

Introduction

1. What is Java?

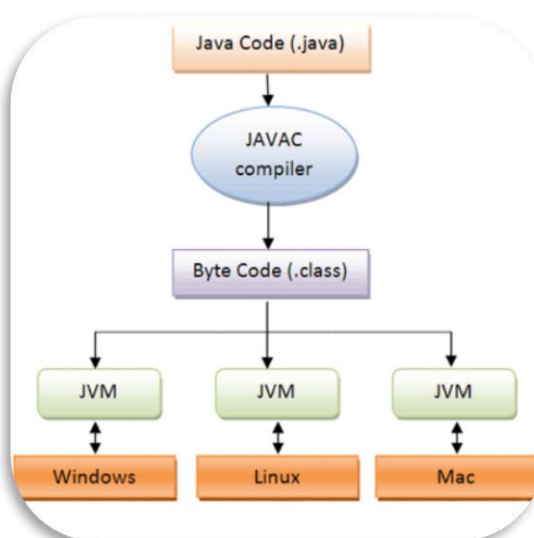
- Java technology is both general-purpose, class-based, object-oriented programming language and a platform (a collection of programs that help programmers to develop and run Java programming applications efficiently) from Oracle Corporation.
- Java was originally developed by **James Gosling** with his colleagues at Sun Microsystems during the early 1990s.
- Initially, it was called a project 'Oak' which had implementation similar to C and C++.
- The name Java has later selected after enough brainstorming and is based on the name of an espresso bean.
- **Java 1.0, the first version was released in 1995 with the tagline of 'write once, run anywhere'.** Later, Sun Microsystems was acquired by Oracle.
- The **latest version** of Java is **Java SE15** released on **15th September 2020**.

2. What are the important features of java?

1. Java is platform independent language. (One java program can be run in different operating system)
 - Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java is guaranteed to be compile-once, run-anywhere language.
 - On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine. Any machine with Java Runtime Environment can run Java Programs.
2. It is open-source language. (Java source code, documentation and jdk software are free available)
3. It is pure object-oriented programming language.
4. Java library (.class file) internal and external are very huge.

3. How java is platform independent?

Java program, once compiled, can be run on any platform without recompiling.



4.	Which are the types in java?								
	<ol style="list-style-type: none"> 1. Class 2. Interface 3. Enum 4. Annotation 								
5.	What is class?								
	<ul style="list-style-type: none"> • Class is nothing but different type of entity. • It is useful designing template or blueprint from which individual objects are created. <p style="text-align: center;"><u>Or</u></p> <ul style="list-style-type: none"> • A class in Java is a blueprint which includes all your data. • A class contains fields (variables) and methods to describe the behaviour of an object. <p>e.g., public class A</p> <pre>{ } </pre>								
6.	What is Object?								
	<ul style="list-style-type: none"> • Object is an instance of a class. • Object it means separate copy of memory. • An object is a real-world entity. Every object has 2 things. <ol style="list-style-type: none"> 1. State 2. Behaviour • State can be change object to object but behaviour will always be same. <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: left;"> <p>Object.java</p> <pre> 1 package com.intro.object; 2 3 public class Object { 4 5 int x; 6 7 public void sum() 8 { 9 10 } 11 12 }</pre> </div> <div style="text-align: left;"> <p>Test.java</p> <pre> 1 package com.intro.object; 2 3 public class Test { 4 5 public static void main(String[] args) 6 { 7 Object obj1=new Object(); 8 obj1.x=100; 9 obj1.sum(); 10 11 Object obj2=new Object(); 12 obj2.x=300; 13 obj2.sum(); 14 15 }</pre> </div> </div>								
7.	What is the difference between class and object?								
	<table border="1"> <thead> <tr> <th>Class</th><th>Object</th></tr> </thead> <tbody> <tr> <td>Class is a blueprint or template from which objects are created.</td><td>Object is an instance of a class.</td></tr> <tr> <td>Class is a group of similar objects.</td><td>Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.</td></tr> <tr> <td>Class is a logical entity.</td><td>Object is a physical entity.</td></tr> </tbody> </table>	Class	Object	Class is a blueprint or template from which objects are created.	Object is an instance of a class.	Class is a group of similar objects.	Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a logical entity.	Object is a physical entity.
Class	Object								
Class is a blueprint or template from which objects are created.	Object is an instance of a class.								
Class is a group of similar objects.	Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.								
Class is a logical entity.	Object is a physical entity.								

	<p>Class is declared using class keyword e.g.</p> <pre>class Student{}</pre> <p>Class is declared once.</p> <p>Class doesn't allocated memory when it is created.</p>	<p>Object is created through new keyword mainly e.g.</p> <pre>Student s1=new Student();</pre> <p>Object is created many times as per requirement.</p> <p>Object allocates memory when it is created.</p>
8.	<p>What will be the initial value of an object reference which is defined as an instance variable?</p> <ul style="list-style-type: none"> All object references are initialized to null in Java. 	
9.	<p>What do you understand by JVM? What it does?</p> <ul style="list-style-type: none"> Java Virtual Machine is a virtual machine (it doesn't physically exist) that enables the computer to run the Java program. JVM acts like a run-time engine which calls the main method present in the Java code. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode. The JVM performs following operation: <ol style="list-style-type: none"> Loads code Verifies code Executes code Provides runtime environment 	
10.	<p>What is the Difference between JDK, JRE and JVM?</p> <p>1. JDK –</p> <ul style="list-style-type: none"> Java Development Kit (in short JDK) is Kit which provides the environment to develop and execute(run) the Java program. It physically exists. JDK is a kit (or package) which includes two things <ol style="list-style-type: none"> Development Tools (to provide an environment to develop your java programs) JRE (to execute your java program). 	
	<pre> graph TD subgraph JDK subgraph JRE JVM((JVM)) Libraries[Set of libraries e.g. rt.jar etc.] Files[Other files] end Tools[Development tools e.g. javac, java etc.] end </pre>	

- JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:
 - Standard Edition Java Platform
 - Enterprise Edition Java Platform
 - Micro Edition Java Platform
- The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It is possible to install more than one JDK version on the same computer.

- **Why use JDK?**

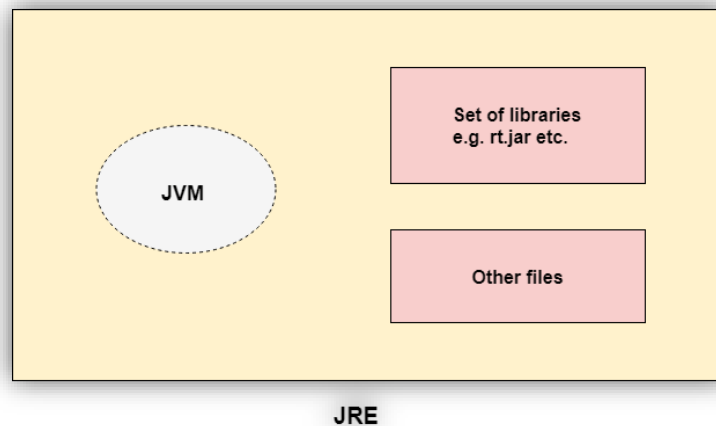
Here are the main reasons for using JDK:

- JDK contains tools required to write Java programs and JRE to execute them.
- It includes a compiler, Java application launcher, Appletviewer, etc.
- Compiler converts code written in Java into byte code.
- Java application launcher opens a JRE, loads the necessary class, and executes its main method.

Note: JDK is only used by Java Developers.

2. JRE –

- Java Runtime Environment (to say JRE) is an installation package which provides environment to only run (not develop) the java program (or application) onto your machine.
- It is also written as Java RTE. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.
- JRE is only used by them who only wants to run the Java Programs i.e. end users of your system.



- **Why use JRE?**

Here are the main reasons of using JRE:

- JRE contains class libraries, JVM, and other supporting files. It does not include any tool for Java development like a debugger, compiler, etc.
- It uses important package classes like math, swing, util, lang, awt, and runtime libraries.
- If you have to run Java applets, then JRE must be installed in your system.

3. JVM –

- Java Virtual machine (JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both.
- Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line hence it is also known as interpreter.
- **Why JVM?**

Here are the important reasons of using JVM:

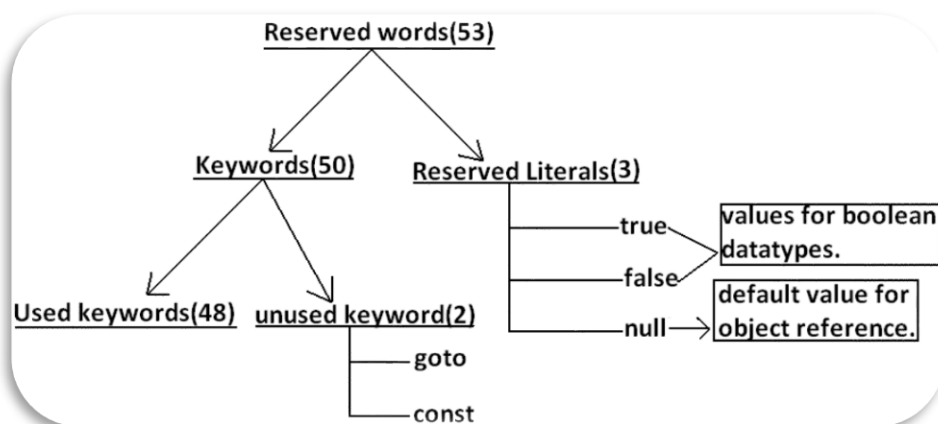
- JVM provides a platform-independent way of executing Java source code.
- It has numerous libraries, tools, and frameworks.
- Once you run a Java program, you can run on any platform and save lots of time.
- JVM comes with JIT (Just-in-Time) compiler that converts Java source code into low-level machine language. Hence, it runs faster than a regular application.

Summary:

JDK	JRE	JVM
It stands for Java Development Kit.	It stands for Java Runtime Environment.	It stands for Java Virtual Machine.
It is the tool necessary to compile, document and package Java programs.	JRE refers to a runtime environment in which Java bytecode can be executed.	It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed.
It contains JRE + development tools.	It's an implementation of the JVM which physically exists.	JVM follows three notations: Specification, Implementation and Runtime Instance

11. How many reserved words in java?

- In java some identifiers are reserved to associate some functionality or meaning such type of reserved identifiers are called reserved words.



Reserved words for data types: (8)	Reserved words for flow control: (11)	Keywords for modifiers: (11)
1) byte 2) short 3) int 4) long 5) float 6) double 7) char 8) Boolean	1) if 2) else 3) switch 4) case 5) default 6) for 7) do 8) while 9) break 10) continue 11) return	1) public 2) private 3) protected 4) static 5) final 6) abstract 7) synchronized 8) native 9) strictfp(1.2 version) 10) transient 11) volatile

Keywords for exception handling: (6)	Class related keywords: (6)	Object related keywords: (4)
1) try 2) catch 3) finally 4) throw 5) throws 6) assert(1.4 version)	1) class 2) package 3) import 4) extends 5) implements 6) interface	1) new 2) instanceof 3) super 4) this

Void return type keyword: (1)	Unused keywords: (2)	Reserved literals: (3)
1) void	1) goto 2) const	1) true 2) false 3) null

Enum keyword: (1)

- 1) **enum** --introduced in 1.5v to define a group of named constants

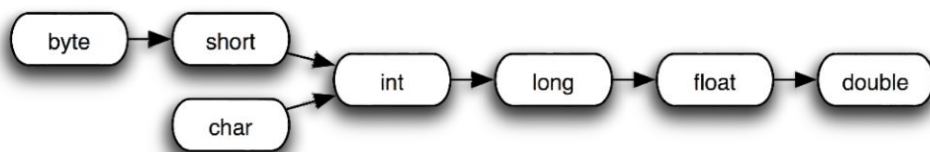
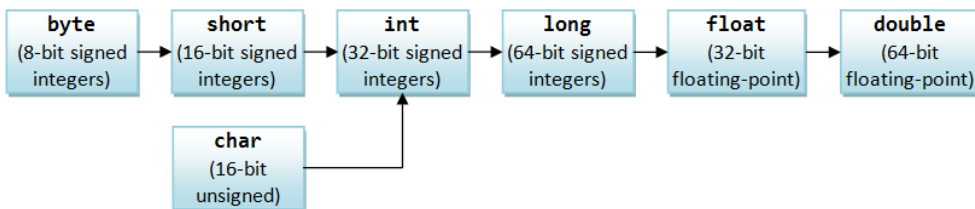
Note: By mistake if we are using **unused keywords** in our program, we will get compile time error.

Conclusion:

- All reserved words in java contain only lowercase alphabet symbols.
- New keywords in java are:
 1. **strictfp**-----1.2v
 2. **assert**-----1.4v
 3. **enum**-----1.5v
- In java we have only new keyword but not delete because destruction of useless objects is the responsibility of Garbage Collection.

instanceof but not instanceOf	extends but not extend
strictfp but not strictFp	implements but not implement
const but not Constant	import but not imports
synchronized but not synchronize	int but not Int

12.	Why Main Method is public?
	<ul style="list-style-type: none"> JVM call main method and it can be access from anywhere that's why it is public
13.	Why Main Method is static?
	<ul style="list-style-type: none"> Main method is static because JVM can call through class name. (no need to create object) <p style="text-align: center;">Or</p> <ul style="list-style-type: none"> JVM need not create the object of class in which main function is defined. <p style="text-align: center;">Or</p> <ul style="list-style-type: none"> If our main() method is not declared as static then the JVM has to create an object first and call which causes the problem of having extra memory allocation. <p>e.g., public class Test</p> <pre> { Test.main() //JVM calls like this } </pre>
14.	What is return type of main method?
	<ul style="list-style-type: none"> Main method return type is void.
15.	Why main method return type is void?
	<ul style="list-style-type: none"> Because JVM doesn't wants any return value from main.
16.	Why this method name is main?
	<ul style="list-style-type: none"> Everyone familiar with this name because it is used in previous programming languages.
17.	What is parameter of main method?
	<ul style="list-style-type: none"> A string type array is parameter of main method.
18.	What is use of main method parameter?
	<ul style="list-style-type: none"> It is used to getting input from command line.
19.	Why main method parameter is string array only? Why not other?
	<ul style="list-style-type: none"> String can hold any java type value in the format of stream.
20.	Can we execute a program without main() method?
	<ul style="list-style-type: none"> No, It was possible before JDK 1.7 using the static block. Since JDK 1.7, it is not possible.
	A person says that he compiled a java class successfully without even having a main method in it? Is it possible?
	<ul style="list-style-type: none"> Main method is an entry point of Java class and is required for execution of the program however; a class gets compiled successfully even if it doesn't have a main method. It can't be run though.

21.	<p>What if the static modifier is removed from the signature of the main method?</p> <ul style="list-style-type: none">• Program compiles fine.• However, at runtime, It throws an error "NoSuchMethodError."																								
22.	<p>Can we declare the main method of our class as private?</p> <ul style="list-style-type: none">• In java, main method must be public static in order to run any application correctly.• If main method is declared as private, developer won't get any compilation error however, it will not get executed and will give a runtime error.																								
23.	<p>Explain data members in java?</p> <ul style="list-style-type: none">• The variables which are declared in any class by using any fundamental data types (like int, char, float etc) or derived data type (like class, structure, pointer etc.) are known as Data Members.• There are 2 types of data member in java.<ol style="list-style-type: none">1. Primitive2. Non-Primitive (User defined) <p>Primitive data types</p> <ul style="list-style-type: none">• There are 8 primitive data members in java. <div><pre>graph LR byte --> short char --> short short --> int int --> long long --> float float --> double</pre><pre>graph LR byte["byte (8-bit signed integers)"] --> short["short (16-bit signed integers)"] short --> int["int (32-bit signed integers)"] int --> long["long (64-bit signed integers)"] long --> float["float (32-bit floating-point)"] float --> double["double (64-bit floating-point)"] char["char (16-bit unsigned)"] --> int</pre></div> <ul style="list-style-type: none">• Every primitive data member size is fixed.• Every primitive data member has their own default value. <table><tr><th>Data type</th><th>Default value</th><th>Data type</th><th>Default value</th></tr><tr><td>Byte</td><td>0</td><td>double</td><td>0.0d</td></tr><tr><td>Short</td><td>0</td><td>char</td><td>'\u0000'</td></tr><tr><td>Int</td><td>0</td><td>boolean</td><td>false</td></tr><tr><td>Long</td><td>0L</td><td>String</td><td>null</td></tr><tr><td>Float</td><td>0.0f</td><td>Any user defined object</td><td>null</td></tr></table> <p>Non-primitive Data types</p> <ul style="list-style-type: none">• Any java type either class type or interface type is called Non-primitive data member.• Every non primitive data member default value is null.• Size will be changed dynamically it means it will change during runtime.• E.g., all user defined classes & interfaces, List, Set, String, Arrays..etc.	Data type	Default value	Data type	Default value	Byte	0	double	0.0d	Short	0	char	'\u0000'	Int	0	boolean	false	Long	0L	String	null	Float	0.0f	Any user defined object	null
Data type	Default value	Data type	Default value																						
Byte	0	double	0.0d																						
Short	0	char	'\u0000'																						
Int	0	boolean	false																						
Long	0L	String	null																						
Float	0.0f	Any user defined object	null																						

24.	What is variable? Types of variables in java?
	<ul style="list-style-type: none"> • Variables are containers for storing data values. • There are 3 types of variables in java. <ol style="list-style-type: none"> 1. Instance variables 2. Static variables 3. Local variables <p>1. Instance Variables</p> <ul style="list-style-type: none"> • Instance variables are those variables which are declared inside the class but outside the method, constructor and block. • If the value of a variable is varied from object to object such type of variables are called instance variables. • For every object a separate copy of instance variables will be created. • Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly same as scope of objects. • Instance variables will be stored on the heap as the part of object. • Instance variables should be declared with in the class directly but outside of any method or block or constructor. • Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area. • But by using object reference we can access instance variables from static area. <p>Example:</p> <pre> class Test { int i=10; public static void main(String[] args) { //System.out.println(i); //C.E:non-static variable i cannot be referenced from a static context(invalid) Test t=new Test(); System.out.println(t.i); //10 (valid) t.methodOne(); } public void methodOne() { System.out.println(i); //10 (valid) } } </pre> <ul style="list-style-type: none"> • For the instance variables it is not required to perform initialization JVM will always provide default values. <p>Example:</p> <pre> class Test { boolean b; public static void main(String[] args) { Test t=new Test(); System.out.println(t.b); //false } } </pre>

```
}  
}
```

- Instance variables also known as object level variables or attributes.

2.Static/Class Variables

- **Static/Class variables** are those variables which are declared inside the class with static keyword.
- If the value of a variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such type of variables at class level by using static modifier.
- In the case of instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created and shared by every object of that class.
- Static variables will be created at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the .class file.
- Static variables will be stored in method area. Static variables should be declared within the class directly but outside of any method or block or constructor.
- Static variables can be accessed from both instance and static areas directly.
- We can access static variables either by class name or by object reference but usage of class name is recommended. But within the same class it is not required to use class name we can access directly.

Example:

```
class Test  
{  
    static int i=10;  
    public static void main(String[] args)  
    {  
        Test t=new Test();  
        System.out.println(t.i); //10  
        System.out.println(Test.i); //10  
        System.out.println(i); //10  
    }  
}
```

- For the static variables it is not required to perform initialization explicitly, JVM will always provide default values.

Example:

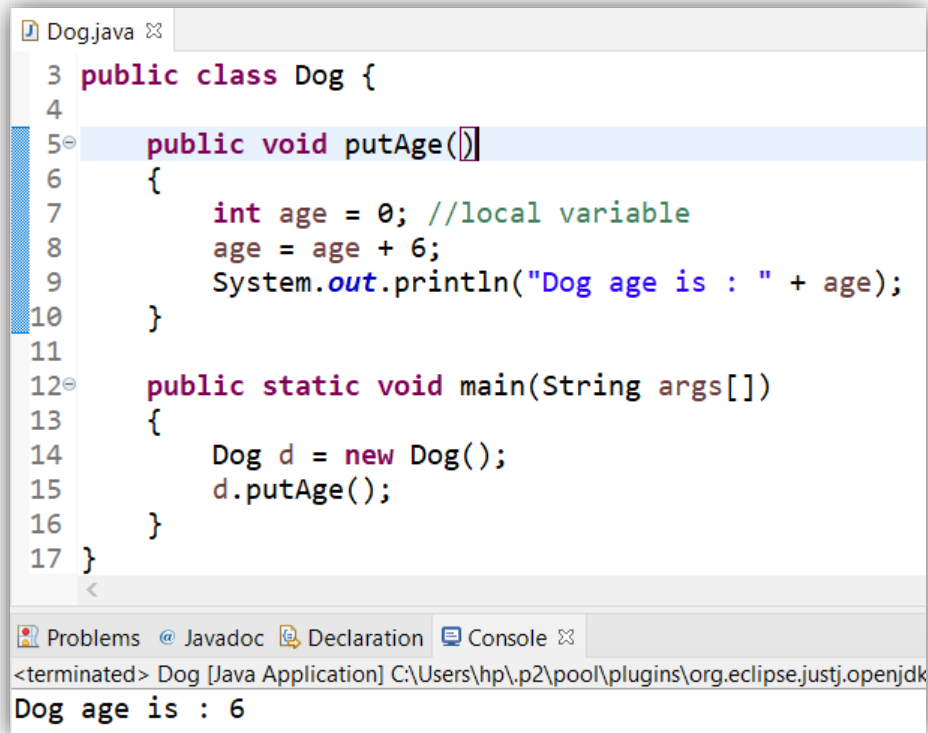
```
class Test  
{  
    static String s;  
    public static void main(String[] args)  
    {  
        System.out.println(s); //null  
    }  
}
```

- Static variables also known as class level variables or fields.

3.Local Variables

- **Local variables** are those variables which are declared inside method, constructor and block to meet temporary requirements of the programmer.

- local variables are also called as automatic variables or temporary variables or stack variables.
- The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- The local variables will be stored on the stack.
- For the local variables JVM won't provide any default values compulsory we should perform initialization explicitly before using that variable.
- It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.



```

Dog.java
3 public class Dog {
4
5     public void putAge()
6     {
7         int age = 0; //local variable
8         age = age + 6;
9         System.out.println("Dog age is : " + age);
10    }
11
12    public static void main(String args[])
13    {
14        Dog d = new Dog();
15        d.putAge();
16    }
17 }

```

Problems Javadoc Declaration Console

<terminated> Dog [Java Application] C:\Users\hp\p2\pool\plugins\org.eclipse.justj.openjdk

Dog age is : 6

- In above program age is a local variable. This variable is defined under putAge() method and its scope is limited to this method only.
- Access modifiers cannot be used for declaring local variables.
- Local Variables can also be defined inside statement blocks i.e. do, for, while loop. These are only visible inside that block.
- For example:


```
for(int i=0;i<=5;i++){.....}
```
- In above example int i=0 is a local variable declaration. Its scope is only limited to the for loop.

Conclusion:

1. For the static and instance variables it is not required to perform initialization explicitly JVM will provide default values. But for the local variables JVM won't provide any default values compulsory we should perform initialization explicitly before using that variable.
2. For every object a separate copy of instance variable will be created whereas for entire class a single copy of static variable will be created. For every Thread a separate copy of local variable will be created.

