JAVA INTERVIEW QUESTIONS

1. What is Java?

Java is a high level programming language and object-oriented programming and platform. Write once and Run Anywhere.

It is statically typed language (every time variables must be declared before they are in use). High level means computer instructions are written in English based language that is not understood by the computer.

2. Current version?

Java 21 or JDK-21.

3. Father of Java?

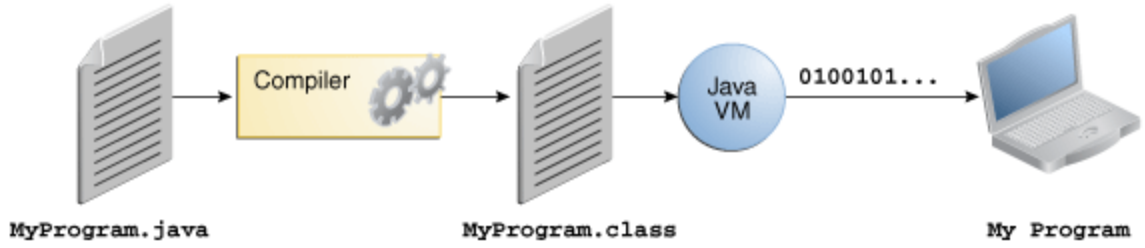
James Gosling.

4. Features of Java?

- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Architecture neutral
- Portable
- High performance
- Robust
- Secure

5. How Java Program will execute?

- All source code is first written in plain text files ending with the .java extension.
- Those source files are then compiled into .class files by the javac compiler.
- A .class file does not contain code that is native to the processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine[1] (Java VM).
- The java launcher tool then runs the application with an instance of the Java Virtual Machine.



6. Who will ignore the comments?

Compiler.

7. What is use of *String [] args* in main method?

The runtime system provides the information to the application.

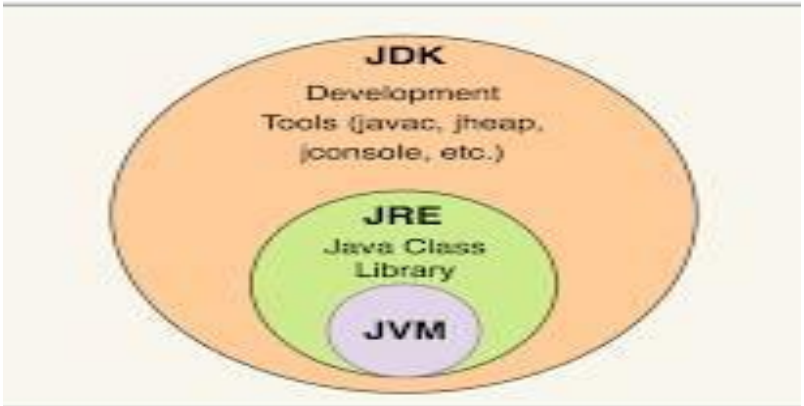
8. Byte code is platform dependent or independent?

Byte code is platform independent but source code (human readable code or .java extension file) is platform dependent.

9. Explain JDK, JRE and JVM?

JDK:

- JDK stands for Java Development Kit.
- It consists of the JRE, JVM, development tools and compiler.
- It is a software development environment.
- Which provides necessary tools and libraries to develop java applications.
- Provides tools such as Javac, debugger and so on to execute a program.



JRE:

- JRE stands for Java Runtime Environment.
- It provides the environment to run java programs.
- It loads classes check memory access and get system resource.
- Once the java program is compiled and converted to byte code, byte code can run on any platform that has the JRE.
- JDK and JRE both interact with each other to create a sustainable environment to run the java application.
- It contains components like ClassLoader, BytecodeVerifier, Interpreter.

JVM:

- It is an abstract machine.
- It does not exist physically.
- Act as run time engine to run java applications.
- It is called main method to run java program.
- It provides the environment to run byte code.

10. Explain *ClassLoader*?

Java ClassLoader will load all the classes that are needed for java program to run. Because all the classes are loaded into memory whenever necessary. To do this task JRE has ClassLoader.

11. What is a *Bytecode verifier*?

The byte code checker will ensure the format and precision of the code is correct, before passing it to the interpreter. If the class is found corrupt and will not load.

12. Difference between the heap memory and stack memory? How does java utilize this?

- stack memory is the portion of the memory that was assigned to every individual program, and it was fixed.
- Heap memory is the portion that was not assigned to every program, but it will be available for the program when it is required.

Memory utilization:

- When we write a java program then all the variables, methods, etc. are stored in the stack memory.
- And when we create an object of the class then that object was created in the heap. And it was referenced from the stack memory.

13. What is a *java compiler*?

Java compiler is a compiler that converts machine dependent source code to machine independent byte code. It includes JAVAC compiler. At run time compiler will first analyses the statements syntactically and then executes the output.

