy form of the lambda expression. Java 8 method reference is used to refer method of functional interface.

148. What is Optional Class?
—> The Optional class in Java 8 is a container object which is used to contain a value that might or might not be present.

149. Advantage of using Optional Class
—> One of the key benefits of using Optional is that it forces you to handle the case where the value is absent. This means that you are less likely to miss important checks in your code and reduces the risk of NullPointerException. If a value is not present, you can either provide a default value or throw an exception.

150. What is Java 8 Stream?
—> A stream represents a sequence of elements and supports different operations (Filter, Sort, Map, and Collect) from a collection.

151. How are Collections different from Stream?
—> Collections store all their elements in memory, whereas streams do not necessarily store all their elements in memory

152. What is the feature of new Date and Time in Java 8?
—> The new date-time API is immutable and does not have setter methods.

153. Difference between findFirst() & findAny()
—> The findAny() method returns any element from a Stream, while the findFirst() method returns the first element in a Stream.

```
Optional<Integer> result = IntStream.range(1, 10)
        .filter(num -> num % 2 == 0)
        .findFirst();

if (result.isPresent()) {
    System.out.println("The first even number is " + result.get());
} else {
    System.out.println("There are no even numbers in the stream");
}
```

———————-

```
Optional<Integer> result = IntStream.range(1, 10)
        .filter(num -> num % 2 == 0)
        .findAny();

if (result.isPresent()) {
    System.out.println("An even number is " + result.get());
} else {
    System.out.println("There are no even numbers in the stream");
}
```

154. Features of Java 11
—> They are:
- HTTP Client
- New String Methods (isBlank, lines, strip, stripLeading, stripTrailing, and repeat)

155. What is Spring Framework?
—> The Spring Framework is an application framework and inversion of control container for the Java platform.

156. Features of Spring Framework?
—> They are:
- Spring MVC
- Dependency Injection
- Loose Coupling

157. What is Spring Configuration File? —> XML file
—> Spring bean configuration file contains spring bean configurations, dependent value configurations, and other miscellaneous configurations. Any name can be given to Spring Bean configuration file with .xml extension.

158. What is IOC Container? Inversion Of Control
—> Spring IOC is a container which creates and manages the objects looking into the configuration file for the classes mentioned in it.

159. What do you mean by Dependency injection?
—> Injecting a value into variables of Java Class.
Two Types of DI:-
1. Setter Dependency Injection:- Using setter method
2. Constructor Dependency Injection:- Using Constructor

DI uses setter method internally.

160. Difference between Constructor & Setter Injection
- Setter injection overrides the constructor injection. If we use both constructor and setter injection, IOC container will use the setter injection.
- We can easily change the value by setter injection. It doesn't create a new bean instance always like constructor. So setter injection is flexible than constructor injection.

161. What is Spring Bean?
—> Spring bean are the objects created by IOC and stored there and ready for using directly in our methods using get bean method of Spring .

162. Bean Scopes available in Spring
—> Spring Bean Scope:-
1. Singleton : Bean will be created once and reused throughout the application
2. Prototype : Bean will be created whenever requested.
3. Request : Spring will create bean whenever HTTP request is called.
4. Session : Spring will create bean for each HTTP session
5. Application : Bean will be created only once
6. Web Socket : Bean will be created only once for web socket environment

Usage: @Bean(“singleton”)

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

164. What is Bean Wiring?
—> Configuring beans and their properties for dependency injection in spring configuration file (.xml) and attaching it to its corresponding object is known as Bean Wiring.

165. What is auto wiring ? Types of Autowiring
—> Auto-wiring, also known as automatic wiring, is a feature of the Spring Framework that automatically wires beans together by matching the types of the properties, constructor arguments, or method arguments of the beans.

Types of Auto-wiring
- no : It is the default autowiring mode. It means no autowiring bydefault.

- byName : The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.

- byType : The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.

- Constructor : The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.

- Autodetect : It is deprecated since Spring 3.

166. Advantage of using Spring Boot over Spring
—>
- You can use it to create standalone applications.
- It doesn't require XML configuration.
- Embeds Tomcat Server

167. Features of Spring Boot
- You can use it to create standalone applications.
- It doesn't require XML configuration.
- Embeds Tomcat Server

168. Explain @SpringBootApplication
—> `@SpringBootApplication` annotation can be used to enable those three features, that is:
- `@EnableAutoConfiguration`: enable Spring Boot’s auto-configuration mechanism
- `@ComponentScan`: enable `@Component` scan on the package where the application is located (see the best practices)
- `@Configuration`: allow to register extra beans in the context or import additional configuration classes

https://www.turing.com/interview-questions/spring-boot

169. What are the effects of running Spring Boot Application as Java Application?
—> When a Spring Boot application is "Run as Java Application," the following happens in the background:

- The main method of the Spring Boot application's entry point class is executed.
- 2. Spring Boot's SpringApplication class is invoked, which initializes and starts the application context.