	<p>3. Instance and static variables can be accessed by multiple Threads simultaneously and hence these are not Thread safe but local variables can be accessed by only one Thread at a time and hence local variables are Thread safe.</p> <p>4. If we are not declaring any modifier explicitly then it means default modifier but this rule is applicable only for static and instance variables but not local variable.</p>
25.	Java is pure object-oriented programming or not?
	<ul style="list-style-type: none"> Java is not considered as pure object-oriented programming language because several oops features (like multiple inheritance, operator overloading) are not supported by java moreover we are depending on primitive data types such as int, byte, long which are not-objects. There are seven qualities to be satisfied for a programming language to be pure Object Oriented. They are: <ol style="list-style-type: none"> 1. Encapsulation/Data Hiding 2. inheritance 3. Polymorphism 4. Abstraction 5. All predefined types are objects 6. All operations are performed by sending messages to objects 7. All user defined types are objects
26.	How java is strong object oriented?
	<ul style="list-style-type: none"> Java is strongly object-oriented programming language because without class and object it is impossible to write any Java program. Java does not allow global members. Everything (variables as well as methods) must be defined inside the class.
27.	Who invokes main() function?
	<ul style="list-style-type: none"> JVM invokes main() function.
28.	Can we define more than one public class in a java source code? what is the rule of public class and file name?
	<ul style="list-style-type: none"> No, we can't define more than one public class in one source file. Name of the public class and the file name must match.
29.	Explain public static void main(String args[]) in Java.
	<ul style="list-style-type: none"> main() in Java is the entry point for any Java program. It is always written as public static void main(String[] args). <ul style="list-style-type: none"> ✓ public: Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class. ✓ static: It is a keyword in java which identifies it is class-based. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler

	<p>will throw an error as main() is called by the JVM before any objects are made and only static methods can be directly invoked via the class.</p> <ul style="list-style-type: none"> ✓ void: It is the return type of the method. Void defines the method which will not return any value. ✓ main: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs. ✓ String args[]: It is the parameter passed to the main method.
30.	What are wrapper classes in Java?
	<ul style="list-style-type: none"> • Wrapper classes convert the Java primitives into the reference types (objects). • Every primitive data type has a class dedicated to it. These are known as wrapper classes because they “wrap” the primitive data type into an object of that class.
31.	What is singleton class in Java and how can we make a class singleton?
	<ul style="list-style-type: none"> • Singleton class is a class whose only one instance can be created at any given time, in one JVM. • A class can be made singleton by making its constructor private.
32.	How can we pass argument to a function by reference instead of pass by value?
	<ul style="list-style-type: none"> • In java, we can pass argument to a function only by value and not by reference.
33.	Can we have two methods in a class with the same name?
	<ul style="list-style-type: none"> • We can define two methods in a class with the same name but with different number/type of parameters. Which method is to get invoked will depend upon the parameters passed. • For example in the class below we have two print methods with same name but different parameters. Depending upon the parameters, appropriate one will be called:

```

1 package com.cjc;
2
3 public class MethodCalling {
4
5     public void print()
6     {
7         System.out.println("Print method without parameters");
8     }
9     public void print(String name)
10    {
11        System.out.println("Print method with parameter");
12    }
13    public static void main(String args[])
14    {
15        MethodCalling obj1= new MethodCalling();
16        obj1.print();
17        obj1.print("xx");
18    }
19 }

```

Problems Javadoc Declaration Console

<terminated> MethodCalling [Java Application] C:\Users\hp\p2\pool\plugins\org.eclipse.justj.o

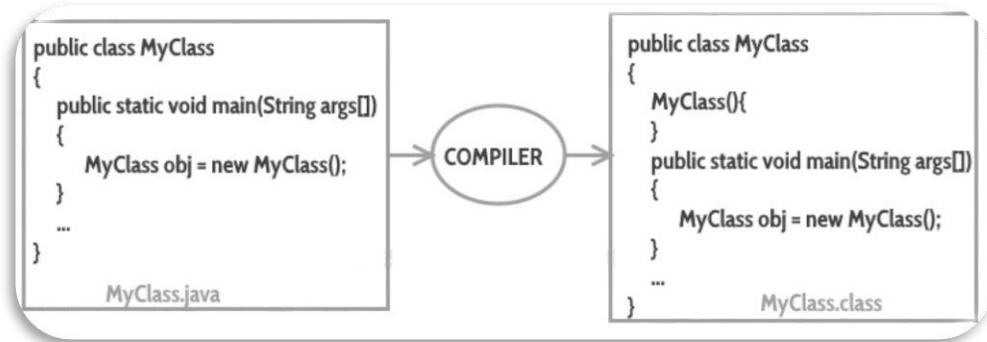
Print method without parameters

Print method with parameter

34.	Can variables be used in Java without initialization?
	<ul style="list-style-type: none"> In Java, if a variable is used in a code without prior initialization by a valid value, program doesn't compile and gives an error as no default value is assigned to variables in Java.
35.	Is JDK required on each machine to run a Java program?
	<ul style="list-style-type: none"> JDK is development Kit of Java and is required for development only and to run a Java program on a machine, JDK isn't required. Only JRE is required.
36.	If an application has multiple classes in it, is it okay to have a main method in more than one class?
	<ul style="list-style-type: none"> If there is main method in more than one classes in a java application, it won't cause any issue as entry point for any application will be a specific class and code will start from the main method of that particular class only.

Constructor

1.	What is constructor in java? How many types of Constructor, which they are?
	<ul style="list-style-type: none"> Constructor is a block of code that initializes the newly created object. Constructor has same name as the class and looks like this in a java code. <p>There are three types of constructors: Default, No-arg constructor and Parameterized.</p> <div style="text-align: center; margin: 20px 0;"> <pre> graph TD A[Types of Constructor] --> B[Default] A --> C[No-arg] A --> D[Parameterized] </pre> </div> <p>1. <u>Default Constructor:</u></p> <p>If you do not implement any constructor in your class, Java compiler inserts a default constructor into your code on your behalf and initializes all member variables to zero. This constructor is known as default constructor. You would not find it in your source code (the java file) as it would be inserted into the code during compilation and exists in .class file. This process is shown in the diagram below:</p>



If you implement any constructor then you no longer receive a default constructor from Java compiler.

2. [no-arg constructor:](#)

As the name specifies the no argument constructors of Java does not accept any parameters instead, using these constructors the instance variables of a method will be initialized with fixed values for all objects.

The signature is same as default constructor so some people claim that that default and no-arg constructor is same but in fact they are not same.

3. [Parameterized Constructor:](#)

Constructor with arguments (or you can say parameters) is known as Parameterized constructor.

Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

2. What is purpose of constructor and is constructor having return type or not?

- The purpose of Constructor is to **initialize instance variable** of class at the time of object creation.
- A constructor in Java is syntactically similar to methods but it's not a method as **it doesn't have a return type**.
- In short constructor and method are different. People often refer constructor as special type of method in Java.
- You need not call a constructor it is invoked implicitly at the time of instantiation.

3. Difference between Constructor and Method?

Constructor	Method
A Constructor is a block of code that initializes a newly created object.	A Method is a collection of statements which returns a value upon its execution.
The purpose of constructor is to initialize the object of a class	The purpose of a method is to perform a task by executing java code.
A Constructor is invoked implicitly by the system.	A Method is invoked explicitly by the programmer.

	A Constructor is invoked when an object is created using the new keyword.	A Method is invoked through method calls.
	A Constructor doesn't have a return type.	A Method must have a return type.
	A Constructor's name must be same as the class name.	A Method's name can be anything.
	A Constructor cannot be inherited by subclasses.	A Method can be inherited by subclasses.
	In case constructor is not present, a default constructor is provided by java compiler.	In the case of a method, no default method is provided.

4. How many Constructor we can write in a class?

- We can write as much Constructors as we can.

5. Difference between local variable and global variable?

Local Variable	Global Variable
Scope will be within a block only.	Scope will be within a class and outside the class.
Local variables we need to initialize compulsory before use.	Global variables by default initialize through JVM. No need to initialize.
Local variable can have only one modifier which is final.	Global variable can have different modifier such as public, static.

6. When does the compiler supply a default constructor for a class?

- The compiler supplies a default constructor for a class if no other constructors are provided.

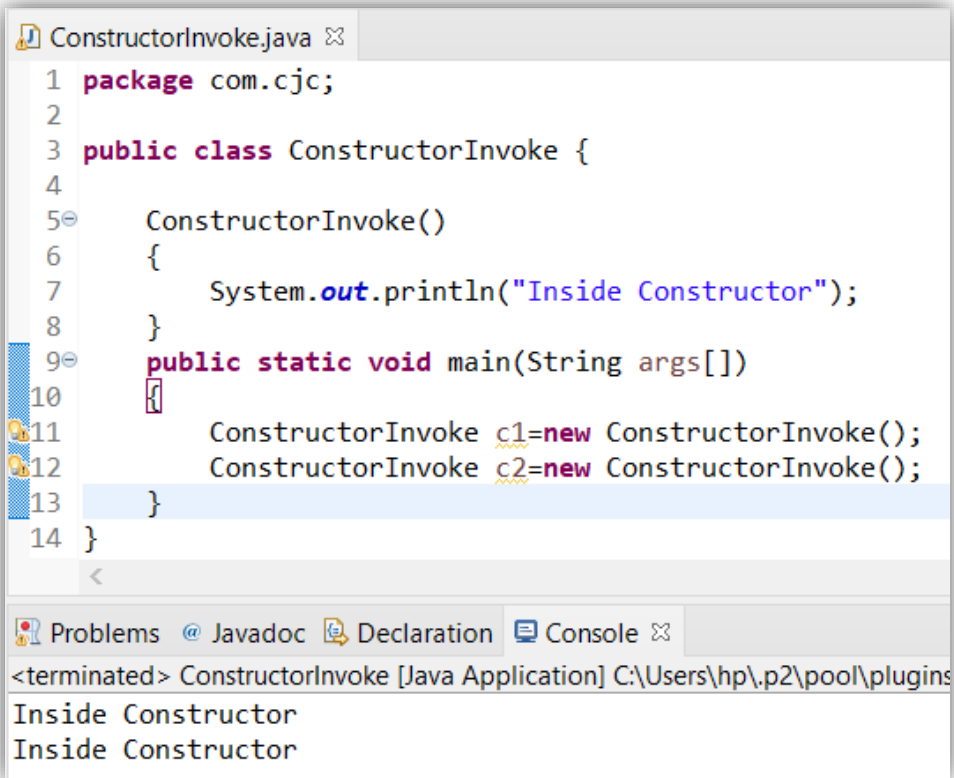
7. Can we declare a constructor as final?

- As we know, constructors are not inherited in java. Therefore, constructors are not subject to hiding or overriding.
- When there is no chance of constructor overriding, there is no chance of modification also.
- When there is no chance of modification, then no sense of restricting modification there.
- We know that the final keyword restricts further modification. So, a java constructor cannot be final because it inherently it cannot be modified.
- Also, a java constructor is internally final. So again, there is no need for final declaration further.

Example:

```
class GFG {
    //GFG() constructor is declared final
    final GFG()
    {
        //This line cannot be executed as compile error will come
        System.out.print{
            "Hey you have declared constructor as final, it's error");
        }
    }
}
```


	<pre> } class Main { public static void main(String[] args) { // Object of GFG class created // Automatically GFG() constructor called GFG obj = new GFG(); } } </pre> <p>Output</p> <pre> prog.java:1: error: modifier final not allowed here final GFG() ^ 1 error </pre> <ul style="list-style-type: none"> From the above example also, it is clear that if we are defining constructor as final the compiler will give an error as modifier final not allowed.
8.	<p>Why java doesn't support static constructor?</p> <ul style="list-style-type: none"> We know static keyword belongs to a class rather than the object of a class for example static method cannot be inherited in the sub class because they belong to the class in which they have been declared. 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. This will violate the whole purpose of inheritance concept and that is reason why a constructor cannot be static. <p>Example:</p> <pre> class GFG { //GFG() constructor is declared final static GFG() { //This line cannot be executed as compile error will come System.out.print{ "Hey you have declared constructor as final, it's error"; } } } class Main { public static void main(String[] args) { // Object of GFG class created // Automatically GFG() constructor called GFG obj = new GFG(); } } </pre> <p>Output</p> <pre> prog.java:2: error: modifier static not allowed here static GFG() ^ 1 error </pre>

	<ul style="list-style-type: none"> From the above example also it is clear that if we are defining constructor as static the compiler will give an error as modifier static not allowed.
9.	When the constructor of a class is invoked? <ul style="list-style-type: none"> The constructor of a class is invoked every time an object is created with new keyword. For example, in the following class two objects are created using new keyword and hence, constructor is invoked two times.
	 <pre> 1 package com.cjc; 2 3 public class ConstructorInvoke { 4 5 ConstructorInvoke() 6 { 7 System.out.println("Inside Constructor"); 8 } 9 public static void main(String args[]) 10 { 11 ConstructorInvoke c1=new ConstructorInvoke(); 12 ConstructorInvoke c2=new ConstructorInvoke(); 13 } 14 } </pre> <p>Problems @ Javadoc Declaration Console</p> <p><terminated> ConstructorInvoke [Java Application] C:\Users\hp\.p2\pool\plugins</p> <p>Inside Constructor Inside Constructor</p>
10.	How objects of a class are created if no constructor is defined in the class? <ul style="list-style-type: none"> Even if no explicit constructor is defined in a java class, objects get created successfully as a default constructor is implicitly used for object creation. This constructor has no parameters.
11.	Can we call the constructor of a class more than once for an object? <ul style="list-style-type: none"> Constructor is called automatically when we create an object using new keyword. It's called only once for an object at the time of object creation and hence, we can't invoke the constructor again for an object after its creation.
12.	Can we use a default constructor of a class even if an explicit constructor is defined? <ul style="list-style-type: none"> Java provides a default no argument constructor if no explicit constructor is defined in a Java class.

	<ul style="list-style-type: none"> But if an explicit constructor has been defined, default constructor can't be invoked and developer can use only those constructors which are defined in the class.
13.	Can a constructor have different name than a Class name in Java?
	<ul style="list-style-type: none"> Constructor in Java must have same name as the class name and if the name is different, it doesn't act as a constructor and compiler thinks of it as a normal method.
<u>Quick Recap of Constructor</u>	
	<ul style="list-style-type: none"> Every class has a constructor whether it's a normal class or an abstract class. Constructors are not methods and they don't have any return type. Constructor name should match with class name. Constructor can use any access specifier; they can be declared as private also. Private constructors are possible in java but their scope is within the class only. Like constructor's method can also have name same as class name, but still, they have return type, though which we can identify them that they are methods not constructors. If you don't implement any constructor within the class, compiler will do it for. this() and super() should be the first statement in the constructor code. If you don't mention them, compiler does it for you accordingly. Constructor overloading is possible but overriding is not possible. Which means we can have overloaded constructor in our class but we can't override a constructor. Constructors cannot be inherited. If Super class doesn't have a no-arg(default) constructor then compiler would not insert a default constructor in child class as it does in normal scenario. Interfaces do not have constructors. Abstract class can have constructor and it gets invoked when a class, which implements interface, is instantiated. (i.e., object creation of concrete class). A constructor can also invoke another constructor of the same class – By using this(). If you want to invoke a parameterized constructor then do it like this: this(parameter list).
<u>Package</u>	
1.	What is package?
	<ul style="list-style-type: none"> Package is nothing but folder. It is used for representing group of similar purpose .class file. The main advantage of package is to overcome name conflict problem.
2.	How to write package in our java source file?
	<ul style="list-style-type: none"> By using package keyword, we can write package in our java source file. It should be the first line of any java source file. Every dot (.) represents a subfolder e.g. package com.inheritance.practice;

3.	How to compile package java source file?
	<p style="text-align: right;">javac -d . A.java</p> <p>javac – compiler -d :- Make directory / make a folder . :- Current location A.java :- source file</p> <ul style="list-style-type: none"> If u want to compile more than 1 .java files then use following command <p style="text-align: right;">javac -d . *.java</p>
4.	How to run package compile file?
	<ul style="list-style-type: none"> By using fully qualified name of the class. <p style="text-align: right;">Java com.inheritance.practice.A</p>
5.	How to communicate two different package class?
	<ul style="list-style-type: none"> By using import keyword, we can communicate two different package class. <p style="text-align: right;">Import com.inheritance.task.*;</p>
6.	Does Importing a package imports its sub-packages as well in Java?
	<ul style="list-style-type: none"> In java, when a package is imported, its sub-packages aren't imported and developer needs to import them separately if required. For example, if a developer imports a package university.*, all classes in the package named university are loaded but no classes from the sub-package are loaded. To load the classes from its sub-package (say department), developer has to import it explicitly as follows: Import university.department.*
Static Concept	
1.	Why we use static keyword in java?
	<ul style="list-style-type: none"> Static keyword is mainly used for memory management. Static means single copy of memory or it is also called as class level memory.
2.	What is Static Block?
	<ul style="list-style-type: none"> A static block is a block of code inside a Java class that will be executed when a class is first loaded into the JVM. Mostly the static block will be used for initializing the variables. The static block will be called only once while loading and it cannot have any return type, or any keywords (this or super). Static block is used to access static members. Static block will be executed before main method, at the time of class loading. You cannot write any method inside static block but you can write variables and print statement inside static block.

	<ul style="list-style-type: none"> You can define more than one static block inside the class. They will get executed in the same order in which they are written. If you want to print some important instructions before main method execution then static block can be used. if you want any logic that needs to be executed at the time of class loading that logic need to place inside the static block so that it will be executed at the time of class loading. <pre> class test { static int val; static { val = 100; } } </pre>
3.	Can a static block exist without a main() method?
	<ul style="list-style-type: none"> You cannot have a static block alone in the class without a main method.
4.	Can we Overload static methods in Java?
	<ul style="list-style-type: none"> Yes, you can overload a static method in Java.
5.	Can we Override static methods in Java?
	<ul style="list-style-type: none"> No, you cannot override a static method in Java as there will not be any Run-time Polymorphism happening. When we declare a method with same signature and static in both Parent and Child class then it is not considered as Method Overriding as there will not be any Run-time Polymorphism happening. When the Child class also has defined the same static method like Parent class, then the method in the Child class hides the method in the Parent class. <pre> class Parent { public static void display() { System.out.println("Welcome to Parent Class"); } } public class Child extends Parent { public static void display() { System.out.println("Welcome to Child class"); } public static void main(String args[]) { //Assign Child class object to Parent reference Parent pc = new Child(); pc.display(); } } </pre> <p>Output: Welcome to Parent Class</p>

6.	What is non-static block?
	<ul style="list-style-type: none"> • If we have many constructors inside a class and those constructors need to have some common statements. Instead of repeating those statements in each constructor, we place those statements in non-static block. • For non-static block to execute compulsory you need to create an object in main method. • If object is not created in main method then non static block will not execute.
7.	Can static member function access non-static data?
	<ul style="list-style-type: none"> • No. because static member function can be called without creating object. • When object is not created, non-static member is not allocated memory.
8.	Can non-static member function access static data?
	<ul style="list-style-type: none"> • Yes. Because in order to invoke non-static member function u need to create object and by that time static members are already allocated memory.
9.	What is a static method?
	<ul style="list-style-type: none"> • A static method belongs to the class rather than an object. It can be called directly by using the class name "<<ClassName>>.<<MethodName>>" • A static method can access static variables directly and it cannot access non-static variables and can only call a static method directly and it cannot call a non-static method from it. • Only the main() method which is static will be called by the JVM automatically, Not all the static method will be called automatically.
10.	What are the restrictions that are applied to the Java static methods?
	<ul style="list-style-type: none"> • Two main restrictions are applied to the static methods. <ol style="list-style-type: none"> 1. The static method cannot use non-static data member or call the non-static method directly. 2. this and super cannot be used in static context as they are non-static.
11.	What is the difference between static and non-static member?
	<ul style="list-style-type: none"> • Static members are common for all the instances(objects) of the class but non-static members are separate for each instance(object) of class.
12.	Can you access non static variable in static context?
	<ul style="list-style-type: none"> • A static variable in Java belongs to its class and its value remains the same for all its instances. • A static variable is initialized when the class is loaded by the JVM. • If your code tries to access a non-static variable, without any instance, the compiler will complain, because those variables are not created yet and they are not associated with any instance.
13.	Can a variable be local and static at the same time?
	<ul style="list-style-type: none"> • No. a variable can't be static as well as local at the same time. Defining a local variable as static gives compilation error.

14.	What's the purpose of Static methods and static variables?								
	<ul style="list-style-type: none"> When there is a requirement to share a method or a variable between multiple objects of a class instead of creating separate copies for each object, we use static keyword to make a method or variable shared for all objects. 								
15.	Differentiate between static and non-static methods in Java.								
	<table> <tr> <th>Static Method</th><th>Non-Static Method</th></tr> <tr> <td>1. The <i>static</i> keyword must be used before the method name</td><td>1. No need to use the <i>static</i> keyword before the method name</td></tr> <tr> <td>2. It is called using the class (className.methodName)</td><td>2. It is can be called like any general method</td></tr> <tr> <td>3. They can't access any non-static instance variables or methods</td><td>3. It can access any static method and any static variable without creating an instance of the class</td></tr> </table>	Static Method	Non-Static Method	1. The <i>static</i> keyword must be used before the method name	1. No need to use the <i>static</i> keyword before the method name	2. It is called using the class (className.methodName)	2. It is can be called like any general method	3. They can't access any non-static instance variables or methods	3. It can access any static method and any static variable without creating an instance of the class
Static Method	Non-Static Method								
1. The <i>static</i> keyword must be used before the method name	1. No need to use the <i>static</i> keyword before the method name								
2. It is called using the class (className.methodName)	2. It is can be called like any general method								
3. They can't access any non-static instance variables or methods	3. It can access any static method and any static variable without creating an instance of the class								
16.	Can we have static methods in an Interface?								
	<ul style="list-style-type: none"> Static methods can't be overridden in any class while any methods in an interface are by default abstract and are supposed to be implemented in the classes being implementing the interface. So, it makes no sense to have static methods in an interface in Java. 								
17.	In a class implementing an interface, can we change the value of any variable defined in the interface?								
	<ul style="list-style-type: none"> No, we can't change the value of any variable of an interface in the implementing class as all variables defined in the interface are by default public, static and Final and final variables are like constants which can't be changed later. 								
<u>Array</u>									
1.	What is an Array?								
	<ul style="list-style-type: none"> Array is used to store homogeneous data types values in a single variable instead of declaring separate variables for each value. Arrays are fixed in size. Array indexing/position starts from 0th index. Arrays only can store Homogeneous data type. 								
2.	In case of an array which variable is created automatically? Can u modify it? If no, why?								
	<ul style="list-style-type: none"> In java whenever u create array, u get "length" variable by default which is nothing but size of array. No u can't modify it because it is final. 								

3.	Is there a way to increase the size of an array after its declaration?
	<ul style="list-style-type: none"> Arrays are static and once we have specified its size, we can't change it. If we want to use such collections where we may require a change of size (no of items), we should prefer vector over array.
String Concept	
1.	What is String?
	<ul style="list-style-type: none"> String is the sequence of characters or say string is an array of characters.
2.	Is String a keyword in java?
	<ul style="list-style-type: none"> No. String is not a keyword in java. String is a final class in java.lang package which is used to represent the set of characters in java.
3.	In how many ways you can create string objects in java?
	<ul style="list-style-type: none"> There are two ways to create string objects in java. <ol style="list-style-type: none"> By using new keyword By using String literals. <p>By using new keyword:</p> <pre>String s1 = new String("abc");</pre> <ul style="list-style-type: none"> In this case 2 objects are created. With new keyword one object is created and that object is stored in heap area and with literal "abc" one more object created and that object is stored in String Constant Pool (SCP). S1 will pointing to the object created in heap area and JVM internally pointing (providing reference) to the object created in SCP. <p>By using String literal:</p> <pre>String s2 = "xyz";</pre> <ul style="list-style-type: none"> In this case only 1 object is created and stored in SCP. This way is recommended because objects created are less.
4.	What is string constant pool?
	<ul style="list-style-type: none"> String Constant Pool is the memory space in heap memory specially allocated to store the string objects created using string literals. In String Constant Pool, there will be no two string objects having the same content. Whenever you create a string object using string literal, JVM first checks the content of the object to be created. If there exist an object in the string constant pool with the same content, then it returns the reference of that object. It doesn't create a new object. If the content is different from the existing objects then only it creates new object.

5.	Is garbage collector deletes the object created in SCP?
	<ul style="list-style-type: none"> No. The string object present in SCP are not applicable for garbage collection because a reference variable internally is maintained by JVM. The object which is not pointing by any variable are applicable for garbage collection or that object is deleted by garbage collector.
6.	What is special about string objects as compared to objects of other derived types?
	<ul style="list-style-type: none"> One special thing about string objects is that you can create string objects without using new operator i.e using string literals. This is not possible with other derived types (except wrapper classes). One more special thing about strings is that you can concatenate two string objects using '+'. This is the relaxation java gives to string objects as they will be used most of the time while coding. And also java provides string constant pool to store the string objects
7.	What do you mean by mutable and immutable objects?
	<ul style="list-style-type: none"> Immutable objects are like constants. You can't modify them once they are created. They are final in nature. Whereas mutable objects are concerned, you can perform modifications to them.
8.	Which is the final class in these three classes – String, StringBuffer and StringBuilder?
	<ul style="list-style-type: none"> All three are final. (Interviewer will ask this type of questions to confuse you)
9.	Why StringBuffer and StringBuilder classes are introduced in java when there already exist String class to represent the set of characters?
	<ul style="list-style-type: none"> The objects of String class are immutable in nature. i.e you can't modify them once they are created. If you try to modify them, a new object will be created with modified content. This may cause memory and performance issues if you are performing lots of string modifications in your code. To overcome these issues, StringBuffer and StringBuilder classes are introduced in java.
10.	How do you create mutable string objects?
	<ul style="list-style-type: none"> Using StringBuffer and StringBuilder classes. These classes provide mutable string objects.
11.	Which one will you prefer among "==" and equals() method to compare two string objects?
	<ul style="list-style-type: none"> prefer equals() method because it compares two string objects based on their content and returns true if the two have same value. That provides more logical comparison of two string objects. If you use "==" operator, it checks only references of two objects are equal or not. It may not be suitable in all situations. So, rather stick to equals() method to compare two string objects.

