350+ Q&A

Java Interview Questions and Answers

Includes Java version till Java 12, Hibernate, Spring, Spring-Boot, Web Services

Pratik Bandal

Java Interview Questions and Answers

Pratik Bandal

About Book

Java Interview Questions and Answers is extremely useful for beginners and professionals who are preparing for interview of Java. This book covers interview questions of Java which is an all-time favourite programming language of the world. The book will prepare you for interview as well as refining your knowledge of Java and will make you ready for working in industry.

For any positive or negative feedback regarding this book, kindly reach me at pratikbandal007@gmail.com

About Author

I am Pratik Bandal currently working in a big multinational company and have over 7 years of experience in software industry. I have done B.E. and Post graduate diploma in Advanced Computing. I have experience in different domains like healthcare, payment and banking sectors.

But on top of all, I am developer like you all guys doing an 8 hour job. Writing is something I do extra and I love doing it. No one is perfect and holds true for me as well. Without all you guys I am not an author.

Writing an interview question book is a really great deal of responsibility. But huge natural variations in interview are something difficult to cover in this small book.

Contents

1. Fundamentals

1. What is Java?

The Java programming language is a high-level language, high level programming language means it has strong abstraction from computer.

It is characterized as Simple, Object oriented, Distributed, Multithreaded, Dynamic, Architecture neutral, Portable, High performance, Robust and Secure.

Java can also be thought of as a software platform which runs on different hardware platforms.

2. What are features of Java?

Object oriented: Java is object oriented language and supports features like polymorphism, inheritance, encapsulation and abstraction.

Simple: Concepts of Java are easier to grasp, programmers can be productive from the very beginning.

Secure: Java is designed in such a way that it is secure from intrusion from outside code which can generate virus or invade file system.

Robust: Java provides compile-time error checking as well as run-time error checking.

Architecture neutral: Java Compiler generates bytecodes which is architecture neutral intermediate code format that can be transported to various hardware and software platforms.

Portable: Bytecode is executed by Java Virtual Machine, JVM is implemented for all different platforms but it will execute same java bytecode and convert it to machine language which makes Java portable.

Multithreaded: Multithreading enables java to perform multiple tasks simultaneously.

Interpreted: Java bytecode is interpreted by JVM at runtime.

High Performance: Java bytecode is highly optimised, uses just in time compilation and hence has high speed of execution.

Distributed: Java was designed for distributed environment of internet in mind.

3. What is JDK?

Java development kit is abbreviated as JDK.

It is software development environment.

The JDK provides tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

It is combination of Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a document generator (Javadoc) and other tools.

4. What is difference between Oracle JDK and openJDK?

Oracle JDK was previously called SUN JDK was the official proprietary implementation of the Java language. After the takeover from Oracle it was named as Oracle JDK and Oracle’s team maintains the JDK.

OpenJDK is an open source implementation of JDK with contribution from Oracle and open Java community.

Starting with JDK 11 accessing the long time support Oracle JDK/Java SE will now require a commercial license. Oracle JDK without subscription could stop working hence you should now pay attention to which JDK you're installing.

OpenJDK is completely free and can be used in accordance with GPL v2 license.

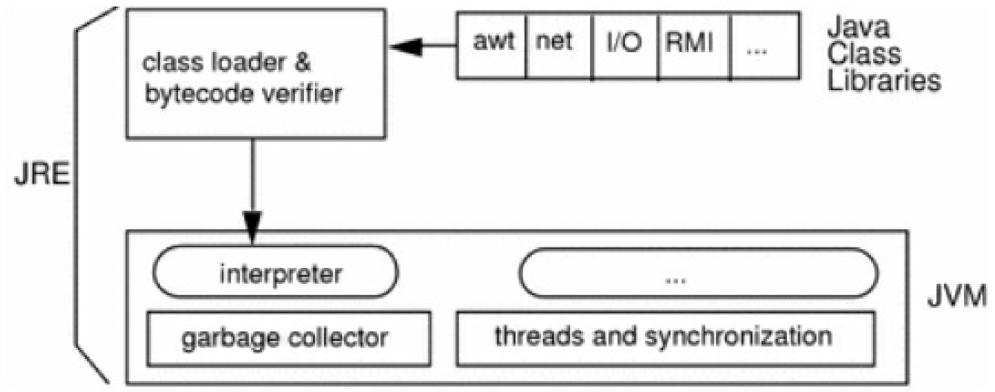
You can create your own version of Java JDK by using OpenJDK as foundation and building on top of it or customizing it.

5. What is JRE?

Java runtime environment provides runtime environment for Java development.

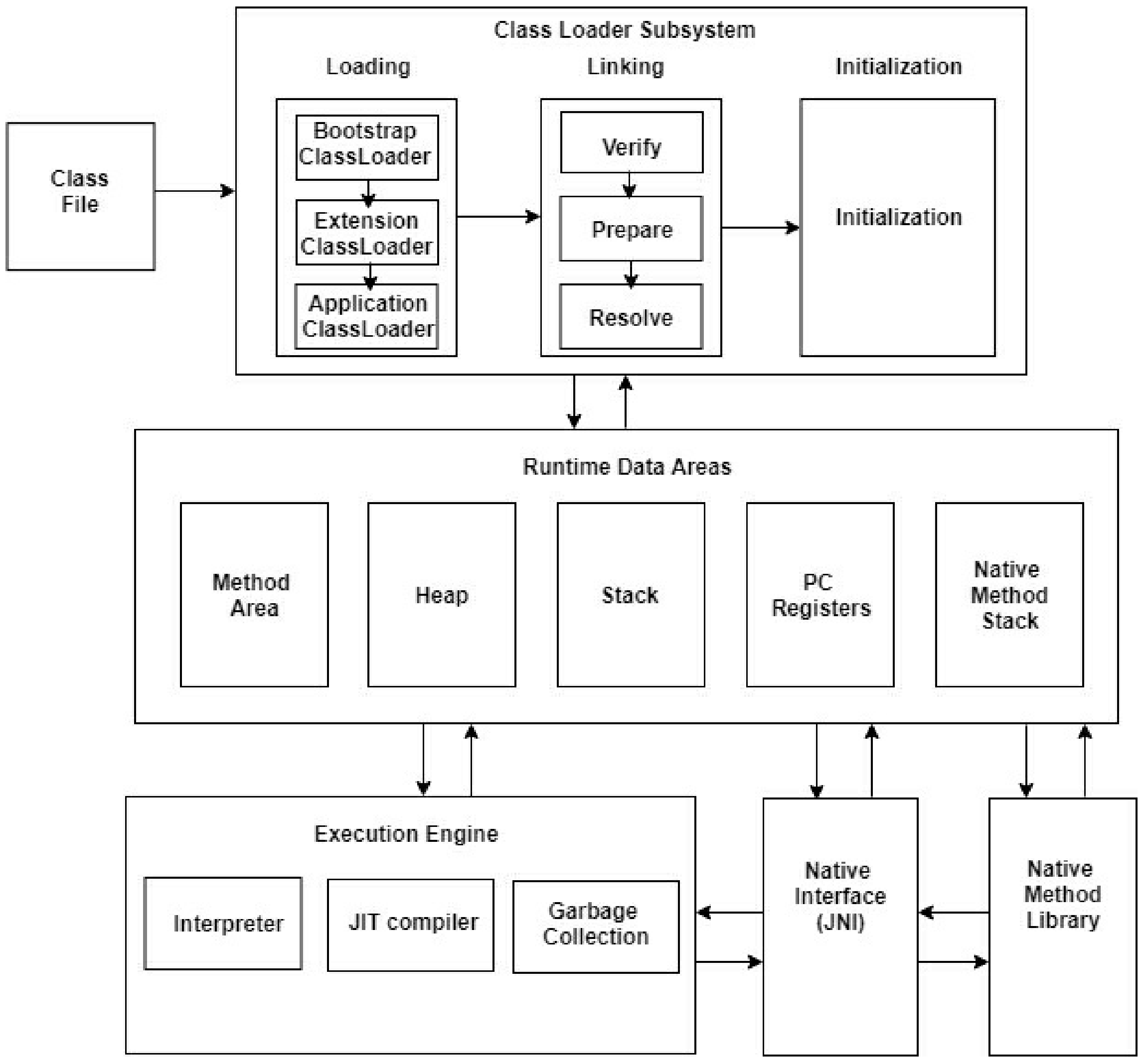
The runtime system includes:

- Implementation of the JVM
- Code to run Java programs, link native methods, handle exceptions and manage memory.



6. What is JVM?

Java virtual machine is abbreviated as JVM. JVM is composed of Class Loader Subsystem, Runtime Data Area, and Execution Engine.



Class Loader Subsystem: It is used for loading, linking and initializing .class file.

Loading: This component loads .class file. Bootstrap class loader, extension class loader and application class loader are the different class loaders.

Bootstrap class loader: Bootstrap class loader loads class file from bootstrap class path i.e. rt.jar.

Extension class loader: Extension class loader loads class file from which are inside the ext folder (jre\lib).

Application class loader: Application class loader loads class file from Application classpath (class path mentioned in environment variable).

Linking: Linking phase has below steps,

Verify: It verifies bytecode, if verification fails you will get the verification error.

Prepare: Memory allocation to all static variables is done, also static variables assigned with default values.

Resolve: Symbolic memory references are replaced with the original references from Method Area.

Initialization: This component performs the final phase of the class loading where all the static variables are assigned the original values and the static blocks are executed from the parent to the child class.

Runtime Data Area:

Runtime Data Area has below components,

Heap: Objects, instance variables and arrays will be stored here. There is also one Heap per JVM. Heap areas is shared by multiple threads hence the data stored is not thread-safe.

Stack Area: For every thread, one runtime stack will be created. For each method call entry is made in the stack area which is called as Stack Frame. Stack memory contains all local variables. The stack area is not shared by multiple thread hence it is thread safe.

Method Area: Class level data is stored in method area, including static variables. Every JVM has only one method area.

Native Method stacks: This component is written in a different language and holds the native method information.

PC Registers: This component holds the address of the JVM instruction which is currently executing. Each thread in Java has its own PC register to hold the address of the currently executing instruction.

Execution Engine:

Bytecode is executed by execution engine.

It has components like interpreter, JIT compiler and garbage collector.

The interpreter interprets the bytecode, when one method is called multiple times, every time a new interpretation is required.

When execution engine finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code.

Repeated method calls will use this native code, which improve the performance of the system.

Garbage collector Collects and removes unreferenced objects.

Java Native Interface (JNI): JNI interacts with the Native Method Libraries and provide required native libraries to the Execution Engine.

Native Method Libraries: Native Method Libraries are required for the Execution Engine.

7. What is Delegation while class loading?

If particular class is not loaded JVM requests the classloader subsystem to load that specific class.

The classloader system passes the request to the Application classloader which in turn delegates this request to the Extension classloader.

The Extension classloader will again delegate this request to the Primordial classloader

The primordial classloader will search this class in the bootstrap classpath (i.e. jdk\jre\lib\rt.jar). If found, the corresponding .class file is loaded

If not, the primordial classloader delegates the request to the extension classloader. So that the class is loaded in the jdk\jre\lib\ext path. If found, the corresponding .class file is loaded

If not, the extension classloader delegates the request to the application classloader and will search the class in the application's classpath. If found, it is loaded otherwise developers will get a ClassNotFoundException at runtime.

8. What is JIT compiler?

The JIT compiler compiles the bytecodes into native code for the platform on which it is running.

9. Is Java fully object oriented?

No, because it supports primitive data type like int, byte, long etc., which are not objects.

10. What is OOPS?

OOPS is abbreviated as Object Oriented Programming system in which object is most important part of your program.

It allows users to create object and methods to perform actions on those objects.

11. What are basic principles of OOPS?

Below are basic principles of language that makes it Object oriented (Explained in detail in later part):

- Encapsulation
- Data Abstraction
- Polymorphism
- Inheritance

12. Which component in JVM will allocate memory to Java program?

The Class Loader subsystem will allocate necessary memory needed by program.

2. Classes and Objects

13. What is class?

Class in java can be considered as blue print to create an object.

14. What does Java Object mean?

An object contains data and actions on the available data.

An object has a state and behaviour.

The state of an object is stored in fields (variables), while methods display the object's behaviour.

Objects are created from classes which are nothing but template.

An object is created using the keyword "new".

15. What are object references?

An object reference is an address which indicates where an object's variables and methods are stored.

16. What are different types of references in Java?

Java has strong, weak, soft and phantom references.

17. Explain access modifiers in Java.

Default: Class, method or variables with no access modifier has default access modifier. The class, variables or methods with default access modifier are accessible only within the same package.

Private: The private keyword is used to declare particular method or data member as private. Private methods or data members are accessible only within the same class in which they are declared.

Protected: The protected keyword is used to declare particular method or data member as protected. Protected methods or data members are accessible within same package or sub classes in different package.

Public: The public keyword is used to declare particular method or data member as public. The public Classes, methods or data members are accessible from everywhere in the program.

18. What is encapsulation? Explain with real life example.

Encapsulation is process of binding code and data together.

Encapsulation basically works on principle of providing restricted access of data(variables) to the code(methods).

Consider example of bank account as object which has balance as data(variable).

If balance variable is declared as public any other bank account object can access it and it will be visible to other person but since balance variable is declared as private it will have only restricted access through code, this code will identify your username, password and other security parameters only then your bank accounts balance can be accessed.

19. What is this keyword?

The ‘this’ keyword refers to the current object.

20. Is it possible to assign reference to this?

No we cannot assign any value to "this" because it always points to current object and it is a final reference in java.

If we try to change or assign value to this compile time error will occur.

21. What are differences between this and super keyword?

“this” refers to current object members whereas parent object members can be referred by using “super”.

Using this(); call we can call other constructor in same class. Using super(); we can call super class constructor from sub class constructor.

22. What is package?

Package is used to group classes, interfaces and sub packages with similar functionalities.

Packages are used for preventing naming conflicts, providing controlled access, making searching of classes, interfaces easier.

23. Can we import same class two times?

Class-loader will load the class only once and memory is allocated to corresponding objects of that class.

So, if Class-loader load class only once then there is no meaning of importing it multiple times.

24. What is static import?

Java 5, introduced static import.

Using static import, static members of a class can be accessed directly without class name or any object.

For Example: we always use out() method of java.lang.System class by using System class i.e. System.out(), but by using static import we can access out () method directly.

25. What does public static void main (String args[]) signify?

Here, main is name of the method.

Public access modifier means main method can be accessed by any class.

Static means method can be accessed without creating an instance of the class.

Void means method does not return anything.

String args[] specifies parameters passed to the method.

26. What is final variable?

When a variable is declared as final, its value can’t change.

It is used when you want your variable to be constant.

Final variable will only initialize once.

Final variable can be initialized when it is declared or a blank final variable can be initialized within a constructor.

27. What happens when you instantiate this class?

```
public class D {
        D a = new D();
}
```

If you try to instantiate D from any other class, you will get java.lang.StackOverflowError

28. What is the default value of the local variables in Java?

Local variables are declared in methods, constructors, or blocks.

Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.

Access modifiers cannot be used for local variables.

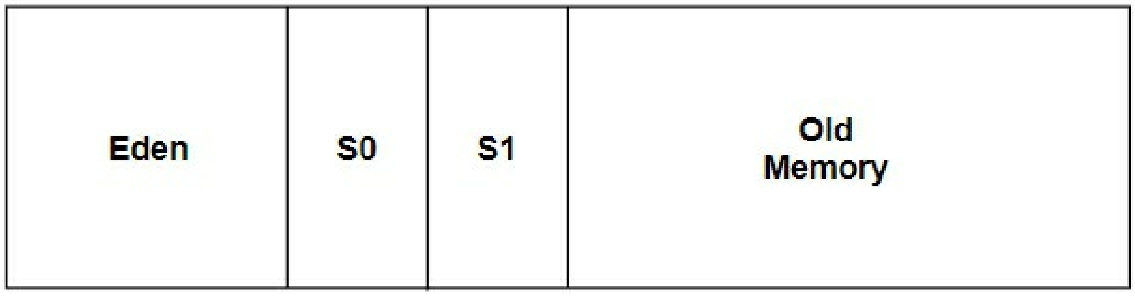
Local variables are visible only within the declared method, constructor, or block.

Local variables are implemented at stack level internally.

There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

3. Garbage Collector

29. How garbage collector manages heap memory?



Newly created object is allocated memory on the Eden space. Eden space is inspected by garbage collector to marks objects as which are alive.

Alive objects after a garbage collecting process are moved into a survivor space S0. The second time the garbage collector runs on the Eden space, it moves all alive marked objects into the S1 space. Also, everything that is currently on S0 is moved into the S1 space.

If an object survives for X rounds of garbage collection, it is most likely that it will survive forever, and it gets moved into the Old space.

The old memory can be also garbage collected, but since it is a bigger part of the memory compared to Eden space, it does not happen that often.

30. Which algorithm is used by garbage collector?

Many algorithms are used by garbage collector but most commonly used algorithm is mark and sweep.

31. How do you identify minor and major garbage collection in Java?

When garbage collection logging is enable minor collection prints “GC” while Major collection prints “Full GC”.

32. How is Garbage Collection managed?

The Garbage Collector is controlled by JVM which decides when to run the Garbage Collector.

JVM runs the Garbage Collector when memory is running low.

There are parameters which can be passed to JVM to tune the behaviour of garbage collector.

33. What is purpose of System.gc() and Runtime.gc() methods?

These methods request JVM to start the Garbage collection, however JVM decides whether to start garbage collection immediately or later.

34. When is the finalize() called ? Why it is protected?

The garbage collector calls java.lang.Object.finalize() method on an object when garbage collection determines that there are no more references to the object.

A subclass overrides the finalize method to provide a chance to free up resources or to perform other clean up, since subclass classes should be able to access finalize() method, it should at least be protected.

35. Which part of memory is involved in Garbage collection?

Heap

36. What are different types of garbage collectors?

There are four types of garbage collectors:

Serial Garbage collector: It uses just a single thread for garbage collection. All the application threads freeze when serial garbage collector performs garbage collection.

Parallel Garbage collector: It is the default garbage collector of the JVM. It uses multiple threads in parallel for garbage collection. All the application threads freeze when parallel garbage collector performs garbage collection.

CMS Garbage collector: Concurrent Mark Sweep (CMS) garbage collector uses different threads to scan the heap memory to mark instances for eviction and then sweep the marked instances. CMS garbage collector holds all the application threads in particular scenarios only.

G1 Garbage collector: It separates the heap memory into regions and does collection within them in parallel. After reclaiming the memory G1 also does compacts the free heap space.

37. Will garbage collection guarantee that a program won't run out of memory?

Garbage collection doesn't guarantee that a program won't run out of memory.

It's possible that programs occupies memory resources quicker than memory resources vacated through garbage collection.

It's also possible that programs might create objects that will not be eligible for garbage collection.

38. What is pass by value and pass by reference?Does Java use pass by value or pass by reference?

Java uses pass by value.

Pass by Value

In pass by value, method receives actual parameter expressions that are evaluated and a value is obtained. This value is stored in a location and then it becomes the formal parameter of the invoked method.

Pass by Reference

In pass by reference, the actual parameter is basically referred by formal parameter. Any changes done to the formal parameter will reflect in actual argument and vice versa.

39. What is Garbage collector interface introduced in Java 10?

Java 10 improves the source code isolation of different garbage collectors by introducing a clean garbage collector (GC) interface.

The garbage collector code before java 10 was scattered all over the HotSpot sources.

New collectors can be easily implemented through clean garbage collector (GC) interface, it would also make the code much cleaner, and simpler to exclude one or several collectors at build time.