14. What is java *interpreter*?

It is a computer program that implements JVM. It is responsible for reading and executing programs. It is designed in such a way that it reads the source code and then converts it to instruction by instruction. Converts byte code into machine code.

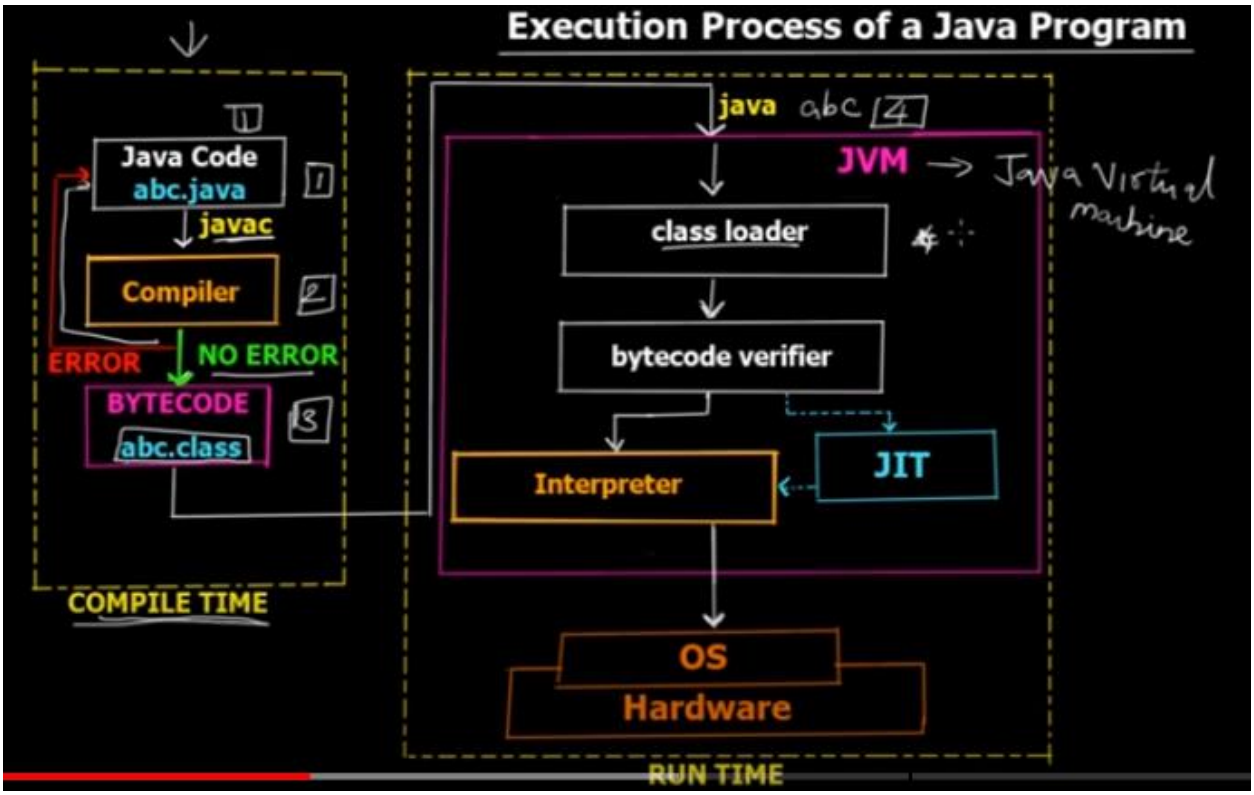
15. Difference between compiler and interpreter.

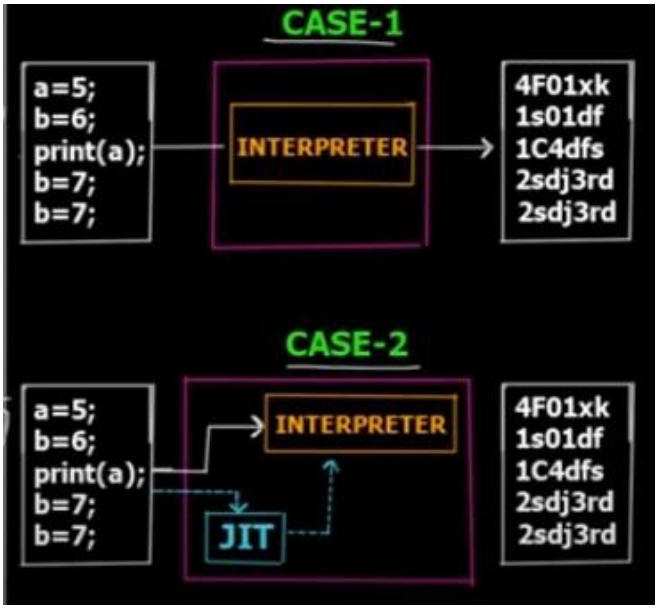
Compiler	***Interpreter***
Translates source code into machine code before execution.	Interprets source code line by line during run-time.
Faster execution.	Slower execution.
Generates an intermediate file that can be executed directly.	Does not generate any intermediate file it directly executes source file.