12.	Which class will you recommend among String, StringBuffer and StringBuilder classes if I want mutable and thread safe objects?
	<ul style="list-style-type: none"> StringBuffer
13.	Why strings have been made immutable in java?
	<ol style="list-style-type: none"> Immutable strings increase security. As they can't be modified once they are created, so we can use them to store sensitive data like username, password etc. Immutable strings are thread safe. So, we can use them in a multi threaded code without synchronization. String objects are used in class loading. If strings are mutable, it is possible that wrong class is being loaded as mutable objects are modifiable.
14.	What do you think about string constant pool? Why they have provided this pool as we can store string objects in the heap memory itself?
	<ul style="list-style-type: none"> String constant pool increases the reusability of existing string objects. When you are creating a string object using string literal, JVM first checks string constant pool. If that object is available, it returns reference of that object rather creating a new object. This will also speed up your application as only reference is returned and also saves the memory as no two objects with same content are created.
15.	String Class Methods
	<pre> Test.java 1 package com.string.methods; 2 3 public class Test { 4 5 public static void main(String[] args) { 6 String s="abc"; 7 8 //1.concat()--->to concat another string with existing one 9 s=s.concat("xyz"); 10 System.out.println(s); //abcxyz 11 12 //2.equals()--->to compare content of two string object 13 String s1="abcxyz"; 14 System.out.println(s1.equals(s)); //true 15 16 //3.toString()--->represents object 17 Test t=new Test(); 18 System.out.println(t.toString()); //com.string.methods.Test@39ed3c8d 19 20 //4.trim--->trims whitespace leading and trailing string object 21 String str=" Complete Java Classes "; 22 String str1=str.trim(); 23 System.out.println(str1); //Complete Java Classes 24 25 //5.length()--->Length of string in terms of number of character 26 System.out.println(str.length()); //23 27 28 //6.subString(int begin)--->subString of string from begin index 29 System.out.println(str.substring(8)); 30 </pre>

	<pre> 31 //7.subString(int begin, int end) 32 System.out.println(str.substring(0, 9)); // Complete 33 34 //8.toUpperCase----> string to UPPERCASE 35 System.out.println(str.toUpperCase()); // COMPLETE JAVA CLASSES 36 37 //9.toLowerCase 38 System.out.println(str.toLowerCase()); // complete java classes 39 40 //10.intern---->for pointing object stored in SCP 41 String ss=new String("nopq"); 42 System.out.println(ss.intern()); 43 String st=ss.intern(); 44 System.out.println(st.equals(ss)); 45 46 //11.charAt---->gives index of number from a given string literal 47 String str2="Core Java"; 48 System.out.println(str2.charAt(5)); //J-->is at 5th index(will not take into account space between core and java) 49 for(int i=0;i<str2.length();i++) 50 { 51 System.out.println(str2.charAt(i)); 52 } 53 54 //12.isEmpty() 55 String str3=""; 56 System.out.println(str3.isEmpty()); 57 58 //13.equalsIgnoreCase()---->ignore Case in between two string literals and then compares 59 String str4="SHARVIL"; 60 String str5="sharvil"; 61 System.out.println(str4.equalsIgnoreCase(str5)); 62 63 //14.compareTo 64 String str6="abhiii"; 65 String str7="abhii"; 66 System.out.println(str6.compareTo(str7)); 67 68 //15.split(String) 69 String str8="Complete, Java, Classes"; 70 String[] sar=str8.split(", "); // Complete Java Classes 71 System.out.println(sar[0]); 72 System.out.println(sar[1]); 73 System.out.println(sar[2]); 74 75 String str9="Core Java"; 76 String[] sarr=str9.split(" "); 77 System.out.println(sarr[0]); 78 System.out.println(sarr[1]); 79 80 //16.indexOf() 81 System.out.println(str8.indexOf('p')); //3 82 System.out.println(str9.indexOf('J')); 83 } 84 } </pre>
16.	Differences between final and immutability
	<ul style="list-style-type: none"> • final: In Java, final is a modifier which is used for class, method and variable also. When a variable is declared with final keyword, it's value can't be modified, essentially, a constant. • Immutability: In simple terms, immutability means unchanging over time or unable to be changed. In Java, we know that String objects are immutable means we can't change anything to the existing String objects. <p>Differences</p> <ul style="list-style-type: none"> • final means that you can't change the object's reference to point to another reference or another object, but you can still mutate its state (using setter methods e.g). Whereas immutable means that the object's actual value can't be changed, but you can change its reference to another one. • final modifier is applicable for variable but not for objects, Whereas immutability applicable for an object but not for variables. • By declaring a reference variable as final, we won't get any immutability nature, even though reference variable is final. We can perform any type of

	<p>change in the corresponding Object. But we can't perform reassignment for that variable.</p> <ul style="list-style-type: none"> • final ensures that the address of the object remains the same whereas the Immutable suggests that we can't change the state of the object once created.
--	---

17. What are two different ways to call garbage collector?

1. System.gc() OR
2. Runtime.getRuntime().gc().

18. Difference between String, StringBuilder, and StringBuffer.

Factor	String	StringBuilder	StringBuffer
Storage Area	Constant String Pool	Heap Area	Heap Area
Mutability	Immutable	Mutable	Mutable
Thread Safety	Yes	No	Yes
Performance	Fast	More efficient	Less efficient

OOPs-Inheritance

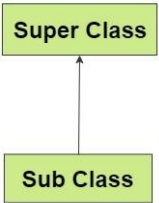
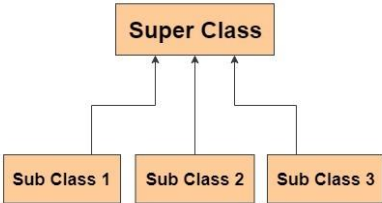
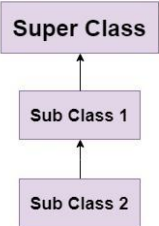
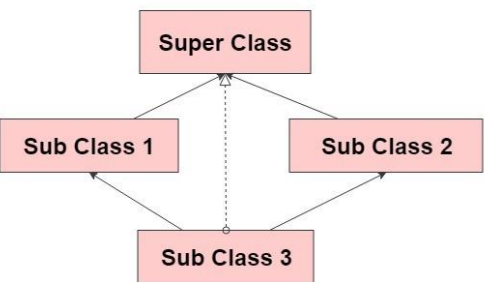
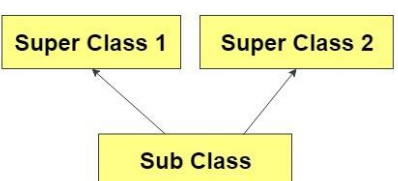
1. What is OOPs? Which are Main Pillars of OOPS?

- OOPs is objected oriented programming Concept.
- As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming.
- Object-Oriented Programming (OOPs) is a type of programming that is based on objects rather than just functions and procedures. Individual objects are grouped into classes. OOPs implements real-world entities like inheritance, polymorphism, hiding, etc into programming. It also allows binding data and code together.
- One of the Language java also uses this concept that's why we call java as OOP language.
- This concept is totally dealing with class and objects and designed from our day-to-day life.
- The main purpose of this concept- make readability and understanding should be easy.

The main Pillars of OOPs-

- 1. Inheritance-** When one object acquires all the properties and behaviours of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
- 2. Polymorphism-** If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently.
In Java, we use method overloading and method overriding to achieve polymorphism.
- 3. Encapsulation-** Binding (or wrapping) code and data together into a single unit are known as encapsulation.

	<p>A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.</p> <p>4. Abstraction- Hiding internal details and showing functionality is known as abstraction. For example, phone call, we don't know the internal processing. In Java, we use abstract class and interface to achieve abstraction.</p>
	Why use OOPs?
	<ul style="list-style-type: none"> • OOPs allows clarity in programming thereby allowing simplicity in solving complex problems. • Code can be reused through inheritance thereby reducing redundancy. • Data and code are bound together by encapsulation. • OOPs allows data hiding, therefore, private data is kept confidential. • Problems can be divided into different parts making it simple to solve. • The concept of polymorphism gives flexibility to the program by allowing the entities to have multiple forms.
	What are the limitations of OOPs?
	<ul style="list-style-type: none"> • Usually not suitable for small problems • Requires intensive testing • Takes more time to solve the problem • Requires proper planning • The programmer should think of solving a problem in terms of objects
2.	What is inheritance and how we can achieve?
	<ul style="list-style-type: none"> • The process by which one class acquires the properties (data members) and functionalities (methods) of another class is called inheritance. • Inheritance is an important pillar of OOP (Object-Oriented Programming). The aim of inheritance is to provide the reusability of code. • Inheritance represents the IS-A relationship which is also known as a parent-child relationship. • The class which inherits the properties of other class is known as subclass (derived class, child class). • The class whose properties are inherited is known as superclass (base class, parent class). • Inheritance in java can be achieved by a keyword called "<u>extends.</u>" <p>Inheritance in java is used for</p> <ol style="list-style-type: none"> 1. For Method Overriding (so runtime polymorphism can be achieved). 2. For Code Reusability.

3.	How many types of inheritance explain with diagram?
	<div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p>Single Inheritance</p>  </div> <div style="text-align: center;"> <p>Hierarchical Inheritance</p>  </div> <div style="text-align: center;"> <p>MultiLevel Inheritance</p>  </div> </div> <div style="display: flex; justify-content: space-around; align-items: flex-start; margin-top: 20px;"> <div style="text-align: center;"> <p>Hybrid Inheritance</p>  </div> <div style="text-align: center;"> <p>Multiple Inheritance</p>  </div> </div> <ol style="list-style-type: none"> 1. Single inheritance consists of one parent class and one child class. The child class inherits parent class methods and data members. 2. Multi-level inheritance is like a parent-child inheritance relationship—the difference is that a child class inherits another child class. 3. When two or more child classes inherits a single parent class, it is known as hierarchical inheritance. 4. Hybrid inheritance can be a combination of any of the three types of inheritances supported in Java. 5. There is also a fifth type of Inheritance, but it is not supported in Java, as multiple class inheritance causes ambiguities. Multiple inheritance is also called a diamond problem. Hence, Java does not support multiple class inheritance.
4.	Which type of inheritance java supports and which are not?
	<ul style="list-style-type: none"> Java supports only single, multilevel, and hierarchical type of inheritance using classes. Java does not support multiple and hybrid inheritance with classes. In java programming, multiple and hybrid inheritance is supported through interface only. To reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class. Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So, whether you have same method or different, there will be compile time error.

5.	<p>What is type casting? Explain types of type casting?</p> <ul style="list-style-type: none"> In Java, type casting is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer. <div style="text-align: center;"> <p>Type Casting in Java</p> </div> <p>There are two types of type casting:</p> <ol style="list-style-type: none"> Widening Type Casting Narrowing Type Casting <p>Widening Type Casting</p> <ul style="list-style-type: none"> Converting a lower data type into a higher one is called widening type casting. It is also known as implicit conversion or casting down. It is done automatically. It is safe because there is no chance to lose data. It takes place when: <ol style="list-style-type: none"> Both data types must be compatible with each other. The target type must be larger than the source type. <p>Narrowing Type Casting</p> <ul style="list-style-type: none"> Converting a higher data type into a lower one is called narrowing type casting. It is also known as explicit conversion or casting up. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error. 								
6.	<p>Does a class inherit the constructors of its superclass?</p> <ul style="list-style-type: none"> A class does not inherit constructors from any of its super classes. 								
7.	<p>Difference between IS-A relationship & HAS-A relationship?</p> <table border="1"> <thead> <tr> <th>IS-A relationship</th><th>HAS-A relationship</th></tr> </thead> <tbody> <tr> <td>It is also known as inheritance; it is acquiring parent class properties.</td><td>It is acquiring class properties by creating instance of that class.</td></tr> <tr> <td>It is achieved by “extends” keyword.</td><td>It is achieved by “new” keyword.</td></tr> <tr> <td>Method with same name, signature but different return types can't be created in parent-child class.</td><td>Method with same name and different return type can be created.</td></tr> </tbody> </table>	IS-A relationship	HAS-A relationship	It is also known as inheritance; it is acquiring parent class properties.	It is acquiring class properties by creating instance of that class.	It is achieved by “extends” keyword.	It is achieved by “new” keyword.	Method with same name, signature but different return types can't be created in parent-child class.	Method with same name and different return type can be created.
IS-A relationship	HAS-A relationship								
It is also known as inheritance; it is acquiring parent class properties.	It is acquiring class properties by creating instance of that class.								
It is achieved by “extends” keyword.	It is achieved by “new” keyword.								
Method with same name, signature but different return types can't be created in parent-child class.	Method with same name and different return type can be created.								

8.	What is constructor chaining in Java?
	<ul style="list-style-type: none"> In Java, constructor chaining is the process of calling one constructor from another with respect to the current object. Constructor chaining is possible only through legacy where a subclass constructor is responsible for invoking the superclass' constructor first. There could be any number of classes in the constructor chain. Constructor chaining can be achieved in two ways: <ol style="list-style-type: none"> Within the same class using this() From base class using super()
9.	Can a class in Java be inherited from more than one class?
	<ul style="list-style-type: none"> In Java, a class can be derived from only one class and not from multiple classes. Multiple inheritances is not supported by Java.
10.	What are the limitations of inheritance?
	<ul style="list-style-type: none"> Increases the time and effort required to execute a program as it requires jumping back and forth between different classes. The parent class and the child class get tightly coupled. Any modifications to the program would require changes both in the parent as well as the child class. Needs careful implementation else would lead to incorrect results.
11.	What is the meaning of "IS-A" and "HAS-A" relationship?
	<ul style="list-style-type: none"> "IS-A" relationship implies inheritance. A sub class object is said to have "IS-A" relationship with the super class or interface. If class A extends B then A "IS-A" B. It is transitive, that is, if class A extends B and class B extends C then A "IS-A" C. The "instanceof" operator in java determines the "IS-A" relationship. When a class A has a member reference variable of type B then A "HAS-A" B. It is also known as Aggregation.

<u>super and this</u>												
1.	Difference between super() and this()?											
	<table><tr><th>Key</th><th>super()</th><th>this()</th></tr><tr><td>Definition</td><td>super() - refers immediate parent class instance.</td><td>this() - refers current class instance.</td></tr><tr><td>Invoke</td><td>Can be used to invoke immediate parent class method.</td><td>Can be used to invoke current class method.</td></tr></table>	Key	super()	this()	Definition	super() - refers immediate parent class instance.	this() - refers current class instance.	Invoke	Can be used to invoke immediate parent class method.	Can be used to invoke current class method.		
Key	super()	this()										
Definition	super() - refers immediate parent class instance.	this() - refers current class instance.										
Invoke	Can be used to invoke immediate parent class method.	Can be used to invoke current class method.										

	Constructor	super() acts as immediate parent class constructor and should be first line in child class constructor.	this() acts as current class constructor and can be used in parametrized constructors.
	Override	When invoking a superclass version of an overridden method the super keyword is used	When invoking a current version of an overridden method the this keyword is used.

2. Difference between super and this?

Key	Super	this
Represent and Reference	super keyword represents the current instance of a parent class	This keyword mainly represents the current instance of a class.
Interaction with class constructor	super keyword used to call default constructor of the parent class.	this keyword used to call default constructor of the same class.
Method accessibility	One can access the method of parent class with the help of super keyword.	this keyword used to access methods of the current class as it has reference of current class.
Static context	On other hand super keyword can't be referred from static context i.e can't be invoked from static instance. For instance, we cannot write <code>System.out.println(super.x)</code> this will leads to compile time error.	this keyword can be referred from static context i.e can be invoked from static instance. For instance, we can write <code>System.out.println(this.x)</code> which will print value of x without any compilation or runtime error.

3. Difference between super(), this() and super, this?

super(), this()	super, this
These are constructor calls, to call super class and current class constructor.	These are keywords to refer super class and current class instance members.
We should use only inside constructors as first line only, if we are using outside of constructor, we will get compile time error.	We can use anywhere (i.e., instance area) except static area, otherwise we will get compile time error.
We can use either super() (or) this() but not both simultaneously.	We can use any number of times.

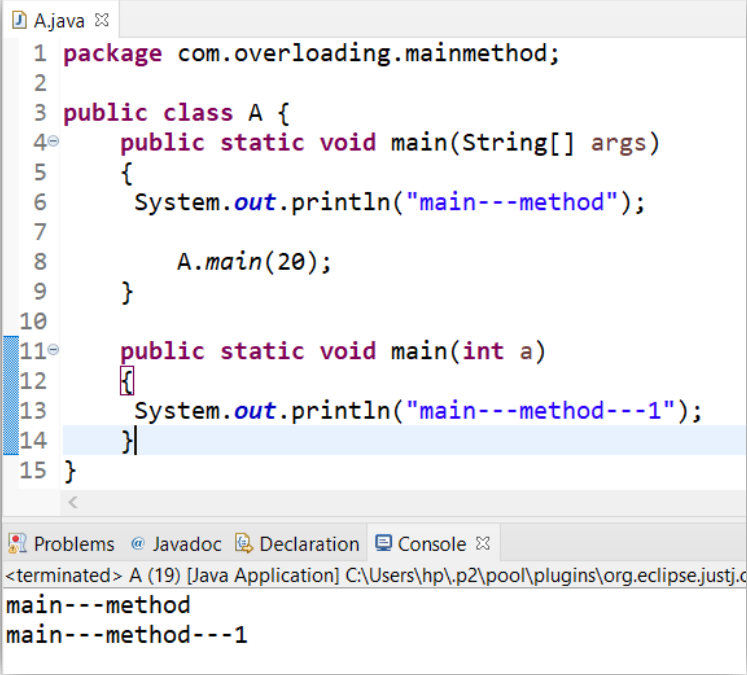
4. Can we use "this" with static member?

<ul style="list-style-type: none"> No. "this" and "static" both are contradictory. i.e. this is a reference to invoking object and static is not at all related to object.

5.	What are the main uses of this keyword?
	<ul style="list-style-type: none"> There are the following uses of this keyword. <ul style="list-style-type: none"> this can be used to refer to the current class instance variable. this can be used to invoke current class method (implicitly) this() can be used to invoke the current class constructor. this can be passed as an argument in the method call. this can be passed as an argument in the constructor call. this can be used to return the current class instance from the method.
6.	What are the main uses of the super keyword?
	<ul style="list-style-type: none"> There are the following uses of super keyword. <ul style="list-style-type: none"> super can be used to refer to the immediate parent class instance variable. super can be used to invoke the immediate parent class method. super() can be used to invoke immediate parent class constructor.
7.	What's the order of call of constructors in inheritance?
	<ul style="list-style-type: none"> In case of inheritance, when a new object of a derived class is created, first the constructor of the super class is invoked and then the constructor of the derived class is invoked.

<u>Polymorphism-Overloading</u>	
1.	What is Polymorphism and how we can achieve it?
	<ul style="list-style-type: none"> Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So, polymorphism means many forms. There are two types of polymorphism in Java: <u>compile-time polymorphism</u> and <u>runtime polymorphism</u>. We can perform polymorphism in java by <u>Method Overloading(Static Binding)</u> and <u>Method Overriding(Dynamic Binding)</u>. <p>Real life example of polymorphism: A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism</p>
2.	Define Overloading. What are advantages and example of overloading?
	<ul style="list-style-type: none"> When a class has two or more methods by the same name but different parameters, it is known as method overloading.

	<p><u>Advantages of method overloading in java</u></p> <ol style="list-style-type: none"> 1. Overloading in Java is the ability to create multiple methods of the same name, but with different parameters. 2. The main advantage of this is cleanliness of code. 3. Method overloading increases the readability of the program. 4. Overloaded methods give programmers the flexibility to call a similar method for different types of data. 5. Overloading is also used on constructors to create new objects given different amounts of data. <p><u>Real-life Example</u></p> <p>Assume, you are supposed just perform the function of talking. Say, you have to tell the story of your day, to a total stranger. Your function will get over pretty quickly. Say, now you are telling the same to your beloved. You will go through more details as compared to the previous one. What has happened here, is, you have performed the same function, but based on the parameter, stranger/beloved, your way of implementing the function changed!</p> <p><u>Java Programming perspective Example</u></p> <pre>void func() { ... } void func(int a) { ... } float func(double a) { ... } float func(int a, float b) { ... }</pre> <p>Here, the func() method is overloaded. These methods have the same name but accept different arguments.</p> <p>Notice that, the return type of these methods is not the same. Overloaded methods may or may not have different return types, but they must differ in parameters they accept.</p> <ul style="list-style-type: none"> • Method overloading is achieved by either: <ul style="list-style-type: none"> ○ changing the number of arguments. ○ or changing the datatype of arguments. • Method overloading is not possible by changing the return type of methods.
3.	<p><u>Explain the rules and also syntax rules of overloading.</u></p> <ul style="list-style-type: none"> • <u>Rules: -</u> <ul style="list-style-type: none"> ➤ We can overload method and constructor both. ➤ We can overload static, private, final method. ➤ We can overload main method also. • <u>Syntax Rules: -</u> <ol style="list-style-type: none"> 1. Method name must be same. 2. Method parameter or signature must be different. 3. Access modifier and return type doesn't matter.

4.	<p>Can we overload main method in java?</p> <ul style="list-style-type: none"> • Yes, we can overload the main() method. • The main() method is the starting point of any Java program. • The JVM starts the execution of any Java program from the main() method. Without the main() method, JVM will not execute the program. • It is a default signature that is predefined in the JVM. • It is called by JVM to execute a program line by line and ends the execution after completion of the method.  <pre> 1 package com.overloading.mainmethod; 2 3 public class A { 4 public static void main(String[] args) 5 { 6 System.out.println("main---method"); 7 8 A.main(20); 9 } 10 11 public static void main(int a) 12 { 13 System.out.println("main---method---1"); 14 } 15 } </pre> <p>main---method main---method---1</p> <p>Note:</p> <ul style="list-style-type: none"> • The JVM always calls the original main() method. It does not call the overloaded main() method.
5.	<p>Why we call overloading as a compile time polymorphism?</p> <ul style="list-style-type: none"> • Every decision will happen at compile time only, is there method is present or not and which method will be executed at runtime.
6.	<p>Can we achieve method overloading by changing the return type of method only?</p> <ul style="list-style-type: none"> • No, in java method overloading is not possible by changing the return type of method only because of ambiguity.
7.	<p>Can we override a method that throws runtime exception without throws clause?</p> <ul style="list-style-type: none"> • Yes, there is no restriction on unchecked exceptions while overriding. • On the other hand, in the case of checked exception, an overriding exception cannot throw a checked exception which comes higher in type hierarchy. • e.g. if the original method is throwing IOException than the overriding method cannot throw java.lang.Exception or java.lang.Throwable.

Polymorphism-Overriding

1. Define Overriding. What are advantages and example of overriding?

- If child class is not satisfied with parent class method implementation then child class rewrite their own implementation that is called method overriding.
- In this case the method in parent class is called overridden method and the method in child class is called overriding method.

Advantages of method overriding in java

1. Main purpose of overriding is to provide additional meaning of existing functionality.
2. Method Overriding is used for Runtime Polymorphism.

Real life example

you learnt driving from you dad! But you both drive the same vehicle differently! (Dad drives cautiously and you drive rash) That is overriding.

Programming perspective example

```
public class Parent
{
    void display()
    {
        System.out.println("parent_display_method");
    }
}
```

```
public class Child extends Parent
{
    @Override
    public void display()
    { System.out.println("child_display_method");
    }
}
```

2. Explain the rules and also syntax rules of overriding.

Rules –

1. We can override method only. We cannot override constructor.
2. We cannot override static, private or final method.
3. We cannot override main method.
4. Overriding always be happen in parent and child class only.

Syntax Rules –

1. Method name must be same.
2. Method parameter or signature must be same.
3. Method return type if void or primitive then it must be same.
4. If method return type is user defined data type then it should be same or their child class.
5. Access modifier should be same or greater but it should not be weaker.

3. Write the difference between method Overloading and method Overriding in java.

Method Overloading	Method Overriding
Method overloading is used to increase the readability of the program.	Method Overriding is used to provide additional meaning of existing functionality.
Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
Inheritance is not involved.	Inheritance is involved.
In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.

	Leads to static binding and static polymorphism.	Leads to dynamic binding and dynamic polymorphism.
	In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.
	One overloaded method does not hide the other method.	Subclass overridden method hides super class method from access by subclass object.
	We can overload Static, private, final method	We cannot override Static, private, final method
	println() of printstream class is example of method overloading.	equals() of object class and equals() of string class are example of method overriding.