Adding a new garbage collector should be a matter of implementing a well-documented set of interfaces, rather than figuring out all the places in HotSpot that needs changing.

40. What is Parallel Full GC for G1 in Java 10?

Garbage collectors uses "incremental" and "full" collection types.

Incremental collection is the better of the two for staying out the way, as it'll do a little bit of work every so often.

Full collection is often more disruptive, as it takes longer and often has to halt the entire program execution while it's running.

Because of this, most modern GCs (including G1) will generally try to ensure that in normal circumstances, the incremental collection will be enough and a full collection will never be required.

However, if lots of objects across different generations are being made eligible for garbage collection in unpredictable ways, then occasionally a full GC may be inevitable.

Currently, only single thread is used for the G1 full collection implementation. And that's where that JEP 307: Parallel Full GC for G1 comes in - it aims to parallelize full GC, so that when a full GC does occur it's faster on systems that can support parallel execution.

41. What is bytecode Generation for Enhanced for Loop in Java 10?

If a huge list of array is iterated using for loop, it keeps a temporary reference to the memory.

This memory might be locked for garbage collection, hence causing out of memory errors. Hence java team updated the byte code generation logic for this in Java 10.

42. What is Epsilon Garbage Collector introduced in Java 11?

Epsilon is a passive or "no-op" GC.

It handles memory allocation but doesn't recycle it when objects are no longer used.

The JVM shuts down, when your application exhausts the Java heap memory.

In other words, Epsilon will allow your application to run out of memory and crash.

There are a few use cases where a no-op GC proves useful:

Performance testing: Having a GC that does almost nothing is a useful tool to do differential performance analysis for other, real GCs.

Memory pressure testing: For Java code testing, you can configure no-op GC with threshold memory and let it crash with a heap dump if that constraint is violated.

Extremely short lived jobs: A short-lived job vacate heap memory by exiting quickly. In this case, clean up the heap with GC is a waste of time, because the heap would be freed on exit anyway.

43. What is Shenandoah in Java 12?

Shenandoah is an ultra-low pause time garbage collector that performs more garbage collection work concurrently with the running Java program to reduce GC pause time.

Concurrent marking of live objects is also performed by CMS and G1.

Shenandoah adds concurrent compaction.

Your heap size also factors into performance.

Testing on Shenandoah was done on larger heap sizes (from 4 to 128 GB). It's recommended that you use it for these large sizes.
Shenandoah GC is less likely to be used with a smaller heap size.

44. How many objects are eligible for garbage collection, after line 8 runs?

```
public class X {
    public static void main(String [] args) {
        X x = new X();
        X x2 = m1(x); /* Line 6 */
        X x4 = new X();
        x2 = x4; /* Line 8 */
        doComplexStuff();
    }
    static X m1(X mx) {
        mx = new X();
        return mx;
    }
}
```

Remember that "Java is pass by value," operation in m1() method does not affect variable x. At line 8 since x2 is referring to new object, only object without a reference is the one generated at line 6.

4. String

45. What is string?

In Java, String is sequence of characters.

String objects are immutable which means it cannot be modified once created.

46. Is string class or data type?

String is class in java.lang package but in Java all classes are also considered as data type hence string are data type as well.

47. What is difference between == and equals()?

== Compares references of string object while equals() method compares contents of string object.

48. Why is string immutable?

The string is immutable because String objects are cached in string pool.

Since cached string literals are shared between multiple clients there is always risk where one client’s action would affect other clients.

Since caching of string was important from performance perspective this risk was avoided by making string class immutable.

49. What is string constant pool?

Java stores literal string values in string constant pool which is an area in heap memory.

String constant pool minimize the redundancy and memory waste caused by storing String objects with duplicate values on the heap.

String Constant Pool stores String objects created without the use of new keyword.

String objects created using new keyword are stored in the normal memory part on the heap.

50. Why string should be created using string literal?

String object created using string literal is stored in String constant pool which minimize the redundancy and memory waste caused by storing String objects with duplicate values on the heap.

51. When to use Stringbuffer over String?

String is immutable whereas stringbuffer is mutable in java, due to which stringbuffer is faster and also consumes less memory than string when performing simple concatenation operations.

52. What is difference between stringbuffer and stringbuilder?

Stringbuffer is synchronized whereas stringbuilder is not synchronized.

Stringbuffer is less efficient than stringbuilder.

However in multithreaded environment Stringbuffer is better than stringbuilder as it is synchronized.

53. How to create immutable class?

To create an immutable class following steps are followed:

Declare the class as final so it could not be inherited.

Declare all variables as private so that it cannot be directly modified.

Do not allocate setter methods for variables only have getter methods.

Declare all variables as final so that it can be initialized once only.

Initialize all the variables via a constructor.

```
public final class Person {
        final String name;
        final String address;
        public Person (String name, String address) {
                this.name = name;
                this.address = address;
        }
        public String getName() {
                return name;
        }
        public String getAddress () {
                return address;
        }
    }
```

54. What is StringJoiner introduced in Java 8?

Java 8 added a new final class StringJoiner in java.util package.

It is used to construct a sequence of characters separated by delimiter.

e.g.

```
StringJoiner joinSports = new StringJoiner(“;”);
joinSports.add(“baseball”);
joinSports.add(“football”);
System.out.println(joinSports);

Output: baseball, football
```

55. What are new String methods introduced in Java 11?

There are lots of new string methods introduced in Java 11, few of them are as below:

String.isEmpty():

String.isEmpty() method determines if String is empty or if it contains only whitespace characters.

String.lines():

String.lines() method returns stream of lines extracted from a given multiline string, separated by line terminators.

String.repeat():

String.repeat() method returns string which is basically concatenation of given string repeated n times. If string is empty or count is zero then empty string is returned.

56. What are Strings new methods introduced in Java 12?

String.indent():

The indent method helps with changing the indentation of a String. We can either pass a positive value or a negative value depending on whether we want to add more white spaces or remove existing white spaces.

e.g.

```
var result = "Hello\nWorld!".indent(3);
System.out.println(result);
Output:
Hello
World!
```

String.transform():

The transform() method takes a String and transforms it into a new String with the help of a Function.

e.g.

```
var result = "Hello".transform(s -> s + ", World!");
System.out.println(result); // Hello, World!
```

57. Write String Palindrome program.

```
import java.util.*;

class Palindrome
{
   public static void main(String args[])
   {
        String originalString, reverseString = "";
        Scanner in = new Scanner(System.in);

        System.out.println("Enter a string:");
        originalString = in.nextLine();

        int length = originalString.length();

        for (int j = length - 1; j >= 0; j--)
            reverseString = reverseString + originalString.charAt(j);

        if (originalString.equals(reverseString)) {
            System.out.println("The string is a palindrome.");
        } else {
            System.out.println("The string isn't a palindrome.");
        }
    }
}
```

5. Methods and Constructors

58. What is constructor?

Constructors are used to initialize the object.

A constructor contains block of statements that are executed at time of object creation.

59. What are types of constructor?

There are two types of constructors,

Default constructor: Default constructor also known as no-arg constructor does not have any parameters.

Default constructor is used to initialize all objects with same value.

Parameterized constructor: Constructor which has parameters are called as parameterized constructor.

Parameterized constructor is used to assign different values to newly created objects.

60. Explain the difference between constructor and object?

Constructor	Method
Used to initialize object.	Used to perform particula functionality
Invoked implicitly while creating an object.	Invoked explicitly to perforr particular action.
Does not have return type	Can have return type.
If constructor is not present Java compiler provides default constructor.	If method is not present Jav compiler does not provide defau method.
Name of constructor should be same as class.	Name of method may or may not b same as class.

61. Why constructor are not inherited?

Constructors have same name as class name. Now consider if constructors were inherited in child class then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name.

Also now suppose if constructors can be inherited then by using a super class's constructor we can access private members of a class from subclass.

A parent class constructor is not inherited in child class and this is why super() is added automatically in child class constructor if there is no explicit call to super or this.

62. Why constructors cannot be final?

When you set a method as final that method cannot be overridden by any class.

But according to the Java Language Specification: Constructor declarations are not members.

They are never inherited and therefore are not available for hiding or overriding, hence no point in making constructor as final.

63. Can constructors be static?

No, if you declare constructor as static, You would get the error as
"modifier static not allowed here".

Everything that is marked static belongs to the class only, Since each constructor is being called by its subclass during creation of the object of its subclass, so if you mark constructor as static the subclass will not be able to access the constructor of its parent class because it is marked static and thus belong to the class only.

Due to this inheritance concept will be violated and that is reason why a constructor cannot be static.

64. Can we overload constructor?

Yes, we can also overload constructors. Based upon the parameters specified overloaded constructor is called.

65. Can we clone Object without using cloneable interface?

The java.lang.Cloneable interface must be implemented by the class whose object clone you want to create.

If you don't implement Cloneable interface, clone() method generates CloneNotSupportedException.

66. Can we override constructor?

No, since constructor cannot be inherited and for overriding to occur inheritance is required constructor overloading is not possible.

67. Why do constructors not return values?

When the constructor exits, it returns the current instance of the class.

So the reason the constructor doesn't return a value is because it's not called directly by your code, it's called by the memory allocation and object initialization code in the runtime.

So basically constructor wasn't designed return any value.

68. What is constructor chaining?

When one constructor is called from another constructor with respect to current object it is called constructor chaining. There are two ways in which you can do this.

this () - For constructors in same class

super ()- For constructors in base class.

This basically occurs during inheritance, it ensures that before the object of the sub class is created, the data members of super class are initialized.

Constructor chaining is performed when we want to perform multiple tasks in one single constructor, rather than implementing in every single constructor.

69. What is difference between this and super?

this is a reference to the object of the current class, and super is a reference to the object of its parent class.

In the constructor, this() calls a constructor defined in the current class. super() calls a constructor defined in the parent class. The constructor may be defined in any parent class, but the one overridden closest to the current class.

Calls to other constructors should be defined as the first line in a constructor.

Calling methods works the same way. Calling this.method() calls a method defined in the current class where super.method() will call the same method as defined in the parent class.

70. Why can't this() and super() both be used together in a constructor?

this() will call current object constructor whereas super() will call a parent object constructor.

Compiler will add super constructor implicitly, if there is no super().

Thus if both were allowed you could end up calling constructor implicitly as well as explicitly, hence this() and super() cannot be used together.

71. What is the static variable?

A static variable is common to all objects of the class because it is a class level variable.

Only single copy of static variable is created which is then shared among all the instances of the class.

Memory is allocated to static variables only once, when the class is loaded in the memory.

72. What is static method?

Static Method is a class level method and does not belong to instances of class.

Only static variables of class are available to static method and it can invoke only static methods of the class.

Usually, static methods are used as utility methods that we expose to other classes so that other classes can utilize them without creating an instance of a class.

73. Can a Non-Static Method Access a Static Variable/Method in Java?

Yes, a non-static method can call a static method and access a static variable.

This is possible, because of static members (static variable/static methods) belongs to a class and depending upon their access modifier can be called from anywhere.

74. Why can't a static method (directly) access non-static members?

Static members is a class level variable and memory is allocated to static variables only once, when the class is loaded in the memory, but for non-static or instance members memory is allocated once when an instance is created for that class.

If we call any non-static member from static context i.e. when class is loaded, it will search for the non-static member which is not in existence till then as we have not created any instance till now.

Hence, static method cannot access non-static methods and fields as static methods are loaded earlier that non static method.

However, non-static method can access static methods and fields as static methods are loaded earlier that non static method.

75. Why is the main method static?

The main() is called by the JVM before any objects are made. Since it is static it can be directly invoked without creating instance of a class.

If main method were not-static then it would have been mandatory for JVM to create instance of main class and since constructor can be overloaded and can have arguments JVM would be uncertain how to find main method.

76. Can you remove static modifier from the signature of the main method?

Static modifier can be removed from signature of main method Program compiles.

However, at runtime, it throws an error "NoSuchMethodError."

77. Can abstract class have static methods?

Yes, But static method must be defined.

You cannot create an abstract static method.

However you can create non abstract static method.

78. Is it possible to use this in static blocks?

No, doing so will give compile time error.

79. Can we use this in static methods?

No, doing so will give compile time error.

80. What happens when you call someMethod() in the below

class?

```
class D {
     int someMethod() {
          return (true ? null : 0);
     }
   }
```

If you try to call someMethod of class D it will generate java.lang.NullPointerException

6. Inheritance

81. What is inheritance?

Inheritance is a mechanism wherein a new class is derived from an existing class.

A class derived from existing class is called a subclass, whereas the existing class from which a subclass is derived is called a superclass.

Below are types of inheritance:

- Single-level inheritance
- Multi-level inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

82. Why do we use inheritance?

We use Inheritance for following reasons:

Using Inheritance you can reuse the code which already exist.

Runtime polymorphism cannot be achieved without using inheritance.

Inheritance provides data hiding. The base class can hide its members from the derived class by making its member private.

Method overriding cannot be achieved without inheritance. Using method overriding, we can give a specific implementation of base class method which is commonly used by all subclasses.

Also through encapsulation only specific attributes parent class can be made accessible to child classes.

83. Private members of a class are inherited to sub class?

No, Private member are not inherited to sub class.

84. What is real life example of inheritance?

Vehicle can be considered for real life example of inheritance.

Vehicle has characteristics like wheels, colour and speed.

Vehicle has subclasses like car and bike.

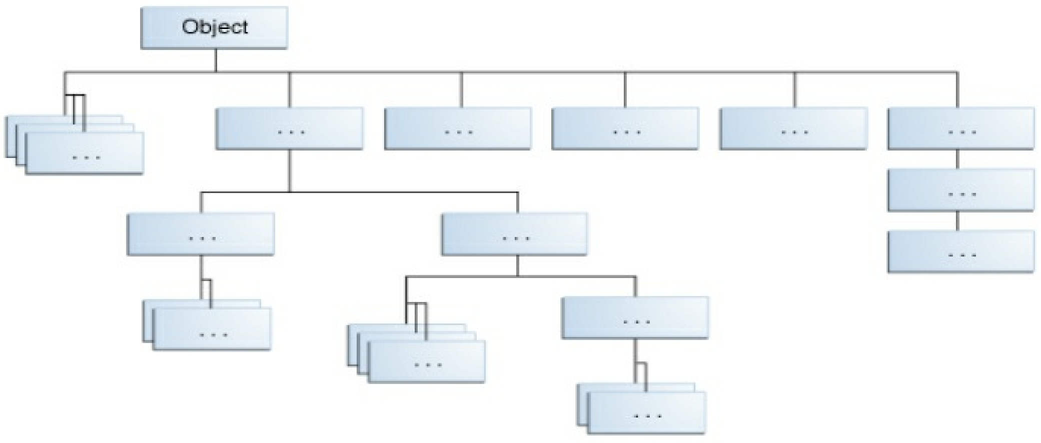
Both car and bike inherit all the characteristics of vehicle like wheels, colour and speed but they will have their own characteristics as well.

85. Why is Object class superclass of all the classes?

Because the Object class defined in the java.lang package is at the top of the class hierarchy tree.

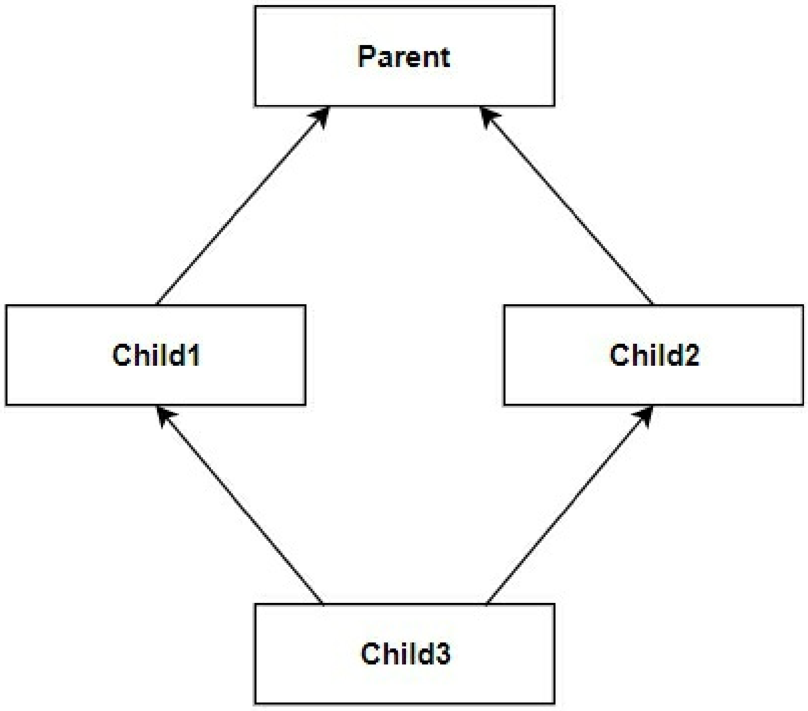
The Object class, defines and implements behaviour common to all classes —including the ones that you write.

In the Java many classes derived directly from Object class, other classes derive from some of those classes, and so on, forming a hierarchy of classes.



86. Why is multiple inheritance not supported in java?

Consider, two classes Child1 and Child2 inheriting from Parent. Assume that Child1 and Child2 are overriding an inherited method and they provide their own implementation. Now Child3 inherits from both Child1 and Child2 doing multiple inheritance. Child3 should inherit that overridden method, which overridden method will be used? Will it be from Child1 or Child2? Here we have an ambiguity. This is called as diamond problem of multiple inheritance.



Since multiple inheritance is rarely required, it can be safely omitted considering the complexity it has for implementation.

87. What is the Association?

Association in basically is a relation between two separate classes which sets up through their Objects.

Composition and aggregation are two types of association.

88. What is aggregation?

Aggregation is a special form of Association where:

Aggregation depicts Has-A relationship.

It represents one way relationship.

In Aggregation, ending one entity will not affect the other entity which means both the entries can survive individually.

e.g. room can exist without a fan and wise versa.

89. What is composition in Java?

Composition depicts Has-A relationship.

Instance variables that refers to other objects are used to achieve composition.

Both the entities are dependent on each other in composition.

In Composition, the composed object cannot exist without the other entity.

e.g. room cannot exist without a roof.

90. What is difference between aggregation and composition?

Aggregation represents weak relationship whereas composition represents weak relationship.

Room has a fan is aggregation as it represents weak relationship whereas room has a roof is composition as it is a strong relationship.

91. What is difference between inheritance and composition?

Inheritance represents IS-A relationship whereas composition represents HAS-A relationship.

Using Inheritance code becomes tightly coupled whereas using composition code becomes loosely coupled.

Shape IS-A circle is example of inheritance whereas Computer HAS-A CPU is example of composition

92. What is final method?

Method cannot be overridden If it is declared as final.

You should make method as final when you are going to have same constant method implementation throughout all derived classes.

93. What is final class?

Class cannot be inherited if it is declared as final.

Since wrapper classes like Float, Integer, Double should not be inherited they are declared as final.

When you want to make your custom class as immutable you use final.

94. What is output of below program?

```
class A {
    int i = 10;
}

class B extends A {
    int i = 20;
}

public class MainClass {
    public static void main(String[] args) {
        A a = new B();
        System.out.println(a.i);
    }
}
```

Output:

```
10
```

7. Polymorphism

95. What does Polymorphism mean?

Polymorphism is ability of a variable, function or object to take on multiple forms.

96. What is real world example of Polymorphism?

Consider Person as a parent class, now if Person is in class room that time Person behave like a student, when Person is in market at that time Person behave like a customer, when Person is at home at that time Person behave like a son or daughter,

Here person behaviour is different at runtime based on situation.