16. What is ***JIT compiler***?

JIT stands for just in time compiler.

- Used for improving performance during the run time.
- Its task is to compile byte code.
- If the byte code has similar functionality at same time thereby reducing time for code to run.





- In case 1 let's assume that JIT is not there.
- Then the interpreter will execute the class file line by line.
- If it has redundant data means duplicate data, it will execute for those lines also.
- For this reason, interpretation is slower.
- In case 2 we have a JIT compiler.

- It will enhance the interpretation process.
- It will check if there is redundant data present or not.
- If present, then it will execute only once.
- Then it will give the non-redundant values to the interpreter to execute.

17. What is ***JAR file***?

The jar file is the Java Archive file is a compressed file consisting of multiple java class files, associated metadata and resources into a single file. The primary purpose is to facilitate the distribution and deployment of java applications and libraries.

JAVA 8 FEATURES:

It has two products JDK 8 and JRE 8. JDK 8 is the superset of JRE. It contains all the tools, compilers, debuggers and all the necessary development applets and applications. JRE provides the libraries and run time environments to run the application.

Features of JAVA 8.

1. Lambda expressions and Functional Interface.
2. Default and Static methods in Interface.
3. Java date time API.
4. Java stream API.
5. Optional class.
6. Method reference.

1) Lambda Expressions and Functional Interface:

Lambda expressions allow us to write more concise and shorter code. Lambda expressions are also helpful for initializing functional interfaces. Can be used in variety of places, such as forEach() of Iterable interface.

Functional interfaces are interfaces that contain only one pure abstract method in it. They are way to represent a function as a value.

2) Default and static methods:

In earlier versions of java, the Interfaces can only contain abstract methods, but in java 8 default and static methods are introduced in the interfaces.

Default methods have a method body, whereas abstract methods don't have. The compiler will not force the subclass to override the default methods, they are optional if they want, they can provide implementation. They start with default keyword.

Static methods that are declared inside the interface can't be overridden in the class that implements the interface. In general, static methods can't be overridden.

3) ***Stream API*** allows to perform operations on collections of data in more efficient

and concise manner.

4) ***Date and Time API*** provides number of classes and methods to work with dates and times.

5) Optional class is introduced in java 8. Present in java.util package, it is a value-based class. It is used to check the NullPointerException that leads to termination of the application. By using the Optional class no need to check too many nulls. It is final class.

```
import java.util.Optional;
public class OptionalExample {
    public static void main(String[] args) {
        // Creating an Optional with a non-null value
        Optional<String> nonNullOptional = Optional.of("Hello, Optional!");

        // Creating an Optional with a potentially null value
        String nullableValue = "Nullable Value";
        Optional<String> nullableOptional = Optional.ofNullable(nullableValue);

        // Creating an empty Optional
        Optional<String> emptyOptional = Optional.empty();

        // Retrieving the value from Optional (if present)
        if (nonNullOptional.isPresent()) {
            String value = nonNullOptional.get();
            System.out.println("Value from nonNullOptional: " + value);
        }

        // Using orElse to provide a default value if Optional is empty
        String result1 = emptyOptional.orElse("Default Value");
        System.out.println("Result from emptyOptional: " + result1);

        // Using orElseGet with a supplier for lazy evaluation
        String result2 = emptyOptional.orElseGet(() -> generateDefaultValue());
```

```
        System.out.println("Result from emptyOptional with supplier: " + result2);

        // Using map to transform the value inside Optional
        String transformedValue = nonNullOptional.map(s -> s.toUpperCase()).orElse("No value");
        System.out.println("Transformed value: " + transformedValue);

        // Using filter to conditionally apply a transformation
        String filteredValue = nonNullOptional.filter(s -> s.length() > 10).orElse("Too short");
        System.out.println("Filtered value: " + filteredValue);
    }
    private static String generateDefaultValue() {
        System.out.println("Generating Default Value");
        return "Lazy Default";
    }
}
```

6) Method reference

- Method reference is a way to refer to a method without calling it.
- They are special type of lambda expressions that make code more concise and readable.
- Method references are created using double colon (::) operator.
- The left operand of the :: operator is the object that contains the method.
- The right operand of the :: is the name of the method.

VARIABLES

1. What are variables? Explain the types of variables?

Variables are like a container used to store the values.

4 types of variables: 1. *Instance* (non-static) 2. *Static* (class variable)

3. *Local* and 4. *Parametrized* variables

1) Instance variable:

Declared inside the class but outside the method, without creating the instance of a class can't access them. The values of the instance variables are unique to each instance of the class.

2) Static variables:

Declared inside the class but outside the method along with static keyword. Static modifier (non-access specifier) tells the compiler that there is exactly one copy of this variable is existed regardless of how many times the class has been instantiated.