4. Can we override main method in java?

- **No**, we cannot override main method in java.
- **Reason:**
We cannot override static methods since main method [public **static** void main(String[] args)] is static, we cannot override it.

5. Can we override the overloaded method?

- Yes we can override the overloaded method.

6. Can we override a method by using same method name and arguments but different return types?

- The basic condition of method overriding is that method name, arguments as well as return type must be exactly same as is that of the method being overridden. Hence using a different return type doesn't override a method.

7. What is object class? How many methods are present in object class?

- The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
- There are total **11** methods present in object class.

1. getClass(): Class<?>	2. hashCode(): int	3. equals(Object): boolean
4. clone(): Object	5. toString(): String	6. notify(): void
7. notifyAll(): void	8. wait(): void	9. wait(long): void
10. wait(long, int): void	11. finalize(): void	

- **hashCode:** It returns hash value of the object
- **equals:** It compares the object references
- **wait:** It causes current thread to wait until notify or notifyAll is not called
- **notify:** Wakes up single thread which is waiting for lock
- **notifyAll:** Wakes up all threads which is waiting for lock
- **toString:** Provides String representation of the object

	<ul style="list-style-type: none"> ➤ clone: This method is used to clone the object ➤ finalize: This method is called when object is being garbage collected.
<u>Encapsulation</u>	
1.	What is Encapsulation? Explain with example.
	<ul style="list-style-type: none"> • Wrapping of methods and variables in class is called Encapsulation. • Every java class is example of Encapsulation. • Setter-Getter class (POJO) is best example of proper Encapsulation.
2.	How to achieve or implement Encapsulation in java?
	<ul style="list-style-type: none"> • With the help of Access Modifiers (public, protected, default, private) we achieve encapsulation. • To achieve encapsulation in Java – <ul style="list-style-type: none"> ✓ Declare the variables of a class as private. ✓ Provide public setter and getter methods to modify and view the variables values.
3.	Why Encapsulation is known as Data Hiding?
	<ul style="list-style-type: none"> • The whole idea behind encapsulation is to hide the implementation details from users. • If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class. • However, if we setup public getter and setter methods to update and read the private data fields then the outside class can access those private data fields via public methods. • This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That's why encapsulation is known as data hiding.
4.	Explain accessibility modifiers in java.
	<ul style="list-style-type: none"> • As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor, variable, method, or data member. • In Java, access modifiers are used to set the accessibility (visibility) of classes, interfaces, variables, methods, constructors, data members, and the setter methods. • Access modifiers can be specified separately for a class, its constructors, fields and methods. Java access modifiers are also sometimes referred to in daily speech as Java access specifiers, but the correct name is Java access modifiers. • There are four types of access modifiers available in java: <ol style="list-style-type: none"> 1. Private - declarations are visible within the class only 2. Default (No keyword required) –declarations are visible only within the package (package private) 3. Protected - declarations are visible within the package or all subclasses 4. Public - declarations are visible everywhere

Note- You cannot set the access modifier of getters methods.

1. private Access Modifier

- If a method or variable is marked as private (has the private access modifier assigned to it), then only code inside the same class can access the variable, or call the method. Code inside subclasses cannot access the variable or method, nor can code from any external class.
- Classes cannot be marked with the private access modifier. Marking a class with the private access modifier would mean that no other class could access it, which means that you could not really use the class at all. Therefore, the private access modifier is not allowed for classes.
- Here is an example of assigning the private access modifier to a field:

```
public class Clock {  
    private long time = 0;  
}
```

- The member variable time has been marked as private. It means, that the member variable time inside the Clock class cannot be accessed from code outside the Clock class.

2.default (package) Access Modifier

- The default Java access modifier is declared by not writing any access modifier at all.
- The default access modifier means that code inside the class itself as well as code inside classes in the same package as this class, can access the class, field, constructor or method which the default access modifier is assigned to. Therefore, the default access modifier is also sometimes referred to as the package access modifier.
- Subclasses cannot access methods and member variables (fields) in the superclass, if these methods and fields are marked with the default access modifier, unless the subclass is located in the same package as the superclass.
- Here is default / package access modifier example:

```
public class Clock {  
    long time = 0;  
}  
public class ClockReader {  
    Clock clock = new Clock();  
  
    public long readClock{  
        return clock.time;  
    }  
}
```

- The time field in the Clock class has no access modifier, which means that it is implicitly assigned the default / package access modifier. Therefore, the ClockReader class can read the time member variable of the Clock object, provided that ClockReader and Clock are located in the same Java package.

3.protected Access Modifier

- The protected access modifier provides the same access as the default access modifier, with the addition that subclasses can access protected methods and member variables (fields) of the superclass. This is true even if the subclass is not located in the same package as the superclass.
- Here is a protected access modifier example:

```
public class Clock {  
    protected long time = 0;    // time in milliseconds  
}  
  
public class SmartClock() extends Clock{  
  
    public long getTimeInSeconds() {  
        return this.time / 1000;  
    }  
}
```

- In the above example the subclass SmartClock has a method called getTimeInSeconds() which accesses the time variable of the superclass Clock. This is possible even if Clock and SmartClock are not located in the same package, because the time field is marked with the protected Java access modifier.

4.public Access Modifier

- The Java access modifier public means that all code can access the class, field, constructor or method, regardless of where the accessing code is located. The accessing code can be in a different class and different package.
- Here is a public access modifier example:

```
public class Clock {  
    public long time = 0;  
}  
  
public class ClockReader {  
    Clock clock = new Clock();  
  
    public long readClock{  
        return clock.time;  
    }  
}
```

- The time field in the Clock class is marked with the public Java access modifier. Therefore, the ClockReader class can access the time field in the Clock no matter what package the ClockReader is located in.

	private	default	protected	public
Class	No	Yes	No	Yes
Constructor	Yes	Yes	Yes	Yes
Method	Yes	Yes	Yes	Yes
Variable	Yes	Yes	Yes	Yes

	<table><tr><th>Accessibility modifiers</th><th>Same class</th><th>Sub class in same package</th><th>Different class in same package</th><th>Sub class in different package</th><th>Different class in different package</th></tr><tr><td>Private</td><td>Yes</td><td>No</td><td>No</td><td>No</td><td>No</td></tr><tr><td>default</td><td>Yes</td><td>Yes</td><td>Yes</td><td>No</td><td>No</td></tr><tr><td>Protected</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td><td>No</td></tr><tr><td>Public</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td></tr></table>	Accessibility modifiers	Same class	Sub class in same package	Different class in same package	Sub class in different package	Different class in different package	Private	Yes	No	No	No	No	default	Yes	Yes	Yes	No	No	Protected	Yes	Yes	Yes	Yes	No	Public	Yes	Yes	Yes	Yes	Yes
Accessibility modifiers	Same class	Sub class in same package	Different class in same package	Sub class in different package	Different class in different package																										
Private	Yes	No	No	No	No																										
default	Yes	Yes	Yes	No	No																										
Protected	Yes	Yes	Yes	Yes	No																										
Public	Yes	Yes	Yes	Yes	Yes																										
5.	Which OOPS concept exposes only the necessary information to the calling functions? <ul style="list-style-type: none">Encapsulation																														
Abstraction																															
1.	What is Abstraction? <ul style="list-style-type: none">Abstraction is the process of hiding certain details and showing only essential information to the user. i.e., we have to highlight set of services what we are offering and we have to hide internal implementation details.E.g., By using ATM GUI screen bank people are highlighting the set of services what they are offering without highlighting internal implementation.In java we achieve abstraction by using Interface and abstract class.We can achieve 100% abstraction using "Interface" and partial by using "Abstract Class". <p>The main advantages of Abstraction are:</p> <ol style="list-style-type: none">We can achieve security as we are not highlighting our internal implementation. (i.e., outside person doesn't aware our internal implementation.)Enhancement will become very easy because without effecting end user we can able to perform any type of changes in our internal system.It provides more flexibility to the end user to use system very easily.It improves maintainability of the application.It improves modularity of the application.It improves easiness to use our system.																														
2.	What is the main purpose of designing interface and abstract class? <ul style="list-style-type: none">To provide same protocol or rules and guidelines.																														
3.	Tell the examples where we need to use abstractions? <ol style="list-style-type: none">While development of project structure and layout.For creating API or learning API. (Application Programming Interface)For third party communication. e.g., IRCTC & Bank- both are communicating each other because of abstraction.																														

4.	What is API? Explain API types with example.
	<ul style="list-style-type: none"> API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API. API is bunch of rules and guidelines. Through API, uniform of application will maintain. There are 3 types of API <ol style="list-style-type: none"> Inbuilt API Open-specification API Third party API <p>1. Inbuilt API: - Their rules & guidelines are written by sun microsystem and their implementation are also written by them. E.g., Multithreading API</p> <p>2. Open-Specification API: - Their rules & guidelines are written by sun microsystem but their implementation are written by some other vendors. E.g., Jdbc API, jsp API, servlet API</p> <p>3. Third Party API: - Their rules & guidelines are written by another person and their implementation are also written by other. E.g., hibernate API, spring API</p>
5.	What is Interface?
	<ul style="list-style-type: none"> An interface in Java is a blueprint of a class. It has static constants and abstract methods. Interface is set of rules and guidelines. We can provide same protocol for class through interface. e.g., RBI and All banks. The interface in Java is a mechanism to achieve abstraction and multiple inheritance. Interfaces can have abstract methods and variables. It cannot have a method body.
6.	How to design interface?
	<ul style="list-style-type: none"> By using interface keyword, we can design interface. e.g., public interface I <pre>{ } </pre>
7.	What modifiers may be used with an interface declaration?
	<ul style="list-style-type: none"> An interface may be declared as public or abstract.

8.	Write the rules of Interface?
	<p>➤ There are some rules that need to be followed by Interface.</p> <ul style="list-style-type: none"> • All interface Methods are implicitly public and abstract (method compulsory ends with ;). Even if you use public abstract keyword it will not create the problem. • Interfaces can declare only Constant. Instance variables are not allowed. This means all variables inside the Interface must be public, static, final (variable we need to initialize compulsory). Variables inside Interface are by default=> public static final. • Interface Methods cannot be static. • Interface Methods cannot be final, strictfp or native. • We cannot create object of interface. • We can create reference of interface by using their implemented class. • The Interface can extend one or more other Interface. Note: The Interface can only extend another interface.
9.	Can we declare an interface as final?
	<ul style="list-style-type: none"> • No, we can't do this. • An interface can't be final because the interface should be implemented by some class according to its definition. • However, if you try to do so, the compiler will show an error.
10.	What is Marker Interface? What is use of marker interface and How many marker interfaces available in Java?
	<ul style="list-style-type: none"> • Interface which has blank body (there is no method, no variable) is called marker or tag interface or null interface. • Marker interface is used for decision making purpose. • JVM will take decision through marker interface. • In java we have the following major marker interfaces as under: <ol style="list-style-type: none"> 1. Serializable interface 2. Cloneable interface 3. Remote interface 4. ThreadSafe interface
11.	Define abstract class. Write the rules of abstract class.
	<ul style="list-style-type: none"> • A class that is declared using "abstract" keyword is known as abstract class. • It can have abstract methods/unimplemented (methods without body) as well as concrete/implemented methods (regular methods with body). <p>Rules of abstract class: -</p> <ol style="list-style-type: none"> 1. An abstract class can have both abstract and non-abstract (or concrete) method. 2. An abstract class can have one or more abstract methods. 3. It cannot be instantiated. (we cannot create object of abstract class) 4. It can have constructors and static methods also. 5. Child class must implement all abstract methods of abstract class.

	<p>6. If child class is unable to implement all methods of abstract class then make that child class as abstract.</p> <p>7. A class can be marked as abstract without containing any abstract method. But if a class has even one abstract method, then the class has to be an abstract class.</p> <p>8. If an abstract class contains multiple methods, it is not necessary that all the methods of the abstract class are implemented in the immediate sub-class. Few of them can be implemented in sub-sub-classes or anywhere else in the sub-class hierarchy. But for a class to be concrete, all the abstract methods in its super-class must be implemented.</p> <p>9. It is not necessary to add the abstract methods only in the super most class, we can add more abstract methods in the sub-classes.</p>
--	--

12. Difference between normal class and abstract class.

Abstract class	Normal (Concrete) class
A class that is declared with abstract keyword is known as an abstract class.	In Java, A simple class (Without abstract keyword) is considered a concrete class.
An abstract class can have abstract methods (Method without body) and concrete/non-abstract methods (Methods with the body) also.	A concrete class can only have concrete methods. Even a single abstract method makes the class abstract.
abstract classes are incomplete classes because they have an abstract method.	Concrete classes are considered a complete class.
An abstract class may or may not have abstract methods	Concrete class cannot have an abstract method.
We can't create an object of an abstract class.	We can create an object of the concrete class.
<p>An Abstract class can't be declared as a final class because the final and abstract are opposite terms in JAVA.</p> <p>Reason: An abstract class must be inherited by any derived class because a derived class is responsible to provide the implementation of abstract methods of an abstract class. But on another hand, if a class is a final class, then it can't be extended(inherited). So, both concepts are opposite to each other.</p>	A concrete class can be declared as a final class. We can make any concrete class as final according to use.

13. Difference between abstract class and interface.

Abstract class	Interface
Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods.

	In abstract class, the keyword 'abstract' is mandatory to declare a method as an abstract	In interfaces, the keyword 'abstract' is optional to declare a method as an abstract because all the methods are abstract by default
	Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
	An abstract class can have instance variables.	An interface cannot have instance variables.
	Abstract class contains constructor.	An interface cannot contain a constructor.
	Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
	The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
	An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
	An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
	Abstract class can have protected and public abstract methods.	Interface can have only public abstract methods.
	abstract class achieves partial abstraction. (0 to 100%)	interface achieves fully abstraction. (100%)
	An abstract class can have any visibility: public, private, protected	An Interface visibility must be public (or) none
	Abstract classes are fast	Interfaces are slow as it requires extra indirection to find the corresponding method in the actual class
14.	Why And When to Use Abstract Classes and Methods?	
	<ul style="list-style-type: none">To achieve security - hide certain details and only show the important details of an object.	
15.	Why And When to Use Interfaces?	
	<ol style="list-style-type: none">To achieve security - hide certain details and only show the important details of an object (interface).Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can implement multiple interfaces. <p>Note: To implement multiple interfaces, separate them with a comma.</p>	
16.	What is the relationship between classes and interfaces?	
	<ul style="list-style-type: none">As shown in the figure given below, a class extends another class, an interface extends another interface, but a class implements an interface.	

	<pre> graph BT C1[class] -- extends --> C2[class] C3[class] -. implements .-> I1[interface] I2[interface] -- extends --> I3[interface] </pre>
17.	<p>Why can't we create the object of an abstract class?</p> <ul style="list-style-type: none"> Because these classes are incomplete, they have abstract methods that have no body. If we tried to access the methods of the abstract class by using the object then the compiler will give errors because it would not be able to execute an abstract method. To avoid this problem, java prohibits the creation of instances of abstract classes.
18.	<p>Why java class does not support multiple inheritance?</p> <ul style="list-style-type: none"> In Java, a class cannot extend more than one class. There are chances of ambiguity while extending multiple class. Consider a case where class C extends class A and Class B and both class A and B have the same method display(). Now java compiler cannot decide, which display method it should inherit. To prevent such situation, multiple inheritances is not allowed in java.
19.	<p>Why we make a class as abstract if class have no abstract method?</p> <ul style="list-style-type: none"> We want to restrict a class, no one can create object of that class.
20.	<p>Why we can restrict a class which has all implemented methods?</p> <ul style="list-style-type: none"> We can make that class abstract.
21.	<p>How we can call abstract class implemented method?</p> <ul style="list-style-type: none"> By using their child class.
22.	<p>How we can create child class of abstract class which has both types of methods implemented and non-implemented?</p> <ul style="list-style-type: none"> Rule 1: - Child class must implement all abstract methods of abstract class. Rule 2: - If child class is unable to implement all methods of abstract class then make child class as abstract.

23.	Is there any constructor inside abstract class? If yes then what is the use of abstract class constructor?														
	<ul style="list-style-type: none"> • Yes, there is constructor inside abstract class. • It is used to initialize instance variable of class. 														
24.	Is there any constructor inside Interface?														
	<ul style="list-style-type: none"> • No, interface cannot have constructors. • An Interface in Java doesn't have a constructor because all data members in interfaces are public static final by default, they are constants (assign the values at the time of declaration). • There are no data members in an interface to initialize them through the constructor. • In order to call a method, we need an object, since the methods in the interface don't have a body there is no need for calling the methods in an interface. • Since we cannot call the methods in the interface, there is no need of creating an object for an interface and there is no need of having a constructor in it. 														
25.	Can a class be defined a both final and abstract?														
	<ul style="list-style-type: none"> • No, because final and abstract are contradictory. i.e. abstract encourages inheritance whereas final stops inheritance. 														
26.	Differences between Abstraction and Encapsulation														
	<ul style="list-style-type: none"> • The major difference between abstraction and encapsulation is that abstraction hides the code complexity while encapsulation hides the internal working from the outside world. <table border="1"> <thead> <tr> <th>Abstraction</th><th>Encapsulation</th></tr> </thead> <tbody> <tr> <td>Abstraction is a feature of OOPs that hides the unnecessary detail but shows the essential information.</td><td>Encapsulation is also a feature of OOPs. It hides the code and data into a single entity or unit so that the data can be protected from the outside world.</td></tr> <tr> <td>It is the process or method of gaining information.</td><td>It is the process or method of containing the information.</td></tr> <tr> <td>It solves an issue at the design or Interface level.</td><td>It solves an issue at implementation level.</td></tr> <tr> <td>It can be implemented using abstract classes and interfaces.</td><td>It can be implemented by using the access modifiers (private, public, protected).</td></tr> <tr> <td>In abstraction, we use abstract classes and interfaces to hide the code complexities</td><td>We use the getters and setters methods to hide the data.</td></tr> <tr> <td>The objects that help to perform abstraction are encapsulated.</td><td>Whereas the objects that result in encapsulation need not be abstracted.</td></tr> </tbody> </table>	Abstraction	Encapsulation	Abstraction is a feature of OOPs that hides the unnecessary detail but shows the essential information.	Encapsulation is also a feature of OOPs. It hides the code and data into a single entity or unit so that the data can be protected from the outside world.	It is the process or method of gaining information.	It is the process or method of containing the information.	It solves an issue at the design or Interface level.	It solves an issue at implementation level.	It can be implemented using abstract classes and interfaces.	It can be implemented by using the access modifiers (private, public, protected).	In abstraction, we use abstract classes and interfaces to hide the code complexities	We use the getters and setters methods to hide the data.	The objects that help to perform abstraction are encapsulated.	Whereas the objects that result in encapsulation need not be abstracted.
Abstraction	Encapsulation														
Abstraction is a feature of OOPs that hides the unnecessary detail but shows the essential information.	Encapsulation is also a feature of OOPs. It hides the code and data into a single entity or unit so that the data can be protected from the outside world.														
It is the process or method of gaining information.	It is the process or method of containing the information.														
It solves an issue at the design or Interface level.	It solves an issue at implementation level.														
It can be implemented using abstract classes and interfaces.	It can be implemented by using the access modifiers (private, public, protected).														
In abstraction, we use abstract classes and interfaces to hide the code complexities	We use the getters and setters methods to hide the data.														
The objects that help to perform abstraction are encapsulated.	Whereas the objects that result in encapsulation need not be abstracted.														

27.	What are the performance implications of Interfaces over abstract classes?
	<ul style="list-style-type: none"> • Interfaces are slower in performance as compared to abstract classes as extra indirections are required for interfaces. • Another key factor for developers to take into consideration is that any class can extend only one abstract class while a class can implement many interfaces. • Use of interfaces also puts an extra burden on the developers as any time an interface is implemented in a class; developer is forced to implement each and every method of interface.
28.	Can we have static method in the interface?
	<ul style="list-style-type: none"> • Yes, we can have static method in the interface from Java 8.
<u>Quick Recap of Interface</u>	
	<ul style="list-style-type: none"> • An interface is 100% abstract class (Implicitly). After Java 8 it doesn't hold true. • Interfaces can be implemented by any class from any inheritance tree. • All methods in Interfaces are abstract. (In Java 8 either abstract/ static / default). • A class implementing an interface can also be an abstract class. • An abstract class which is implementing an interface need not implement all abstract method. • A class can Implement more than one Interface. • Interfaces cannot extend a class or implement an Interface. • An interface can extend another Interface. • Like abstract classes, interfaces cannot be used to create objects. • Interface methods do not have a body - the body is provided by the "implement" class • On implementation of an interface, you must override all of its methods • Interface methods are by default abstract and public • Interface attributes are by default public, static and final • An interface cannot contain a constructor (as it cannot be used to create objects) • We can't instantiate an interface in java. That means we cannot create the object of an interface • Class that implements any interface must implement all the methods of that interface; else the class should be declared abstract. • Interface cannot be declared as private, protected or transient. • A class cannot implement two interfaces that have methods with same name but different return type.
28.	What is an association?
	<ul style="list-style-type: none"> • Association is a relationship where all object have their own lifecycle and there is no owner.

	<ul style="list-style-type: none"> Let's take the example of Teacher and Student. Multiple students can associate with a single teacher and a single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. These relationships can be one to one, one to many, many to one and many to many.
29.	What do you mean by aggregation?
	<ul style="list-style-type: none"> An aggregation is a specialized form of Association where all object has their own lifecycle but there is ownership and child object cannot belong to another parent object. Let's take an example of Department and teacher. A single teacher cannot belong to multiple departments, but if we delete the department teacher object will not destroy.
30.	What is composition in Java?
	<ul style="list-style-type: none"> Composition is again a specialized form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object does not have their lifecycle and if parent object deletes all child object will also be deleted. Let's take again an example of a relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room cannot belongs to two different houses if we delete the house room will automatically delete.
<u>Exception Handling</u>	
1.	What is exception? Draw the hierarchy.
	<ul style="list-style-type: none"> Exception is an abnormal condition or unwanted situation which occurs during the execution of a program and disrupts normal flow of the program. It is an object which is thrown at runtime. When exception occurs then the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. <p>Hierarchy of Java Exception classes</p> <ul style="list-style-type: none"> The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: <ol style="list-style-type: none"> Exception and Error. A hierarchy of Java Exception classes are given below:

	<pre> graph BT Object[Object] --> Throwable[Throwable] Throwable --> Exceptions[Exceptions] Throwable --> Errors[Errors] Exceptions --> Check[Check Exceptions] Exceptions --> Unchecked[Unchecked Exceptions] Check --> IOException[IOException] Check --> SQLException[SQLException] Check --> ClassNotFoundException[ClassNotFoundException] Unchecked --> ArithmeticException[ArithmeticException] Unchecked --> NullPointerException[NullPointerException] Unchecked --> IndexOutOfBoundsException[IndexOutOfBoundsException] IndexOutOfBoundsException --> ArrayIndexOutOfBoundsException[ArrayIndexOutOfBoundsException] IndexOutOfBoundsException --> StringIndexOutOfBoundsException[StringIndexOutOfBoundsException] Errors --> StackOverflowError[StackOverflowError] Errors --> VirtualMachineError[VirtualMachineError] Errors --> OutOfMemoryError[OutOfMemoryError] </pre>
2.	<p>What is exception handling?</p> <ul style="list-style-type: none"> The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors caused by exceptions so that normal flow of the application can be maintained.
3.	<p>Write the reasons why exceptions will occur?</p> <ul style="list-style-type: none"> An exception can occur for many different reasons. Following are some scenarios where an exception occurs. <ol style="list-style-type: none"> Wrong input provided by end user. Sometimes we developer make logical mistakes. A file that needs to be opened cannot be found i.e., file is misplaced from existing location. A network connection has been lost in the middle of communications or the JVM has run out of memory. Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.
4.	<p>How the exceptions are handled in java? OR Explain exception handling mechanism in java?</p> <ul style="list-style-type: none"> Exceptions in java are handled using try, catch and finally blocks. try block: The code or set of statements which are to be monitored for exception are kept in this block. catch block: This block catches the exceptions occurred in the try block. finally block: This block is always executed whether exception is occurred in the try block or not and occurred exception is caught in the catch block or not.