97. What are types of polymorphism?

There are two types of Polymorphism: Runtime Polymorphism (Dynamic Binding) and Compile time Polymorphism (Static Binding).

Runtime polymorphism: method overriding can be categorized as runtime polymorphism.

Compile time polymorphism: method overloading can be categorized as compile time polymorphism.

98. What is method overloading?

Writing two or more methods with same name but different signature in a class is called as method overloading.

Method overloading can be achieved by changing the number of parameters passed to method, by changing data type of parameters passed to method or changing the order of data types passed to method.

99. What is method overriding?

Writing method with same name and signature as parent class in child class is known as method overriding.

100. What is difference between method overloading and overriding?

Method Overloading	Method Overriding
Writing two or more methods with same name but different signature in same class is called as method overloading.	Writing two or more methods with same name and same signature in child class is called as method overriding.
In method overloading parameters of methods must be different.	In method overriding parameters of methods must be same.
Method overloading extends currently available functionality.	Method overriding provides different implementation for currently available functionality.

101. Is method overloading by changing the return type possible?

No, because return value alone is not sufficient for the compiler to figure out which method to call.

102. Why doesn't Java allow overriding of static methods?

Overriding needs an instance of a class.

Subclass provides specific implementation of superclass method.

At run-time based on object used for calling this method it is decided that which method will be executed.

Hence polymorphism is dependent on object/instance which is used to call method.

A static method is not associated with any instance of a class so overriding is not applicable.

103. Can you write virtual methods in Java?

Virtual function is the function whose behaviour can overridden to provide polymorphic behaviour at runtime.

All non-static methods are by default virtual functions in Java.

However, final methods which cannot be overridden and private methods which cannot be inherited are non-virtual methods.

104. What is variable hiding?

When the child and parent class both have a variable with the same name, Child class's variable hides the parent class's variable, even if their types are different, this concept is known as Variable Hiding.

105. Why data member overriding is not allowed?

Overriding data members would break code in the superclass, hence data member overriding is not allowed.

106. What is covariant return type?

Java 5 supports covariant return types.

Before Java 5, when you override a superclass method, the name, argument types and return type of the overriding method has to be exactly same as that of superclass method.

After Java 5, you can override any method by changing the return type if the return type of the subclass overriding method is subclass type. It is known as covariant return type.

107. What will be the output of the following program?

```
class X {
    void method(int a) {
        System.out.println("ONE");
    }
    void method(double d) {
        System.out.println("TWO");
    }
}

class Y extends X {
    @Override
    void method(double d) {
        System.out.println("THREE");
    }
}

public class MainClass {
    public static void main(String[] args) {
        new Y().method(100);
    }
}
```

Output:

```
ONE
```

8. Abstract Classes and Interfaces

108. What is abstraction?

Abstraction is a process of hiding unnecessary information and showing only what is essential to the end user.

For example, when you want to turn on television you just turn on the switch and you don’t actually know the internal working of how television started.

109. What is abstract class?

Abstract class is declared using abstract keyword.

Zero or more abstract methods are present in abstract class.

An abstract method is basically a method without body, the code will be added in its derived classes.

Objects of abstract class type cannot be created, because the code to instantiate an object of the abstract class type will result in a compilation error.

110. What is use of abstract method?

Abstract classes are classes that contain zero or more abstract methods. An abstract method is basically a method without body (no implementation).

Abstract classes require subclasses to provide implementations for the abstract methods and may not be instantiated.

Abstract methods are useful when there are some common actions to be performed in different way by base classes whereas all other common actions are performed in similarly.

For example, consider example of animal class which has child classes as cow and dog. Now when it has sleep method which will be implemented similarly in both cow and dog hence it is concrete method in base class but it also has makeNoise method which has to be implemented differently for both cow and dog as dog barks whereas cow does moo, hence makeNoise should be abstract method.

111. Can you make abstract class final?

No, abstract class cannot be final.

An abstract class is incomplete and concrete subclass implements it, while final class cannot be inherited.

Hence final and abstract are opposite concepts and cannot be used together.

112. Why does abstract class cannot be instantiated but have constructor?

Constructor is used to initialize fields.

Now, if you have subclass to such abstract class when you initialize subclass its constructor will be called and then the parent constructor will be called which will initialize fields of abstract class.

Hence it is possible to initialize abstract class fields through subclass.

113. What is an interface?

An interface is basically blueprint of a class.

An interface is used to achieve abstraction.

Interface has static constants and abstract methods, from Java 8 interface can have default and static methods, from Java 9 interface can have private methods.

114. Why interface method cannot be final?

Interface provides abstract methods which has to be implemented by implementing class as interface acts as blueprint of a class.

A final method can't be overridden.

It doesn’t make sense of having an interface if you cannot actually implement the method.

115. What is default method?

Before Java 8, only abstract methods could be present in interfaces.

Abstract method has to be implemented in its subclasses.

So, if method is to be removed from an interface, then it would break implementation in all its subclasses.

In Java 8, the concept of default method was introduced which allow the interfaces to have methods with implementation without affecting the classes that implement the interface.

116. How to resolve conflict caused by default methods of same signature?

With default methods there is possibility a class may implement two interfaces having method with same signature.

In this case to resolve ambiguity of which method will be called can be resolved by calling method as <super-interface-name>.super.<method-name>.

117. Why interface cannot be final?

A final class cannot have any sub-classes.

An interface is basically blueprint of a class, it is considered best practice to only use interfaces for defining method(s) of sub-classes.

So the two are contradictory.

118. What is marker interface?

A marker interface is interface with no data member and member functions.

Serializable, Cloneable are marker interfaces.

119. What is use of marker interface?

Marker interface means empty inteface.

They are used because you tell JVM that class implementing marker interface have special behaviour.

Serializable interface tells JVM class is Serializable.

Cloneable interface tells JVM class is Cloneable.

120. Why methods of interface are public and abstract by default?

Interface methods are public and abstract by default because they should be available to third party vendors to provide implementation.

121. Can you implement one interface from another?

No, implementing interface means writing body for abstract methods in interface which cannot be done in another interface.

However you can extent one interface from another.

122. Is it possible to create an inner class within an interface?

Yes, Specifying a class inside an interface ties that class directly to that interface, clients which use that interface will have access to that class and all the functionality that it provides.

123. What is functional interface?

Interface with only one abstract method is called as functional interface, which was introduced in Java 8.

If interface is annotated with @FunctionalInterface and we try to have more than one abstract method it throws compile time error.

124. What is difference between interface and abstract class?

Abstract class	Interface

Abstract class is extended using extend keyword.	Interface is implemented using implements keyword.
Can have instance variables.	Cannot have instance variables.
Can have the constructor.	Cannot have the constructor.
Can provide the implementation of the interface.	Cannot provide the implementation of the abstract class.
An abstract class is declared using abstract keyword.	An interface is declared using interface keyword.
An abstract class can extend another Java class as well as implement multiple interfaces.	An interface can extend another Java interface only.

125. For every interface written in a java file, .class file will be generated after compilation? True or False?

True. The .class file will be generated for every interface after compilation.

126. What is private interface methods in Java 9?

Since Java 9, it is possible to include private methods in interfaces.

Now encapsulation is possible in interfaces using private methods.

Java 9 private interface methods can be instance or static.

The private method cannot inherited by sub-interfaces or implementations.

Private methods in interfaces are mainly there to improve code re-usability within interface only: thus improving encapsulation.

127. What is functional interface introduced Java 8?

Interface with only abstract method is called as functional interface.

If interface is annotated with @FunctionalInterface and we try to have more than one abstract method it throws compile time error.

Runnable, Comparable are examples of functional Interface. Java.util.function package contains many builtin functional interfaces in Java 8.

128. Why the below code is showing compile time error?

```
interface X {
    void methodX();
}

class Y implements X {
    void methodX(){
        System.out.println("Method X");
    }
}
```

Interface methods must be implemented as public.

Interface methods are public by default, however you should not reduce the visibility of any methods while overriding.

9. Wrapper Classes

129. What are wrapper classes?

Wrapper classes are used to convert primitive datatypes into objects.

Below are wrapper classes available for primitive data types:

Primitive data type	Wrapper class
Boolean	java.lang.Boolean
Byte	java.lang.Byte
Char	java.lang.Character
Double	java.lang.Double
Float	java.lang.Float
Int	java.lang.Integer
Long	java.lang.Long
Short	java.lang.Short

130. Why we need wrapper classes?

Java is an object-oriented language and in pure object oriented language everything should be presentable as object.

The primitive data types are not objects.

Sometimes, it is necessary to convert primitive data types into objects in Java.

For example, up to JDK1.4, data type is to be converted into an object and then added to a Stack or Vector as these data structures in collection framework store objects only.

Also in multithreading environment object is needed to support synchronization.

131. What is Autoboxing and Unboxing?

Autoboxing is the automatic conversion that the Java compiler makes from the primitive types to their corresponding object wrapper classes.

For example, int to an Integer, a double to a Double etc.

If the conversion goes the other way i.e. from wrapper class to primitive type it is called unboxing.

For example, Converting an object of a wrapper type (Integer) to its corresponding primitive (int) value is called unboxing.

132. When to use wrapper class and primitive type?

Scenarios to use over primitive data type:

Wrapper classes should be used when you need to use them collection.

Wrapper classes can be handy to initialize Objects to null or send null parameters into a method/constructor to indicate state or function. This can't be done with primitives.

Wrapper classes will cause a NullPointerException when something is being used incorrectly, which is much more programmer-friendly than unknowingly inserting default values through primitive date types.

Scenarios to use primitive data type over wrapper classes:

It is slower to use the wrapper classes compared to the primitives as object instantiation, method calls, boxing, unboxing cost some extra resources.

133. Are wrapper classes immutable?

Yes, Wrapper classes are immutable.

JDK provided wrapper classes also provide this in form of instance pooling i.e. each wrapper class store a list of commonly used instances of own type in form of cache and whenever required, you can use them in your code.

It helps in saving lots of byes in your program runtime.

java.lang.Boolean store two inbuilt instances TRUE and FALSE, and return their reference if new keyword is not used.

java.lang.Character has a cache for chars between unicodes 0 and 127 (ascii-7 / us-ascii).

java.lang.Long has a cache for long between -128 to +127.

java.lang.String has a whole new concept of string pool.

134. What happens when you override someMethod(int) as someMethod(Integer) like in the below example?

```
class Parent {
    void someMethod(int i) {
        //some processing
    }
}

class Child extends Parent {
    @Override
    void someMethod(Integer i) {
        //some processing
    }
}
```

No. It gives compile time error. While overriding Compiler treats int and Integer as two different types. Auto-boxing doesn’t happen here.

135. Write program to find Factorial of a number using BigInteger.

```
import java.math.BigInteger;
public class Factorial {
    public static void main(String[] args) {
        int num = 20;
        BigInteger factorial = BigInteger.ONE;
        for(int i = 1; i <= num; ++i)
        {
            // factorial = factorial * i;
            factorial = factorial.multiply(BigInteger.valueOf(i));
        }
        System.out.printf("Factorial of %d = %d", num, factorial);
    }
}
```

10. Serialization

136. What is serialization and deserialization?

Serialization is the process of turning an object in memory into a stream of bytes so you can do stuff like store it on disk or send it over the network.

Deserialization coverts a stream of bytes into an object in memory.

137. What is SerialVersionUID?

The serialVersionUID is used to uniquely identify Serializable classes.

It is ensured that during deserialization the same class that was used during serialize process is loaded using SerialVersionUID.

138. What is need for serialization?

Serialization is usually used to send your data over network or when data is required to be stored in files.

Hardware components like Network infrastructure and Hard disk understand bits and bytes but not Java objects.

Serialization is the process of turning an object in memory into a stream of bytes so you can do stuff like store it on disk or send it over the network.

139. What happens if the object to be serialized includes the references to other serializable objects?

If the object to be serialized includes references to the other objects, then referred object's state will also be saved as the part of the serialized state of the main object.

140. What happens if an object is serializable but it includes a reference to a non-serializable object?

If you try to serialize an object of a class which implements serializable, but the object includes a reference to non-serializable class then a ‘NotSerializableException’ will be thrown at runtime.

141. What is the difference between Serializable and Externalizable interface in Java?

Serializable	Externalizable
Serializable is a marker interface i.e. does not contain any method.	Externalizable interface contains two methods writeExternal()and readExternal() which implementing classes MUST override
Serializable interface pass the responsibility of serialization to JVM and it’s default algorithm.	Externalizable provides control of serialization logic to programmer.

142. Which kind of variables is not serialized during Serialization?

Since static variables belong to the class and not to an object they are not the part of the state of object so they are not saved during Serialization process.

As Serialization only persist state of object and not object itself.

Transient variables are also not included in serialization process and are not the part of the object’s serialized state.

11. Exception Handling

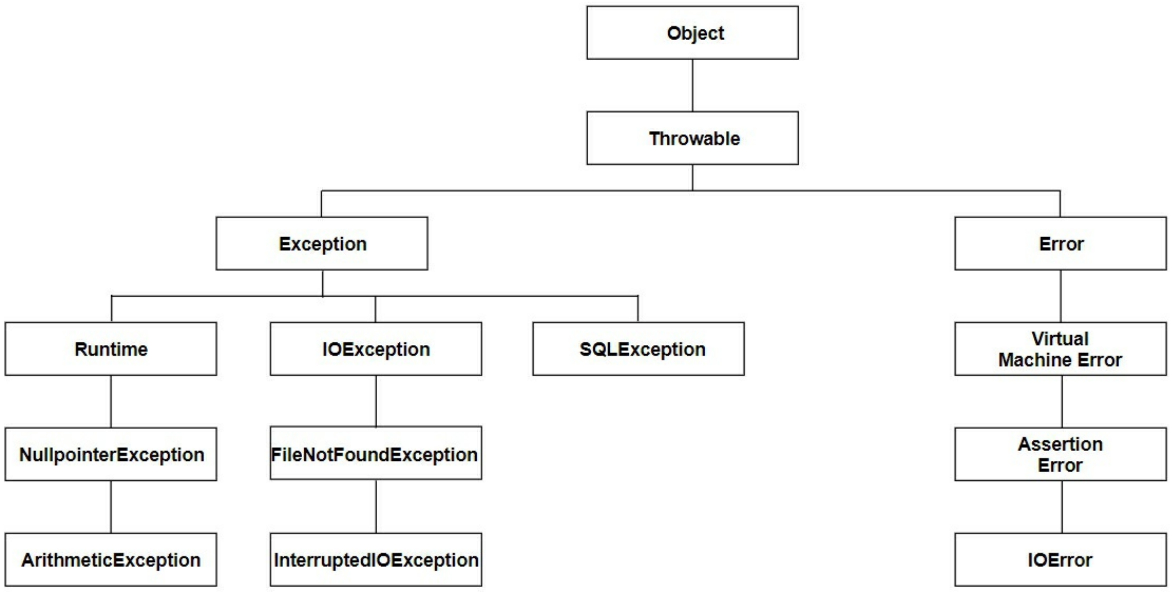
143. What is exception handling?

Exception is an error that occur at runtime.

Exception handling is the mechanism used to handle runtime errors only, compile time errors are not handled by exception handling.

144. Explain exception hierarchy?

Throwable class is the base class for exception and error.



Exception is super class for all types of exceptions.

Exceptions are categorized as checked and unchecked exceptions.

Error is parent class for all types of error.

145. What is difference between checked and unchecked exception?

Checked exceptions are checked at compile time.

The method must either handle the exception or it must specify the exception using throws keyword, if some code within a method throws a checked exception.

e.g. SQLException, IOException are checked exceptions.

Unchecked are not checked at compiled time.

Error and Runtime Exception are unchecked exceptions.

e.g. NullpointerException, ArithmeticException are unchecked exceptions.

146. What is difference between throw and throws keyword?

The throws clause is used to declare an exception and throw keyword is used to throw an exception explicitly.

Keyword throw is followed by an instance variable whereas throws is followed by exception class names.

The keyword throw is used inside method body to invoke an exception and throws clause is used in method signature.

147. Why are empty catch block a bad idea?

Empty catch block is example of bad programming, its often sign that developer saw an exception, didn’t knew what to do about it and so used empty catch block to silence the problem.

It is just equivalent to ignoring problem instead of solving it.

148. What is difference between final, finally and finalize?

Final: Final is a keyword, when used with variable it can be initialized only once.

Method declared as final cannot be overridden.

Class declared as final cannot be inherited.

Finally: Finally is block which will always be executed once try block exits . It is used to perform clean-up operations.

Finalize: finalize is a method which is called before an object is garbage collected. It can be used to release global resources.

149. What happens when exception is thrown by main method?

When exception is thrown by main method Java runtime terminates the program and print the exception message.

150. Can subclass overriding method declare an exception if parent class method doesn't throw an exception?

Overridden methods can throw exceptions as long as method being overridden also throws the same exception unless it is unchecked exception.

The reason you cannot introduce new checked exception is the, if method from superclass or interface doesn't throw an exception and you refer to your object as that type you would get unexpected behaviour.

151. What is collection framework?

Collection framework in java is set of interfaces and classes which are useful to store and manipulate group of objects.

152. What is exception propagation?

An exception is first thrown from top of the stack and if it is not caught it drops down the call stack to previous method.

When exception occurs propagation is a process in which the exception is being dropped from the top to the bottom of the stack, If not caught once the exception again drops down to previous method and so on until it gets caught. This is called as exception propagation.

153. What will be output of below program?

```
public class Exception2 {

    /**
     * @param args
     */
    public static void main(String[] args) {
        String returnVal = method1();
        System.out.println(returnVal);
    }

    public static String method1() {
        try {
            int i = 9/0;
            System.out.println(i);
        } catch (Exception e) {
            System.out.println("exception caught");
            return "from catch";
        } finally {
            System.out.println("finally block executing");
        }
        System.out.println("end");
        return "from end";
    }

}
```

Output:

```
exception caught
finally block executing
from catch
```

12. Collection

154. What is collection framework?

The Java Collections Framework is a collection of interfaces and classes which helps in storing and manipulating group of objects.

155. What are benefits of collection framework?

The Java Collections Framework provides the following benefits:

Reduces effort: The Collections Framework frees you to concentrate on the important parts of your program by providing useful data structures and algorithms.

Increases speed and quality: The Collections Framework provides high-performance, high-quality implementations of useful data structures and algorithms. The various implementations of each interface are interchangeable, so code can be easily modified by switching collection implementations. Because you're freed from writing your own data structures, you'll have more time to improve code quality and performance.

Allows interoperability among unrelated APIs: Different unrelated APIs using collection Framework can understand each other easily.

Maintenance: Since Collections framework code is open source and API documents is widely available, it is easy to maintain the code written with the help of Java Collections framework. One developer can easily understand the code of previous developer.