3) Local Variables:

Variables declared inside the method body, if variable is declared then they are not visible to the outside of the method body or variables that are present in the other methods.

4) Parameterized variables:

Variables that are declared inside the parenthesis of the method. When we call the method, we must pass the arguments for that method are called parameterized variables.

2. Where are instance and static variables stored?

In *heap-memory* instance and static variables are stored. Static methods are also stored in the heap memory.

3. What is the default value of the local variables?

No. There are no default values for the variables declared inside (local variables) the methods.

4. Can we declare variables inside the constructor?

In constructor it is allowed to declare instance variables and can call the reference of the instance variables that are declared outside the constructor.

But in case of static variables, reference can be called but can't declare inside the constructor.

5. What are types of initializing variables?

There are 4 type they are literals, constructors, methods, and references.

6. Who will assign the default values to instance variables?

If the instance variables are not initialized, then compiler will assign the default values to the variables.

7. What are final variables?

A final variable can be explicitly initialized only once. A variable declared final can never be reassigned to refer to a different object.

DATATYPES

1. What is datatype?

Datatype specifies what type and size of data can be stored in a variable and is called datatype. There are two types *primitive* and *non-primitive* datatypes.

2. Primitive datatypes default values

NAME	DEFAULT	SIZE	TYPE	EXAMPLE
byte	0	8-bit	Integral	byte b = 100;
short	0	16-bit	Integral	short s = 10000;
int	0	32-bit	Integral	int i = 100000;
long	0L	64-bit	Integral	long l = 9999999;
float	0.0f	32-bit	Floating Point	float f = 123.4f;
double	0.0d	64-bit	Floating Point	double d = 12.4;
boolean	false	1-bit	Boolean	boolean b = true;
char	'\u0000'	16-bit	Character	char c = 'C';

3. What is the difference between primitive and non-primitive datatypes?

Primitive: Primitive datatypes are called using object reference and are predefined or already defined by the java.

Non-primitive: Theses are defined by the users and are non-predefined by java.

Primitive	***Non-primitive***
Predefined by java.	Non-predefined .
Primitives does not have any methods.	Non-primitives used to call certain methods to perform operations .
Primitives have default values.	Default value is null.

ARRAY

1. What is an Array?

Array is container that holds fixed number of values. The size of the array is declared when creating. Each item is called element and is accessed by its index number. There are two types of creating an array one is directly storing values in curly braces, i.e., implicit method and another one is by using new keyword for the creation of array.

2. Can we change the size of an array after creation? Can we assign negative value as size of array?

No. We can't change the size of an array once it has been created.

No. We can't pass the negative value has size of the array.

3. Advantages of array

- At the same time, it helps to sort multiple items.
- Using index, accessing elements of array is easier.
- Arrays can be used to perform operations quickly.

4. Disadvantages of array

- Once an array is created, it can't decrease or increase the size of an array.
- Allows only to store contiguous datatypes.
- Inefficient insertion or deletion of elements from the array.

5. Difference between Array and ArrayList?

- Array has fixed size, but ArrayList has dynamic size means can be increased or decreased.
- Array supports both datatypes (primitive and non-primitive) but ArrayList supports only non-primitives or objects.

Array	*ArrayList*
Supports both primitives and non-primitive types.	Supports only non-primitive types.
Has fixed size.	Does not have fixed size.
Arrays immutable. Inserting or deleting an element can't be done.	ArrayList are mutable.
Uses more memory storage.	Uses less memory storage.

6. What are jagged Array?

Arrays containing arrays of different length are called jagged or multi-dimensional arrays.

7. How to achieve infinite loop using for and while loops?

Using for loop:

```
for (; ; ) {
}
```

Using while loop:

```
While (true) {
}
```

METHODS

Method is a set of statements or instructions which perform specific operations.

1. Explain method?

- method is a set of statements to execute whenever method is called.
- Methods are used to code reusability.
- It is declared with access modifier then if it is a static method then use static keyword, if non static then there is no static keyword.
- After the access modifier there is a return type if it returns any result, return types like int, float, boolean etc. if not then void then name of the method followed by pair of parentheses.
- Variables inside the parenthesis are parameterized variables and declared inside the method block are local variables.

2. Method overloading?

- In a class, two or more methods can have the same method name with different parameters list.
- Even though they have different return types with same datatype of the parameter, it does not allow to overload the method.
- Methods with the same name and same parameters but differ in sequence are also not allowed.
- Cannot declare the methods with the same name and the same type of arguments because compiler does not treat them separately.
- Overloading is a type of compile time polymorphism.