5.	Write the difference between exception and error?		
	Key	Exception	Error
	Definition	Exception is an abnormal condition or unwanted situation which occurs during the execution of a program and disrupts normal flow of the program.	These are not exceptions at all, but problems that arise beyond the control of the user or the programmer.
	Time of occurrence	Exceptions can occur at compile time or runtime, depending on the type of exception occurred. For example, NullPointerException is a runtime exception on the other hand IOException is a compile-time exception	Errors occur only at runtime. They are not known to the compiler.
	Recovery	Programs can recover from Exceptions by handling them appropriately using a try-catch block or throw keyword in java.	Programs cannot recover from Errors once they occur. Errors will definitely cause termination of the program.
	Package	Exceptions are defined in java.lang.Exception package.	Errors are defined in java.lang.Error package.
	Checked/Unchecked	Exceptions can be Checked (compile-time exceptions) or Unchecked exceptions (runtime exceptions).	Errors are a part of Unchecked exceptions in java.
	Cause	Exceptions are caused by the program/application itself.	Errors are caused by the environment in which the program runs.
	Examples	Checked Exception examples are IOException, SQLException, etc. Unchecked Exception examples are NullPointerException, ArrayIndexOutOfBoundsException etc.	Errors examples are java.lang.OutOfMemoryError, java.lang.StackOverflowError etc.
6.	What are checked exception and unchecked exception?		
	Checked exceptions – <ul style="list-style-type: none"> A checked exception is an exception that is checked (notified) by the compiler at compilation-time to see whether the programmer has handled them or not. These are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions. If these exceptions are not handled/declared in the program, you will get compilation error. 		

	Unchecked exceptions – <ul style="list-style-type: none">• An unchecked exception is an exception that occurs at the time of execution.• These are also called as Runtime Exceptions.• These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit.• These include programming bugs, such as logic errors or improper use of an API.• Runtime exceptions are ignored at the time of compilation.													
7.	Write the difference between checked and unchecked exception?													
	<table><tr><th>Checked Exception</th><th>Unchecked Exception</th></tr><tr><td>Exceptions that are checked and handled at compile time are checked exception. (Compile time exception)</td><td>Exceptions that are checked and handled at run time are unchecked exception. (Runtime exception)</td></tr><tr><td>They are direct subclasses of exception but do not inherit from RuntimeException.</td><td>They are direct subclass of RuntimeException class.</td></tr><tr><td>The program gives a compilation error if a method throws a checked exception and the compiler is not able to handle the exception on its own.</td><td>The program compiles fine because the exceptions escape the notice of compiler. Exceptions occur due to errors in programming logic.</td></tr><tr><td>A checked exception occurs when the chances of failure are too high.</td><td>Unchecked exception occurs mostly due to programming mistakes.</td></tr><tr><td>Common checked exceptions include IOException, ClassNotFoundException, SQLException, FileNotFoundException etc.</td><td>Common unchecked exceptions include ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException</td></tr></table>	Checked Exception	Unchecked Exception	Exceptions that are checked and handled at compile time are checked exception. (Compile time exception)	Exceptions that are checked and handled at run time are unchecked exception. (Runtime exception)	They are direct subclasses of exception but do not inherit from RuntimeException.	They are direct subclass of RuntimeException class.	The program gives a compilation error if a method throws a checked exception and the compiler is not able to handle the exception on its own.	The program compiles fine because the exceptions escape the notice of compiler. Exceptions occur due to errors in programming logic.	A checked exception occurs when the chances of failure are too high.	Unchecked exception occurs mostly due to programming mistakes.	Common checked exceptions include IOException, ClassNotFoundException, SQLException, FileNotFoundException etc.	Common unchecked exceptions include ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException	
Checked Exception	Unchecked Exception													
Exceptions that are checked and handled at compile time are checked exception. (Compile time exception)	Exceptions that are checked and handled at run time are unchecked exception. (Runtime exception)													
They are direct subclasses of exception but do not inherit from RuntimeException.	They are direct subclass of RuntimeException class.													
The program gives a compilation error if a method throws a checked exception and the compiler is not able to handle the exception on its own.	The program compiles fine because the exceptions escape the notice of compiler. Exceptions occur due to errors in programming logic.													
A checked exception occurs when the chances of failure are too high.	Unchecked exception occurs mostly due to programming mistakes.													
Common checked exceptions include IOException, ClassNotFoundException, SQLException, FileNotFoundException etc.	Common unchecked exceptions include ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException													
8.	Write the JVM steps for exception? (Default Exception Handling)													
	<ol style="list-style-type: none">1. Code Analyse2. Find out problem3. Find out the class of the problem4. Creating object5. Throw object6. JVM will catch7. JVM will display message													
9.	Which class is the super class for all types of errors and exceptions in java?													
	<ul style="list-style-type: none">• java.lang.Throwable is the super class for all types of errors and exceptions in java.													
10.	What is OutOfMemoryError in Java?													
	<ul style="list-style-type: none">• OutOfMemoryError is the subclass of java.lang.Error which generally occurs when our JVM runs out of memory.													

11.	Keywords used in Exception Handling?				
There are 5 keywords which are used in handling exceptions in Java.					
Keyword	Description				
try	<ul style="list-style-type: none">The "try" keyword is used to specify a block where we should place exception code.If an exception occurs within the try block, that exception is handled by an exception handler associated with it.The try block must be followed by either catch or finally. It means, we can't use try block alone.				
	<table><tr><th>The syntax of java try-catch</th><th>The syntax of a try-finally block</th></tr><tr><td>Try { //code that may throw exception } catch(Exception_class_Name ref) { }</td><td>try { //code that may throw exception } finally { }</td></tr></table>	The syntax of java try-catch	The syntax of a try-finally block	Try { //code that may throw exception } catch(Exception_class_Name ref) { }	try { //code that may throw exception } finally { }
	The syntax of java try-catch	The syntax of a try-finally block			
Try { //code that may throw exception } catch(Exception_class_Name ref) { }	try { //code that may throw exception } finally { }				
catch	<ul style="list-style-type: none">The "catch" block is used to handle the exception.It must be used after the try block only means we can't use catch block alone.You can use multiple catch block with a single try. It can be followed by finally block later. <p>Syntax</p> <pre>try { //code that cause exception; } catch(Exception_type e) { //exception handling code }</pre>				
finally	<ul style="list-style-type: none">The "finally" block is used to execute the important code of the program.It is executed whether an exception is handled or not.Java finally block follows try or catch block.For each try block, there can be zero or more catch blocks, but only one finally block.The finally block will not be executed if program exits (either by calling System.exit() or by causing a fatal error that causes the process to abort).				
throw	<ul style="list-style-type: none">The "throw" keyword is used to throw an exception.				

		<ul style="list-style-type: none"> The flow of execution stops immediately after the throw statement; any subsequent statements are not executed. The nearest enclosing try block is inspected to see if it has a catch statement that matches the type of exception. If it does find a match, control is transferred to that statement. If not, then the next enclosing try statement is inspected, and so on. If no matching catch is found, then the default exception handler halts the program and prints the stack trace.
	throws	<ul style="list-style-type: none"> The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained. It is always used with method signature. We declare only checked exception using a throws keyword. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing checkup before the code is used. <p>The syntax of java throws</p> <pre>return_type method_name() throws exception_class_name { //method code }</pre>
12.	Can an exception be rethrown?	
		<ul style="list-style-type: none"> Yes, an exception can be rethrown.
13.	Can we write only try block without catch and finally blocks?	
		<ul style="list-style-type: none"> No, it shows compilation error. The try block must be followed by either catch or finally block. You can remove either catch block or finally block but not both.
14.	There are three statements in a try block – statement1, statement2 and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement2. Does statement3 get executed or not?	
		<ul style="list-style-type: none"> No. Once a try block throws an exception, remaining statements will not be executed. control comes directly to catch block.
15.	Why use java finally?	
		<ul style="list-style-type: none"> Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

16.	<p>Can we keep other statements in between try, catch and finally blocks?</p> <ul style="list-style-type: none"> No. We shouldn't write any other statements in between try, catch and finally blocks. They form a one unit. <pre> try { // Statements to be monitored for exceptions } //You can't keep statements here catch(Exception ex) { //Catching the exceptions here } //You can't keep statements here finally { // This block is always executed } </pre>
17.	<p>What are the legal combinations of try, catch and finally blocks?</p> <div> <pre> try { //try block } catch(Exception ex) { //catch block } </pre> <pre> try { //try block } finally { //finally block } </pre> <pre> try { //try block } catch(Exception ex) { //catch block } finally { //finally block } </pre> </div> <ul style="list-style-type: none"> A try block should associate with at least a catch or a finally block. The sequence of try, catch and finally matters a lot. If you modify the order of these then the code won't compile. Adding to this there can be multiple catch blocks associated with a try block. The final concept is there should be a single try, multiple catch blocks and a single finally block in a try-catch-finally block.
18.	<p>What is unreachable catch block error?</p> <ul style="list-style-type: none"> When you are keeping multiple catch blocks, the order of catch blocks must be from most specific to most general ones. i.e sub classes of Exception must come first and super classes later. If you keep super classes first and sub classes later, compiler will show unreachable catch block error.

	<pre> public class ExceptionHandling { public static void main(String[] args) { try { int i = Integer.parseInt("abc"); //This statement throws NumberFormatException } catch(Exception ex) { System.out.println("This block handles all exception types"); } catch(NumberFormatException ex) { //Compile time error //This block becomes unreachable as //exception is already caught by above catch block } } } </pre>
19.	What is the purpose of finally block?
	<ul style="list-style-type: none"> Java finally block is a block that is used to execute important code such as resource release code, closing connection, stream etc. e.g., <ol style="list-style-type: none"> If file is open inside try block then file should be close inside finally block. If database connection is open inside try block then connection should be close inside finally block. Java finally block is always executed whether exception is handled or not (either there is problem inside try block or there is no problem inside try block). Java finally block follows try or catch block. <pre> try { //Statements that may cause an exception } catch { //Handling exception } finally { //Statements to be executed } </pre>

20.	Is it compulsory to use the finally block?									
	<ul style="list-style-type: none">It is always a good practice to use the finally block.The reason for using the finally block is, any unreleased resources can be released and the memory can be freed.For example, while closing a connection object an exception has occurred. In finally block we can close that object.Coming to the question, you can omit the finally block when there is a catch block associated with that try block.A try block should have at least a catch or a finally block.									
21.	Can we keep the statements after finally block If the control is returning from the finally block itself?									
	<ul style="list-style-type: none">No, it gives unreachable code error. Because, control is returning from the finally block itself.Compiler will not see the statements after it. That's why it shows unreachable code error.									
22.	Why it is always recommended that clean-up operations like closing the DB resources to keep inside a finally block?									
	<ul style="list-style-type: none">Because finally block is always executed whether exceptions are raised in the try block or not and raised exceptions are caught in the catch block or not.By keeping the clean-up operations in finally block, you will ensure that those operations will be always executed irrespective of whether exception is occurred or not.									
23.	What is purpose of throw keyword?									
	<ul style="list-style-type: none">The Java throw keyword is used to explicitly throw an exception.We can throw either checked or unchecked exception in java by throw keyword.The throw keyword is mainly used to throw custom exception.									
24.	Write the Difference between throw and throws?									
	<table><tr><th>throw</th><th>Throws</th></tr><tr><td>throw keyword is used to throw an exception explicitly.</td><td>throws keyword is used for propagating or delegating the exception.</td></tr><tr><td>throw is followed by an instance of Exception class. throw new ArithmeticException(); or ArithmeticException e=new ArithmeticException(); throw e;</td><td>throws is followed by exception class names. throws ArithmeticException;</td></tr><tr><td>Throw is used within the method. e.g., public void m1() { Throw new ArithmeticException(); }</td><td>Throws is used with the method signature. e.g., public void m1() throws ClassNotFoundException { Class.forName("A"); }</td></tr></table>	throw	Throws	throw keyword is used to throw an exception explicitly.	throws keyword is used for propagating or delegating the exception.	throw is followed by an instance of Exception class. throw new ArithmeticException(); or ArithmeticException e=new ArithmeticException(); throw e;	throws is followed by exception class names. throws ArithmeticException;	Throw is used within the method. e.g., public void m1() { Throw new ArithmeticException(); }	Throws is used with the method signature. e.g., public void m1() throws ClassNotFoundException { Class.forName("A"); }	
throw	Throws									
throw keyword is used to throw an exception explicitly.	throws keyword is used for propagating or delegating the exception.									
throw is followed by an instance of Exception class. throw new ArithmeticException(); or ArithmeticException e=new ArithmeticException(); throw e;	throws is followed by exception class names. throws ArithmeticException;									
Throw is used within the method. e.g., public void m1() { Throw new ArithmeticException(); }	Throws is used with the method signature. e.g., public void m1() throws ClassNotFoundException { Class.forName("A"); }									

	<p>You can throw one exception at a time. You cannot throw multiple exceptions.</p> <p>e.g.,</p> <pre>public void m1() { Throw new ArithmeticException(); }</pre>	<p>you can handle multiple exceptions by declaring them using throws keyword.</p> <p>e.g.,</p> <pre>public void m1() throws IOException, ClassNotFoundException { Class.forName("A"); }</pre>
--	---	---

25. What is customized exception explain with example.

- In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as **user-defined** or **custom exceptions**.
- By the help of custom exception, you can have your own exception and message.
- E.g.,

```
public class AgeInvalidException extends Exception {

    public AgeInvalidException(String msg)
    {
        super(msg);
    }

}
```

26. Throws keyword with method overriding

Unchecked Exception		Checked Exception	
Parent Class Method	Child Class Method	Parent Class Method	Child Class Method
Does not throws any exception	We cannot throws any exception	Does not throws any exception	We cannot throws any exception
	Can throws any unchecked exception Partial exception/checked exception not allowed	throws Partial exception	Can throws same exception class or their child exception class but cannot throws parent exception class
throws unchecked exception	No need to throws exception	throws checked exception	No need to throws exception
	Can throws any unchecked exception Partial exception/checked exception not allowed		Can throws same exception class or their child exception class but cannot throws parent exception class

27.	Throws keyword with superclass and subclass constructor.			
	Unchecked Exception		Checked Exception	
	Parent Class Constructor	Child Class Constructor	Parent Class Constructor	Child Class Constructor
	Does not throws any exception	No need to throws exception Can throws any of the exception class (Partial also accepted)	Does not throws any exception	We cannot throws any exception
	throws unchecked exception	No need to throws exception Can throws any of the exception class (Partial also accepted)	throws checked exception	Can throws same exception class or their parent exception class but cannot throws child exception class
28.	Can we override a super class method which is throwing an unchecked exception with checked exception in the sub class?			
	<ul style="list-style-type: none"> No. If a super class method is throwing an unchecked exception, then it can be overridden in the sub class with same exception or any other unchecked exceptions but cannot be overridden with checked exceptions. 			
29.	What is the use of printStackTrace() method?			
	<ul style="list-style-type: none"> printStackTrace() method is used to print the detailed information about the exception occurred. 			
30.	When do you get NullPointerException?			
	<ul style="list-style-type: none"> Any reference variable other than Primitive, may have null value and if we call any method of java class with that null reference then in that case, we will get NullPointerException. When reference variable containing null, used to invoke member, we will get NullPointerException. <p><u>Example:</u></p> <pre> public class A { int x; } public class Test { public static void main(String[] args) { // Case 1: A a= null; //System.out.println(a.getClass()); //NullPointerException // Case 2: String s=null; //System.out.println(s.length()); //NullPointerException // Case 3: A aa[]= new A[2]; } } </pre>			

	<pre> aa[0]=null; //System.out.println(aa[0].getClass());//NullPointerException // Case 4: A a1= new A(); System.out.println(a1.x); //0 a1.x=100; System.out.println(a1.x); //100 // Case 5: try { Class.forName("com.cjc.A"); //Full Qualified Path Required } catch (ClassNotFoundException e) { e.printStackTrace(); } } </pre>
31.	<p>What is "ClassCastException"?</p> <ul style="list-style-type: none"> When u try to cast the classes out of hierarchy, you get ClassCastException. The java.lang.ClassCastException is one of the unchecked exception in Java. It can occur in our program when we tried to convert an object of one class type into an object of another class type. <pre> public class ClassCastExceptionDemo { public static void main(String[] args) { Object o = new String(); Integer i = (Integer) o; } } </pre> <ul style="list-style-type: none"> We all know that every class in java is a sub class of java.lang.Object class. String is also a subclass of Obeject class and Integer is also a subclass of Object class. In the above example, String object is created and it is automatically up casted to Object type. Further, this object is explicitly downcasted to Integer type. This causes ClassCastException, because, String object is not an Integer type.
32.	<p>What are methods in Exception class?</p> <ul style="list-style-type: none"> ⇒ public String getMessage() Returns a detailed message about the exception that has occurred. ⇒ public Throwable getCause() Returns the cause of the exception.

	<p>⇒ public String toString() Returns the name of the class concatenated with the result of getMessage().</p> <p>⇒ public void printStackTrace() Prints the result of toString() along with the stack trace, the error output stream.</p> <p>⇒ public StackTraceElement [] getStackTrace() Returns an array containing each element on the stack trace.</p> <p>⇒ public Throwable fillInStackTrace() Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.</p>
33.	What is final, finally, finalized?
	<ul style="list-style-type: none"> ➤ The final keyword in java is used to restrict the user. The java final keyword can be used in many contexts. Final can be used for variables, methods, class. <ul style="list-style-type: none"> ✓ Final variable once assigned can't be changed after. ✓ Final method can't be rewritten, can't be inherited. ✓ Final class can't be accessed by creating child of it. ➤ The finally is a block that always be executed either there is exception occur inside "try" block or there is no exception occur inside "try" block. In both situations, "finally" block code will be executed. ➤ The finalize() is called by the garbage collector on an object when garbage collection determines that there are no more references to the object, it is used to perform clean-up activity.
34.	Can we Create Custom Checked Exception in Java?
	<ul style="list-style-type: none"> • Yes. we can create Custom Checked Exception in java. • The only way of doing it is to extend Exception (or a subclass thereof) for a checked exception, and extending RuntimeException (or a subclass thereof) for an unchecked exception.
35.	How to decide that the custom exceptions are checked or unchecked?
	<ul style="list-style-type: none"> • If you extend Exception then it is "checked", i.e., if you throw it, it must be caught or declared in the method signature. • Unchecked exceptions extend RuntimeException and do not need to be declared or caught.
36.	New Exception related feature in JDK 1.7 version?
	<ol style="list-style-type: none"> 1. Multiple exceptions handling in only one catch block. 2. Finally out of scope. Try-Catch with resources is new feature.
37.	Which type of statements can be written in try with resources?
	<ul style="list-style-type: none"> • We can write those classes/interfaces which have implemented/extended Autoclosable interface only.

	<ul style="list-style-type: none"> For user-defined class/interface, we need to implement/extend Autoclosable interface explicitly. So that we can write that class in try with resources context.
--	---

38. Difference between ClassCastException and NoClassDefFound Error?

ClassCastException	NoClassDefFound Error
It occurs when we are trying to type cast parent object to Child.	It occurs when it is unable to find required .class file at runtime.
It can be avoided by correcting type while type casting, by changing in code.	It can be avoided by providing required .class file.
It is child class of Runtime Exception.	It is child class Error.

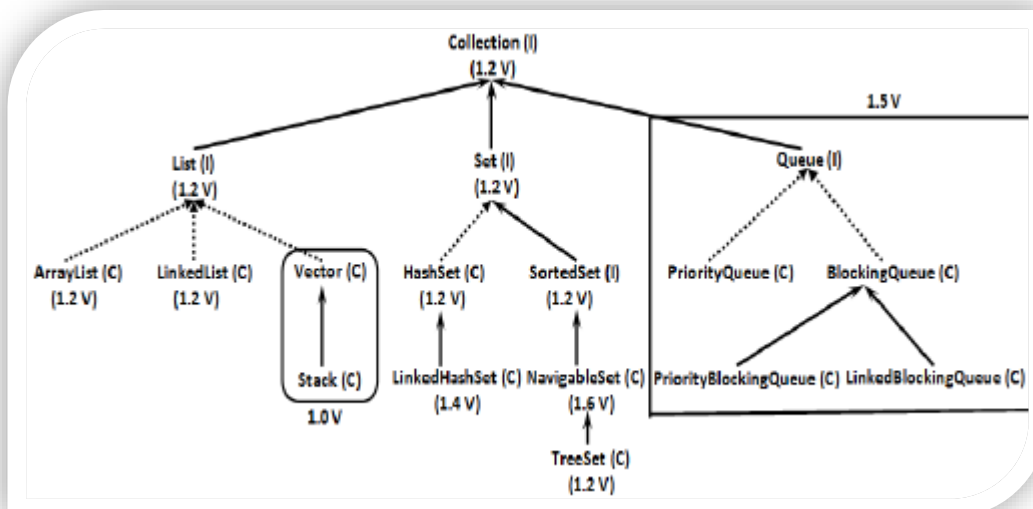
39. How do you handled Exception in your project?

<ul style="list-style-type: none"> We created separate package for Exception classes needed in project where handling code was written, then at a time of exception handling scenario in project used throws keyword.
--

Collection Framework

1. What do you mean by collection in java? Draw the hierarchy of collection

<ul style="list-style-type: none"> The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects. Collection represents group of objects: Homogeneous or Heterogeneous. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion. Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet)
--



2.	What is List?															
	<ul style="list-style-type: none">List in Java provides the facility to maintain the ordered collection.It contains the index-based methods to insert, update, delete and search the elements.It can have the duplicate elements also. We can also store the null elements in the list.															
3.	What is Set?															
	<ul style="list-style-type: none">A Set is a Collection that cannot contain duplicate elements.It models the mathematical set abstraction.The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.															
4.	Difference between List and Set															
	<table><tr><th>List</th><th>Set</th></tr><tr><td>List is an ordered collection it maintains the insertion order, which means upon displaying the list content it will display the elements in the same order in which they got inserted into the list.</td><td>Set is an unordered collection; it doesn't maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet.</td></tr><tr><td>List allows duplicate elements.</td><td>Set doesn't allow duplicate elements.</td></tr><tr><td>List implementations: ArrayList, LinkedList etc.</td><td>Set implementations: HashSet, LinkedHashSet, TreeSet etc.</td></tr><tr><td>List allows any number of null values.</td><td>Set can have only a single null value at most.</td></tr><tr><td>ListIterator can be used to traverse a List in both the directions (forward and backward).</td><td>ListIterator cannot be used to traverse a Set. We can use Iterator (It works with List too) to traverse a Set.</td></tr><tr><td>List interface has one legacy class called Vector</td><td>Set interface does not have any legacy class.</td></tr></table>	List	Set	List is an ordered collection it maintains the insertion order, which means upon displaying the list content it will display the elements in the same order in which they got inserted into the list.	Set is an unordered collection; it doesn't maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet.	List allows duplicate elements.	Set doesn't allow duplicate elements.	List implementations: ArrayList, LinkedList etc.	Set implementations: HashSet, LinkedHashSet, TreeSet etc.	List allows any number of null values.	Set can have only a single null value at most.	ListIterator can be used to traverse a List in both the directions (forward and backward).	ListIterator cannot be used to traverse a Set. We can use Iterator (It works with List too) to traverse a Set.	List interface has one legacy class called Vector	Set interface does not have any legacy class.	
List	Set															
List is an ordered collection it maintains the insertion order, which means upon displaying the list content it will display the elements in the same order in which they got inserted into the list.	Set is an unordered collection; it doesn't maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet.															
List allows duplicate elements.	Set doesn't allow duplicate elements.															
List implementations: ArrayList, LinkedList etc.	Set implementations: HashSet, LinkedHashSet, TreeSet etc.															
List allows any number of null values.	Set can have only a single null value at most.															
ListIterator can be used to traverse a List in both the directions (forward and backward).	ListIterator cannot be used to traverse a Set. We can use Iterator (It works with List too) to traverse a Set.															
List interface has one legacy class called Vector	Set interface does not have any legacy class.															
5.	Difference between Array and ArrayList															
	<table><tr><th>Array</th><th>ArrayList</th></tr><tr><td>An array is basic functionality provided by Java; therefore, array members are accessed using [].</td><td>ArrayList is class of collection framework in Java. ArrayList has a set of methods to access elements and modify them.</td></tr><tr><td>Array is a fixed length data structure. We cannot change length of array once created.</td><td>ArrayList is a variable length Collection class. Length of ArrayList can be changed.</td></tr><tr><td>It is mandatory to provide the size of an array while initializing it directly or indirectly.</td><td>We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size.</td></tr><tr><td>array can store both primitives and objects in Java.</td><td>We cannot store primitives in ArrayList, it can only store objects.</td></tr><tr><td>It performs fast in comparison to ArrayList because of fixed size.</td><td>The resize operation in ArrayList slows down the performance.</td></tr></table>	Array	ArrayList	An array is basic functionality provided by Java; therefore, array members are accessed using [].	ArrayList is class of collection framework in Java. ArrayList has a set of methods to access elements and modify them.	Array is a fixed length data structure. We cannot change length of array once created.	ArrayList is a variable length Collection class. Length of ArrayList can be changed.	It is mandatory to provide the size of an array while initializing it directly or indirectly.	We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size.	array can store both primitives and objects in Java.	We cannot store primitives in ArrayList, it can only store objects.	It performs fast in comparison to ArrayList because of fixed size.	The resize operation in ArrayList slows down the performance.			
Array	ArrayList															
An array is basic functionality provided by Java; therefore, array members are accessed using [].	ArrayList is class of collection framework in Java. ArrayList has a set of methods to access elements and modify them.															
Array is a fixed length data structure. We cannot change length of array once created.	ArrayList is a variable length Collection class. Length of ArrayList can be changed.															
It is mandatory to provide the size of an array while initializing it directly or indirectly.	We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size.															
array can store both primitives and objects in Java.	We cannot store primitives in ArrayList, it can only store objects.															
It performs fast in comparison to ArrayList because of fixed size.	The resize operation in ArrayList slows down the performance.															