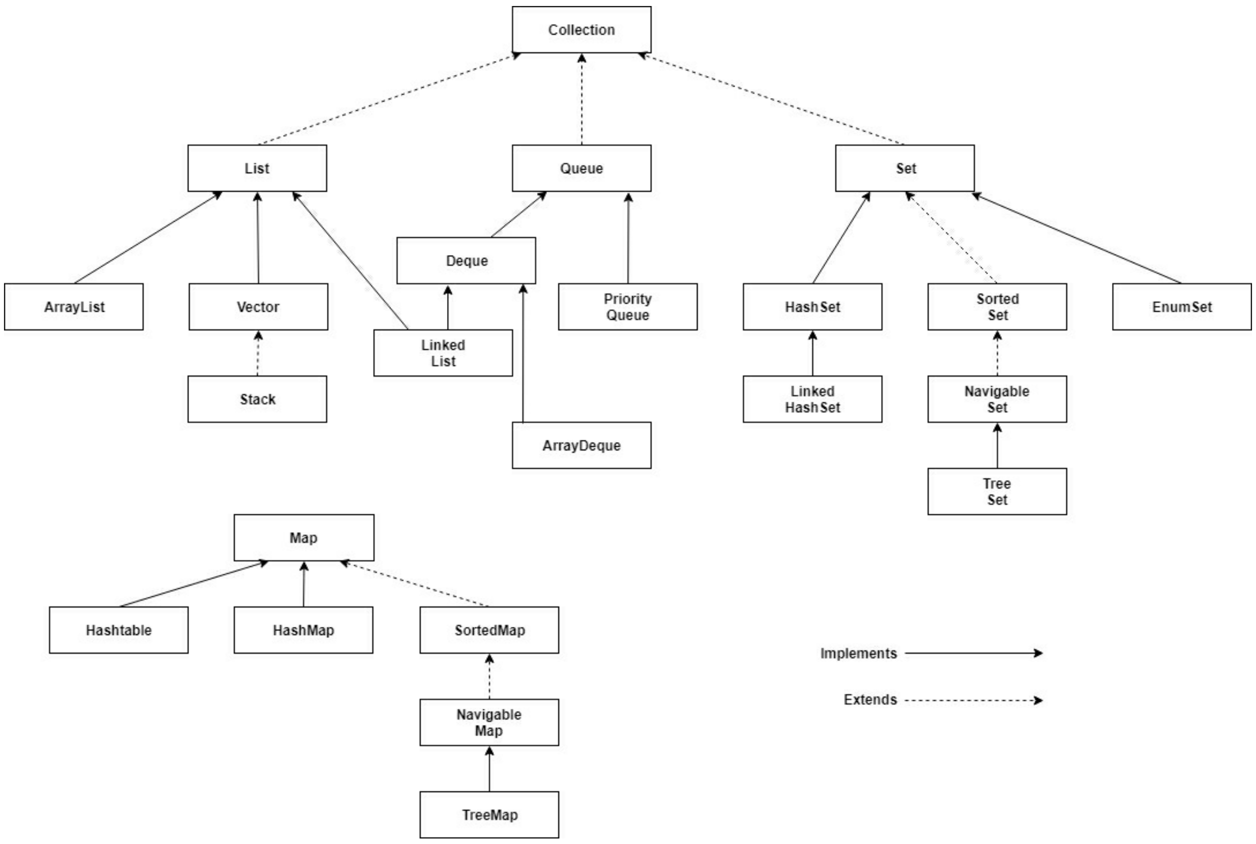
156. What is difference between List and Set?

List	Set
Duplicates are allowed in List.	Duplicates are not allowed in Set.
Elements in List are ordered.	Order of elements in Set depends upon set implementation.
Multiple nulls are allowed in List.	Only one null is allowed in Set.
Positional access is possible in List.	Positional access is not possible in

	Set.

157. Explain Collection hierarchy.

The actual hierarchy of what extends what, and what implements what, is fairly intricate. Here is a simplified hierarchy of the collections framework:



158. What is difference between ArrayList and LinkedList?

ArrayList	LinkedList
ArrayList internally uses array data structure which is contiguous block of memory.	LinkedList internally uses doubly linked list.
ArrayList is not efficient for	LinkedList is efficient for

manipulation as a lot of shifting is required.	manipulation.
ArrayList is better to store and fetch data.	LinkedList is better to manipulation data.

159. What is difference between ArrayList and Vector?

ArrayList	Vector
ArrayList is not synchronized which means multiple thread can work on it.	Vector is synchronized which means only single thread can work on it at a time.
ArrayList increases its size by 50% when elements exceeds its capacity.	Vector increases its size by 100% which doubles its size when elements exceeds its capacity.
ArrayList uses iterator to traverse through its elements	Vector can use iterator as well as enumeration to traverse through its elements

160. What is difference between HashSet, LinkedHashSet and TreeSet?

HashSet	LinkedHashSet	TreeSet
HashSet uses HashMap internally to store elements.	LinkedHashSet uses LinkedHashMap internally to store elements.	TreeSet uses TreeMap internally to store its elements.
HashSet doesn't maintain any order of elements.	linkedHashSet maintain insertion order of elements i.e. elements are placed	TreeSet orders the elements according to comparator. If no comparator is supplied

	as they are inserted.	elements are placed in their natural ascending order.
HashSet allow maximum one null element.	LinkedHashSet allow maximum one null element.	TreeSet doesn’t allow even single null element.
HashSet gives performance of order O(1) for insert, remove and retrieve operations.	LinkedHashSet also gives performance of order O(1) for insert, remove and retrieve operations.	TreeSet gives performance of order 0(log[n]) for insert, remove and retrieve operations.
If you don’t want to maintain any order of elements, use HashSet	If you want to maintain insertion order of elements, use LinkedHashSet.	If you want to sort the elements according to some comparator, use TreeSet.

161. What is difference between HashMap and HashTable?

HashMap	Hashtable
HashMap can contain one null key.	Hashtable cannot contain any null key.
HashMap contain multiple null values.	Hashtable cannot contain any null values.
HashMap is not synchronized.	Hashtable is synchronized.

162. What is load factor of ArrayList?

Load factor of ArrayList is 1.

163. What are default capacities of collections?

Below are default capacities of collections:

- ArrayList = 10
- Vector = 10
- HashSet = 16

- HashMap = 16
- Hashtable = 11
- HashSet =16

164. What is difference between collections and collection?

The collection is root interface in the collection hierarchy. A collection is basically group of objects, known as its elements.

Collections is a class which includes static methods that operate on or return collections.

165. What is iterator?

The java.util.Iterator interface is a member of the Java Collections Framework.

Iterator interface contains methods to iterate over any Collection.

166. What is difference Iterator and Enumeration?

Iterators differ from enumerations in number of ways:

Using Iterators it is possible to remove elements from the underlying collection during the iteration.

Method names have been improved. e.g. hasnext(), next(), remove() in Iterator interface vs hasMoreElements(), nextElement() in Enumeration.

Iterator is a universal cursor as it is applicable for all the collection classes whereas Enumeration applies only to legacy classes.

167. What are different ways to iterate over list?

Below are the different ways to iterate through list:

For Loop:

```
for (int i = 0; i < list.size(); i++) {//do something}
```

Advanced For Loop:

```
for (String variable : list) {//do something }
```

Iterator:

```
Iterator<String> listIterator = list.iterator();
while (listIterator.hasNext()) { // do something}
```

While Loop:

```
int i = 0;
while (i < list.size()) {
        //do something
        i++;
}
```

forEach method(Java 8):

```
list.forEach((temp) -> {
        //do something
});
```

168. How to remove duplicates from ArrayList?

LinkedHashSet is used to remove duplicates from Arraylist. LinkedHashSet removes duplicates as well as preserving their insertion order.

e.g.

LinkedHashSet<Integer> hashSet = new LinkedHashSet<>(integerArrayList);

In Java 8 we can also remove duplicates from ArrayList using stream api. We can use streams distinct method which will return distinct elements from ArrayList.

e.g.

List<Integer> distinctList = list.stream().distinct().collect(Collectors.toList());

169. How HashMap works internally?

HashMap contains an array of Node. HashMap has an inner class Entry which can be considered as a node has following fields:

```
final K key;
V value;
Entry<K ,V> next;
final int hash;
```

These nodes are stored in bucket. A bucket is one element of HashMap array.

Objects are stored by using put(key, value) method of HashMap and retrieved by using get(key) method.

So when you call put method as map.put(“Pratik”,”Bandal”); “Pratik” is key and “Bandal” value.

When we call put method, hashcode() method of the key object is called which returns hashcode to identify a bucket location to store value object, which is actually an index of the internal array.

The hashcode() method basically returns hashcode. A hashcode is an integer value which represents the state of the object upon which it was called. Integer that is set to 2 will return a hashcode of "2" because an Integer's hashcode and its value are same. A character's hashcode and it's ASCII character code are same.

Now suppose hashcode for key Pratik is 1222.

Index of bucket array in which node will be inserted is calculated as follows:

index = hashCode(key) & (n-1).

Here, n is number of buckets or the size of array.

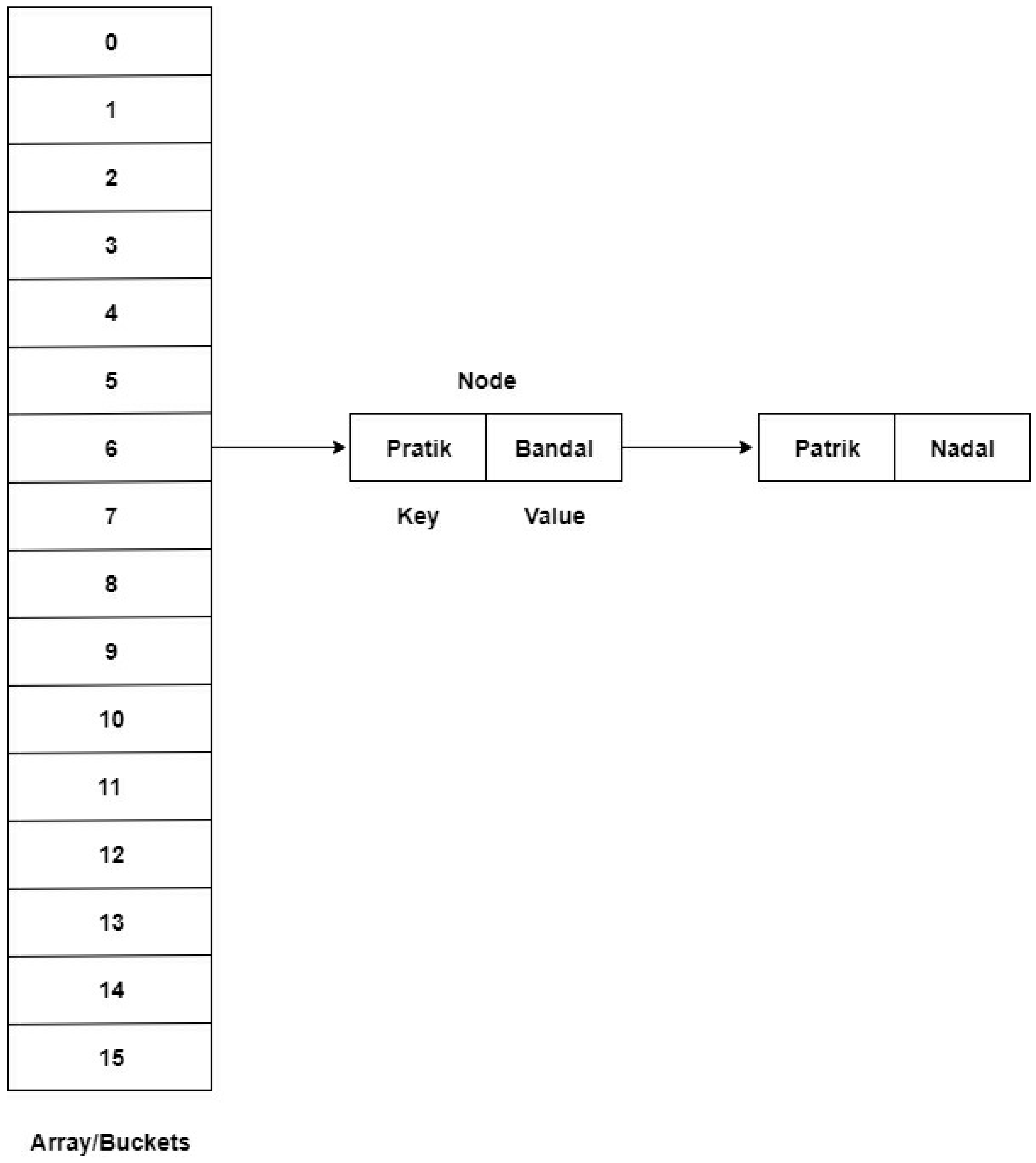
Lets say bucket evaluated for hashcode 1222 is 6, then node will be placed at 6th bucket.

Since the internal array of HashMap is of fixed size(16 by default), and if you keep storing objects, hash function will return same bucket location for two different keys having same hashcode, this is called collision in HashMap. In this scenario, a linked list is formed at that bucket location and a new entry is stored as next node.

Now, Lets say you want to store another key value pair in map with key as “Patrik” and value as “Nadal”.

Now suppose hashcode for key Patrik is also 1238.

Lets say bucket evaluated for hashcode 1238 is 6, then node will be placed at 6th bucket. This node will encounter first key-value pair having key as “Pratik”. Now Hashmap will compare if two keys “Pratik” and “Patrik” are same using equals() method. Since both are not same latest node having key as “Patrik” added next in buckets linked list and first node will point to the second node.



170. How to iterate over HashMap?

Using EntrySet:

For(Map.Entry<Long, String> a : map.entrySet()) {//do Something}

Using EntrySet and Iterator:

Iterator<Map.Entry<Long, String>> a = map.entrySet().iterator();

```
while(it.hasNext()) {//do Something}
```

Using KeySet:

```
for(Long a: map.keySet()) {//do Something }
```

Using KeySet and Iterator:

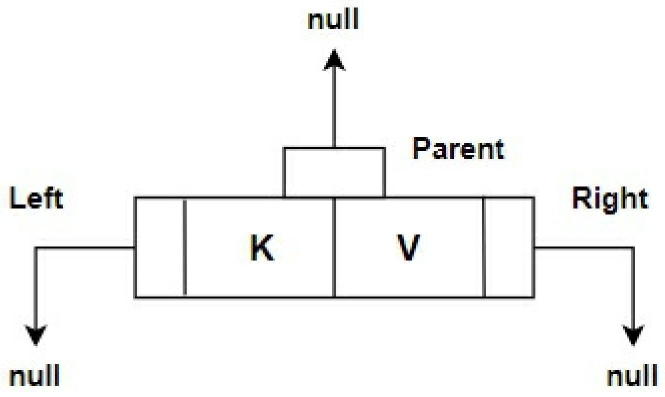
```
Iterator<Long> it = map.keySet();
while(it.hasNext()) {//do Something}
```

171. How TreeMap works in Java?

TreeMap internally uses Red-Black tree to sort elements in natural order, it also allows you to use Comparator for custom sorting implementation.

TreeMap is based on tree datastructure.

Node in tree will have three references i.e. parent, right, left.



Tree Node

When node is inserted tree formation happens in such a way that left element is always less that parent element and right element will always be greater than or equal to parent element.

172. What is difference between HashMap and TreeMap?

HashMap	TreeMap

HashMap doesn’t provide any guarantee regarding the elements order in the Map.	Elements in TreeMap are sorted in their natural order.
HashMap allows storing one null key and many null values.	TreeMap doesn’t allow storing null key but allows storing many null values.

173. How ConcurrentHashMap works?

ConcurrentHashMap object is divided into 32 segments so that at a time 32 threads can work on it.

Threads create lock on segment and not on entire object.

It doesn’t throw concurrent modification exception if one thread is modifying it and other thread is reading it.

Lock is applied on update operation and not on read operation.

174. What is Generic? What are its advantages?

Generics enable types (classes/interfaces) to be parameters when defining classes, interfaces and methods.

Generic provides below advantages:

Type Safety: We can hold only single type of values in generics. It doesn’t allow to store other objects.

Type casting not required: There is no need to type cast.

Before Generics,

```
List list = new ArrayList();
list.add(“hello”);
String s = (String) list.get(0);
```

After Generics,

```
List<String> list = new ArrayList <String> ( );
list.add(“hello”);
```

```
String s= list.get(0);
```

Compile time checking: It is checked at compile time so problem want occur at runtime.

e.g

```
List<String> list = new ArrayList<String>();
list.add(“String”);
list.add(32); //compile time error.
```

175. What are concurrent collections?

The java.util.concurrent package provides number of additions to the Java Collections Framework. These are categorized by the collection interfaces provided:

BlockingQueue is a first-in-first-out data structure that blocks when you attempt to add to a full queue, or retrieve from an empty queue.

CopyOnWriteArrayList is a concurrent alternative of synchronized List. CopyOnWriteArrayList provides better concurrency than synchronized List as it allowes multiple concurrent reader and replaces the whole list on write operation. CopyOnWriteArrayList performs better when there are multiple reader and requirement of iteration is more than writing. CopyOnWriteArrayList eliminates need to lock the collection during iteration as it don't throw ConcurrencModificationException.

ConcurrentMap is a subinterface of java.util.Map that provides useful atomic operations. These operations decides to replace or remove a key-value pair only if the key is present or decides to add a key-value pair only if the key is absent. Making these operations atomic helps avoid synchronization.

The standard general-purpose implementation of ConcurrentMap is ConcurrentHashMap, which is a concurrent parallel of HashMap.

ConcurrentNavigableMap is a subinterface of ConcurrentMap provides supports to approximate matches. The standard general-purpose implementation of ConcurrentNavigableMap is ConcurrentSkipListMap, which is a concurrent parallel of TreeMap.

176. Implement ArrayList using Array.

ArrayList internally uses array.

```
public class MyArrayList {
    private static final int SIZE_FACTOR=5;
    private Object data[];
    private int index;
    private int size;
    public MyArrayList(){
            this.data=new Object[SIZE_FACTOR];
            this.size=SIZE_FACTOR;
    }
    public void add(Object obj){
            System.out.println("index:"+this.index+"size:"+this.size+"d
            if(this.index==this.size-1){
            //we need to increase the size of data[]
            increaseSizeAndReallocate();
            }
            data[this.index]=obj;
            this.index++;
    }
    private void increaseSizeAndReallocate() {
            this.size=this.size+SIZE_FACTOR;
            Object newData[]=new Object[this.size];
            for(int i=0; i<data.length;i++){
            newData[i]=data[i];
            }
            this.data=newData;
```

```
                System.out.println("***index:"+this.index+"size:"+this.size
size:"+this.data.length);
        }
        public Object get(int i) throws Exception{
                if(i>this.index-1){
                throw new Exception("ArrayIndexOutOfBound");
                }
                if(i<0){
                throw new Exception("Negative Value");
                }
                return this.data[i];
        }
        public void remove(int i) throws Exception{
                if(i>this.index-1){
                throw new Exception("ArrayIndexOutOfBound");
                }
                if(i<0){
                throw new Exception("Negative Value");
                }
                System.out.println("Object getting removed:"+this.data[i]);
                for(int x=i; x<this.data.length-1;x++){
                data[x]=data[x+1];
                }
                this.index--;
        }

        public static void main(String[] args) throws Exception {
                MyArrayList mal = new MyArrayList();
                mal.add("0");
```

```
                    mal.add("1");
                    mal.add("2");
                    mal.add("3");
                    mal.add("4");
                    mal.add("5");
                    mal.add("6");
                    mal.add("7");
                    mal.add("8");
                    mal.add("9");
                    mal.remove(5);
                    System.out.println(mal.get(7));
            }
        }
```

13. Multithreading

177. What is multithreading?

Multithreading is a process of executing two or more threads concurrently.

178. What is thread?

Thread is basically light weight subprocess as it consumes less CPU resources as compared to process.

Each process has atleast one thread know as main thread and can have multiple threads.

JVM creates main thread which invokes main() method.

Thread is separate path of execution.

179. What is context switching?

Context switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from same point as earlier. This is feature of multitasking operating system and allows a single CPU to be shared by multiple processes.

