OBJECT ORIENTED PROGRAMMING

- BY TAMIL TELEGRAM TECH INTERVIEWERS

Q #1) Explain in brief what do you mean by Object Oriented Programming in Java?

Answer: OOP deals with objects, like real-life entities such as pen, mobile, bank account which has state (data) and behavior (methods).

With help of access, specifiers access to this data and methods is made secured. Concepts of encapsulation and abstraction offer data hiding and access to essentials, inheritance, and polymorphism help code reuse and overloading/overriding of methods and constructors, making applications platform-independent, secured and robust using languages like Java.

Q #2) Explain Is Java a pure Object Oriented language?

Answer: Java is not an entirely pure object-oriented programming language. **The following are the reasons:**

- Java supports and uses primitive data types such as int, float, double, char, etc.
- Primitive data types are stored as variables or on the stack instead of the heap.
- In Java, static methods can access static variables without using an object, contrary to object-oriented concepts.

Q #3) Describe class and object in Java?

Answer: Class and object play an integral role in object-oriented programming languages like Java.

- Class is a prototype or a template that has state and behavior supported by an object and used in the creation of objects.
- The object is an instance of the class, **for example,** Human is a class with the state as having a vertebral system, brain, color, and height and has behavior such as canThink(), ableToSpeak(), etc.

Q #4) What are the differences between class and objects in Java?

Answer: Following are a few major differences between class and objects in Java:

Class	Object
Class is a logical entity	Object is physical entity
Class is a template from which object can be created	Object is an instance of the class
Class is a prototype that has the state and behavior of similar objects	Objects are entities that exist in real life such as mobile, mouse, or intellectual objects such as bank account
Class is declared with class key word like class Classname { }	Object is created via new keyword as Employee emp = new Employee();
During class creation, there is no allocation of memory	During object creation, memory is allocated to the object
There is only one-way class is defined using the class keyword	Object creation can be done many ways such as using new keyword, newInstance() method, clone() and factory method.
Real-life examples of Class can be a •A recipe to prepare food. •Blue prints for an automobile engine.	Real-life examples of Object can be •A food prepared from recipe. •Engine constructed as per blue-prints.

Q #5) Why is a need for Object-oriented programming?

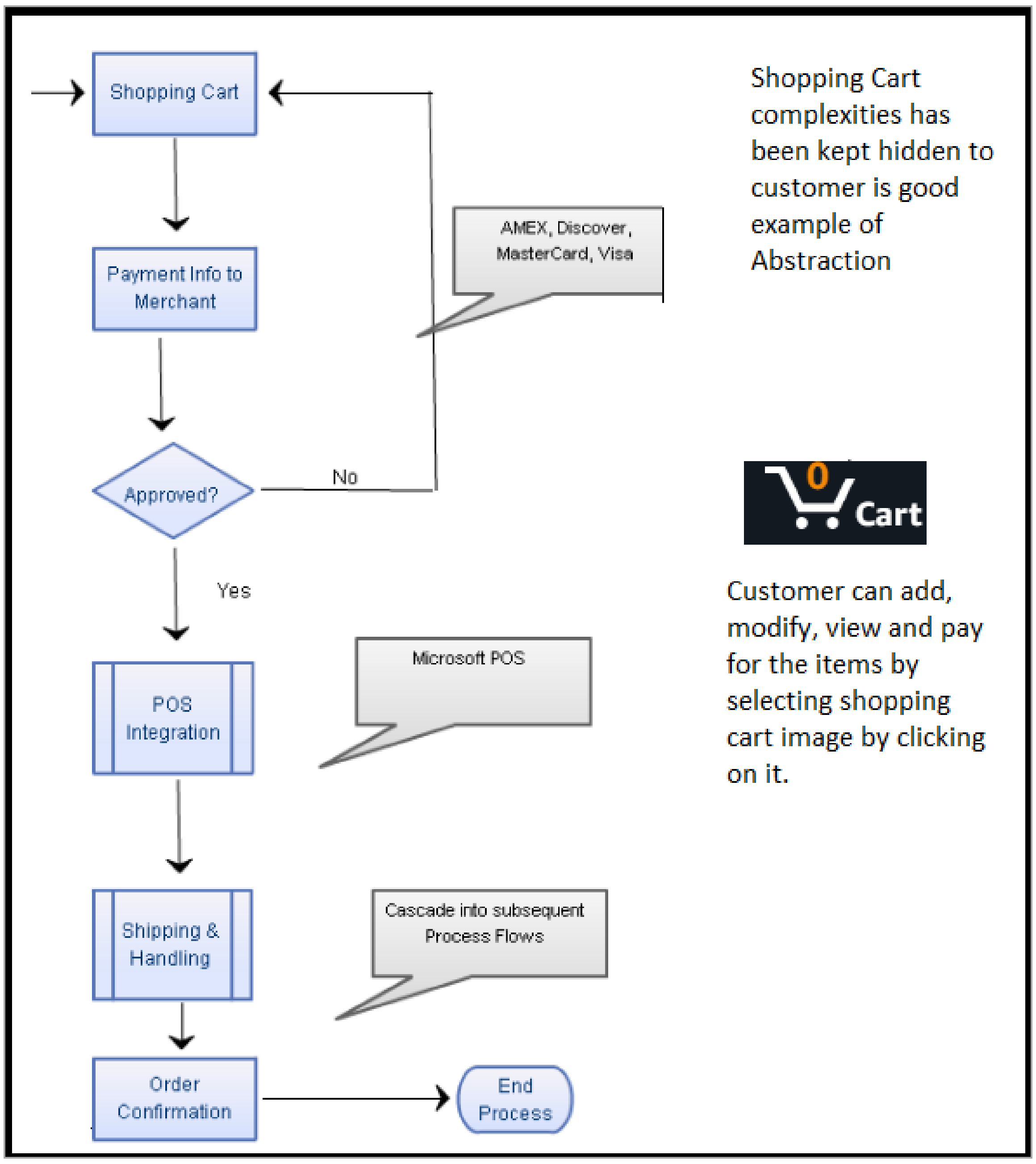
Answer: OOP provides access specifiers and data hiding features for more security and control data access, overloading can be achieved with function and operator overloading, Code Reuse is possible as already created objects in one program can be used in other programs.