	We use for loop or for each loop to iterate over an array	We use an iterator to iterate over ArrayList.																					
6.	Why ArrayList is better than Array?																						
	<ul style="list-style-type: none">ArrayList is a variable-length data structure. It can be resized itself when needed.We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size.																						
7.	What is difference between Collections and Arrays?																						
	<table><tr><th>Arrays</th><th>Collections</th></tr><tr><td>Arrays are Fixed in Size that is Once we created an Array there is No Chance of Increasing OR Decreasing Size based on Our Requirement.</td><td>Collections are Growable in Nature. That is based on Our Requirement we can Increase OR Decrease the Size.</td></tr><tr><td>With Respect to Memory Arrays are Not Recommended to Use.</td><td>With Respect to Memory Collections are Recommended to Use.</td></tr><tr><td>With Respect to Performance Arrays are Recommended to Use.</td><td>With Respect to Performance Collections are not Recommended to Use.</td></tr><tr><td>Arrays can Hold Only Homogeneous Data Elements.</td><td>Collections can Hold Both Homogeneous and Heterogeneous Elements.</td></tr><tr><td>Arrays can hold both object and primitive type data.</td><td>Collections can hold only object types but not the primitive type of data.</td></tr><tr><td>Arrays Concept is Not implemented based on Some Standard Data Structure. Hence Readymade Method Support is Not Available.</td><td>For every Collection class underlying Data Structure is Available Hence Readymade Method Support is Available for Every Requirement</td></tr></table>	Arrays	Collections	Arrays are Fixed in Size that is Once we created an Array there is No Chance of Increasing OR Decreasing Size based on Our Requirement.	Collections are Growable in Nature. That is based on Our Requirement we can Increase OR Decrease the Size.	With Respect to Memory Arrays are Not Recommended to Use.	With Respect to Memory Collections are Recommended to Use.	With Respect to Performance Arrays are Recommended to Use.	With Respect to Performance Collections are not Recommended to Use.	Arrays can Hold Only Homogeneous Data Elements.	Collections can Hold Both Homogeneous and Heterogeneous Elements.	Arrays can hold both object and primitive type data.	Collections can hold only object types but not the primitive type of data.	Arrays Concept is Not implemented based on Some Standard Data Structure. Hence Readymade Method Support is Not Available.	For every Collection class underlying Data Structure is Available Hence Readymade Method Support is Available for Every Requirement								
Arrays	Collections																						
Arrays are Fixed in Size that is Once we created an Array there is No Chance of Increasing OR Decreasing Size based on Our Requirement.	Collections are Growable in Nature. That is based on Our Requirement we can Increase OR Decrease the Size.																						
With Respect to Memory Arrays are Not Recommended to Use.	With Respect to Memory Collections are Recommended to Use.																						
With Respect to Performance Arrays are Recommended to Use.	With Respect to Performance Collections are not Recommended to Use.																						
Arrays can Hold Only Homogeneous Data Elements.	Collections can Hold Both Homogeneous and Heterogeneous Elements.																						
Arrays can hold both object and primitive type data.	Collections can hold only object types but not the primitive type of data.																						
Arrays Concept is Not implemented based on Some Standard Data Structure. Hence Readymade Method Support is Not Available.	For every Collection class underlying Data Structure is Available Hence Readymade Method Support is Available for Every Requirement																						
8.	Write the difference between List and ArrayList?																						
	<table><tr><th>Key</th><th>List</th><th>ArrayList</th></tr><tr><td>General</td><td>It is an interface.</td><td>It is a class.</td></tr><tr><td>Work</td><td>It creates a list of objects that can be accessed by the individual index number.</td><td>It creates a dynamic array that can be expanded when needed.</td></tr><tr><td>Implementation</td><td>List <data-type> list1= new ArrayList();</td><td>ArrayList myList = new ArrayList();</td></tr><tr><td>Extend/Implement</td><td>It extends the collection framework.</td><td>It extends AbstractList class and implements the List interface.</td></tr><tr><td>Performance</td><td>It provides faster manipulation of objects.</td><td>It provides slow manipulation on objects compared to List.</td></tr><tr><td>Instantiation</td><td>It cannot be instantiated.</td><td>It can be instantiated.</td></tr></table>	Key	List	ArrayList	General	It is an interface.	It is a class.	Work	It creates a list of objects that can be accessed by the individual index number.	It creates a dynamic array that can be expanded when needed.	Implementation	List <data-type> list1= new ArrayList();	ArrayList myList = new ArrayList();	Extend/Implement	It extends the collection framework.	It extends AbstractList class and implements the List interface.	Performance	It provides faster manipulation of objects.	It provides slow manipulation on objects compared to List.	Instantiation	It cannot be instantiated.	It can be instantiated.	
Key	List	ArrayList																					
General	It is an interface.	It is a class.																					
Work	It creates a list of objects that can be accessed by the individual index number.	It creates a dynamic array that can be expanded when needed.																					
Implementation	List <data-type> list1= new ArrayList();	ArrayList myList = new ArrayList();																					
Extend/Implement	It extends the collection framework.	It extends AbstractList class and implements the List interface.																					
Performance	It provides faster manipulation of objects.	It provides slow manipulation on objects compared to List.																					
Instantiation	It cannot be instantiated.	It can be instantiated.																					

9.	<p>What is generic in java? Explain with example.</p> <ul style="list-style-type: none"> Java Generics is a set of related methods or a set of similar types. Generics allow types Integer, String, or even user-defined types to be passed as a parameter to classes, methods, or interfaces. Generics are mostly used by classes like HashSet or HashMap. Generic feature comes from jdk 1.5 version. <p>Example:</p> <pre>List<Integer> l=new ArrayList<>(); //main purpose is to provide type safety l.add(5); l.add(20); l.add(30); //l.add("abc"); int x=l.get(0); //no need to typecast System.out.println(x); //String s=(String)l.get(0); //cannot cast from integer to string-->it will overcome class cast exception problem</pre>
10.	<p>Explain advantage of generic?</p> <ul style="list-style-type: none"> There are mainly 3 advantages of generics. They are as follows: <ol style="list-style-type: none"> <u>Type-safety:</u> We can hold only a single type of objects in generics. It doesn't allow to store other objects. Without Generics, we can store any type of objects. E.g.:- <pre>List list = new ArrayList(); list.add(10); list.add("10");</pre> With Generics, it is required to specify the type of object we need to store. <pre>List<Integer> list = new ArrayList<Integer>(); list.add(10); list.add("10");// compile-time error</pre> <u>Type casting is not required:</u> There is no need to typecast the object. Before Generics, we need to type cast. E.g.:- <pre>List list = new ArrayList(); list.add("hello"); String s = (String) list.get(0);//typecasting</pre> After Generics, we don't need to typecast the object. <pre>List<String> list = new ArrayList<String>(); list.add("hello"); String s = list.get(0);</pre> <u>Compile-Time Checking:</u> It is checked at compile time so problem will not occur at runtime.

	The good programming strategy says it is far better to handle the problem at compile time than runtime. E.g: List<String> list = new ArrayList<String>(); list.add("hello"); list.add(32); //Compile Time Error							
11.	Is Iterator a Class or Interface? What is its use?							
	Iterator is an interface which is used to step through the elements of a Collection.							
12.	What is difference between Collection(I) and Collections(C)?							
	<table><tr><th>Collection(I)</th><th>Collections(C)</th></tr><tr><td>Collection is an Interface which can be used to Represent a Group of Individual Objects as a Single Entity.</td><td>Whereas Collections is an Utility Class Present in java.util Package to Define Several Utility Methods for Collection Objects.</td></tr><tr><td>It is the root interface of the Collection framework.</td><td>It is a utility class.</td></tr></table>	Collection(I)	Collections(C)	Collection is an Interface which can be used to Represent a Group of Individual Objects as a Single Entity.	Whereas Collections is an Utility Class Present in java.util Package to Define Several Utility Methods for Collection Objects.	It is the root interface of the Collection framework.	It is a utility class.	
Collection(I)	Collections(C)							
Collection is an Interface which can be used to Represent a Group of Individual Objects as a Single Entity.	Whereas Collections is an Utility Class Present in java.util Package to Define Several Utility Methods for Collection Objects.							
It is the root interface of the Collection framework.	It is a utility class.							
13.	What are legacy classes and interfaces present in Collections framework?							
	<ul style="list-style-type: none">• Early version of java did not include the Collections framework. In the early version of Java, we have several classes and interfaces which allow us to store objects.• After adding the Collection framework in JSE 1.2, for supporting the collections framework, these classes were re-engineered.• So, classes and interfaces that formed the collections framework in the older version of Java are known as Legacy classes.• For supporting generic in JDK5, these classes were re-engineered.• All the legacy classes are synchronized.• The java.util package defines the following legacy classes:<ul style="list-style-type: none">1. HashTable2. Stack3. Dictionary4. Properties5. Vector• There is only one legacy interface called Enumeration.							
14.	What are the basic interfaces of Java Collections Framework?							
	<ul style="list-style-type: none">• The most basic interfaces that reside in the Java Collections Framework are:<ul style="list-style-type: none">1. Collection, which represents a group of objects known as its elements.2. Set, which is an unordered collection that cannot contain duplicate elements.3. List, which is an ordered collection and can contain duplicate elements.4. Map, which is an object that maps keys to values and cannot contain duplicate keys.							

15.	What is an Iterator?													
	<ul style="list-style-type: none">• In Java, Iterator is an interface available in Collection framework in java.util package.• It is a Java Cursor used to iterate a collection of objects.• Iterators are used in Collection framework in Java to retrieve elements one by one.• Iterator object can be created by calling iterator() method present in Collection interface.													
16.	Why Iterator is called as Universal Java Cursor?													
	<ul style="list-style-type: none">• We can apply Iterator to any Collection object or classes that's why it is called as universal cursor.• By using Iterator, we can perform both read and remove operations.• Compare to Enumeration interface, Iterator method names are simple and easy to use• Iterator must be used whenever we want to enumerate elements in all Collection framework implemented interfaces like Set, List, Queue, Deque and also in all implemented classes of Map interface.• Iterator is the only cursor available for entire collection framework.													
17.	What differences exist between Iterator and ListIterator?													
	<table><tr><th>Iterator</th><th>ListIterator</th></tr><tr><td>An Iterator can be used to traverse the Set, List and Map collections.</td><td>The ListIterator can be used to iterate over only Lists.</td></tr><tr><td>The Iterator can traverse a collection only in forward direction.</td><td>The ListIterator can traverse a List in both directions.</td></tr><tr><td>Using Iterator, you cannot add any element to a collection.</td><td>Using ListIterator you can add elements to a collection.</td></tr><tr><td>Using Iterator, you cannot remove an element in a collection.</td><td>You can remove an element from a collection using ListIterator.</td></tr><tr><td>Methods of Iterator:<ul style="list-style-type: none">• hasNext()• next()• remove()</td><td>Methods of ListIterator:<ul style="list-style-type: none">• add(E e)• hasNext()• hasPreviousus()• next()• remove()• previous()• previousIndex()• set(E e)• nextIndex()</td></tr></table>	Iterator	ListIterator	An Iterator can be used to traverse the Set, List and Map collections.	The ListIterator can be used to iterate over only Lists.	The Iterator can traverse a collection only in forward direction.	The ListIterator can traverse a List in both directions.	Using Iterator, you cannot add any element to a collection.	Using ListIterator you can add elements to a collection.	Using Iterator, you cannot remove an element in a collection.	You can remove an element from a collection using ListIterator.	Methods of Iterator: <ul style="list-style-type: none">• hasNext()• next()• remove()	Methods of ListIterator: <ul style="list-style-type: none">• add(E e)• hasNext()• hasPreviousus()• next()• remove()• previous()• previousIndex()• set(E e)• nextIndex()	
Iterator	ListIterator													
An Iterator can be used to traverse the Set, List and Map collections.	The ListIterator can be used to iterate over only Lists.													
The Iterator can traverse a collection only in forward direction.	The ListIterator can traverse a List in both directions.													
Using Iterator, you cannot add any element to a collection.	Using ListIterator you can add elements to a collection.													
Using Iterator, you cannot remove an element in a collection.	You can remove an element from a collection using ListIterator.													
Methods of Iterator: <ul style="list-style-type: none">• hasNext()• next()• remove()	Methods of ListIterator: <ul style="list-style-type: none">• add(E e)• hasNext()• hasPreviousus()• next()• remove()• previous()• previousIndex()• set(E e)• nextIndex()													
	Difference Between Iterable and Iterator.													
	<table><tr><th>Iterable</th><th>Iterator</th></tr><tr><td>Iterable is an interface</td><td>Iterator is an interface</td></tr><tr><td>Belongs to java.lang package</td><td>Belongs to java.util package</td></tr><tr><td>Provides one single abstract method called iterator()</td><td>Provides two abstract methods called hasNext() and next()</td></tr></table>	Iterable	Iterator	Iterable is an interface	Iterator is an interface	Belongs to java.lang package	Belongs to java.util package	Provides one single abstract method called iterator()	Provides two abstract methods called hasNext() and next()					
Iterable	Iterator													
Iterable is an interface	Iterator is an interface													
Belongs to java.lang package	Belongs to java.util package													
Provides one single abstract method called iterator()	Provides two abstract methods called hasNext() and next()													

	It is a representation of a series of elements that can be traversed	It represents the object with iteration state
18.	How to make ArrayList synchronized?	
	<pre>List<String> l=new ArrayList<>(); l.add("aaa"); l.add("bbb"); l.add("ccc"); collections.synchronizedList(l); //Synchronized for(String nm:l) { System.out.println(nm); }</pre>	
19.	How to make list as read only?	
	<pre>List<String> l=new ArrayList<>(); l.add("aaa"); l.add("bbb"); collections.unmodifiableList(l); //cannot modify l.add("ccc"); //Runtime error</pre>	
20.	Which Collection object you have used in your Project?	
	<p>⇒ List - ArrayList for getting lists of objects from Database.</p> <p>⇒ Set – HashSet for mapping POJOs using hibernate. (One-to-Many & Many-to-One)</p>	
21.	How Arraylist works?	
	<p>⇒ When we create object of Arraylist, it create Arraylist instance with default capacity 10.</p> <p>⇒ Arraylist capacity increases with formula –</p> <p style="text-align: center;">New Capacity = ((3/2) x Old Capacity)+1</p> <p>⇒ When Arraylist increments with new capacity then data from old Arraylist is copied into new instance and old instance is destroyed.</p> <p>⇒ When we add or delete data into the Arraylist then multiple data shift operations are performed.</p> <p>⇒ Arraylist follows Indexing.</p>	
22.	Why Arraylist is fast for retrieval operation?	
	<ul style="list-style-type: none">It implements RandomAccess Interface, hence Arraylist is fast for retrieval operation.	
23.	Which API is provided by Java for operations on set of objects?	
	<ul style="list-style-type: none">Java provides a Collection API which provides many useful methods which can be applied on a set of objects.	

	<ul style="list-style-type: none"> Some of the important classes provided by Collection API include ArrayList, HashMap, TreeSet and TreeMap.
--	---

24. Difference between ArrayList and LinkedList

Parameter	ArrayList	LinkedList
Internal data structure	It uses dynamic array to store elements internally	It uses doubly Linked List to store elements internally
Manipulation	If We need to insert or delete element in ArrayList, it may take $O(n)$, as it internally uses array and we may have to shift elements in case of insertion or deletion	If We need to insert or delete element in LinkedList, it will take $O(1)$, as it internally uses doubly LinkedList
Search	Search is faster in ArrayList as uses array internally which is index based. So here time complexity is $O(1)$	Search is slower in LinkedList as uses doubly Linked List internally So here time complexity is $O(n)$
Interfaces	ArrayList implements List interface only, so it can be used as List only	LinkedList implements List, Deque interfaces, so it can be used as List, Stack or Queue

25. When to use ArrayList or LinkedList?

<ul style="list-style-type: none"> It actually depends on our need. <ul style="list-style-type: none"> ⇒ If we have more insertion or deletion then we should use LinkedList. ⇒ If we have less insertion or deletion and more search operations then we should use ArrayList.
--

26. How will you reverse a List?

<ul style="list-style-type: none"> ArrayList can be reversed using the reverse() method of the Collections class.
--

```

ReverseList.java
1 package com.collectionframework.list.reverselist;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.List;
6
7 public class ReverseList {
8
9     public static void main(String[] args) {
10         List<String> myList = new ArrayList<String>();
11         myList.add("AWS");
12         myList.add("Java");
13         myList.add("Python");
14         myList.add("Blockchain");
15         System.out.println("Before Reversing");
16         System.out.println(myList.toString());
17         Collections.reverse(myList);
18         System.out.println("After Reversing");
19         System.out.println(myList);
20     }
21 }

```

Problems Javadoc Declaration Console

```

<terminated> ReverseList [Java Application] C:\Users\hp\p2\pool\plugins\org.eclipse
Before Reversing
[AWS, Java, Python, Blockchain]
After Reversing
[Blockchain, Python, Java, AWS]

```

27.	Differences Between HashSet, LinkedHashSet, and TreeSet			
	Features	HashSet	LinkedHashSet	TreeSet
	Internal Working	HashSet internally uses HashMap for storing objects	LinkedHashSet uses LinkedHashMap internally to store objects	TreeSet uses TreeMap internally to store objects
	When To Use	If you don't want to maintain insertion order but want to store unique objects	If you want to maintain the insertion order of elements then you can use LinkedHashSet	If you want to sort the elements according to some Comparator then use TreeSet
	Order	HashSet does not maintain insertion order	LinkedHashSet maintains the insertion order of objects	While TreeSet orders the elements according to supplied Comparator. By default, objects will be placed according to their natural ascending order.
	Complexity of Operations	HashSet gives O(1) complexity for insertion, removing, and retrieving objects	LinkedHashSet gives insertion, removing, and retrieving operations performance in order O(1).	While TreeSet gives the performance of order O(log(n)) for insertion, removing, and retrieving operations.
	Performance	The performance of HashSet is better when compared to LinkedHashSet and TreeSet.	The performance of LinkedHashSet is slower than TreeSet. It is almost similar to HashSet but slower because LinkedHashSet internally maintains LinkedList to maintain the insertion order of elements	TreeSet performance is better than LinkedHashSet except for insertion and removal operations because it has to sort the elements after each insertion and removal operation.

	Compare	HashSet uses equals() and hashCode() methods to compare the objects	LinkedHashSet uses equals() and hashCode() methods to compare it's objects	TreeSet uses compare() and compareTo() methods to compare the objects
	Null Elements	HashSet allows only one null value.	LinkedHashSet allows only one null value.	TreeSet does not permit null value. If you insert null value into TreeSet, it will throw NullPointerException.
	Syntax	HashSet obj = new HashSet();	LinkedHashSet obj = new LinkedHashSet();	TreeSet obj = new TreeSet();

28. Why set doesn't allow duplicates?

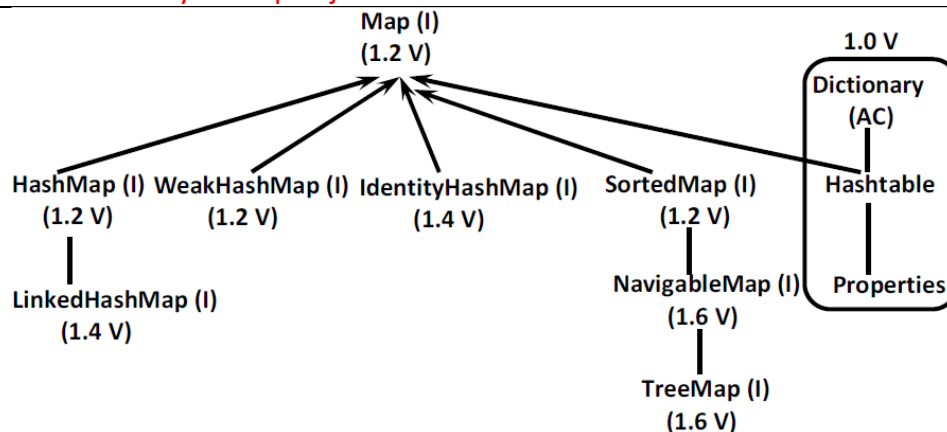
- **Set internally uses HashMap.**
- HashMap object is created in every Set implemented class.
- Here HashMap stores key as all inserted elements and value as a dummy object created with new keyword.
- As HashMap doesn't accept duplicate keys, so set don't allow duplicate values.

29. How Linkedlist works? (Why insertion & deletion is fast in Linkedlist?)

- When we create an object of Linkedlist and add an element to it then it stores element as a node in which previous & next node address is also stored.
- Node format = || prev. node addr. | (value) | next node addr. ||
- Due to previous & next node address is stored, hence while updation or insertion & deletion operation data shift operation need not to perform and it makes Linkedlist fast.

Map

1. Draw the hierarchy of map in java?



2.	Write the difference between HashMap and Hashtable.																			
	<table><tr><th>HashMap</th><th>Hashtable(Legacy Class)</th></tr><tr><td>Not synchronized (Not Threadsafe) Every method present in HashMap is not synchronized.</td><td>Synchronized (Threadsafe) Every method present in Hashtable is synchronized.</td></tr><tr><td>At a time, Multiple Threads are allowed to Operate on HashMap Object simultaneously and Hence it is Not ThreadSafe.</td><td>At a time, Only One Thread is allowed to Operate on the Hashtable Object and Hence it is Thread Safe.</td></tr><tr><td>Relatively Performance is High because threads are not required to wait to operate on HashMap object.</td><td>Relatively Performance is low because threads are required to wait to operate on Hashtable object.</td></tr><tr><td>null is allowed for Both Key and Value.</td><td>null is not allowed for Both Keys and Values otherwise we will get NullPointerException.</td></tr><tr><td>Retrieval of elements can be achieved through Iterator</td><td>Retrieval of elements can be achieved through Enumeration (Legacy Interface)</td></tr><tr><td>Introduced in jdk 1.2 version and it is not legacy.</td><td>Introduced in jdk 1.0 version and it is legacy.</td></tr></table>	HashMap	Hashtable(Legacy Class)	Not synchronized (Not Threadsafe) Every method present in HashMap is not synchronized.	Synchronized (Threadsafe) Every method present in Hashtable is synchronized.	At a time, Multiple Threads are allowed to Operate on HashMap Object simultaneously and Hence it is Not ThreadSafe.	At a time, Only One Thread is allowed to Operate on the Hashtable Object and Hence it is Thread Safe.	Relatively Performance is High because threads are not required to wait to operate on HashMap object.	Relatively Performance is low because threads are required to wait to operate on Hashtable object.	null is allowed for Both Key and Value.	null is not allowed for Both Keys and Values otherwise we will get NullPointerException.	Retrieval of elements can be achieved through Iterator	Retrieval of elements can be achieved through Enumeration (Legacy Interface)	Introduced in jdk 1.2 version and it is not legacy.	Introduced in jdk 1.0 version and it is legacy.					
HashMap	Hashtable(Legacy Class)																			
Not synchronized (Not Threadsafe) Every method present in HashMap is not synchronized.	Synchronized (Threadsafe) Every method present in Hashtable is synchronized.																			
At a time, Multiple Threads are allowed to Operate on HashMap Object simultaneously and Hence it is Not ThreadSafe.	At a time, Only One Thread is allowed to Operate on the Hashtable Object and Hence it is Thread Safe.																			
Relatively Performance is High because threads are not required to wait to operate on HashMap object.	Relatively Performance is low because threads are required to wait to operate on Hashtable object.																			
null is allowed for Both Key and Value.	null is not allowed for Both Keys and Values otherwise we will get NullPointerException.																			
Retrieval of elements can be achieved through Iterator	Retrieval of elements can be achieved through Enumeration (Legacy Interface)																			
Introduced in jdk 1.2 version and it is not legacy.	Introduced in jdk 1.0 version and it is legacy.																			
	Difference Between HashMap and HashSet																			
	<table><tr><th>Parameter</th><th>HashMap</th><th>HashSet</th></tr><tr><td>Interface</td><td>This is core difference among them. HashMap implements Map interface</td><td>HashSet implement Set interface</td></tr><tr><td>Method for storing data</td><td>It stores data in a form of key->value pair. So, it uses put(key,value) method for storing data</td><td>It uses add(value) method for storing data</td></tr><tr><td>Duplicates</td><td>HashMap allows duplicate value but not duplicate keys</td><td>HashSet does not allow duplicate values.</td></tr><tr><td>Performance</td><td>It is faster than hashset as values are stored with unique keys</td><td>It is slower than HashMap</td></tr><tr><td>HashCode Calculation</td><td>In HashMap hashcode value is calculated using key object</td><td>In this, hashcode is calculated on the basis of value object. Hashcode can be same for two value object so we have to implement equals() method. If equals() method return false then two objects are different.</td></tr></table>	Parameter	HashMap	HashSet	Interface	This is core difference among them. HashMap implements Map interface	HashSet implement Set interface	Method for storing data	It stores data in a form of key->value pair. So, it uses put(key,value) method for storing data	It uses add(value) method for storing data	Duplicates	HashMap allows duplicate value but not duplicate keys	HashSet does not allow duplicate values.	Performance	It is faster than hashset as values are stored with unique keys	It is slower than HashMap	HashCode Calculation	In HashMap hashcode value is calculated using key object	In this, hashcode is calculated on the basis of value object. Hashcode can be same for two value object so we have to implement equals() method. If equals() method return false then two objects are different.	
Parameter	HashMap	HashSet																		
Interface	This is core difference among them. HashMap implements Map interface	HashSet implement Set interface																		
Method for storing data	It stores data in a form of key->value pair. So, it uses put(key,value) method for storing data	It uses add(value) method for storing data																		
Duplicates	HashMap allows duplicate value but not duplicate keys	HashSet does not allow duplicate values.																		
Performance	It is faster than hashset as values are stored with unique keys	It is slower than HashMap																		
HashCode Calculation	In HashMap hashcode value is calculated using key object	In this, hashcode is calculated on the basis of value object. Hashcode can be same for two value object so we have to implement equals() method. If equals() method return false then two objects are different.																		