180. What is difference between process and thread?

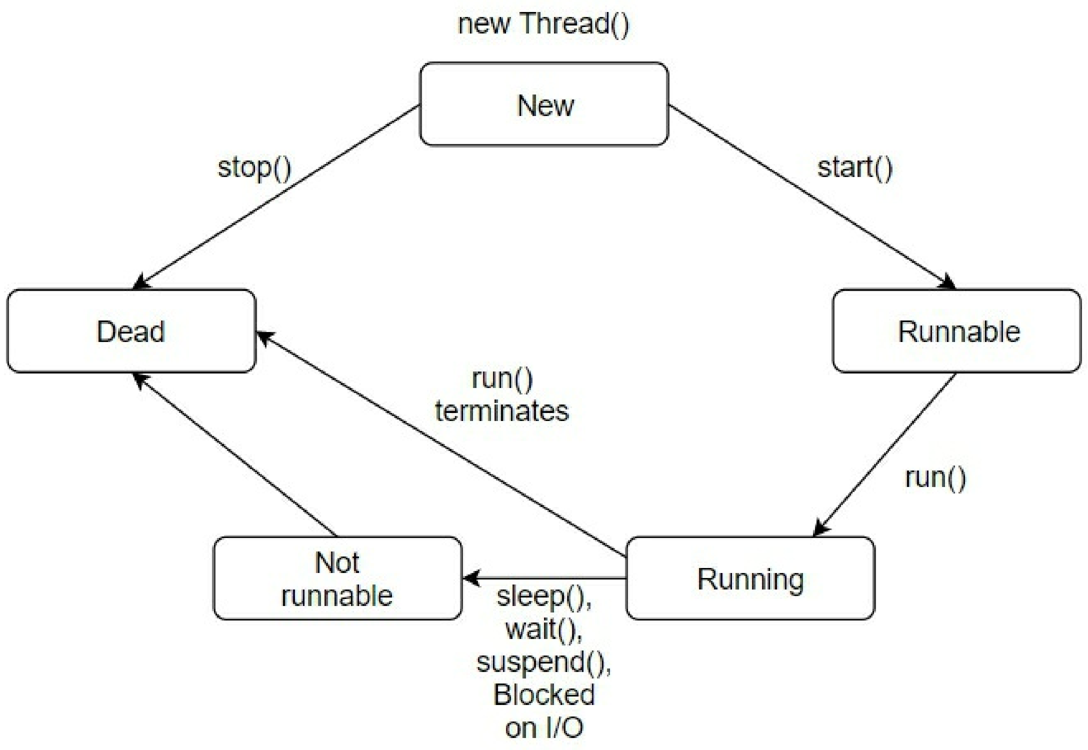
Process	Thread
Process is basically program in execution.	Thread is lightweight process, one process can have multiple threads.
Process do not share memory with each other.	Thread do share memory with each other.
Context switching slower as compared to thread.	Context switching faster.
Consumes more CPU resources i.e. memory and processor time.	Consumes less CPU resources i.e. memory and processor time.

181. What is difference between ‘extends Thread’ and ‘implements Runnable’?

Both are used to define thread but when you write extends Thread there is no scope to extend another class.

But with implements Runnable interface you can still extend other class and take benefit of code reusability through inheritance.

182. Explain Thread life cycle.



Upon creation of thread it is in NEW state.

Once start() method is called on thread object it goes into RUNNABLE state.

When start() method calls run() method thread is in RUNNING state.

When other threads are holding on the resources the current thread is said to be in NOT RUNNABLE state.

When execution of run method terminates thread is said to be in DEAD state.

183. Which method is executed by thread by default?

public void run()

184. Why can’t we start thread twice?

We can only start a thread having status “NEW”. When we call thread.start() for the first time it is in “NEW” state and then goes into READY state.

If you allow creating many threads what would you expect getState() and getStackTrace() methods to return?

When we call it second time it checks status in Threads start() method. If it is other than “NEW” it will throw IllegalThreadStateException();

185. Why do we set thread priorities as there is no guarantee by which a thread will be executed?

Thread priorities can change from 1 to 10.

However Thread priority is only hint to OS task scheduler.

Task scheduler will try to allocate more resources to thread with highest priority, but there are no explicit guarantees.

186. What is synchronized method and synchronized block? Which is more preferred?

Synchronized block is useful to synchronize a block of statement, Synchronized keyword is useful to synchronize entire method.

Synchronized block doesn’t lock the object, hence it is more preferred way.

Synchronized method lock entire object of class in which that method is present and if there are multiple synchronization blocks or method in the same class even though they are not related will be stopped from execution.

187. What is daemon thread?

Daemon thread is low priority thread that runs in background to perform tasks such as garbage collection.

The setDaemon(true) method can be used to create daemon thread. If you call setDaemon(true) after calling start method it will throw IllegalThreadStateException.

188. Why wait(), notify() and notifyAll() must be called from synchronized block or method?

We use wait(), notify() and notifyAll() method mostly for inter thread communication.

wait() method causes current thread to wait until another thread invokes notify() or notifyAll() method for this object. The current thread must own this objects monitor(mutually exclusive lock) that it will leave and goes into wait state until another thread calls notify() method on this object.

Similarly when thread calls notify() method on any object it leaves the monitor on the object so that other waiting threads can get monitor on that object.

Since these methods require object to have monitor it can be achieved through synchronized method or block.

189. What is volatile keyword?

Volatile keyword is used with variable which indicate java compiler and JVM to read its value directly from the memory and not to cache it.

Volatile variable can be used in multithreading so that multiple threads can access it simultaneously.

However, volatile keyword does not provide atomicity as multiple threads can simultaneously update it and hence its use is restricted to very limited cases.

190. What is thread local?

The ThreadLocal class enables you to create variables that can only be read and written by same thread.

Every thread has its own thread local variable and they can use its own get() and set() methods to get the default value.

All thread of an object share its variable so if variable is not thread safe we can use synchronization but if we want to avoid synchronization we can use ThreadLocal.

191. What is Thread dump?

Thread dump is list of all active threads in JVM.

Thread dumps are very useful in analysing bottleneck in application and analysing deadlock situation.

There are many ways using which we can generate thread dumps-Profiler, Kill -3 command, jstock tools etc.

192. What is deadlock?

Deadlock is programming situation where two or more threads are blocked forever waiting for each other.

193. What is Thread pool?

Pool of worker threads is managed by Thread pool.

A thread pool reuses previously created threads to execute current tasks and also provides a solution to the problem of thread cycle overhead and resource thrashing.

java.util.concurrent.Executors provide implementation of java.util.concurrent.Executor interface to create the thread pool which is used to manage worker thread.

194. Why Thread.destroy() and Thread.stop(Throwable) methods are removed in Java 11?

The Thread.destroy() and Thread.stop(Throwable) methods have been deprecated for many years, and in Java 9 they were both deprecated for removal.

The Thread.destroy() method never actually destroyed a thread in any release.

Instead, it would throw NoSuchMethodError to the caller.

The Thread.stop(Throwable) method had been implemented in the past, but in JDK 8 the implementation was changed not to stop the target thread, but instead to throw UnsuppportedOperationException to the caller.

14. JDBC

195. What is JDBC?

JDBC is API using which application can execute SQL statements, retrieve results and propagate results back to an underlying data source.

196. What is JDBC driver?

JDBC drivers are client side adaptors that convert request from application to a protocol that DBMS can understand.

There are 4 types of JDBC drivers:

Type-1 driver or JDBC-ODBC bridge driver.

Type-2 driver or Native-API driver.

Type-3 driver or Network Protocol driver.

Type-4 driver or Thin driver.

197. What is JDBC DriverManager?

JDBC Driver manager is factory class which gives the database connection object.

The DriverManager provides a service for managing a set of JDBC drivers.

JDBC Driver is registered with DriverManager.

When getConnection is called the DriverManager will attempt to locate a suitable driver from amongst those loaded at initialization and those loaded explicitly using the same classloader as the current application.

198. Write JDBC program to select data from table.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;

public class JdbcSelectFromTableExample {

    public static void main(String[] args) throws Exception {

        // Step - I
        Class.forName("com.mysql.jdbc.Driver");
        // Step - II
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/EMP", "username",
"password");
        // Step -III
        Statement stmt = con.createStatement();
        // Step - IV
        ResultSet rs = stmt.executeQuery("select * from employee");
        // Step - V
        while (rs.next()) {
            System.out.println(rs.getInt(1) + " " + rs.getString(2) + " "
                    + rs.getString(3));
        }
        // Step - VI
        rs.close();
        stmt.close();
        con.close();
    }

}
```

199. What are the types of JDBC Statements available?

Below are types of JDBC statements,

Statement:

It can be used for general-purpose(retrieve, modify) access to the database. It provides methods to execute queries with the database.

PreparedStatement:

The PreparedStatement interface accepts input parameters at runtime. It can be used instead of executing same SQL statement many times.

CallableStatement:

CallableStatement can be used when you want to access stored procedures in database.

200. What are the advantages of PreparedStatement over Statement?

PreparedStatement prevents SQL injection attack.

PreparedStatement is pre-compiled and then retained in memory by the system, rather than being compiled each and every time the statement is called. This allows for faster execution as compared to Statement.

201. What is ResultSet?

The java.sql.ResultSet interface represents a database result set, which is usually generated by executing a statement that queries the database.

A ResultSet object also maintain a cursor which points to its current row of data.

The cursor is positioned before the first row initially.

The next method moves the cursor to the next row and it returns false when there are no more rows in the ResultSet object, this behaviour can be useful in a while loop for iterating through the result set.

202. What is scrollable resultset?

It is result set in which moving in forward and backward direction is possible.

It also provides methods which can be used to update rows in result set.

203. What is BLOB?

BLOB is abbreviated as binary large object which is SQL datatype used to store binary data like images.

204. What is CLOB?

CLOB is abbreviated as character large object which is SQL datatype used to store large text data.

205. How to turn off auto commit in JDBC?

Using connection.setAutoCommit(false) it is possible to turn off the auto commit in JDBC.

206. What is difference between execute, executeQuery and executeUpdate?

Any SQL query can be executed using execute() method.

Select Query can be executed by executeQuery() method.

Queries used to manipulate/update table are executed using executeUpdate() method.

15. Version wise features

207. What are important features introduced in Java 5?

The important features of Java 5 are as below:

- For-each loop
- Varargs
- Static import
- Autoboxing and unboxing
- Enum
- Covariant return type
- Annotations
- Generics

208. What is for-each loop introduced in Java 5?

Java 5 introduced an for-each loop, which is called enhanced for each loop.

It is used to iterate over elements of an array and the collection.

It is known as for-each loop because it traverse the array or collection one by one.

The syntax of java for-each loop consist of data_type with variable followed by : then array or collection.

```
for(data_type variable: collection) {
        //body
 }
```

209. What is Varargs introduced in Java 5?

In JDK 5, Java provides a feature that simplifies the creation of methods that need to take a variable number of arguments. This feature is called varargs which is short-form for variable-length arguments.

Although multiple arguments must be passed in an array, the varargs feature automates and hides the process. Furthermore, it is upward compatible with preexisting APIs.

So, for example, consider format method has this declaration:

```
public static String format(String pattern, Object... arguments);
```

The three periods after the final parameter's type specify that the final argument may be passed as an array or as a sequence of arguments.

210. What is static import?

This is already explained in Q24.

211. What is Autoboxing and Unboxing?

This is already explained in Q132.

212. What is Enum?

An enum type is a special data type which is basically collection of constants.

Examples of include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week.

The names of an enum type's fields are in uppercase letters because they are constants.

You have to define an enum type by using the enum keyword.

Below is the example of enum,

```
public enum Day {
        SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
        THURSDAY, FRIDAY, SATURDAY
    }
```

213. What is Covariant return type introduced in Java 5?

This is already explained in Q107.

214. What is generics in Java 5?

This is already explained in Q176.

215. What are new features introduced in Java 6?

The important feature of Java 6 are as below:

- Premain method

216. What is premain method introduced in Java 6?

The premain method is a mechanism associated with java.lang.instrument package, used for loading "Agents" which makes byte-code changes in Java programs.

217. What are new features introduced in Java 7?

Few important features introduces in Java 7 are as below:

- String in switch statement
- Binary literals
- Try with resources
- Caching multiple exceptions by single catch
- Underscore in numeric literals

218. What is String in switch statement in Java 7?

In Java 7 we can use string with switch statement.

The switch statement compares string object in its expression with the expression associated with each case label as if it were using String.equals method.

Comparison of string object in switch statement is case sensitive.

The java compiler generates more efficient bytecode from switch statements that use string object than from chained if-then-else statements.

219. What is binary literals in Java 7?

Java 7 added binary literals which allows you to express integral types (byte, short, int and long) through binary number system.

To specify binary literal add prefix 0b or 0B to the integral value.

e.g.

```
// An 8-bit 'byte' value:
byte aByte = (byte)0b00100001;

// A 16-bit 'short' value:
short aShort = (short)0b1010000101000101;

// Some 32-bit 'int' values:
int anInt1 = 0b10100001010001011010000101000101;
int anInt2 = 0b101;
int anInt6 = 0B101;

// A 64-bit 'long' value. Note the "L" suffix:
long                                          aLong
0b101000010100010110100001010001011010000101000101101000
```

220. What is try with resources in Java 7?

As name suggests, the try-with-resources statement is a try statement that declares one or more resources.

A resource is an object which must be closed after the program is finished with it.

The try-with-resources statement is used to ensure that each resource is closed at the end of the statement.

e.g.

```
static String readFirstLineFromFile(String path) throws IOException {
    try (BufferedReader br =
                   new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

In this example, BufferedReader is the resource declared in the try-with-resources statement.

The declaration statement of resource appears within parentheses immediately after the try keyword.

In Java 7 and later, The BufferedReader class implements the interface java.lang.AutoCloseable.

BufferedReader instance will be closed regardless of whether the try statement completes normally or abruptly Because it is declared in a try-with-resource statement

221. Explain catching multiple exception with single catch in Java 7.

Since Java 7, a single catch block can be used handle more than one type of exception.

This feature can reduce code duplication and lessen the inclination to catch broad exception.

The following example of single catch for multiple exception is valid in Java 7 and later, eliminates the duplicated code:

```
catch (IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
```

}

222. Explain underscore in numeric literals in Java 7?

In Java 7 and later, any number of underscore characters (_) can appear anywhere between digits in a numerical literal.

Using this feature you will be able to separate groups of digits in numeric literals.

This can improve the readability of your code.

In a scenario where your code contains numbers with many digits, you can use an underscore character to separate digits in multiple groups.

e.g.

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi =      3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

223. What are new features introduced in Java 8?

There are lots of new features introduced in Java 8, the most significant ones are as below:

- Lambda expression
- Stream
- Java 8 Date/time
- Functional Interface
- Default methods
- Method references
- forEach method

- Collectors
- StringJoiner
- Optional Class
- Nashhorn

224. What are lambda expressions introduced in Java 8?

Lambda expression add functional processing capability to Java.

Lambda expression is treated as function so compiler does not create .class file.

Lambda expression has 3 parts:

Parameters: Lambda expression has zero or more parameters.

Arrow operator: Arrow operator “->” separates parameters from body.

Body: It contains expressions and statements for lambda expressions.

e.g.

()-> System.out.println(“Hello World”);

Lambda expression helps to iterate, filter and extract data from collection as well as it is used to provide implementation to functional interface.

225. What is stream API introduced in Java 8?

Stream interface is defined in java.util.Stream package.

Stream can be defined as a sequence of elements from source that supports aggregate functions.

Common aggregate operation are filter, map, reduce, find, match, sort.

Stream supports pipelining and internal iterations.

In java 8 stream are designed in such way that most of its stream operations return stream only.

This help us creating chain of various stream operation. This is called pipelining.

e.g.

List<string> names =

students.stream().map(student::getName). filter(name –> name.startswith('A')).collect(Collectors.toList());

226. What are date time improvements in Java 8?

Java 8 introduced new Date-Time API to handle issues like thread safety and difficulty in time zone handling in old date time APIs.

Java 8 introduces Local and Zoned classes in java.time package.

Local Date-Time API: LocalDate/LocalTime and LocalDateTime classes simplify the development where time zones are not required.

Zoned Date-Time API: Zoned Date-Time API are used when time zones are considered.

Date Time API provides below methods:

To get Zone Id:

ZoneId zoneId = ZoneId.of("Europe/Paris");

To get all Zone ids:

Set<String> allZoneIds = ZoneId.getAvailableZoneIds();

To convert LocalDateTime to specific zone:

ZonedDateTime zonedDateTime = ZonedDateTime.of(localDateTime, zoneId);

227. What is functional interface introduced Java 8?

This is already explained in Q128.

228. What is default method introduced in Java 8?

Java 8 introduces default method which allows developer to add new methods to interface without breaking existing implementation of these interfaces.

Default method of interface can have body, this method will be available to implementations.

e.g.

```
public interface oldInterface {
        public void existingMethod();
        default public void newDefaultMethod() {
                    System.out.println(“New method in interface”);
        }
}
```

229. What is method reference in Java 8?

A method reference is short hand syntax for a lambda expression that executes only one method.

In a method reference you can place object or class that contains method before the :: operator and the name of the method after it without arguments.

e.g.

```
Math::max is equivalent to Math.max(x);
```

230. What is forEach method in Java 8?

Java 8 has introduced forEach method which can be used to loop over List or Collection.

Using forEach method you can iterate through any collection e.g. List, Set or Map by converting them into Stream instance and then calling forEach() method.

e.g.

```
List<Integer> listOfIntegers = Arrays.asList(1,2,3,4,5,6,7);
listOfIntegers.stream().forEach(i->System.out.println(i));
```

231. What is Collectors introduced in Java 8?

The Java.util.Collectors implement various useful reduction operation such as accumulating elements into collections, summarizing elements according to various criteria etc.

e.g.

```
// Accumulate names into a List
List<String>list=people.stream().map(Person::getName).collect(Collec
String joined = things.stream().map(Object::toString)
                                .collect(Collectors.joining(", "));

// Compute sum of salaries of employee
int total = employees.stream()
                               .collect(Collectors.summingInt(Employee::get
```

232. What is StringJoiner introduced in Java 8?

This is already explained in Q54.

233. What is Optional Class introduced in Java 8?

Optional is basically a container object that is used to contain not-null objects.

With Optional object null is represented as absent value.

This class has various utility methods which enables you to handle values as ‘available’ or ‘not available’ instead of checking null values.

234. What is Nashhorn introduced in Java 8?

Nashorn is javascript engine.

It is used to execute Javascript code dynamically in JVM.

Java 8 provides a command-line tool jjs which is used to execute Javascript code.

e.g.

jjs sample.js

235. What are new features introduced in Java 9?

There are lots of new features introduced in Java 9, the most significant ones are as below:

- Java Platform Module System
- JShell: The Java Shell (REPL)
- Private interface methods
- New Version-String Scheme
- Stream API improvements
- Improved Javadoc

236. What is java platform module system introduced in Java 9?

Java 9 introduces java platform module system which is basically group of closely related packages and resources along with a new module descriptor file.

In Java 8 and earlier versions of java as JDK is too big it is difficult to scale down to small devices, Java 9 will divide JDK, JRE and JARs into smaller modules so that we can use whatever module that we want.

When you install Java 9, you can see that the JDK now has a new structure.

The modules are split into four major groups: java, javafx, jdk, and Oracle.

Modular JAR files contains modular descriptor. In this modular descriptor dependencies on other modules are expressed through ‘requires’ statement.

Also, ‘export’ statement define which packages inside module are accessible to other modules.

All non-exported packages are encapsulated in module by default.

237. What is JShell introduced in Java 9?

Jshell which stands for Java shell is interactive tool for learning the Java programming language.