3. Can we overload the main in Java? Which main method JVM will call?

Yes. We can overload the main method in java because it also a method nothing wrong in that. If main is overloaded, then JVM will executes only main having the signature of

public static void main (String [] args) or *public static void main (String args…)*

4. Can we override static method in java?

No, because method overriding is based on dynamic binding at runtime and static methods bounded using static binding at compile time. So static method cannot be override. It is also applicable for the main method because the main method is also a static method.

5. Can we make the main method final?

Yes, we can make the final. JVM does not allow the subclass to override the main method. In java we cannot override the any method in the subclass that is final.

6. Can we make the main synchronized?

Yes, the main can be synchronized in Java, a synchronized modifier is allowed in the main signature and you can make your main method synchronized in Java.

7. Explain main method in java?

- The main () method is a static method.
- We can overload the main () method in Java.
- We cannot override the main () method in Java.
- We can make the main method final in Java.

- We can make the main method synchronized in Java.

8. Can we make the main to non-static?

No, we cannot make main to non-static.

- It is a standard method that the JVM starts execution of java application.
- There is *no object* of the *class existing* when the java runtime starts.
- JVM will load the class into memory and calls the main.
- If the main method is non-static, then JVM will be unable to do it because there is no object of the class present.

9. Explain public static void main(String[] args) {}

Public: Public is access modifier which specifies where and who can access the method. Making method has a public means it is globally available. Because of this JVM will invoke it from the outside of the class as it is not present in the class.

Static: Static is a keyword. By making main() to static JVM can invoke it without instantiating the class.

Void: Void is a keyword which specifies that method does not return anything.

Main(String[] args): Main is the name of the method. String[] args specifies runtime system provides information to application.

10. Explain System.out.println();

It is used to print the statements that are passed. System is a final class, out is the instance of System class and has a type PrintStream. It has a public final access specifier. Println() is used to display the content.

11. Can we declare static variables inside the method?

In the non-static methods (instance method), variables can be declared and call the reference of the instance method, but in the case of static methods static variables can't declared and call the reference also.

In static methods non-static variables can be declared but can't call the reference of the non-static variables. But in the case of the static variable can't be declared and call the reference also.

CLASSES, OBJECTS and CONSTRUCTORS:

1. What is class?

- Class is just a keyword 'class', it is a blueprint to create objects of the class.
- It is a non-primitive or custom datatype.
- Every class implicitly has a default constructor which will not take any parameters if any of the constructors are not declared.
- It is declared with the access specifier followed by the class keyword and then name of the class.
- Every class implicitly extends the Object class has super (parent) class.
- Class signature
    - public class NameOfTheClass {}
- We can create n number of objects (instances) for the class.

OBJECT CLASS:

Object class is the root class of the class hierarchy; every class implicitly extends the object class. All objects, including arrays, implement methods of this class.

Object class present inside the java.lang package.

2 Explain the methods of object class?

There are 11 methods present inside the object class.

Methods	***Description***
1. clone()	Clone method creates and returns copy of the object.
2. equals(Object obj)	Indicate whether some other object is equal to this object. Return true if same otherwise false.
3. finalize()	Called by the garbage collector the of an object when the garbage collection determines that there are no more references to the object.
4. toString()	Returns string representation of the object.
5. getClass()	Returns the runtime class of the object.
6. hashCode()	Returns the hash code value of the object.
7. notify()	Wakes up a single thread that is waiting on this object's monitor.

8. notifyAll()	Wakes up all the threads that are waiting on this object's monitor.
9. wait()	Causes the current thread to wait until another thread invokes notify() or notifyAll() of the object.
10. wait(long timeout)	Causes the current thread to wait until either another thread invokes the notify() or notifyAll() of the object or specified amount of time elapsed.
11. wait(long timeout, int nanso)	Causes the current thread to wait until either another thread invokes the notify() or notifyAll() of the object or some other threads interrupts the current thread or specified amount of time elapsed.

CONSTRUCTORS

3. What is constructor?

- A class contains constructors that are invoked to create objects of the class.
- Every class has a constructor if not declared then have implicitly default (no argument).
- Constructor is a special type of method.
- It is same having class name.
- It does not have any return type.
- It is used with new keyword.
- Constructors are used to initialize the instance variables.
- Whenever instantiating a class or creating the object of the class an instance memory is allocated to that instance of a class.
- We cannot write two constructors that have the same number and types of arguments.

4. What are the types of constructors?

There are two types of constructors: *no-argument* and *parameterized* constructors.

5. Default (no argument) constructor calls which constructor?

It is called the constructor of its super class.

Ex: Default constructor of the super class (Object class) if a class does not extend or inherit from the any other class.

6. How many ways to pass arguments to the methods or constructor?

There are two types of passing arguments to methods or a constructor.

- Implicit arguments passing to method or constructor.
- Explicitly means storing the value to variable then passing the reference of that value to method or constructor.

7. What is constructor overloading?

Two constructors of the same class have the same name with different parameters means that initializing the different instance variables of the same class is called constructor overloading.