3. The application context is configured, including component scanning to detect beans, auto-configuration, and loading external configurations.
4. The embedded server (e.g., Tomcat, Jetty) is started, allowing the application to handle incoming requests.
5. Application-specific beans and dependencies are instantiated and wired together via dependency injection.
6. The application context is refreshed, performing any necessary initialization and configuration.
7. Any initialization callbacks or lifecycle events defined in beans are triggered.
8. The application starts listening for incoming requests on the configured port and is ready to process them.

170. Can we change default port number of embedded tomcat server?
—> Yes, we can change the port number of server in application.properties file using server.port

171. Difference between @Controller & @RestController

@Controller	**@RestController**
@Controller is used to mark classes as Spring MVC Controller.	@RestController annotation is a special controller used in RESTful Web services, and it's the combination of @Controller and @ResponseBody annotation.
In @Controller, we can return a view in Spring Web MVC.	In @RestController, we can not return a view.
In @Controller, we need to use @ResponseBody on every handler method.	In @RestController, we don't need to use @ResponseBody on every handler method.

172. Difference between @PathVariable & @RequestParam
- The @PathVariable annotation is used to retrieve data from the URL path.
- On the other hand, the @RequestParam annotation enables you to extract data from the query parameters in the request URL

173. Explain @RequestMapping, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping
- @RequestMapping : The @RequestMapping annotation can be applied to class-level and/or method-level in a controller
- @GetMapping : used to retrieve data from db.
- @PostMapping : used to insert/save data in db
- @PutMapping : used to update existing data in db
- @DeleteMapping : used to delete existing data from db

174. Explain @Autowired, @Bean, @Configuration, @RequestBody, @Valid , @Qualifier
- @Autowired : used to perform dependency injection on beans
- @Bean : used to indicate that a method instantiates, configures, and initializes a new object to be managed by the Spring IoC container
- @Configuration : Tags the class as a source of bean definitions for the application context.

- @RequestBody : indicates that Spring should deserialize a request body into an object
- @Valid : When you apply the @Valid annotation to a method parameter, Spring Boot automatically triggers validation for that parameter before the method is invoked
- @Qualifier : This annotation is useful when we want to specify which bean of a certain type should be injected by default

175. Explain Bean Validation
—> It is done at entity classes. To validate the java variables.

176. Explain @Repository, @Service, @RestControllerAdvice, @ExceptionHandler
- @Repository : used to indicate the class as repository/dao class
- @Service : It is a stereotype. used to indicate the class as service implementation class
- @RestControllerAdvice : used to create a global exception handler in a Spring Boot application
- @ExceptionHandler : used to handle the specific exceptions and sending the custom responses to the client
- @Entity : used to indicate the class as entity/model class

177. What is Hibernate Framework?
—> It is a ORM (Object Relational Mapping) tool.

178. Advantage of Hibernate over JDBC
- Hibernate provides good support for lazy loading whereas JDBC does not support lazy loading.
- Hibernate manages exceptions itself by marking them as unchecked whereas JDBC code needs to be written in a try-catch block as it throws checked exceptions.

179. What are the some important interface of Hibernate?
- Session: It maintains a connection between the hibernate application and database. It provides methods to store, update, delete or fetch data from the database such as persist(), update(), delete(), load(), get() etc.

- Configuration

- SessionFactory: SessionFactory provides the instance of Session. It is a factory of Session. It holds the data of second level cache that is not enabled by default.

- Query

- Criteria : The objects of criteria are used for the creation and execution of the object-oriented criteria queries.

- Transaction

180. How is SQL query created in Hibernate?
—> The SQL query is created with the help of the following syntax:
Session.createSQLQuery

181. How is HQL query created?
—> The HQL query is created with the help of the following syntax:
Session.createQuery

182. How can we add criteria to a SQL query?
—> A criterion is added to a SQL query by using the Session.createCriteria.

183. Define persistent classes.
—> Classes whose objects are stored in a database table are called as persistent classes.

184. Is SessionFactory a thread-safe object?
—> Yes, SessionFactory is a thread-safe object, many threads cannot access it simultaneously.

185. 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.