Jshell is standard component of JDK 9.

Jshell can be started by simply executing “jshell” command.

e.g.

```
$jshell
```

To display the “Hello, World” in jshell all you have to write is this:

```
jshell>System.out.println(“Hello, World!!!”);
Hello, World!!!
```

Main advantage of jshell is that you can test your individual statements, methods etc.

Jshell can be stopped by executing “/exit” command.

238. What is private interface methods in Java 9?

This is already explained in Q127.

239. What is New version string scheme in Java 9?

In Java 9, a new version-string scheme is introduced that follows below pattern:

$MAJOR.$MINOR.$SECURITY

$MAJOR: The major version number, incremented for a major release that contains significant new features as specified in a new edition of the Java SE Platform Specification. The $MAJOR of JDK 8 is 8; the $MAJOR of JDK 9 is 9. When $MAJOR is incremented, all elements coming after that are removed.

$MINOR: The minor version number, incremented for a minor update release that may contain compatible bug fixes, revisions to standard APIs and implementation features outside the scope of that Specification.

$SECURITY: The security level, incremented for a security-update release that contains critical fixes including those necessary to improve security. $SECURITY does not reset to zero when $MINOR is incremented. A higher value of $SECURITY for a given $MAJOR value, therefore, always indicates a more secure release, regardless of the value of $MINOR.

240. What are stream API improvements in Java 9?

Java 9 has introduced following methods to java.util.Stream interface:

dropWhile: dropWhile() method drops all the values until the predicate returns true.

e.g.

```
Stream<Integer> stream = Stream.of(1,2,3,4,5,6,7,8);
Stream.dropWhile(x->x<5).forEach(a->System.out.println(a));
```

```
5
6
7
8
```

takeWhile: takeWhile() method takes all the values until the predicate returns false. In case of ordered stream it returns a stream consisting of the longest prefix of elements taken from this stream matching the given predicate.

e.g.

```
Stream<Integer> stream = Stream.of(1,2,3,4,5,6,7,8);
Stream.takeWhile(x->x<5).forEach(a->System.out.println(a));
```

```
1
2
3
```

iterate: iterate() method now has hasNext predicate as parameter which stops the loop when hasNext returns false.

e.g.

```
Stream<Integer> stream = Stream.of(1,2,3,4);
Stream.iterate(3, x->x<10, x->x+3).forEach(a->System.out.println(a));
```

```
3
6
9
```

ofNullable: ofNullable() method returns a sequential stream containing single element if not null else if it receives null element it returns empty stream.

e.g.

```
long count = Stream.ofNullable(1).count()
System.out.println(count);
1
long count = Stream.ofNullable(null).count()
System.out.println(count);
0
```

241. What is improved Javadoc feature in Java 9?

Before Java 9, Javadoc used to be generated in html 4 format.

From Java 9 you can generate javadocs in html 5 format by using –html5 option in command line arguments.

242. What are new features introduced in Java 10?

There are lots of new features introduced in Java 10, the most significant ones are as below:

- Time-Based Release Versioning
- Local-Variable Type Inference
- Garbage Collector Interface
- Parallel Full GC for G1
- Root Certificates
- Bytecode Generation for Enhanced for Loop

243. What is Local-Variable Type Inference introduced in Java 10?

This new feature will allow you to declare and initialize local variables with var, rather than specifying a type.

When var is used instead of a type, the type of the variable is concluded from its initial value by compiler.

This is especially useful if the type has a long name or is a complex parameterized type or the type is redundant with the initial value.

Using var you can possibly make code more concise without sacrificing readability.

In some cases var can improve readability by removing redundancy.

e.g.

```
var someVar = "Value for someVar";
```

```
var messages = new ArrayList<String>();
```

244. What is Time-Based Release Versioning introduced in java 10?

With Java 10 versioning semantics have been changed to $FEATURE.$INTERIM.$UPDATE.$PATCH.

$FEATURE is incremented every six months. For example, March 2018 release is JDK 10, September 2018 release is JDK 11, and so on.

$INTERIM is always zero, because the six-month model does not include interim releases. We reserve it here for flexibility, so that a future revision to the release model could include such releases. For examples, the JDK 1.4.1 and 1.4.2 releases were basically interim releases, and would have been numbered 4.1 and 4.2 under this scheme.

$UPDATE is incremented one month after $FEATURE is incremented, and every three months after that. For example, April 2018 release is JDK 10.0.1, July release is JDK 10.0.2, and so on.

$PATCH: The emergency patch-release counter, incremented only when it's necessary to produce an emergency release to fix a critical issue.

Now, consider below example of java version information,

```
$ java -version
```

```
java version "10" 2018-03-20
```

Java(TM) SE Runtime Environment 18.3 (build 10+46)

The version is “10”.The date of release is added. 18.3 can be read as Year 2018 & 3rd Month, build 10+46 is 46th build for version 10.

245. What is Garbage collector interface introduced in Java 10?

This is already explained in Q39.

246. What is Parallel Full GC for G1 in Java 10?

This is already explained in Q40.

247. What is significance of Root Certificates introduced in Java 10?

Java 10, provides a default set of root Certification Authority (CA) certificates in the JDK.

The root (CA) certificates make OpenJDK builds more attractive to developers, and to reduce the differences between those builds and Oracle JDK builds.

The cacerts keystore, is intended to contain a set of root certificates that can be used to establish trust in the certificate chains employed in various security protocols.

The cacerts keystore in the JDK source code, however, before Java 10 were empty. As a result, critical security components such as TLS did not work by default in OpenJDK builds.

248. What is bytecode Generation for Enhanced for Loop in Java 10?

This is already explained in Q41

249. What are new features introduced in Java 11?

There are lots of new features introduced in Java 11, the most significant ones are as below:

- New String methods like isBlank(), lines(), strip(), repeat()
- New File methods like writeString(), readString(), isSameFile()
- TimeUnit conversion method
- Local variable syntax for lambda parameters
- Deprecate the Nashorn JavaScript Engine
- Epsilon, A No-Op Garbage Collector
- Removal of Thread.destroy() and Thread.stop(Throwable) Methods

250. What are new String methods introduced in Java 11?

This is already explained in Q55.

251. What are new file methods introduced in Java 11?

writeString():

The java.nio.file.Files class has two overloaded static methods to write content to file.

public static Path writeString(Path path, CharSequence csq, OpenOption... options) throws IOException

Write a CharSequence to a file. Characters are encoded into bytes using the UTF-8 charset.

This method is equivalent to: writeString(path, test, StandardCharsets.UTF_8, options)

public static Path writeString(Path path, CharSequence csq, Charset cs, OpenOption... options) throws IOException

Write a CharSequence to a file. Using the specified charset characters are encoded into bytes.

All characters are written as they are, including the line separators in the char sequence. No extra characters are added.

The options parameter specifies how the file is created or opened. This method opens the file for writing, creating the file if it doesn't exist, or initially truncating an existing regular-file to a size of 0.

readString():

The java.nio.file.Files class has two overloaded methods to read content from file.

readString(Path path)

Reads all content from a file into a string, uses the UTF-8 charset for decoding bytes to characters.

readString(Path path, Charset cs)

Reads all characters from a file into a string, uses the specified charset for decoding bytes to characters.

isSameFile():

isSameFile(Path path, Path path2)

Tests if two paths locate the same file.

252. What is TimeUnit conversation method in Java 11?

The java.util.concurrent.TimeUnit has public long convert(Duration duration) method added in Java 11.

Converts the given time duration to this unit.

e.g.

TimeUnit time = TimeUnit.NANOSECONDS;

time.convert(35L, TimeUnit.MINUTES);

In above example, 35L which is time in minutes will be converted to TimeUnit objects unit i.e. NANOSECONDS.

253. What is local variable syntax for lambda parameters in Java 11?

Local var syntax can be used with lambda expression in Java 11.

e.g. (var a1, var a2) -> a1 + a2

This makes the usage of var uniform in both local variables and lambda parameters.

254. What is Epsilon Garbage Collector introduced in Java 11?

This is already explained in Q42.

255. Why Thread.destroy() and Thread.stop(Throwable) methods are removed in Java 11?

This is already explained in Q196.

256. What are new features introduced in Java 12?

There are lots of new features introduced in Java 12, the most significant ones are as below:

- Switch expressions
- Default CDS archives
- Shenandoah
- Strings New Methods
- JVM constants API
- Support for Unicode 11
- Support for Compact Number Formatting
- Teeing Collectors

257. What is Switch expression in Java 12?

Expression label:

Instead of having different cases, you can use the new switch label "case L -> expression" which allows the expression on the right to execute if the label matches. This assists in making code easier to read/understand in addition to make switch statements quicker to type.

e.g.

```
switch (x) {
    case 1 -> System.out.println("Foo");
    default -> System.out.println("Bar");
}
```

Multiple case labels:

Java 12 will allow you to list multiple case labels on the same line rather than forcing the fall through semantics of switch statements. This has been done to make code both easier to read and easier to understand.

An example of this in use would be:

```
switch (x) {
    case 1, 2, 4 -> System.out.println("Foo");
    default -> System.out.println("Bar");
}
```

Returning values from the switch statement:

Java 12 allows switch statements to return values.

Before Java 12, in a lot of scenarios depending on the input provided switches are used to return specific values. This removes the necessity to create a variable specifically for the purpose of returning a set value.

Example:

```
int y = switch (x) {
    case 1 -> 2;
    case 2 -> 4;
    case 3 -> 3;
    default -> 1;
};
```

258. What JVM improvements are done through default CDS Archives in Java 12?

For each used class JVM loads the bytes, verifies the bytecode and then pulls it into an internal data structure, this process is repeated every time the JVM is relaunched, even though, as long as the class is unchanged, it always leads to the same result.

Class-data sharing (CDS) removes the redundancy by storing the internal data structure called as class-data archive, in a file and then mapping it in memory on future launches.

Now Java 12 ships with an archive for the JDK classes and uses it by default, which can be turned off by -Xshare:off.

It is observed that launching java application in Java 12 with CDS on(enabled by default) takes lot less time than CDS off(disabled using -Xshare:off).

259. What is Shenandoah in Java 12?

This is already explained in Q43.

260. What are Strings new methods introduced in Java 12?

This is already explained in Q56.

261. What does support for Unicode 11 in Java 12 signify?

In a time in which emojis play a crucial role in communicating on social media channels, it’s more important than ever to support the latest Unicode specification.

Hence, Java 12 now supports Unicode 11.

Unicode 11 adds 684 characters in a total of 137,374 characters and seven new scripts in a total of 146 scripts.

262. What does Support for Compact Number Formatting in Java 12 signify?

NumberFormat adds support for formatting a number in its compact form.

Compact number formatting refers to the representation of a number in a short or human readable form.

For example, 1000 can be formatted as "1K" and 1000000 can be formatted as "1M" in the en_US locale.

To obtain an instance, use one of the factory methods given by NumberFormat for compact number formatting. For example:

```
NumberFormat fmt = NumberFormat.getCompactNumberInstance(Locale.US, NumberFormat.Style.SHORT);
String result = fmt.format(1000);
```

The example above results in "1K".

263. What are teeing collectors in Java 12?

Java 12 has introduced Teeing Collector utility in the Streams API.

This collector has three arguments – Two collectors and a Bi-function.

All input values are passed to each collector and the result is available in the Bi-function.

e.g.

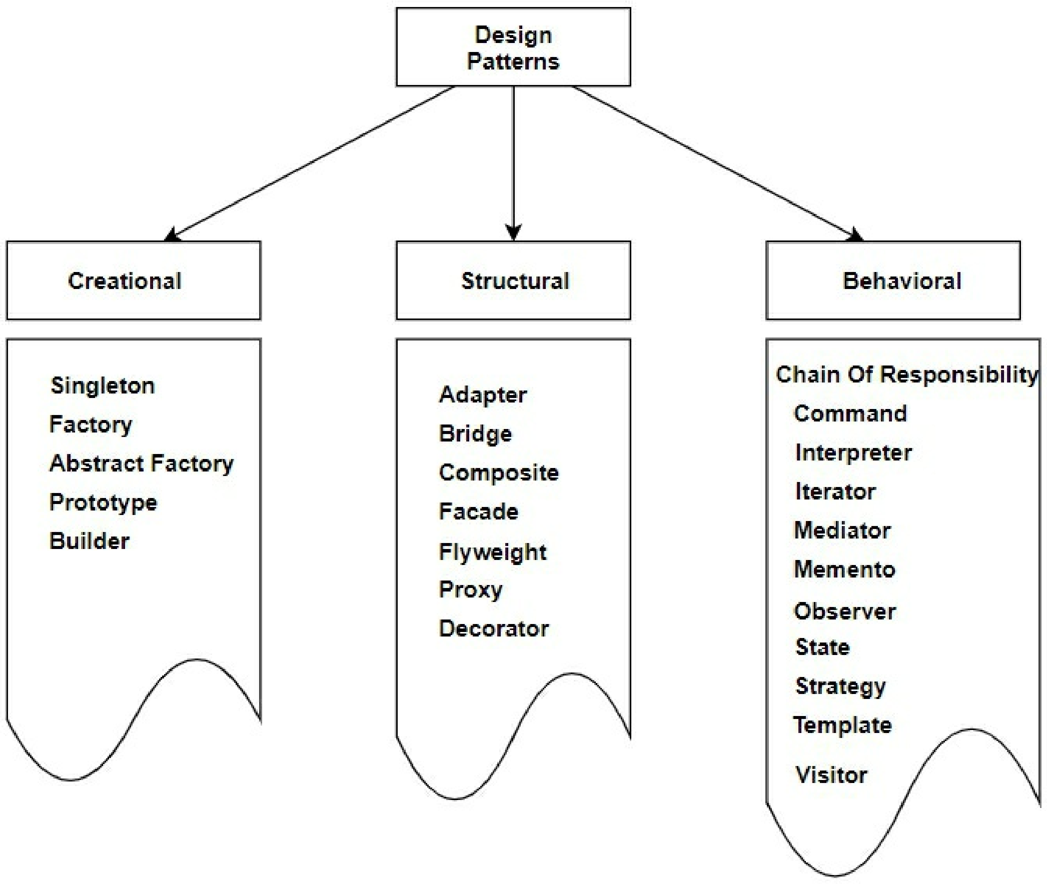
```
double mean = Stream.of(1, 2, 3, 4, 5)
                .collect(Collectors.teeing(
                        summingDouble(i -> i),
                        counting(),
                        (sum, n) -> sum / n));
```

16. Design Pattern

264. What are different categories of Java design pattern?

Three categories of design patterns are creational, structural and behavioural.

Below are different design patterns:



265. Explain Singleton design pattern.

The singleton design pattern is used to ensure that only one instance of the singleton class is created and also provides global point of access to that instance.

It is a creational design pattern.

266. In how many ways object is initialized in singleton design pattern?

There are two ways of creating a Singleton pattern:

Early Instantiation

In Early Instantiation the creation of instance happen at load time.

Lazy Instantiation

In Lazy Instantiation the creation of instance happen when required.

267. Write program for singleton design pattern.

```
class Singleton {
    private static Singleton obj;

    private Singleton() {}

    public static Singleton getInstance(){
        if (obj==null)
      obj = new Singleton();
        return obj;
    }
}
```

268. Explain factory design pattern.

Factory design pattern uses factory method to create object without exposing creation logic to the client.

It is a creational design pattern.

269. Explain abstract factory design pattern.

Abstract factory design pattern is used to create factory which is responsible for creation of other factories.

It is a creational design pattern.

270. Explain Prototype design pattern.

Prototype design pattern allows you to create duplicate object while keeping performance in mind.

It is used when direct object creation is costly.

E.g. while performing database operation we can cache the object and return its clone on next request and update the database as and when required. This reduces database calls.

271. Explain builder design pattern.

Builder pattern allows you to build a complex object using simple objects.

Home is example if builder design pattern.

It is a structural design pattern.

272. Explain Adapter design pattern.

Adapter pattern let classes work together that couldn’t otherwise because of incompatible interfaces.

Adapter pattern involves a class which is responsible to join functionality of independent or incompatible interfaces.

The object that joins these unrelated interfaces is called an Adapter

E.g. card reader which acts as adapter between memory card and a laptop.

This is a structural design pattern.

273. Explain Bridge design pattern.

The Bridge design pattern lets you separate the abstraction from the implementation.

It is a structural design pattern.

274. Explain composite design pattern.

The composite design pattern lets you treat individual objects and composition of objects uniformly.

Composite pattern composes objects as a tree structure to represent part as well as whole hierarchy.

It is a structural design pattern.

275. Explain Decorator Design Pattern.

The decorator design pattern enables you to dynamically add functionality and behavior to an object without affecting the behavior of other existing objects in the same class.

We use inheritance to extend the behavior of the class.

This takes place at compile time, and all of the instances of that class get the extended behavior.

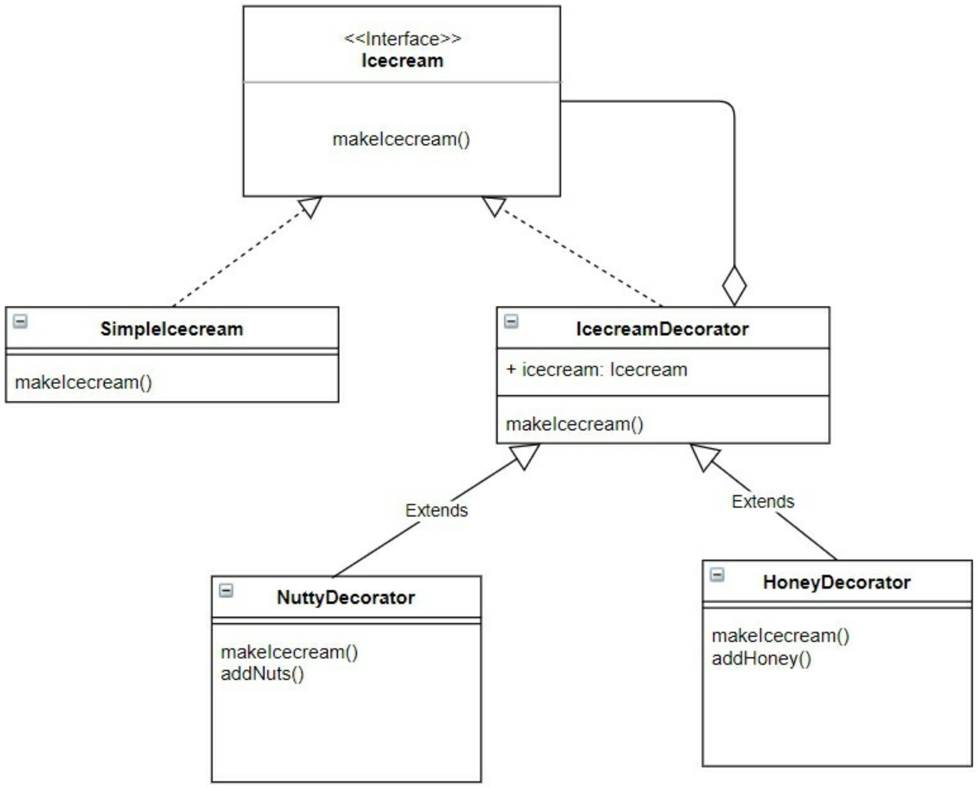
Decorator design patterns allow us to add functionality to an object (not the class) at runtime, and we can apply this customized functionality to an individual object based on our requirement and choice.