8. What is the use of new keyword when creating object?

The new keyword is used to create a new object of the class by instantiating the class. It will allocate a memory to that instance and then return the memory address of that instance.

9. What is the use of this keyword?

It is used for differentiating *instance* variables with the *local* variable.

10. What is constructor chaining?

Constructor chaining is the process of calling the one constructor from the other constructor.

- It is used for code reusability.
- It can be done in two ways.
    - Within the same class one constructor can call another constructor using this ().
    - From base class chaining is done by using the super ().

11. What are the different ways to create an object of a class or ways to instantiate a class?

There are 5 ways to create the object of a class in java they are.

1) Using new keyword.
    - ClassName *name* = new ClassName();
2) Using new instance
    - If the name of the class is known, then we can use Class.forName() method but it does not create object, to create object need to use new instance method.
    - Class *cls* = Class.forName("ClassName");
    - ClassName *ref* = (ClassName)cls.newInstance();

3) Clone method
   - The clone method is declared in the object class using this we can create object of the class using existing object by implementing *Cloneable* interface.
   - MyObject obj = new MyObject();
   - MyObject *myObject* = (MyObject) obj.clone();
4) Using deserialization:
   - Whenever we serialize and deserialize an object JVM creates a separate object. To deserialize an object we need to implement *Serializable* interface.
5) Using *newInstance*() method of Constructor class.

12. What are nested classes?

Defining a class within another class such class is called nested classes.

```
class OuterClass {
   ...
   class NestedClass {
      ...
   }}
```

- It is divided into two categories *non-static* and *static* inner class.
- We can make inner class static or non-static.
- The nested class is a member of the *enclosed class (outer class)*.
- Non-static class can *access members* of the outer or enclosed class even if they are declared as private.
- Static inner class does *not have access* to members of the outer class.

13. What is the use of nested class?

- Way of logically grouping classes that are used in one place.
- It increases the encapsulation.
- Lead to more readable and maintainable code.

14. Explain inner class?

Inner class instance is associated with instance of outer class like associated with instance methods and fields. The instance of the inner class is present only inside the outer class i.e., enclosed class.

```
class OuterClass {
   ...
   class InnerClass {
      ...
   }
        OuterClass outerObject = new OuterClass();
        OuterClass.InnerClass innerObject = outerObject.new  InnerClass();
}
```

15. What are local classes?

Local classes are like inner classes. Local classes are typically defined inside the block or method body are called local classes.

16. Anonymous classes?

Anonymous classes are like local classes except they do not have names. They enable us to declare and instantiate at same time.

ABSTRACT CLASS AND METHODS:

17. What is abstract class?

Abstract classes are classes that may or may not contain abstract methods in it. If it contains abstract methods, then it is declared as abstract class.

- Abstract classes can't be instantiated.
- Can be subclassed, and subclass will provide the implementation for the abstract methods.
- If subclass also does not provide the implementation for the abstract methods, then it is also abstract class.
- Abstract class signature.

```
abstract class ClassName{
    //abstract methods
    //non abstract methods if present
}
```

18. What are abstract methods?

Abstract methods are like a normal method, but they do not have method body. To provide implementation subclass is necessary. If a class contain abstract method, then class itself is declared as an abstract.

19. Difference between abstract class and interface?

Abstract class	***Interface***
Abstract class may or may not contain abstract methods.	Interface contains only abstract methods, except that may contain default or static methods.
Can't instantiate.	Can't instantiate.
To instantiate it subclass is necessary.	Same as abstract class.
Fields declared inside the abstract class can have any of the access modifier.	Fields declared inside the interface have by default ***public static final*** modifier.

Abstract class extend only class.	Interface can implement any number of interfaces.

ACCESS SPECIFIERS

1. What are access specifiers?

Access specifiers in java are keywords used to control the visibility of class, members, methods, constructors, and variables. They can be used to restrict access to members within class, within package or outside the package.

Access specifiers are: package default, public, private, and protected.

2. Explain access specifiers?

1) Public: This access specifier allows members and class to access from anywhere in the package whether it is in the same package or different package.

2) Private: This access specifier allows you to access only the members of the class which are declared within the class only.

3) Protected: This specifier allows you to access members of the class within the class and by the subclass of that class within the same package. Outside the package is not accessible.

4) Package Default: This specifier allows members to access within the same package.

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

OBJCET ORIENTED PROGRAMING (OOP's)

1. What are the pillars of Java?

There are 4 pillars of java.