186. What is the difference between session.save() and session.persist() method?
- The first difference between save and persist is there return type. Similar to save method, persist also INSERT records into the database, but return type of persist is void while return type of save is Serializable Object.

187. What is the difference between get and load method?

get()	**load()**
Returns **null** if an object is not found.	Throws **ObjectNotFoundException** if an object is not found.
get() method always **hit the database**.	load() method **doesn't hit** the database.
It returns the real object, not the proxy.	It returns proxy object.

188. What are the states of the object in hibernate?
—> There are 3 states of the object (instance) in hibernate.
1. **Transient**: The object is in a transient state if it is just created but has no primary key (identifier) and not associated with a session.
2. **Persistent**: The object is in a persistent state if a session is open, and you just saved the instance in the database or retrieved the instance from the database.
3. **Detached**: The object is in a detached state if a session is closed. After detached state, the object comes to persistent state if you call lock() or update() method.

189. How to make an immutable class in hibernate?
—> If you mark a class as mutable="false", the class will be treated as an immutable class. By default, it is mutable="true".

190. How many types of association mapping are possible in hibernate?
There can be 4 types of association mapping in hibernate.
1. One to One
2. One to Many
3. Many to One
4. Many to Many

191. Is it possible to perform collection mapping with One-to-One and Many-to-One?
No, collection mapping can only be performed with One-to-Many and Many-to-Many.

192. What is lazy loading in hibernate?
—> Lazy loading in hibernate improves the performance. It loads the child objects on demand. Since Hibernate 3, lazy loading is enabled by default, and you don't need to do lazy="true". It means not to load the child objects when the parent is loaded.

193. What is HQL (Hibernate Query Language)?
—> Hibernate Query Language is known as an object-oriented query language. It is like a structured query language (SQL). The main advantage of HQL over SQL is:
1. You don't need to learn SQL
2. Database independent
3. Simple to write a query

194. What is the difference between first level cache and second level cache?
- First Level Cache is associated with Session whereas Second Level Cache is associated with SessionFactory.
- First Level Cache is **enabled** by default whereas Second Level Cache is not enabled by default.

195. Hibernate Configuration File
—> hibernate.cfg.xml

196. What are the common annotation used for hibernate mapping
Commonly used JPA/Hibernate annotations
- @Id
- @GeneratedValue
- @Table
- @Entity
- @Column
- @Transient
- @Temporal
- @Embedded
- @Embeddable
- @ElementCollection
- @OneToMany
- @ManyToOne
- @ManyToMany
- @OneToOne
- @Lob
- @JoinColumn

197. What is Microservice?
—> Breaking the application into smaller modules that implements business logics.

198. Features of micro services
- If any of the module gets failed then it will not affect other modules.
- If we want to update logic of a specific module then we don't need to deploy whole application, simply we will deploy the updated module only.

199. Difference between Monolithic & Microservices
- In Microservices, If any of the module gets failed then it will not affect other modules. But in Monolithic, if any module gets failed then application gets crashed.
- In Microservices, If we want to update logic of a specific module then we don't need to deploy whole application, simply we will deploy the updated module only. But in Monolithic, we will need to re-deploy the application.

200. Explain Spring Cloud
—> Spring Cloud is a collection of projects like load balancing, service discovery, circuit breakers, routing, micro-proxy, etc will be given by Spring Cloud. So spring Cloud basically provides some of the common tools and techniques and projects to quickly develop some common patterns of the microservices.

201. Explain Spring Boot Actuator
—> In essence, Actuator brings production-ready features to our application. Monitoring our app, gathering metrics, and understanding traffic or the state of our database becomes trivial with this dependency.

202. What do you mean by Eureka Server/Service discovery?
- Spring Cloud Netflix Eureka
- Registering microservices to eureka server
- Keeps the track of port number, url, name

203. What is API Gateway?
—> An API gateway manages incoming requests and routes them based on key factors such as request path, headers, and query parameters

204. Microservices communication
1. Synchronous
- Rest Template
- Web Client
- Spring Cloud Open Feign

2. Asynchronous
- Apache Kafka
- RabbitMQ

205. Circuit Breaker Using Resilience
—> Fallback method, Retry, Circuit Breaker

206. Config Server
- Keeping the configuration file in GIT

207. What is Docker Container?
—> A container is a runnable instance of an image.

208. What is Docker Image?
—> Docker images are read-only templates that contain instructions for creating a container. A Docker image is a snapshot or blueprint of the libraries and dependencies required inside a container for an application to run.

209. What is DockerFile?
—> A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

210. What can you tell about Docker compose?
—> Docker Compose is a tool that helps you define and share multi-container applications

211. What is Docker Image registry?
—> A Docker registry is a system for versioning, storing and distributing Docker images.