276. Write Program for Decorator Design pattern.

```
public interface Icecream {
  public String makeIcecream();
}
==========================================================
public class SimpleIcecream implements Icecream {

  @Override
  public String makeIcecream() {
    return "Base Icecream";
  }

}
=====================================================
abstract class IcecreamDecorator implements Icecream {
  protected Icecream specialIcecream;

  public IcecreamDecorator(Icecream specialIcecream) {
    this.specialIcecream = specialIcecream;
  }

  public String makeIcecream() {
    return specialIcecream.makeIcecream();
  }
}
==================================================

public class NuttyDecorator extends IcecreamDecorator {

  public NuttyDecorator(Icecream specialIcecream) {
    super(specialIcecream);
  }

  public String makeIcecream() {
    return specialIcecream.makeIcecream() + addNuts();
  }

  private String addNuts() {
    return " + cruncy nuts";
  }
}
==================================================
public class HoneyDecorator extends IcecreamDecorator {

  public HoneyDecorator(Icecream specialIcecream) {
    super(specialIcecream);
  }

  public String makeIcecream() {
    return specialIcecream.makeIcecream() + addHoney();
  }

  private String addHoney() {
    return " + sweet honey";
  }
```

```
}
```

277. Draw UML diagram for decorator design pattern.



278. Explain facade design pattern.

Provide a unified interface to a set of interfaces in a subsystem and therefore it hides the complexities of the subsystem from the client.

It is a structural design pattern.

279. Explain flyweight design pattern.

Flyweight design pattern allows you to create a lot of Objects of a class.

As object uses memory that can be crucial for low memory devices, such as mobile devices flyweight design pattern can reduce the memory consumption by sharing objects.

It is a structural design pattern.

280. Explain Proxy design pattern.

The proxy design pattern allows you to provide controlled access of a functionality.

It is a structural design pattern.

In hibernate, we get entities from the database. Hibernate creates an object which is a proxy to the underlying entity. The client reads the data using proxy.

These proxy entity classes are useful in lazy loading scenarios where associated entities are fetched only when they are requested explicitly. It improves performance of DAO operations.

281. Explain Chain of responsibility design pattern.

Chain of responsibility pattern allows you to have loose coupling in software design, where a request is passed to a chain of objects.

The object in the chain will identify whether request should be processed by itself or request should be sent to the next object in the chain.

It is behavioural design pattern.

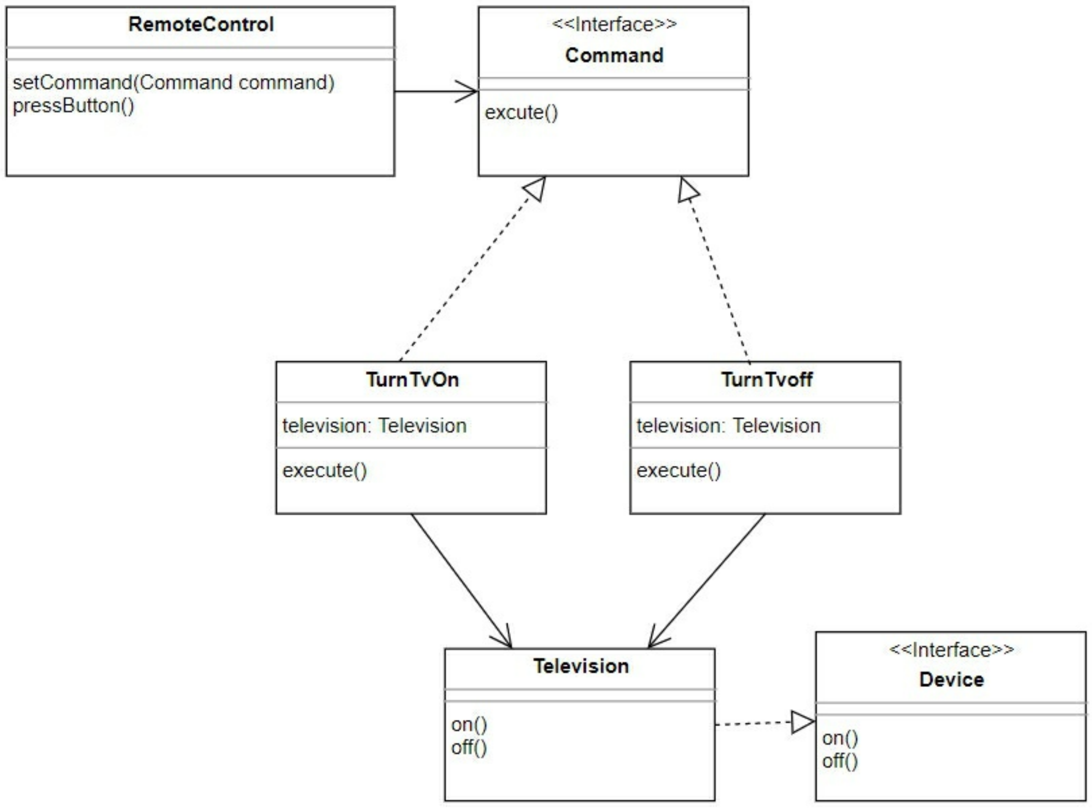
282. Explain Command design pattern.

In command design pattern command is basically an object which is passed to invoker object. (i.e. Press button command is passed to remote control)

Invoker object passes the command to the appropriate receiver object which executes the command. (i.e. Remote control passes the command to television, command contains request to turn on or turn off television)

It is behavioural design pattern.

283. Draw UML for Command design pattern.



284. Write Program for command design pattern.

```
// An interface for command
interface Command
{
    public void execute();
}

==================================================
class Television
{
    public void on()
    {
        System.out.println("Television is on");
    }
    public void off()
    {
        System.out.println("Television is off");
    }
}
class TurnTVOn implements Command
{
    Television television;

    public TurnTVOn(Television television)
    {
       this.television = television;
    }
    public void execute()
    {
       television.on();
    }
}
class TurnTVOff implements Command
{
    Television television;

    public TurnTVOff(Television television)
    {
       this.television = television;
    }
    public void execute()
    {
       television.off();
    }
}


class RemoteControl
{
    Command command;
```

```
        public RemoteControl()
        {
        }

        public void setCommand(Command command)
        {
            this.command = command;
        }

        public void buttonWasPressed()
        {
            this.command.execute();
        }
    }

    // Driver class
    class RemoteControlTest
    {
        public static void main(String[] args)
        {
            RemoteControl remote = new RemoteControl();
            Television television = new Television();


            remote.setCommand(new
                      TurnTVOn(television));
            remote.buttonWasPressed();

         }
      }
```

285. Explain Interpreter design pattern.

The interpreter design pattern defines a grammatical representation for a language and provides an interpreter to deal with this grammar.

The java compiler that interprets the java source code into byte code that is understandable by JVM is best example of interpreter design pattern.

It is behavioural design pattern.

286. Explain Iterator design pattern.

Iterator design pattern allows you to traverse through group of Objects.

Iterator interface provides methods to iterate over any Collection which is example of iterator design pattern.

It is behavioural design pattern.

287. Explain Mediator design pattern.

Mediator pattern provides a mediator between objects for communication and allows you to implement lose-coupling between objects.

Air traffic controller is example of mediator pattern where the airport control room works can be considered as a mediator object for communication between different flights.

It is behavioural design pattern.

288. Explain Observer design pattern.

In Observer design pattern observer object watch the state of other object (Subject).

Model-View-Controller (MVC) frameworks uses Observer pattern where Model is the Subject and Views are observers that to get notified of any change to the model.

It is behavioural design pattern.

289. Explain Strategy design pattern.

The strategy pattern allows you to select appropriate algorithm at runtime.

It is behavioural design pattern.

290. Explain Template method design pattern.

Template Method design pattern allows you to define a skeleton of an algorithm in a base class and let subclasses override the steps without changing the overall algorithm’s structure.

It is behavioural design pattern.

291. Explain Visitor design pattern.

Visitor design pattern allows you to add new behaviors to existing class hierarchy without altering any existing code.

It is behavioural design pattern.

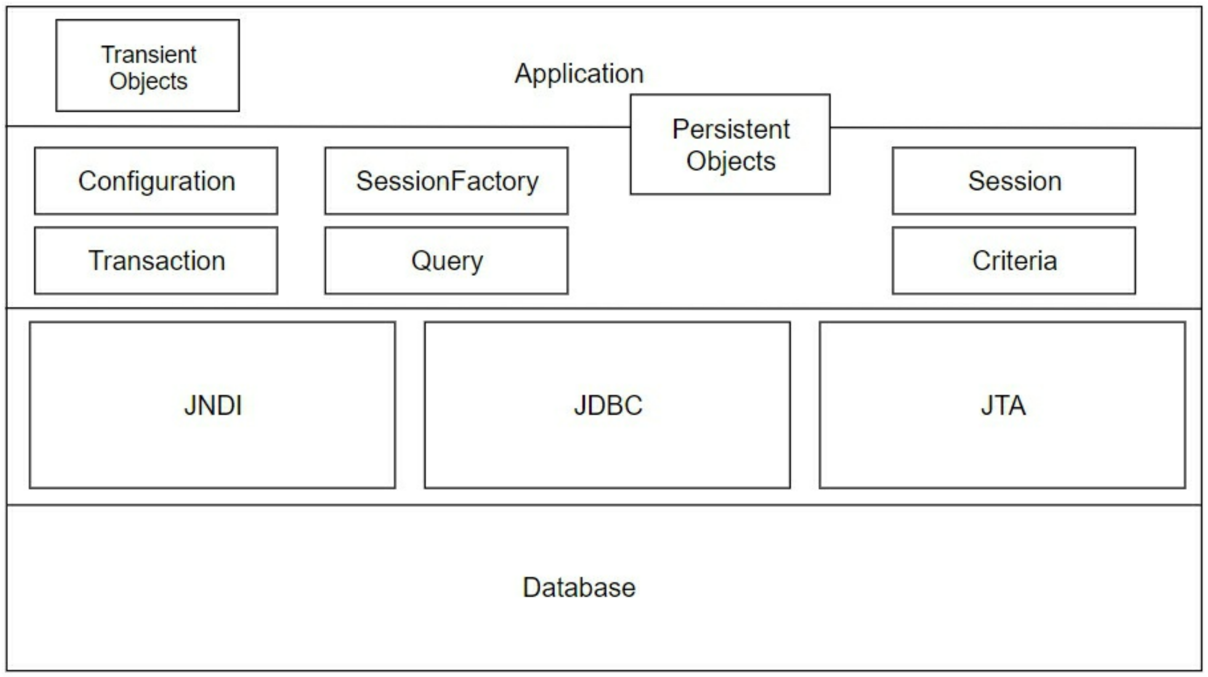
17. Hibernate

292. What is Hibernate?

Hibernate is lightweight ORM tool which is basically used to communicate with database.

It is useful for storing, manipulating and retrieving data from database.

293. Explain hibernate architecture.



Configuration

The Configuration object reads properties Hibernate uses to get connected to a database and configure itself for work.

SessionFactory

The SessionFactory is obtained from a Configuration object. It is factory of Session objects. One SessionFactory object is created per database. For connection with multiple database multiple SessionFactory objects are created with different configurations. The SessionFactory provide second level caching. It is a thread safe object.

Session

Session maintains connection between application and database. It provides methods like load(), get(), save(), persist(), update() using which you can store, manipulate data from database. It is factory of Query, Criteria and Transaction. Session Object is not thread safe. Multiple threads can access it.

Transaction

It represents unit of works.

Query and Criteria

These objects are used to retrieve (and recreate) persistent objects.

294. What is ORM?

ORM stands for Object/Relational mapping.

ORM lets programmers map object with the data stored in the database.

It makes it easier to data creation, data manipulation, and data access.

295. What is JPA?

Java Persistence API (JPA) gives specification for managing the relational data in applications.

JPA specifications are specified with annotations in javax.persistence package.

With help of JPA annotation it is possible to write implementation (Hibernate, EclipseLink) independent code.

296. What is SessionFactory?

SessionFactory is a factory of sessions, it provides session instance.

SessionFactory is associated with second level cache which is not enabled by default.

SessionFactory is also thread safe as it is immutable.

297. What is Session?

Session basically provides connection between hibernate application and database.

Session provides methods like save(), persist(), get(), load(), update(), merge(), delete() to fetch and manipulate data from database.

Session also provides instances of Query, Criteria and Transaction.

Session is not thread-safe.

298. What is Transaction?

Transaction is sequence of operation which work as an atomic unit.

Transaction has Atomicity, Consistency, isolation and durability properties.

Atomicity: All changes are done as if they are single operation. All operations are executed or none of them are. i.e. If one account is debited other account is credited.

Consistency: Upon completion of successful transaction the data in datastore has to be consistent or reliable.

Isolation: If transaction are happening on same data one transaction will not disturb other transaction.

Durability: After a transaction is done data is stored permanently.

299. What is difference between save() and persist()?

save()	persist()
It returns the identifier of the instance.	It returns nothing because its return type is void.
Can be used outside transaction.	Can not be used outside transaction.

save() for detached object will create new row in table.	persist() will throw PersistentObject exception for detached object.

300. What is difference between get() and load()?

get()	load()
Returns null if object is not present.	Throws ObjectNotFoundException if object is not present.
The get() method hits the database and returns real object.	The load() method doesn’t hit the database and returns proxy object.
The get() method should be used if you are not sure about the existence of object.	The load() method should be used if you are sure about existence of object.

301. What is difference between update and merge method?

update()	merge()
After closing a session if next session is already having persistent object with same primary key and we call update method then it will throw error.	After closing a session if next session is already having persistent object with same primary key and we call merge method then the changes from previous session will be merged into current.
Update is used to edit record.	Merge is used to combine record.

302. Select Query example in hibernate.

String queryString = “From Student S where S.id := student_id”;

Query query = session.createQuery(queryString);

query.setParameter(“student_id”, 10);

List results = query.list();

303. What are the states of object in hibernate?

Transient: The object is in transient state when it is just created but not associated with any session.

Persistent: The object is in persistent state when session is open and object has been saved or retrieved from database.

Detached: When session is closed the object is in detached state.

304. Update Query example in hibernate.

String queryString = “Update Student S set marks := marks where S.id := student_id”;

Query query = session.createQuery(queryString);

query.setParameter(“marks”, 90);

query.setParameter(“student_id”, 10);

List results = query.list();

305. What are inheritance mapping strategies in hibernate?

There are 3 inheritance mapping strategies:

Single Table: Single table is created per inheritance hierarchy.

@Inheritance(strategy=InheritanceType.SINGLE_TABLE)

Table per class: Number of tables are created equivalent to number of concrete entities.

@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)

Joined Table: All entries will be done in one table.

```
@Inheritance(strategy=InheritanceType.JOINED)
```

18. Spring

306. What is Spring?

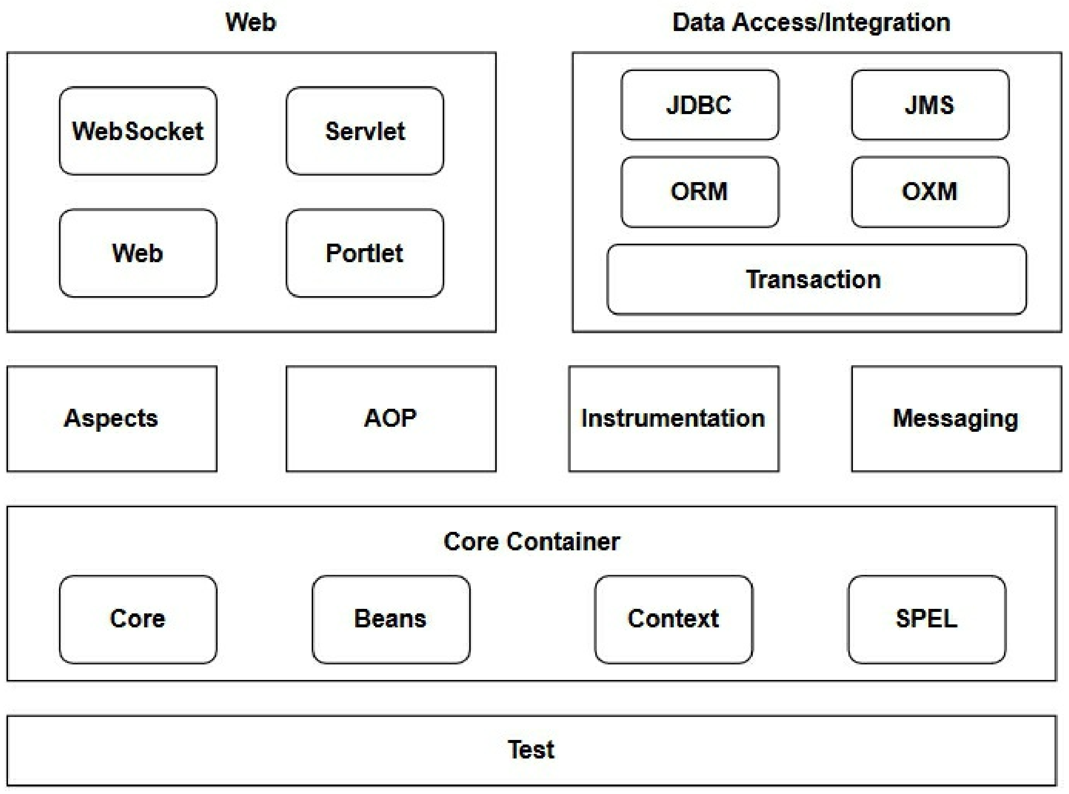
Spring is framework for developing enterprise application in Java.

It is lightweight, integrated and loosely coupled.

It is built on core concepts like dependency injection and aspect oriented programming.

304. Explain the modules of spring framework.

Below are the modules of Spring :



Core Container

The Core Container comprises of the Beans, Core, Context, and Expression Language modules.

The Core and Beans modules enable IoC and Dependency Injection features.

Context is responsible for initializing, configuring the instance and assembling the dependencies.

The Expression Language module provides a powerful expression language which is used for method invocation of bean, bean injection. Expression language also supports regular expression, mathematical, relational, logical, conditional operators.

Data Access/Integration

JDBC, ORM, OXM, JMS and Transaction modules are part of The Data Access/Integration layer.

The JDBC module removes the need to do redundant JDBC coding.

The ORM module integrates with popular object-relational mapping APIs such as JPA, Hibernate and more.

The OXM module provides supports for Object/XML mapping implementations like JAXB, XStream and more.

The Java Messaging Service (JMS) module is used for producing and consuming messages.

The Transaction module is used for programmatic and declarative transaction management.

Web

Web, Web-Servlet, WebSocket and Web-Portlet modules are part of Web layer.

Web module provides basic web integration features.

The Web-Servlet module consists of model-view-controller (MVC) implementation which is used in web applications.

AOP

AOP module provides aspect-oriented programming implementation which divides program into aspects which increases modularity of program.

Aspect is a class which is used across multiple classes for particular requirement. For example, Logging aspect can be a class which is used across multiple classes for logging.

Test

The Test module supports the testing using JUnit or TestNG.

It provides capability to load Spring ApplicationContexts and cache those contexts.

It also provides mock objects that can be used to test the code.

305. Explain advantages of spring framework?

Spring simplify the development process by providing an abstraction layer on existing technologies like servlets, jsps, jdbc, jndi, rmi, jms, Java mail and more.

Spring makes integration easy through technologies like ORM framework, logging framework, J2EE and JDK Timers and more

Spring web MVC framework makes it easy to develop web applications

Spring can get rid of creation of the singleton and factory classes.

Spring provides transaction management interface.

Spring can be used for the development of standalone applications, standalone GUI applications, Web applications and applets as well.

Spring supports configurations using both xml and annotation.

304. What is IOC?

Spring IoC is used to achieve loose-coupling between objects dependencies.