3.	How to get Synchronized Version of HashMap? <ul style="list-style-type: none"> By default, HashMap is non-Synchronized. But we can get Synchronized Version of HashMap by using synchronizedMap() of Collections Class. E.g., <pre>HashMap m=new HashMap(); Map m1=Collections.synchronizedMap(m);</pre> In above example, m is not synchronized but m1 is synchronized.								
4.	Write the difference between HashMap and LinkedHashMap. <table> <tr> <th>HashMap</th><th>LinkedHashMap</th></tr> <tr> <td>The Underlying Data Structure is Hashtable.</td><td>The Underlying Data Structure is Combination of Hashtable and LinkedList.</td></tr> <tr> <td>Insertion order is Not Preserved and it is based on hashCode of keys.</td><td>Insertion order is Preserved.</td></tr> <tr> <td>Introduced in jdk 1.2 version</td><td>Introduced in jdk 1.4 version</td></tr> </table>	HashMap	LinkedHashMap	The Underlying Data Structure is Hashtable.	The Underlying Data Structure is Combination of Hashtable and LinkedList.	Insertion order is Not Preserved and it is based on hashCode of keys.	Insertion order is Preserved.	Introduced in jdk 1.2 version	Introduced in jdk 1.4 version
HashMap	LinkedHashMap								
The Underlying Data Structure is Hashtable.	The Underlying Data Structure is Combination of Hashtable and LinkedList.								
Insertion order is Not Preserved and it is based on hashCode of keys.	Insertion order is Preserved.								
Introduced in jdk 1.2 version	Introduced in jdk 1.4 version								
5.	Write the difference between HashMap and IdentityHashMap. <table> <tr> <th>HashMap</th><th>IdentityHashMap</th></tr> <tr> <td>HashMap differentiate key by value. In HashMap JVM will Use .equals() to Identify Duplicate Keys, which is Meant for Content Comparison.</td><td>IdentityHashMap differentiate key by reference. In IdentityHashMap JVM will Use == Operator to Identify Duplicate Keys, which is Meant for Reference Comparison.</td></tr> <tr> <td> <pre>Map m=new HashMap(); Integer i1=new Integer(5); Integer i2=new Integer(5); m.put(i1, "Java"); m.put(i2, "CJC"); System.out.println(m);</pre> //5=CJC </td><td> <pre>Map m=new IdentityHashMap(); Integer i1=new Integer(5); Integer i2=new Integer(5); m.put(i1, "Java"); m.put(i2, "CJC"); System.out.println(m);</pre> //5=Java, 5=CJC </td></tr> </table>	HashMap	IdentityHashMap	HashMap differentiate key by value. In HashMap JVM will Use .equals() to Identify Duplicate Keys, which is Meant for Content Comparison .	IdentityHashMap differentiate key by reference. In IdentityHashMap JVM will Use == Operator to Identify Duplicate Keys, which is Meant for Reference Comparison .	<pre>Map m=new HashMap(); Integer i1=new Integer(5); Integer i2=new Integer(5); m.put(i1, "Java"); m.put(i2, "CJC"); System.out.println(m);</pre> //5=CJC	<pre>Map m=new IdentityHashMap(); Integer i1=new Integer(5); Integer i2=new Integer(5); m.put(i1, "Java"); m.put(i2, "CJC"); System.out.println(m);</pre> //5=Java, 5=CJC		
HashMap	IdentityHashMap								
HashMap differentiate key by value. In HashMap JVM will Use .equals() to Identify Duplicate Keys, which is Meant for Content Comparison .	IdentityHashMap differentiate key by reference. In IdentityHashMap JVM will Use == Operator to Identify Duplicate Keys, which is Meant for Reference Comparison .								
<pre>Map m=new HashMap(); Integer i1=new Integer(5); Integer i2=new Integer(5); m.put(i1, "Java"); m.put(i2, "CJC"); System.out.println(m);</pre> //5=CJC	<pre>Map m=new IdentityHashMap(); Integer i1=new Integer(5); Integer i2=new Integer(5); m.put(i1, "Java"); m.put(i2, "CJC"); System.out.println(m);</pre> //5=Java, 5=CJC								
6.	Write the difference between ArrayList and Vector <table> <tr> <th>ArrayList</th><th>Vector</th></tr> <tr> <td>Not synchronized (Not ThreadSafe) Every method present in ArrayList is not synchronized.</td><td>Synchronized (ThreadSafe) Every method present in Vector is synchronized.</td></tr> <tr> <td>At a time, Multiple Threads are allowed to Operate on ArrayList Object simultaneously and Hence it is Not ThreadSafe.</td><td>At a time, Only One Thread is allowed to Operate on the Vector Object and Hence it is Thread Safe.</td></tr> </table>	ArrayList	Vector	Not synchronized (Not ThreadSafe) Every method present in ArrayList is not synchronized.	Synchronized (ThreadSafe) Every method present in Vector is synchronized.	At a time, Multiple Threads are allowed to Operate on ArrayList Object simultaneously and Hence it is Not ThreadSafe.	At a time, Only One Thread is allowed to Operate on the Vector Object and Hence it is Thread Safe.		
ArrayList	Vector								
Not synchronized (Not ThreadSafe) Every method present in ArrayList is not synchronized.	Synchronized (ThreadSafe) Every method present in Vector is synchronized.								
At a time, Multiple Threads are allowed to Operate on ArrayList Object simultaneously and Hence it is Not ThreadSafe.	At a time, Only One Thread is allowed to Operate on the Vector Object and Hence it is Thread Safe.								

	Relatively Performance is High because threads are not required to wait to operate on ArrayList object.	Relatively Performance is low because threads are required to wait to operate on Vector object.
	Default capacity=10 New capacity=75% increment of old capacity	Default capacity=10 New capacity=100% increment of old capacity
	We can use Iterator, ListIterator only in ArrayList.	We can use Iterator, ListIterator, Enumeration in Vector.
	Introduced in jdk 1.2 version and it is not legacy.	Introduced in jdk 1.0 version and it is legacy.
7.	Why map doesn't extend the Collection Interface?	
	<ul style="list-style-type: none">• The map Interface in java follows a key-value pair structure whereas the Collection Interface is a collection of objects which are stored in a structured manner with a specified access mechanism.• The main reason Map doesn't extend the Collection interface is that the add(E e) method of the Collection Interface doesn't support the key-value pair like Map Interface's put(K, V) method.• It might not extend the Collection Interface but still is an integral part of the java Collection Framework.	
8.	What happens when we put same keys in Map?	
	<ul style="list-style-type: none">• If we add a key-value pair where the key exists already, put method replaces the existing value of the key with the new value.	
9.	Which two methods should you override while putting the custom object as Key in HashMap?	
	<ul style="list-style-type: none">• You need to override hashCode and equals method in custom class while putting objects of custom class in HashMap.	
10.	How HashMap internally works?	
	<ul style="list-style-type: none">• What is HashMap?<ul style="list-style-type: none">➤ HashMap is a part of the Java collection framework. It uses a technique called Hashing.➤ Hashing: It is the process of converting an object into an integer value. The integer value helps in indexing and faster searches.➤ This HashMap class extends AbstractMap class that implements the Map interface. It stores the data in the pair of Key and Value.➤ HashMap contains an array of the nodes, and the node is represented as a class having following objects.➤ It uses an array and LinkedList data structure internally for storing Key and Value.➤ HashMap internally uses HashTable implementation.	

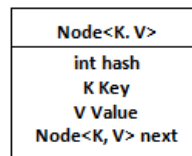


Figure: Representation of a Node

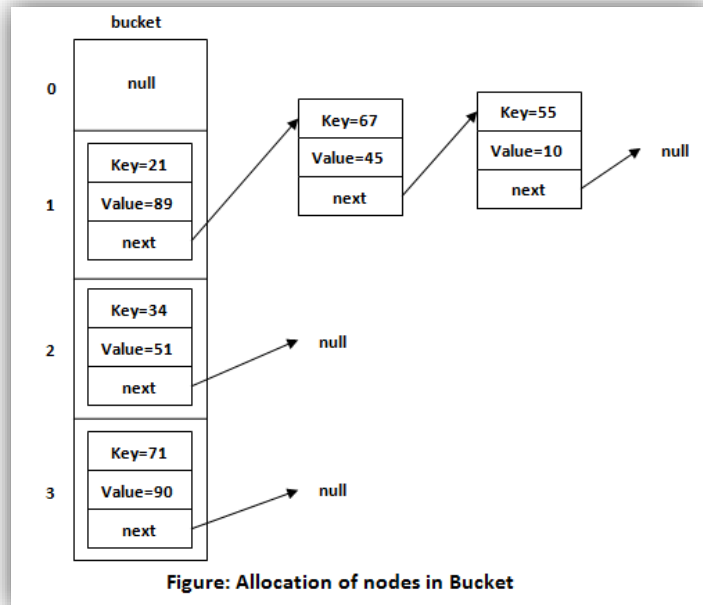
- **Few important points to about HashMap:**

- ✓ HashMap uses its static inner class Node<K,V> for storing the entries into the map.
- ✓ HashMap allows at most one null key and multiple null values.
- ✓ The HashMap class does not preserve the order of insertion of entries into the map.
- ✓ HashMap has multiple buckets or bins which contain a head reference to a singly linked list. That means there would be as many linked lists as there are buckets. Initially, it has a bucket size of 16 which grows to 32 when the number of entries in the map crosses the 75%. (That means after inserting in 12 buckets bucket size becomes 32)
- ✓ HashMap is almost similar to Hashtable except that it's unsynchronized and allows at max one null key and multiple null values.
- ✓ HashMap uses hashCode() and equals() methods on keys for the get and put operations. So HashMap key objects should provide a good implementation of these methods.
- ✓ That's why the Wrapper classes like Integer and String classes are a good choice for keys for HashMap as they are immutable and their object state won't change over the course of the execution of the program.

Note: Java HashMap is not thread-safe and hence it should not be used in multithreaded application. For the multi-threaded application, we should use ConcurrentHashMap class.

Before understanding the internal working of HashMap, you must be aware of hashCode() and equals() method.

- **equals():** It checks the equality of two objects. It compares the Key, whether they are equal or not. It is a method of the Object class. It can be overridden. **If you override the equals() method, then it is mandatory to override the hashCode() method.**
- **hashCode():** This is the method of the object class. It returns the memory reference of the object in integer form. The value received from the method is used as the bucket number. The bucket number is the address of the element inside the map. In HashMap, hashCode() is used to calculate the bucket and therefore calculate the index. **Hash code of null Key is 0.**
- **Buckets:** A bucket is one element of HashMap array. It is used to store nodes. Two or more nodes can have the same bucket. In that case LinkedList structure is used to connect the nodes. Buckets are different in capacity.



Insert Key, Value pair in HashMap

- We use **put()** method to insert the Key and Value pair in the HashMap.
- **The default size of HashMap is 16 (0 to 15).**
- In the following example, we want to insert three (Key, Value) pair in the HashMap.

```
HashMap<String, Integer> map = new HashMap<>();
map.put("Aman", 19);
map.put("Sunny", 29);
map.put("Ritesh", 39);
```

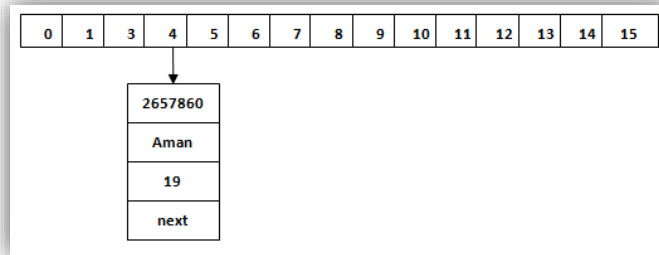
- Let's see at which index the Key, value pair will be saved into HashMap. When we call the put() method, then it calculates the hash code of the Key "Aman."
- Hash code of key may be large enough to create an array. hash code generated may be in the range of integer and if we create arrays for such a range, then it will easily cause **outOfMemoryException**. So, we generate index to minimize the size of array.
- Suppose the hash code of "Aman" is 2657860. To store the Key in memory, we have to calculate the index.
- Basically, following operation is performed to calculate index.

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

where n is number of buckets or the size of array.

Index = 2657860 & (16-1) = 4

- The value 4 is the computed index value where the Key and value will store in HashMap.



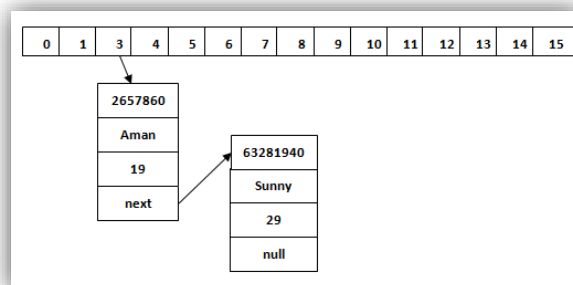
- If the key is null, the value is stored in **table[0]** position, because **hashcode for null is always 0**.

Hash Collision

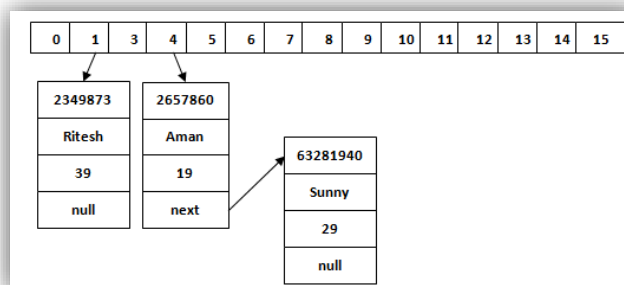
- This is the case when the calculated index value is the same for two or more Keys.
- Let's calculate the hash code for another Key "Sunny." Suppose the hash code for "Sunny" is 63281940. To store the Key in the memory, we have to calculate index by using the index formula.

$$\text{Index} = 63281940 \ \& \ (16-1) = 4$$

- Place this object at index 4 if no other object is presented there.
- In this case a node object is found at the index 4 – this is a case of collision.
- In this case, equals() method check that both Keys are equal or not. If Keys are same, replace the value with the current value. Otherwise, connect this node object to the existing node object through the LinkedList. Hence both Keys will be stored at index 4.
- In case of collision, i.e. index of two or more nodes are same, nodes are joined by link list i.e. second node is referenced by first node and third by second and so on.



- Similarly, we will store the Key "Ritesh." Suppose hash code for the Key is 2349873. The index value will be 1. Hence this Key will be stored at index 1.



get() method in HashMap

- get() method is used to get the value by its Key. It will not fetch the value if you don't know the Key. When get(K Key) method is called, it calculates the hash code of the Key.
- Suppose we have to fetch the Key "Aman." The following method will be called.

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

- It generates the hash code as 2657860. Now calculate the index value of 2657860 by using index formula. The index value will be 4, as we have calculated above.
- get() method search for the index value 4. It compares the first element Key with the given Key. If both keys are equal, then it returns the value else check for the next element in the node if it exists.
- In our scenario, it is found as the first element of the node and return the value 19.
- Let's fetch another Key "Sunny." The hash code of the Key "Sunny" is 63281940. The calculated index value of 63281940 is 4, as we have calculated for put() method.
- Go to index 4 of array and compare first element's key with given key. If both are equals then return the value, otherwise check for next element if it exists.
- In our case it is not found as first element and next of node object is not null.
- If next of node is not null traverse to the second element and repeat the process until key is not found or next is not null. It compares the second element Key with the specified Key and returns the value 29.
- If next of node is null then return null.

Re-Hashing:

- Whenever the number of entries in the hashmap crosses the threshold value then the bucket size of the hashmap is doubled and rehashing is performed and all the already existing entries of the map are copied and new entries are added to this increased hashmap.

Threshold value = Bucket size * Load factor

- Eg. If bucket size is 16 and the load factor is 0.75 then the threshold value is 12.

Time complexity

- Time complexity is almost constant for put and get method until rehashing is not done.
- In a fairly distributed hashMap where the entries go to all the buckets in such a scenario, the hashMap has **O(1)** time for search, insertion, and deletion operations.
- In the worst case, where all the entries go to the same bucket and the singly linked list stores these entries, **O(n)** time is required for operations like search, insert, and delete.
- In a case where the threshold for converting this linked list to a self-balancing binary search tree(i.e. AVL/Red black) is used then for the operations, search, insert and delete **O(logN)** is required as AVL/Red Black tree has a max length of logN in the worst case.

	<p>Answer in Short:</p> <ul style="list-style-type: none"> • HashMap uses its static inner class Node<K,V> for storing map entries. That means each entry in hashMap is a Node. • Internally HashMap uses a hashCode of the key Object and this hashCode is further used by the hash function to find the index of the bucket where the new entry can be added. • HashMap uses multiple buckets and each bucket points to a Singly Linked List where the entries (nodes) are stored. • Once the bucket is identified by the hash function using hashCode, then hashCode is used to check if there is already a key with the same hashCode or not in the bucket(singly linked list). • If there already exists a key with the same hashCode, then the equals() method is used on the keys. If the equals method returns true, that means there is already a node with the same key and hence the value against that key is overwritten in the entry(node), otherwise, a new node is created and added to this Singly Linked List of that bucket. • If there is no key with the same hashCode in the bucket found by the hash function then the new Node is added into the bucket found. <p style="text-align: center;">Or</p> <p>⇒ When we create HashMap object, HashMap instance as per default capacity 16 buckets is created.</p> <p>⇒ When we perform add (put ()) operation, it accepts data in key & value format.</p> <p>⇒ Internally hashing technique is used, that generates hashCode for key and also calculate index to find bucket location for inserting data in HashMap instance.</p> <p>⇒ It will store element at that location as a node format.</p> <p style="text-align: center;"> previous node address (Key) (Value) next node address </p> <p>⇒ Now when we perform retrieval (get ()) operation, it asks for key.</p> <p>⇒ Again hashing technique is used and bucket location is identified, then equals () method is used to compare key content and if it returns true then value is retrieved.</p>
11.	How Collisions Are Resolved?
	<ul style="list-style-type: none"> • If two unequal objects can have the same hash code value, then how those two different objects will be stored in the same array location called a bucket. • The answer is LinkedList. • The Entry class has an attribute "next." This attribute always points to the next object in the chain. This is exactly the behavior of the LinkedList.
12.	hashCode() and equals() contracts
	<ul style="list-style-type: none"> • Some principles of equals() method of Object class : If some other object is equal to a given object, then it follows these rules: 1. Reflexive: for any non-null reference value a, a.equals(a) should return true.

	<p>2. Symmetric: for any non-null reference values a and b, if a.equals(b) should return true then b.equals(a) must return true.</p> <p>3. Transitive: for any non-null reference values a, b, and c, if a.equals(b) returns true and b.equals(c) returns true, then a.equals(c) should return true.</p> <p>4. Consistent: Multiple calling of a.equals(b) should consistently return true or consistently return false If the value of the object is not modified for either object.</p> <p>Note: For any non-null reference value a, a.equals(null) should return false.</p> <ul style="list-style-type: none"> • The general contract of hashCode is: <ol style="list-style-type: none"> 1. During the execution of the application, if hashCode() is invoked more than once on the same Object then it must consistently return the same Integer value, provided no information used in equals(Object) comparison on the Object is modified. It is not necessary that this Integer value to be remained same from one execution of the application to another execution of the same application. 2. If two Objects are equal, according to the equals(Object) method, then hashCode() method must produce the same Integer on each of the two Objects. 3. If two Objects are unequal, according to the equals(Object) method, It is not necessary the Integer value produced by hashCode() method on each of the two Objects will be distinct. It can be same but producing the distinct Integer on each of the two Objects is better for improving the performance of hashing-based Collections like HashMap, HashTable...etc. <p>Note: Equal objects must produce the same hash code as long as they are equal, however unequal objects need not produce distinct hash codes.</p> <p>Key points to remember:</p> <ul style="list-style-type: none"> ⇒ If you are overriding equals method then you should override hashCode() also. ⇒ If two objects are equal then they must have same hashCode. ⇒ If two objects have same hashCode then they may or may not be equal. ⇒ Always use same attributes to generate equals and hashCode. ⇒ Overriding equals() alone will make your business fail with hashing data structures like: HashSet, HashMap, HashTable ... etc. ⇒ Overriding hashCode() alone doesn't force Java to ignore memory addresses when comparing two objects.
13.	Why String, Integer and other wrapper classes are considered good keys for HashMap?
	<ul style="list-style-type: none"> • String, Integer and other wrapper classes are natural candidates of HashMap key, and String is most frequently used key as well because String is immutable and final, and overrides equals and hashCode() method. • Other wrapper class also shares similar property.