212. How many Docker Components are there?
—> Docker follows Client-Server architecture, which includes the three main components that are **Docker Client**, **Docker Host**, and **Docker Registry**.

213. What is Docker Hub?
—> Docker Hub is a container registry built for developers and open source contributors to find, use, and share their container images.

214. Can a container restarts by itself?
—> Docker provides restart policies to control whether your containers start automatically when they exit, or when Docker restarts.

215. What is EC2? (Amazon Elastic Compute Cloud)
—> Amazon Elastic Compute Cloud (EC2) is the Amazon Web Service you use to create and run virtual machines in the cloud

216. What is S3?
—> Amazon Simple Storage Service (Amazon S3) is a scalable, high-speed, web-based cloud storage service. The service is designed for online backup and archiving of data and applications on Amazon Web Services (AWS).

217. What is SnowBall?
—> Snowball is a petabyte-scale data transport solution that uses secure appliances to transfer large amounts of data into and out of the AWS cloud.

218. What is CloudWatch?
—> Amazon CloudWatch is a service that monitors applications, responds to performance changes, optimizes resource use, and provides insights into operational health.

219. What do you mean by VPC?
—> A virtual private cloud (VPC) is a secure, isolated private cloud hosted within a public cloud. VPC customers can run code, store data, host websites, and do anything else they could do in an ordinary private cloud, but the private cloud is hosted remotely by a public cloud provider.

220. What do you mean by Serverless?
—> A serverless architecture is a way to build and run applications and services without having to manage infrastructure. Your application still runs on servers, but all the server management is done by AWS.

221. What are Key-Pairs in AWS?
—> A key pair is a combination of a public key that is used to encrypt data and a private key that is used to decrypt data.

222. How many subnets can you have per VPC?
—> Currently you can create 200 subnets per VPC.

223. DNS and Load Balancer comes under which type of cloud service?
—> IAAS-storage cloud service. (Infrastructure as a service (IaaS))

224. What is SQL?
—> Structured query language (SQL) is a standard language for database creation and manipulation. SQL is used to communicate with a database.

225. What is database?
—> A database is an organized collection of structured information, or data, typically stored electronically in a computer system.

226. What is DDL? Data Definition Language
—> Create, Drop, Alert
DDL (Data Definition Language) provides the ability to define, create and modify database objects such as tables, views, indexes, and users.

227. What is DML? Data Manipulation Language
—> Insert, Delete, Retrieve, Update
DML (Data Manipulation Language) allows for manipulating data in a database, such as inserting, updating, and deleting records.

228. What is View ?
—> A SQL view is **a virtual table created by a query**, which can include columns from one or more tables in a database.

229. What is foreign Key?
—> A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

230. What is primary Key?
—> Primary keys must contain UNIQUE values, and cannot contain NULL values.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

231. What is Normalisation?
—> Normalization is the process of organizing data in a database. It includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.

232. What is Denormalisation?
—> Denormalization is the process of adding precomputed redundant data to an otherwise normalized relational database to improve read performance.

233. What are the different operators available in SQL?
- SQL Arithmetic Operators
- SQL Bitwise Operators
- SQL Comparison Operators
- SQL Compound Operators
- SQL Logical Operators

234. Difference between BETWEEN & IN operator
—> The BETWEEN operator is utilized to compare two values inside a range, whereas the IN operator is utilized to compare a value with a set of values.

```
SELECT * FROM orders
WHERE order_date BETWEEN '2020-01-01' AND '2020-12-31';
```

```
SELECT * FROM Customers
WHERE CustomerName IN ('IBM', 'Microsoft', 'Apple');
```

235. Write an SQL query to find the names of employees starting with 'A'.

```
SELECT emp_name
FROM employees
WHERE emp_name LIKE 'A%';
```

236. What is Joins in SQL? Types of Joins
—> A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Here are the different types of the JOINs in SQL:

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table

237. What is index?
—> Indexes are used to quickly locate data without having to search every row in a database table every time said table is accessed.

238. Difference between DELETE & IN TRUNCATE
- DELETE is a SQL command that removes one or multiple rows from a table using conditions. TRUNCATE is a SQL command that removes all the rows from a table without using any condition.
- Truncate is a DDL command so this command change structure of table. Delete is a DML command. It only remove rows from a table, leaving the table structure untouched.
- You can’t rollback in Truncate but you can rollback in Delete.

239. What are ACID properties?
- Atomicity
- Consistency
- Isolation
- Durability