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

2. What is an Object?

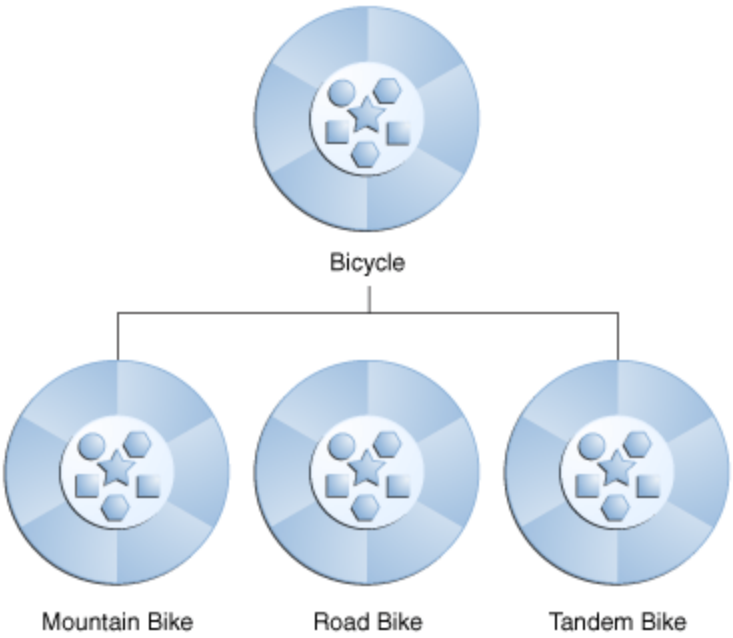
Anything that exists in the real world that has *two characteristics* like *state* and *behaviors* are called objects.

Object is a class that is super class it does not have any of the parent classes. In java language Object is only parent class from this other class inherit the properties from Object class.

3. What is inheritance?

Inheritance is the process of inheriting ***states and behaviors*** (fields and methods) of the *super class* to the *derived* or child class.

- Every class is implicitly or explicitly inherited from the other class.
- Class from which states and behaviors are inherited is ***super or base*** *(parent)* class and class that inherits the super class is called ***derived class***.
- If any of the classes is not explicitly inherited from the parent class, then it is implicitly inherited from the *Object class*.
- Object class in java is the only class that does not have any parent class.
- If a class has explicit inheritance, then must use *extends* keyword.
- Derived or child class can inherit fields of only one class means it must extend one parent class properties. But parent classes can have any number of child classes.



- The child class is necessary for creating instances of the class when the parent class has *protected access modifier*.
- By inheritance can achieve ***Is-a relation***.
- Used for code reusability.
- Private methods do not get inherited.
- Cyclic inheritance is not permitted.

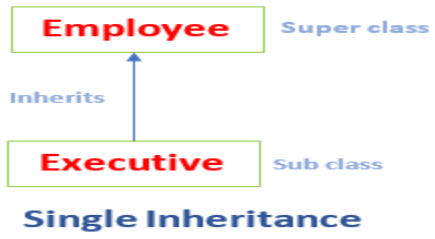
4. What are the types of inheritance?

There are 5 types of inheritance.

1. *Single level*
2. *Multi-level*
3. *Hierarchical*
4. *Multiple*
5. *Hybrid*

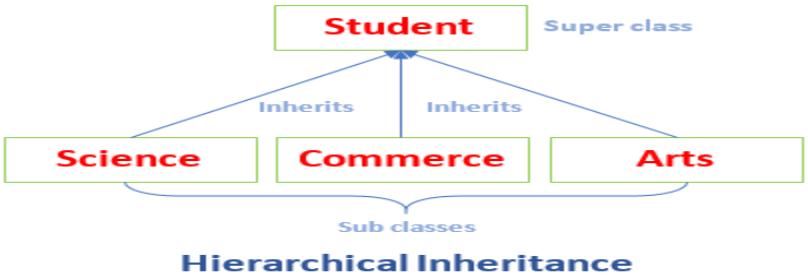
6. Explain the types of inheritance?

1. Single inheritance: In this one class will inherit the properties from the only one parent class.

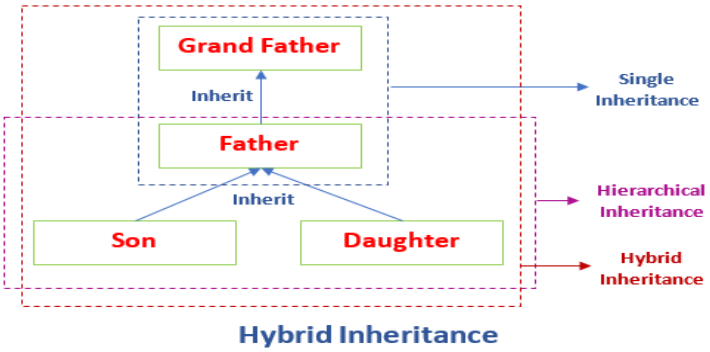


2. multi-level: A class is derived from a class which is also derived from another class called multi-level inheritance. In multi-level inheritance, an important fact is to maintain the classes in different levels.

3. hierarchical inheritance: The number of classes that are inherited from the single base class is called hierarchical inheritance.



4. hybrid inheritance: Hybrid means more than one. Hybrid inheritance is the combination of two or more types of inheritance.