Loose coupling of the objects is achieved at runtime by injecting dependency from outside i.e. Spring IoC container.

This is called as IOC (Inversion Of Control).

305. What is dependency Injection?

Using Dependency injection objects get other required objects from outside i.e. Spring IoC container.

IOC is achieved through dependency Injection.

306. What is IoC container in Spring?

IoC container is responsible for initializing, configuring the instance and assembling the dependencies.

Types of IOC containers in spring framework are BeanFactory and ApplicationContext.

The BeanFactory is basic version of IOC containers whereas ApplicationContext is advanced version.

BeanFactory uses lazy loading while ApplicationContext uses eager loading for bean initialization.

307. What are the known implementations of the ApplicationContext?

The most commonly used implementation of 'Application Context' are:

FileSystemXmlApplicationContext: It loads the definitions of the beans from an XML file. You have to provide the full path of the XML bean configuration file to the constructor.

ClassPathXmlApplicationContext: It loads the definitions of the beans from an XML file. This container will search for bean configuration XML file in CLASSPATH.

WebXmlApplicationContext: It loads the XML file with definitions of all beans from within a web application.

308. What is @Bean annotation?

@Bean annotation is applied on a method to specify that it returns a bean managed by spring context.

309. What is @Configuration annotation?

@Configuration annotation specifies that annotated class contains one or more @Bean methods and may be processed by spring container to generate bean definition and service request for those beans at runtime.

310. What is @ComponentScan annotation?

@ComponentScan without arguments signals Spring to scan the components of the current package and all of its sub-packages.

311. What is autowiring in Spring?

Autowiring is used to inject the bean automatically, explicit injection logic is not required.

Below are modes of Autowiring:

no: it means autowiring is not enabled.

byName : injects the bean based on the name of property.

byType : injects the bean based on the type of property.

constructor : injects the bean using constructor.

312. What is @Autowired annotation?

@Autowired annotation is used for spring bean autowiring.

@Autowired annotation is used with variables and methods for autowiring byType.

It is also possible to use @Autowired annotation on constructor for constructor based spring autowiring.

313. What are the different bean scopes in spring?

Singleton

Single instance of bean is created per Spring IoC container.

Prototype

New bean instance will be created each time request is made.

Request

New bean instance will be created each HTTP request.

Session

New bean instance will be created each HTTP session.

Global session

New bean instance will be created each global HTTP session.

314. What is AOP?

AOP stands for Aspect Oriented Programming.

It divides program into aspects which increases modularity of program.

315. What is Aspect?

Aspect is a class which is used across multiple classes for particular requirement.

For example, Logging aspect can be a class which is used across multiple classes for logging.

316. What is JoinPoint?

Joinpoint is any point in program execution.

For example, execution of a method or the handling of an exception.

Join point represents a method execution in Spring.

317. What is Advice?

Advice is basically action taken by aspect.

It is generally a method declaration which is executed “Before”, “After”, “AfterReturning”, “Throws”, “Around” particular method.

318. Explain types of Advice.

@Before Advice: Runs before method execution.

@After Advice: Runs after method execution.

@AfterReturning Advice: Runs after method returns results.

@AfterThrowing Advice: Runs after method throws exception.

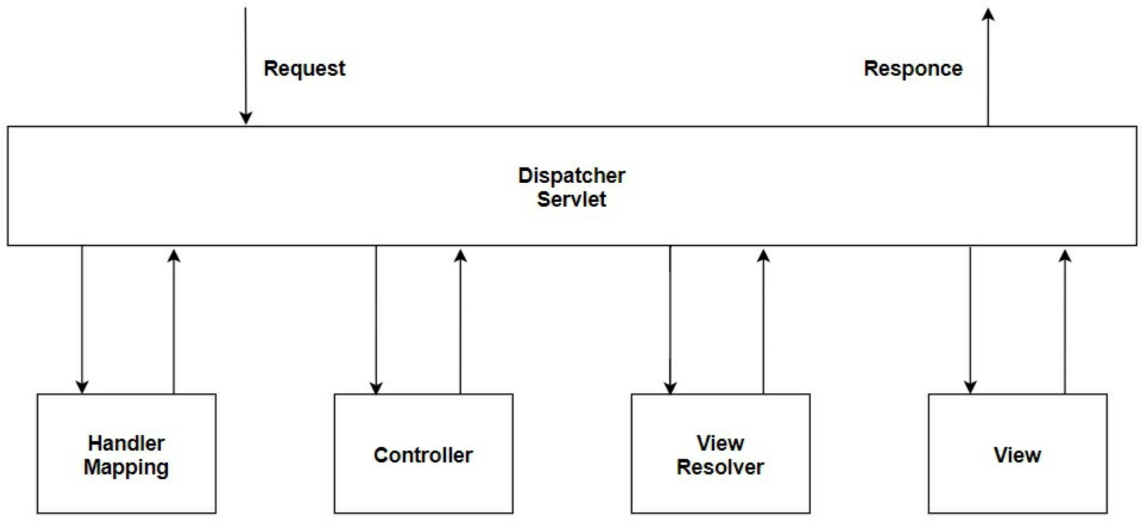
@Around Advice: Runs before & after method execution.

319. What is Pointcut?

Pointcut is basically expression which is matched with joinpoint to determine whether advice should be executed or not.

For example, there can be a pointcut expression which is matched with method name to determine whether advice should be executed or not.

320. Explain Spring MVC framework flow.



Dispatcher servlet receives HTTP Request.

Dispatcher servlet consults with handler mapping to identify controller.

Controller receives request and call services method which will model data based on business logic.

Controller will return Model and View name to Dispatcher servlet.

Dispatcher servlet will consult with View Resolver to identify appropriate view from view name.

Once view is obtained Dispatcher servlet binds model data to view which is then rendered on browser or app.

321. What is @ResponceBody annotation?

@ResponceBody annotation tells controller that object returned is automatically serialized into JSON and passed in HTTP Response object.

322. What is @RequestBody annotation?

@RequestBody basically map HTTP Request body to Java Object.

323. What is @RestController annotation?

@RestController annotation is basically combination of @Controller and @RequestBody annotations.

324. What is @RequestHeader annotation?

@RequestHeader annotation maps HTTP Request header to variables.

325. How exception is handled in Spring?

@ControllerAdvice can be used to handle exceptions across application in spring.

It can be thought of as interceptor of exceptions which are thrown by methods annotated with @RequestMapping.

19. Web Services

326. What is web service?

Web services generally stick to client server architecture where client can access web services over network.

Web services are stateless and don’t maintain user sessions.

327. What are advantages of web services?

Below are the advantages of web services:

Reusability: Same web service can be used by many client applications.

Interoperability: Web services can be written in different languages i.e. Java, C# etc. and they can still communicate with each other.

Loose Coupling: Web service client is loosely coupled with web service.

328. What are different types of web services?

Two types of web services are: SOAP and RESTful web services.

329. What is difference between SOAP and REST web services?

SOAP	REST
SOAP is stands for Simple Object Access Protocol.	REST stands for Representational State Transfer.
SOAP is a protocol.	REST is architectural style.
SOAP consumes more bandwidth as SOAP has additional header for every message.	REST consumes less bandwidth as it utilizes HTTP header.
SOAP only supports XML data format.	REST supports XML, JSON, image and many more formats.
SOAP web service and client are	REST web service and client are

tightly coupled with contract.	loosely coupled.

330. What is difference between the Accept and Content-Type HTTP headers?

Accept headers indicate data format of response client is accepting.

Content-Type header is used to indicate format of data being sent in the request from client.

331. What is @Controller annotation?

@Controller annotation is used to mark class as a web controller.

Controller can handle client requests and send a response back to the client.

Classes with @Controller are auto detected through classpath scanning.

332. What is @RequestMapping annotation?

The @RequestMapping annotation can be used with class or method in a controller.

The @RequestParam annotation may or may not have a value.

RequestMapping annotation maps web requests to specific handler classes or handler methods.

333. What is @RequestParam annotation?

The @RequestParam is used to map a web request parameter to a method parameter.

```
@RequestMapping("/person")
public String findPerson(@RequestParam("firstname") String firstName)
{
                .
                .
}
```

334. What is @PathVariable?

@PathVariable is used to indicate a method parameter which should be bound to a URI template variable.

```
@RequestMapping(value = "/user/{name}")
public void process(@PathVariable String name)
{

        .
      .
  }
```

335. What are different HTTP methods?

Below are different HTTP methods:

POST: It is generally used to create new entity.

GET: It is generally used to read entity.

PUT: It is generally used to update entity.

PATCH: It is generally used to modify entity.

DELETE: It is generally used to delete entity.

HEAD: It is similar to GET except that server must not return message body.

336. What is difference between GET and POST?

In GET data is sent through URL, in POST data is sent through message body.

POST is more secure way of transferring the data as data is in message body and not in URL.

337. What is difference between PUT and POST?

POST is used for creating new entity whereas PUT is used for replacing existing entity.

338. What is difference between PUT and PATCH?

PUT is used update entity entirely whereas PATCH is used for updating entity partially.

339. What is JAX-RS?

JAX-RS is API used for creating REST web services.

JAX-RS comes with JDK so there is no need to include external library.

340. What are important annotations used in JAX-RS?

@Path specifies relative path of class and methods.

@GET: The Java method with @GET annotation will process HTTP GET requests.

@POST: The Java method with @POST annotation will process HTTP POST requests.

@PUT: The Java method with @PUT annotation will process HTTP PUT requests.

@DELETE: The Java method with @DELETE annotation will process HTTP DELETE requests.

@Consumes: It is used to specify the MIME media types that a resource can consume.

@Produces: It is used to specify the MIME media types that resource can produce.

@PathParam: It is used to map method parameters to path parameters that are extracted from URI.

@QueryParam: It is used to map method parameters to query parameters that are extracted from request URIs query parameters.

341. What are important HTTP status codes?

Below are few important HTTP status codes:

1xx Informational

2xx Success

200 OK

201 Created

202 Accepted

203 Non Authoritative Information

204 No Content

3xx Redirection

4xx Client Error

400 Bad Request

401 Unauthorized

403 Forbidden

404 Not Found

5xx Server Error

500 Internal Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

20. Spring Boot

342. What is Spring Boot?

Spring boot makes it easy to create spring based applications that you can just run as java –jar.

Spring boot is useful for creating microservices applications.

Spring boot provides embedded tomcat due to which you don’t need to install tomcat separately for application deployment.

It provides starter POMs which provides bunch of dependencies.

It supports auto configuration, Spring Boot auto-configuration automatically configures Spring application based on the dependencies that are available to project.

Spring actuator provides several production grade services to your application.

Spring boot Initializer makes it easy to create spring boot application.

Spring Boot CLI is used to run and test spring boot applications from command prompt.

343. What is an embedded server in Spring Boot?

Spring boot provides embedded tomcat in jar.

Since deployable jar has tomcat binaries embedded within it by default, hence you don’t need to install tomcat separately.

Due to embedded tomcat you can just run your spring boot application as java application.

344. How to switch from default embedded server in Spring Boot?

Spring boot application has spring-boot-starter-web dependency which provides tomcat server embedded within application by default.

You can exclude default tomcat as below:

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

After exclusion you can add new dependency of application server that you want to include as below:

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

345. How to change port number of Spring Boot application?

In Spring Boot application, tomcat starts by default on port 8080.

You can customize this port in application.properties or application.yml as below:

```
server.port=8081
```

346. What is starter POM?

Starter POMs are a set of dependency descriptors that are included in your application.

Spring Boot starters are useful to reduce the number of manually added dependencies just by adding one dependency (Starter POM).

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

The spring-boot-starter-web contains following spring web dependencies, hence you don't need to include these dependencies separately:

- spring-boot-starter
- jackson
- spring-core
- spring-mvc
- spring-boot-starter-tomcat

347. What is auto configuration?

Spring Boot supports auto configuration.

Spring Boot auto-configuration automatically configures spring application based on the dependencies that are available to project.

An Auto Configuration class resembles any regular spring @Configuration class, but it also has @Conditional annotation.

```
@Configuration
@ConditionalOnClass(KafkaTemplate.class)
public class KafkaAutoConfiguration {
            .
            .
            .
}
```

This configuration class would only be enabled if the KafkaTemplate class is present in the application classpath.

If specific Apache Kafka dependency is available to the project spring will auto configure itself with the appropriate beans.

To register the class as an auto-configuration candidate, add the name of the class under the key org.springframework.boot.autoconfigure.EnableAutoConfiguration in the standard file resources/META-INF/spring.factories as:

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.config.KafkaAutoConfiguration
```

348. What is Spring Boot actuator?

Spring actuator provides several production grade services to your application.

Actuator is used to expose information about the running application like health, env, info, dump, metrics etc.

These services are available as HTTP endpoints.

The spring-boot-actuator dependency is required to be added as below to enable actuator in your application:

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

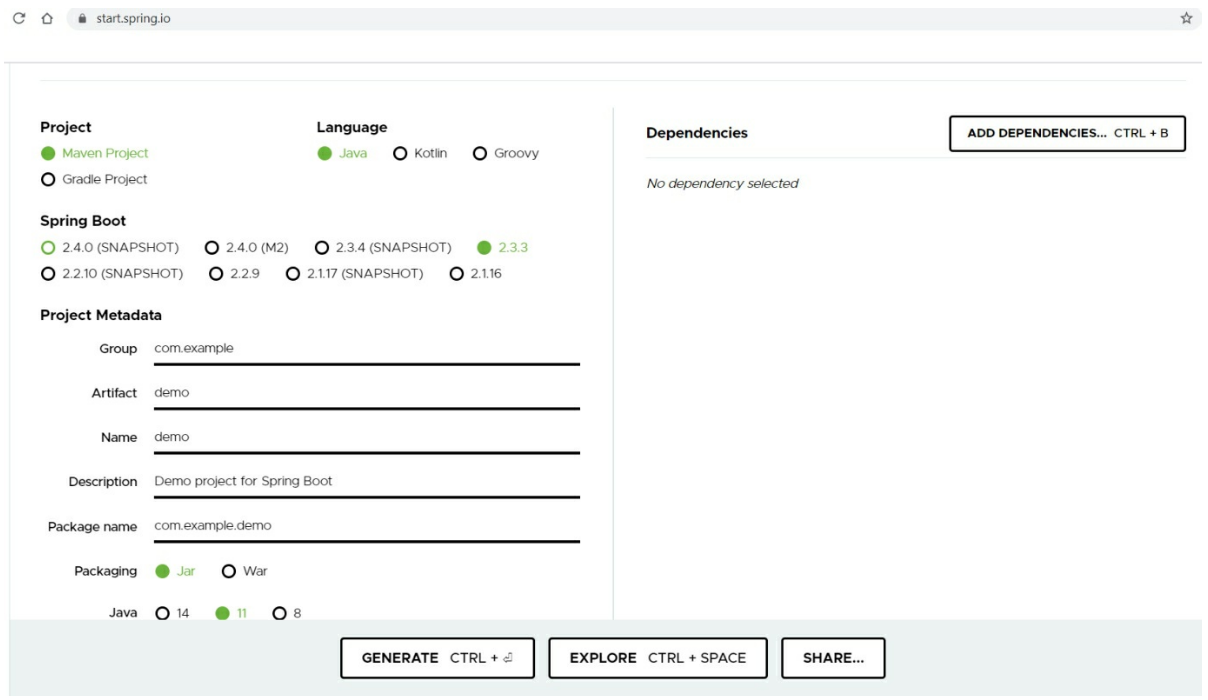
349. What is Spring Boot initializer?

Spring boot Initializer makes it easy to create spring boot application.

The Spring Initializer is basically a web application using which you can generate a Spring Boot project structure.

As project structure is ready with Spring Boot initializer all you need to do is write the application code.

Below is the screen shot of spring initializer web application:


start.spring.io

Project
[✓ Maven Project] Gradle Project

Language
[✓ Java] Kotlin Groovy

Spring Boot
2.4.0 (SNAPSHOT) 2.4.0 (M2) 2.3.4 (SNAPSHOT) [✓ 2.3.3]
2.2.10 (SNAPSHOT) 2.2.9 2.1.17 (SNAPSHOT) 2.1.16

Project Metadata

Group: com.example
Artifact: demo
Name: demo
Description: Demo project for Spring Boot
Package name: com.example.demo
Packaging: [✓ Jar] War
Java: 14 [✓ 11] 8

Dependencies
No dependency selected

To generate new project you need to specify project build(maven/gradle), Language(Java/Kotlin/Groovy), Spring Boot version, Project Metadata(Group, Artifact, Name, Description, Package name, Packaging, Java version) and dependencies.

350. What is Spring Boot CLI?

Spring Boot CLI is used to run and test spring boot applications from command prompt.

351. How do you connect to database in your Spring Boot project?

Add spring-boot-starter-data-jpa dependency to POM.

Add Oracle Driver dependency as well.

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

```
                <groupId>com.oracle</groupId>
                <artifactId>ojdbc7</artifactId>
                <version>12.1.0.1</version>
</dependency>
```

Add connection properties to application.properties as:

```
oracle.url=jdbc:oracle:thin:@//abcd.com:1521/ORCL
oracle.username=system
oracle.password=manager
```

Create configuration class with below method:

```
@Configuration
public class DBConfiguration {

    private String username;

    private String password;

    private String url;
    public void setUsername(String username) {
        this.username = username;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    @Bean
    DataSource dataSource() throws SQLException {
        OracleDataSource dataSource = new OracleDataSource();
        dataSource.setUser(username);
        dataSource.setPassword(password);
        dataSource.setURL(url);
        return dataSource;
    }
}
```

Create Repository.

Create entity class that is mapped to database table.

352. Is it mandatory to use @SpringBootApplication annotation to create spring boot application?

No.

353. @SpringBootApplication annotation is combination of which annotations?

@SpringBootApplication is combination of @EnableAutoConfiguration+@ComponentScan+@Configuration.

21. Programs

354. Write a program to print fibonacci series program.

```
public static void main(String[] args) {
		int fib1 = 0, fib2 = 1, fib;
		System.out.print(fib1 + " " + fib2);
		int count = 8;
		for(int i  = 0; i < count; i++) {
		fib = fib1 + fib2;
		System.out.print(" " + fib);
		fib1 = fib2;
		fib2 = fib;
		}
}
```

355. Write a program to check if String is palindrome.

```
public static void main(String[] args) {
		System.out.println(isPalindrome("bob"));
}
public static boolean isPalindrome(String originalString) {
		String reverse = "";
		for(int i = 0; i < originalString.length(); i++) {
		reverse = reverse + originalString.charAt(i);
		}
		if(originalString.equals(reverse)) {
		return true;
		}
		return false;
}
```

356. Write a program to check if integer is palindrome.

```
public static void main(String[] args) {
		System.out.println(isPalindrome(1221));
}
public static boolean isPalindrome(int original) {
		int temp = original;
		int reverse = 0;
		while (temp > 0) {
		int reminder = temp % 10;
		reverse = reverse*10 + reminder;
		temp = temp/10;
		}
		if(original == reverse) {
		return true;
		}
		return false;
}
```

357. Write a program to find factorial of number.

```
public static void main(String[] args) {
		System.out.println(factorial(4));
}
public static int factorial(int original) {
		int factorial = 1;
		while(original > 0) {
		factorial = factorial*original;
		original--;
		}
		return factorial;
}
```

You may also like to read (available on amazon):

Angular 8 Interview Questions and Answers: Includes Angular 8, 7, 6, 5, 4 and 2

Angular 7 Interview Questions and Answers: Includes Angular 7, 6, 5, 4 and 2

Javascript Interview Questions and Answers