	<ul style="list-style-type: none"> Immutability is required, in order to prevent changes on fields used to calculate hashCode() because if key object returns different hashCode during insertion and retrieval than it won't be possible to get an object from HashMap. 														
14.	Can we use any custom object as a key in HashMap?														
	<ul style="list-style-type: none"> Yes, we can use any Object as key in Java HashMap provided it follows equals and hashCode contract and its hashCode should not vary once the object is inserted into Map. 														
15.	Difference between HashMap and ConcurrentHashMap?														
	<table> <tr> <th>HashMap</th><th>ConcurrentHashMap</th></tr> <tr> <td>HashMap is not synchronized internally and it is not thread safe. You can make HashMap synchronized externally using Collections.synchronizedMap() method.</td><td>ConcurrentHashMap is internally synchronized and hence it is thread safe.</td></tr> <tr> <td>HashMap performance is relatively high because it is non-synchronized in nature and any number of threads can perform simultaneously.</td><td>ConcurrentHashMap performance is low sometimes because sometimes Threads are required to wait on ConcurrentHashMap.</td></tr> <tr> <td>HashMap is the part of Java collection framework since JDK 1.2.</td><td>ConcurrentHashMap is introduced in JDK 1.5 as an alternative to HashTable.</td></tr> <tr> <td>HashMap allows maximum one null key and any number of null values.</td><td>ConcurrentHashMap doesn't allow even a single null key and a null value.</td></tr> <tr> <td>While one thread is iterating the HashMap object, if other thread tries to add/modify the contents of Object then we will get Run-time exception saying ConcurrentModificationException. i.e., Iterators returned by HashMap are fail-fast in nature.</td><td>In ConcurrentHashMap we won't get any exception while performing any modification at the time of iteration. i.e., Iterators returned by ConcurrentHashMap are fail-safe in nature.</td></tr> <tr> <td>HashMap is not synchronized internally and it is most suitable for single threaded applications.</td><td>ConcurrentHashMap is internally synchronized and hence it is most suitable for multi-threaded applications.</td></tr> </table>	HashMap	ConcurrentHashMap	HashMap is not synchronized internally and it is not thread safe. You can make HashMap synchronized externally using Collections.synchronizedMap() method.	ConcurrentHashMap is internally synchronized and hence it is thread safe.	HashMap performance is relatively high because it is non-synchronized in nature and any number of threads can perform simultaneously.	ConcurrentHashMap performance is low sometimes because sometimes Threads are required to wait on ConcurrentHashMap.	HashMap is the part of Java collection framework since JDK 1.2.	ConcurrentHashMap is introduced in JDK 1.5 as an alternative to HashTable.	HashMap allows maximum one null key and any number of null values.	ConcurrentHashMap doesn't allow even a single null key and a null value.	While one thread is iterating the HashMap object, if other thread tries to add/modify the contents of Object then we will get Run-time exception saying ConcurrentModificationException. i.e., Iterators returned by HashMap are fail-fast in nature.	In ConcurrentHashMap we won't get any exception while performing any modification at the time of iteration. i.e., Iterators returned by ConcurrentHashMap are fail-safe in nature.	HashMap is not synchronized internally and it is most suitable for single threaded applications.	ConcurrentHashMap is internally synchronized and hence it is most suitable for multi-threaded applications.
HashMap	ConcurrentHashMap														
HashMap is not synchronized internally and it is not thread safe. You can make HashMap synchronized externally using Collections.synchronizedMap() method.	ConcurrentHashMap is internally synchronized and hence it is thread safe.														
HashMap performance is relatively high because it is non-synchronized in nature and any number of threads can perform simultaneously.	ConcurrentHashMap performance is low sometimes because sometimes Threads are required to wait on ConcurrentHashMap.														
HashMap is the part of Java collection framework since JDK 1.2.	ConcurrentHashMap is introduced in JDK 1.5 as an alternative to HashTable.														
HashMap allows maximum one null key and any number of null values.	ConcurrentHashMap doesn't allow even a single null key and a null value.														
While one thread is iterating the HashMap object, if other thread tries to add/modify the contents of Object then we will get Run-time exception saying ConcurrentModificationException. i.e., Iterators returned by HashMap are fail-fast in nature.	In ConcurrentHashMap we won't get any exception while performing any modification at the time of iteration. i.e., Iterators returned by ConcurrentHashMap are fail-safe in nature.														
HashMap is not synchronized internally and it is most suitable for single threaded applications.	ConcurrentHashMap is internally synchronized and hence it is most suitable for multi-threaded applications.														
<u>Comparable & Comparator</u>															
1.	Explain Comparable with its method														
	<ul style="list-style-type: none"> Comparable interface is mainly used to sort the arrays (or lists) of custom objects. Lists (and arrays) of objects that implement Comparable interface can be sorted automatically by Collections.sort (and Arrays.sort). This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be rollno, name, age or anything else. 														

[compareTo\(Object obj\) method](#)

public int compareTo(Object obj): It is used to compare the current object with the specified object.

It returns

- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

2. Explain Comparator with its method

- Java Comparator interface is used to order the objects of a user-defined class.
- This interface is found in java.util package and contains 2 methods `compare(Object obj1, Object obj2)` and `equals(Object element)`.
- It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member, for example, rollno, name, age or anything else.

The compare Method	The equals Method
<ul style="list-style-type: none">• int <code>compare(Object obj1, Object obj2)</code>• It compares the first object with the second object.• This method returns zero if the objects are equal.• It returns a positive value if obj1 is greater than obj2. Otherwise, a negative value is returned.• By overriding <code>compare()</code>, you can alter the way that objects are ordered.• For example, to sort in a reverse order, you can create a comparator that reverses the outcome of a comparison.	<ul style="list-style-type: none">• boolean <code>equals (Object obj)</code>• It is used to compare the current object with the specified object.• The method returns true if obj and the invoking object are both Comparator objects and use the same ordering. Otherwise, it returns false.• Overriding <code>equals()</code> is unnecessary, and most simple comparators will not do so.

3. Write the difference between comparable and comparator.

Comparable	Comparator
Comparable meant for default natural sorting order.	Comparator meant for customized sorting order.

	Comparable provides a single sorting sequence . In other words, we can sort the collection on the basis of a single element such as id, name, and price	The Comparator provides multiple sorting sequences . In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
	Comparable present in java.lang package .	Comparator present in java.util package .
	It contains only one method i.e. compareTo() .	It contain two methods i.e. compare() and equals() .
	All wrapper classes and String class implements comparable interface.	The only implemented classes of comparator are i. Collator ii. RuleBasedCollator
4.	Difference between compareTo() and compare().	
	compareTo()	compare()
	The Comparable interface provides a compareTo() method for the ordering of objects.	The Comparator interface provides a compare() method for the ordering of objects.
	public int compareTo(Object obj)	int compare(Object obj1, Object obj2)
	It is used to compare the current object with the specified object and returns an integer.	It compares the first object with the second object and returns an integer.
	It returns -ve number if & only if current object is less than obj. It returns +ve number if & only if current object is greater than obj. It returns 0 if & only if current object is equal to obj.	It returns -ve number if & only if obj1 is less than obj2 It returns +ve number if & only if obj1 is greater than obj2 It returns 0 if & only if obj1 is equal to obj2
5.	While implementing comparator interface only implementation is provided for compare() but not for equals(). Why?	
	<ul style="list-style-type: none">Whenever we are implementing comparator interface compulsory we should provide implementation only for compare() and we are not required to provide implementation for equals() because it is already available to our class from object class through inheritance. equals() is the dummy.	
6.	How the Collection objects are sorted in Java?	
	<ul style="list-style-type: none">Sorting in Java Collections is implemented via Comparable and Comparator interfaces.When Collections.sort() method is used the elements get sorted based on the natural order that is specified in the compareTo() method.On the other hand when Collections.sort(Comparator) method is used it sorts the objects based on compare() method of the Comparator interface.	

Multithreading	
1.	What is Thread? <ul style="list-style-type: none"> Thread is small part of execution. It is light weight process.
2.	What is Multithreading? <ul style="list-style-type: none"> The process of executing multiple threads simultaneously is known as multithreading. Multithreading in java is process of execution of two or more parts of a program to maximum utilize the CPU time. A multithreaded program contains two or more parts that can run concurrently. Each such part of a program called thread.' Threads are lightweight sub-processes, they share the common memory space. Java Multithreading is mostly used in games, animation, etc. Advantages of Java Multithreading <ol style="list-style-type: none"> It doesn't block the user because threads are independent and you can perform multiple operations at the same time. You can perform many operations together, so it saves time. Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.
3.	How the Thread implementation in java can be achieved? <p>Thread implementation in java can be achieved in two ways:</p> <ol style="list-style-type: none"> Extending the java.lang.Thread class Implementing the java.lang.Runnable Interface <p>Note: The Thread and Runnable are available in the java.lang.* package</p> <p>1)By extending thread class</p> <ul style="list-style-type: none"> The class should extend Java Thread class. The class should override the run() method. The functionality that is expected by the Thread to be executed is written in the run() method. <p>void start(): Creates a new thread and makes it runnable. void run(): The new thread begins its life inside this method.</p> <p>Example</p> <pre>public class MyThread extends Thread { public void run(){ System.out.println("thread is running..."); } public static void main(String[] args) { MyThread obj = new MyThread(); obj.start(); } }</pre> <p>2) By Implementing Runnable interface</p> <ul style="list-style-type: none"> The class should implement the Runnable interface

	<ul style="list-style-type: none"> The class should implement the run() method in the Runnable interface The functionality that is expected by the Thread to be executed is put in the run() method <p>Example</p> <pre>public class MyThread implements Runnable { public void run(){ System.out.println("thread is running.."); } public static void main(String[] args) { Thread t = new Thread(new MyThread()); t.start(); } }</pre>																								
4.	Can we start a Thread twice in Java?																								
	<ul style="list-style-type: none"> No, once a thread is started, it can never be started again. Doing so will throw an IllegalThreadStateException. 																								
5.	What is lock?																								
	<ul style="list-style-type: none"> Every object in java has a unique lock. Whenever we are using Synchronized keyword the only lock concept will come into the picture. 																								
6.	Why we need to apply lock? How many types of lock explain with example?																								
7.	Explain Thread class methods?																								
	<table> <tr> <th>Method</th><th>Description</th></tr> <tr> <td>public void run()</td><td>Used to perform action for a thread</td></tr> <tr> <td>public void start()</td><td>Starts the execution of the thread by calling run() method</td></tr> <tr> <td>public void sleep(long miliseconds)</td><td>Causes the currently executing thread to sleep (temporarily cease/suspend execution) for the specified number of milliseconds.</td></tr> <tr> <td>public void join()</td><td>Waits for a thread to end/ die.</td></tr> <tr> <td>public void join(long miliseconds)</td><td>Waits for a thread to die for the specified milliseconds.</td></tr> <tr> <td>public void setName(String name)</td><td>To give thread a name</td></tr> <tr> <td>public String getName()</td><td>Returns the name of the thread.</td></tr> <tr> <td>public Thread currentThread()</td><td>Returns the reference of currently executing thread</td></tr> <tr> <td>public void yield()</td><td>Causes the currently executing thread object to temporarily pause and allow other threads to execute.</td></tr> <tr> <td>public int setPriority(int priority)</td><td>Changes the priority of the thread.</td></tr> <tr> <td>public int getPriority()</td><td>Returns the priority of the thread.</td></tr> </table>	Method	Description	public void run()	Used to perform action for a thread	public void start()	Starts the execution of the thread by calling run() method	public void sleep(long miliseconds)	Causes the currently executing thread to sleep (temporarily cease/suspend execution) for the specified number of milliseconds.	public void join()	Waits for a thread to end/ die.	public void join(long miliseconds)	Waits for a thread to die for the specified milliseconds.	public void setName(String name)	To give thread a name	public String getName()	Returns the name of the thread.	public Thread currentThread()	Returns the reference of currently executing thread	public void yield()	Causes the currently executing thread object to temporarily pause and allow other threads to execute.	public int setPriority(int priority)	Changes the priority of the thread.	public int getPriority()	Returns the priority of the thread.
Method	Description																								
public void run()	Used to perform action for a thread																								
public void start()	Starts the execution of the thread by calling run() method																								
public void sleep(long miliseconds)	Causes the currently executing thread to sleep (temporarily cease/suspend execution) for the specified number of milliseconds.																								
public void join()	Waits for a thread to end/ die.																								
public void join(long miliseconds)	Waits for a thread to die for the specified milliseconds.																								
public void setName(String name)	To give thread a name																								
public String getName()	Returns the name of the thread.																								
public Thread currentThread()	Returns the reference of currently executing thread																								
public void yield()	Causes the currently executing thread object to temporarily pause and allow other threads to execute.																								
public int setPriority(int priority)	Changes the priority of the thread.																								
public int getPriority()	Returns the priority of the thread.																								

8.	Difference between Thread.start() and Thread.run() in Java	
	start()	run()
	When a program calls the start() method, a new thread is created and then the run() method is executed.	If we directly call the run() method then no new thread will be created and run() method will be executed as a normal method call on the current calling thread itself and no multi-threading will take place.
	<pre> class MyThread extends Thread { public void run() { System.out.println("Current thread name: "+ Thread.currentThread().getName()); System.out.println("run() method called"); } } class Test { public static void main(String[] args) { MyThread t = new MyThread(); t.start(); } } </pre> <p>Output: Current thread name: Thread-0 run() method called</p> <p>in the above program, when we call the start() method of our thread class instance, a new thread is created with default name Thread-0 and then run() method is called and everything inside it is executed on the newly created thread.</p>	<pre> class MyThread extends Thread { public void run() { System.out.println("Current thread name: "+ Thread.currentThread().getName()); System.out.println("run() method called"); } } class Test { public static void main(String[] args) { MyThread t = new MyThread(); t.run(); } } </pre> <p>Output: Current thread name: main run() method called</p> <p>in the above example, when we called the run() method of our MyThread class, no new thread is created and the run() method is executed on the current thread i.e. main thread. Hence, no multi-threading took place. The run() method is called as a normal function call.</p>
	we can't call the start() method twice otherwise it will throw an IllegalStateException	run() method can be called multiple times as it is just a normal method calling.
	Defined in java.lang.Thread class and we should not override start()	Defined in java.lang.Runnable interface and must be overridden in the implementing class.

9.	Write difference between sleep() and wait() method.		
	sleep()		wait()
	Through sleep() lock will be hold.		Through wait() lock will be released.
	Through sleep() running thread will wait only for time interval.		Through wait() running thread will wait for time interval as well as till the notify() or notifyAll() will get call.
	sleep is static method of Thread class.		wait is non-static method of Object class.
	sleep() we can call inside any of the context.		wait() we can call only inside synchronized context (Synchronized Method/ Block).
10.	Explain about synchronized keyword? What are its advantages and disadvantages?		
	<ul style="list-style-type: none">• If a multiple thread wants to operate on a same object simultaneously, there may be problem of data inconsistency (Race Condition).• To overcome this problem, we need to apply lock and for applying lock we can use synchronized keyword.• Synchronized keyword is only applicable to methods & blocks and not to variables & class.• If a method or block declared as the Synchronized then at a time one Thread is allow to execute that methods of block on the given object.		
	Advantages: The main advantage of synchronization is that by using the synchronized keyword we can resolve the data inconsistency problem.		
	Disadvantages: It increases waiting time of thread. At a time only one thread can operate on object so other threads have to wait. So it affects performance of the system.		
11.	What is object level lock? Explain with example.		
	<ul style="list-style-type: none">• Object level locking means when we want to synchronize non static method or no static code block so that it can be accessed by only one thread at a time for that instance.• It is used if you want to protect non static data.		
	<pre>public class A1 extends Thread { Hello1 h; String msg; public A1(Hello1 h, String msg) { this.h=h; this.msg=msg; } }</pre>	<pre>public class Hello1 { public synchronized void display(String msg) { System.out.println("[""); System.out.println(msg); } }</pre>	<pre>public class Test1 { public static void main(String[] args) { Hello1 h=new Hello1(); A1 t1=new A1(h,"Java"); } }</pre>

	<pre> public void run() { h.display(msg); } } </pre>	<pre> System.out.println("]); } //Synchronized Block-Object Level Lock /*public void display(String msg) { synchronized(this) { System.out.println("["); System.out.println(msg); System.out.println("]); } }*/ } </pre>	<pre> A1 t2=new A1(h,"Classes"); Hello1 h1=new Hello1(); A1 t3=new A1(h1,"CJC"); A1 t4=new A1(h1,"Pune"); t1.start(); t2.start(); t3.start(); t4.start(); } } </pre>
	<ul style="list-style-type: none"> • In above program thread t1 & t2 are of same object (h) and thread t3 & t4 are of same object (h1). • Any one thread from each object will come inside the method/block and executes parallelly. For example, thread t1 of h object and thread t3 of h1 object will execute method/block parallelly and at the same time remaining threads from each object i.e., thread t2 and thread t4 will wait outside the method. • When thread t1 and t3 performs their operation completely then and then only thread t2 and t4 will enter inside method and executes parallelly. 		
12.	What is Class level lock? Explain with example.		
	<ul style="list-style-type: none"> • Class level locking means you want to synchronize static method or block so that it can be accessed by only one thread for whole class. • If you have 10 instances of class, only one thread will be able to access only one method or block of any one instance at a time. • It is used if you want to protect static data. • Every class in java has a unique lock. If a Thread wants to execute a static synchronized method then it required class level lock. • Once a Thread got class level lock then it is allowed to execute any static synchronized method of that class. • While a Thread executing any static synchronized method, the remaining Threads are not allowed to execute any static synchronized method of that class simultaneously. • But remaining Threads are allowed to execute normal synchronized methods, normal static methods, and normal instance methods simultaneously. 		

<ul style="list-style-type: none">Class level lock and object lock both are different and there is no relationship between these two.	<pre>public class A2 extends Thread { Hello2 h; String msg; public A2(Hello2 h, String msg) { this.h=h; this.msg=msg; } public void run() { h.display(msg); } }</pre>	<pre>public class Hello2 { public static synchronized void display(String msg) { System.out.println([""); System.out.println(msg); System.out.println("]"); } } //Synchronized Block-Class Level Lock /*public void display(String msg) { synchronized(Hello2 .class) { System.out.println([""); System.out.println(msg); System.out.println("]"); } }*/ }</pre>	<pre>public class Test2 { public static void main(String[] args) { Hello2 h=new Hello2(); A2 t1=new A2(h,"Java"); A2 t2=new A2(h,"Classes"); Hello2 h1=new Hello2(); A2 t3=new A2(h1,"CJC"); A2 t4=new A2(h1,"Pune"); t1.start(); t2.start(); t3.start(); t4.start(); } }</pre>						
	<ul style="list-style-type: none">In above program any 1 thread from each object comes for execution inside the method/block at a time then it will be decided which thread will execute method/block first. After complete execution of method/block 2nd thread will execute the method/blockRemaining threads of each objects will wait outside the method until 1st set of threads performs their operation completely.								
13.	What is difference between object level lock and class level lock?								
	<table><tr><th>Key</th><th>Object Level Lock</th><th>Class Level Lock</th></tr><tr><td>Basic</td><td>It can be used when you want non-static method or non-static block of the</td><td>Class level locking means you want to synchronize static method or block so</td></tr></table>	Key	Object Level Lock	Class Level Lock	Basic	It can be used when you want non-static method or non-static block of the	Class level locking means you want to synchronize static method or block so		
Key	Object Level Lock	Class Level Lock							
Basic	It can be used when you want non-static method or non-static block of the	Class level locking means you want to synchronize static method or block so							

		code should be accessed by only one thread	that it can be accessed by only one thread for whole class.
	Static/Non Static	It should always be used to make non-static data thread safe	It should always be used to make static data thread safe.
	Number of Locks	Every object in the class may have their own lock	Multiple objects of class may exist but there is always one class's class object lock available

14. What is synchronized block? Explain its declaration.

- If very few lines of the code required synchronization then it's never recommended to declare entire method as synchronized we have to enclose those few lines of the code with in synchronized block.
- Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
- If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Note:

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

Syntax:
synchronized (object reference expression)
{
//code block
}

Example 1: To get lock of current object we can declare synchronized block as follows.
If Thread got lock of **current object** then only it is allowed to execute this block.
synchronized(**this**)

```
{
}
```

Example 2: To get the lock of a particular object 'b' we have to declare a synchronized block as follows.

If thread got lock of **'b'** object then only it is allowed to execute this block.
synchronized(**b**)

```
{
}
```

Example 3: To get **class level lock** we have to declare synchronized block as follows.
If thread got class level lock of Display then only it allowed to execute this block.

synchronized(**Display.class**)

```
{
}
```

	<p>Note: As the argument to the synchronized block, we can pass either object reference or ".class file" and we can't pass primitive values as argument because lock concept is dependent only for objects and classes but not for primitives.</p> <div><div><p>Example:</p><pre>Int x=b; Synchronized(x) { }</pre></div><div><p>Output:</p><p>Compile time error. Unexpected type. Found: int Required: reference</p></div></div>											
15.	What is advantage of synchronize block over synchronize method? (Which is more preffered?)											
	<ul style="list-style-type: none">• The main advantage of synchronized block over synchronized method is it reduces waiting time of Thread and improves performance of the system.• Synchronized method locks the entire object. This means no other thread can use any synchronized method in the whole object while the method is being run by one thread.• Synchronized block just locks the code within the block. This means no other thread can acquire a lock on the locked object until the synchronized block exits.											
16.	Write the difference between Runnable Interface and Callable Interface.											
	<table><tr><th>Runnable Interface</th><th>Callable Interface</th></tr><tr><td>Runnable Interface have run().</td><td>Callable Interface have call().</td></tr><tr><td>run() does not return any value.</td><td>call() returns object.</td></tr><tr><td>run() does not throws any exception.</td><td>call() throws exception.</td></tr><tr><td>It simply belongs to Java.lang.</td><td>It simply belongs to java.util.concurrent.</td></tr></table>	Runnable Interface	Callable Interface	Runnable Interface have run().	Callable Interface have call().	run() does not return any value.	call() returns object.	run() does not throws any exception.	call() throws exception.	It simply belongs to Java.lang.	It simply belongs to java.util.concurrent.	
Runnable Interface	Callable Interface											
Runnable Interface have run().	Callable Interface have call().											
run() does not return any value.	call() returns object.											
run() does not throws any exception.	call() throws exception.											
It simply belongs to Java.lang.	It simply belongs to java.util.concurrent.											
17.	While a Thread executing a synchronized method on the given object is the remaining Threads are allowed to execute other synchronized methods simultaneously on the same object?											
	<ul style="list-style-type: none">• No											
18.	What is synchronized statement?											
	<ul style="list-style-type: none">• The statements which present inside synchronized method and synchronized block are called synchronized statements. [Interview people created terminology].											
19.	Why wait(), notify() and notifyAll() methods are available in Object class but not in Thread class ?											
	<ul style="list-style-type: none">• Because Thread can call these methods on any common object.											
20.	Every java program contains by default how many Threads and which Thread by default runs?											
	<ul style="list-style-type: none">• Only 1 Thread and that is main thread and by default main Thread runs in every java program.											

21.	Which method is executed by any Thread?										
	<ul style="list-style-type: none"> A Thread executes only public void run() method. 										
22.	In multi-threading how can we ensure that a resource isn't used by multiple threads simultaneously?										
	<ul style="list-style-type: none"> In multi-threading, access to the resources which are shared among multiple threads can be controlled by using the concept of synchronization. Using synchronized keyword, we can ensure that only one thread can use shared resource at a time and others can get control of the resource only once it has become free from the other one using it. 										
23.	I want to control database connections in my program and want that only one thread should be able to make database connection at a time. How can I implement this logic?										
	<ul style="list-style-type: none"> This can be implemented by use of the concept of synchronization. Database related code can be placed in a method which has synchronized keyword so that only one thread can access it at a time. 										
24.	What is the Difference between thread and a process?										
	<table border="1"> <thead> <tr> <th>Process</th><th>Thread</th></tr> </thead> <tbody> <tr> <td>A process is a single application or program.</td><td>A thread is a subprocess within that application or program.</td></tr> <tr> <td>Processes are quite heavyweight and have more overhead.</td><td>Thread is light weight and have less overhead.</td></tr> <tr> <td>Each process has its own address space in memory.</td><td>Multiple threads share same address space of process.</td></tr> <tr> <td>You do not require synchronization in case of process.</td><td>Threads require synchronization to avoid unexpected scenarios.</td></tr> </tbody> </table>	Process	Thread	A process is a single application or program.	A thread is a subprocess within that application or program.	Processes are quite heavyweight and have more overhead.	Thread is light weight and have less overhead.	Each process has its own address space in memory.	Multiple threads share same address space of process.	You do not require synchronization in case of process.	Threads require synchronization to avoid unexpected scenarios.
Process	Thread										
A process is a single application or program.	A thread is a subprocess within that application or program.										
Processes are quite heavyweight and have more overhead.	Thread is light weight and have less overhead.										
Each process has its own address space in memory.	Multiple threads share same address space of process.										
You do not require synchronization in case of process.	Threads require synchronization to avoid unexpected scenarios.										
25.	How can we pause the execution of a Thread for specific time?										
	<ul style="list-style-type: none"> We can use Thread class sleep() method to pause the execution of thread for certain time. 										
26.	How does thread communicate with each other?										
	<ul style="list-style-type: none"> Threads can communicate using three methods i.e., wait(), notify(), and notifyAll(). 										
27.	Thread vs Runnable which is better approach to create a thread?										
	<ul style="list-style-type: none"> Implementing Runnable interface is considered to be better approach than Extending Thread because Java does not support multiple inheritance so if you extend Thread class and you cannot extend any other class which is needed in most of the cases. 										
28.	How can we achieve thread safety in java?										
	<ul style="list-style-type: none"> By Synchronized keyword. 										

29.	How to make a main thread wait until all other threads finished execution?
	<ul style="list-style-type: none"> You can make use of join method to achieve above scenario.
30.	What are the states in the lifecycle of a Thread?
	<ul style="list-style-type: none"> A thread can have one of the following states during its lifetime: <ol style="list-style-type: none"> New: In this state, a Thread class object is created using a new operator, but the thread is not alive. Thread doesn't start until we call the start() method. Runnable: In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selected by the thread scheduler. Running: In this state, the thread scheduler picks the thread from the ready state, and the thread is running. Waiting/Blocked: In this state, a thread is not running but still alive, or it is waiting for the other thread to finish. Dead/Terminated: A thread is in terminated or dead state when the run() method exits.