5. multiple inheritance: Java does not support multiple inheritance through classes, but it can be achieved by using ***interfaces***.

7. What is polymorphism?

As the name specifies poly means many, morphism means forms. Polymorphisms defined as a single property can have many forms. Polymorphisms can be *implemented* through *overriding* or *overloading*.

8. Types of polymorphism?

There are two types of polymorphisms compile-time and run-time polymorphisms.

Compile-time polymorphisms:

Compile time polymorphism means compiler determines which method to execute based on the number of argument passed is called compile-time polymorphism.

- Compile-time polymorphisms are achieved through method overloading or constructor overloading.
- Method overloading means methods have same name with different parameters list.
- The compiler chooses the method to call based on the argument passed.
- It is also a static binding because decision of which method to call is made at compile time.

Run-time polymorphism:

Runtime polymorphism means a method implementation is provided in both sub-class and super class then the decision of which method to execute is made at run-time is called run-time polymorphism.

- Run-time polymorphism is achieved by method overriding.
- Overriding means method in the sub class have same name, return type, parameter list, as a method in the super class is called run-time polymorphism.
- JVM will decide which method has to execute whether from the sub class or parent class.

9. What is *method hiding*?

Static methods from the parent class are not inherit directly to child class, but explicitly want to inherit it has same method signature as in parent class then it is called method hiding.

10. What is ***encapsulation***?

Encapsulation is concept of object-oriented programing language; it means restricting the accessing of properties of the class and providing access over the methods (getters and setters).

- This ensures that the data can only be accessed through methods that are declared in the class.
- Achieved by making properties private.
- Improve the maintainability and security of the code.
- It provides control over how data is used.
- Protects data from being accessed and modified from the other classes.

11. What is ***abstraction***?

Abstraction is the process of using the functionality without knowing the implementation is called abstraction. It is basically hiding the implementation details.

- Abstraction can be achieved in two ways, by abstract class and interface.
- Abstract classes and interfaces can't instantiate.
- In both the cases subclass is necessary to provide the implementation for the abstract methods.
- If a concrete class (non-abstract) contain any abstract method in it, then it is declared as an abstract class.

INTERFACES AND ASSOCIATION

INTERFACE:

1. What is interface?

- Interface is a keyword.
- Interface is a medium which is used to connect to classes or clients and service provider.
- Interface *does not* extend Object class.
- It is used to achieve 100% abstraction and *multiple inheritance*.
- Interfaces *can't be* instantiated.
- They can only be implemented by classes or extended by other interfaces; interface can extend any number of interfaces.
- They contain only abstract methods, final variables (like constants) it is optional, static methods and default methods.
- Methods inside the interface do not have a method body except default methods and static methods.
- To use interface, we must write class that *implements* interface.
- Abstract methods and static methods inside the interface have the modifier ***public*** implicitly. Default methods have ***default modifier***.
- All constants' declarations in interface implicitly ***public static final***.
- Static methods defined inside the interface are not overridden by the implementation class.

2. What are the types of interfaces?

Functional interface and Marker interface.

1. Functional interface:

- Functional interface has only pure *one abstract method*.
- The interface that contains one abstract method but any number of default methods and static methods those interfaces are called *functional interface*.

Ex: 1) *Runnable* interface contains only run () method.

2) *ActionListener* interface contains only actionPerformed () method.

3) *ItemListener* contains only itemStateChanged () method.

2. Marker interface:

- Interface that does not contain any methods like abstract methods, default methods, constants.
- Those interfaces are called Marker interface.
- *Serializable* and *Cloneable* interfaces are examples of Marker interface.

3. What is an association?

- Association is the process of *connecting two classes* that are set up through their objects is called association.
- It is based on the ***Has a relation***.
- It shows how objects communicate with each other.
- It can be in any form one-to-one, one-to-many, many-to-one, many-to-many.

4. Types of association?

Two types of association are present Is-A and Has-A association.

Is-A association is also known as inheritance.

Has-A association:

Has-A association is further divided into two parts Aggregation and Composition.

Aggregation:

If two entities are connected to each other, if *one of them fails* then will not affect the other one is called aggregation association. Aggregation can exist without the other object; they are *not dependent* on each other.

Ex: Toy and battery are connected to each other, if toy broken then it will not affect the battery means battery remains same.

Composition:

In composition *entities are dependent* on each other. If one object fails to work, then the other one also goes down.

Ex: Mobile and mother board are connected to each other if mother board fails to work then the mobile also not work.

5. Differentiate between "==" and "equals()"

"=="	***"equals()"***
Used to check whether two objects are pointing to same memory location.	equals() used to check contents of two objects are same.
Applicable to primitive data types.	Applicable on non-primitive data types.
Can be directly use.	Before use it must be override.

6. Explain final class and methods?

Methods that are declared final are accessible only by the subclass of that class.

If class is declared as final, then it is not sub classed.

Ex: String class.