Data redundancy, code maintenance, data security, and advantage of concepts such as encapsulation, abstraction, polymorphism, and inheritance in object-oriented programming provide an advantage over previously used procedural programming languages.

Q #6) Explain Abstraction with a real-time example.

Answer: Abstraction in object-oriented programming means hiding complex internals but to expose only essential characteristics and behavior with respect to context. In real life, an example of abstraction is an online shopping cart, say at any e-commerce site. Once you select a product and book order, you are just interested in receiving your product on time.

How things happen is not what you are interested in, as it is complex and kept hidden. This is known as abstraction. Similarly, take the example of ATM, the complexity of internals of how money is debited from your account is kept hidden, and you receive cash via a network. Similarly for cars, how petrol makes the engine run the automobile is extremely complex.


Shopping Cart
Payment Info to Merchant
AMEX, Discover, MasterCard, Visa
Approved?
No
Yes
POS Integration
Microsoft POS
Shipping & Handling
Cascade into subsequent Process Flows
Order Confirmation
End Process
Shopping Cart complexities has been kept hidden to customer is good example of Abstraction
0 Cart
Customer can add, modify, view and pay for the items by selecting shopping cart image by clicking on it.

Q #7) Give some real-time examples and explain Inheritance.

Answer: Inheritance means one class (sub class) acquiring properties of another class (super class) by inheritance. In real life, take an example of inheritance of a normal bicycle where it is a parent class and a sports bike can be a child class, where sports bike has inherited properties and behavior of rotating wheels with pedals via gears that of a normal bike.

Q #8) How polymorphism works in Java, explain with real-life examples?

Answer: Polymorphism is an ability to have multiple forms or capability of the method to do different things. In real life, the same person performing different duties behaves differently. In-Office he is an employee, at home, he is a father, during or in after school tuitions he is a student, on weekends he plays cricket and is a player in the playground.

In Java, there are two types of polymorphism

- **Compile-time polymorphism:** This is achieved by method overloading or operator overloading.
- **Runtime polymorphism:** This is achieved by method overriding.

Q #9) How many types of inheritance are present?

Answer: Various types of inheritance are listed below:

- **Single Inheritance:** Single child class inherits characteristics of the single-parent class.
- **Multiple Inheritance:** One class inherits features of more than one base class and is not supported in Java, but the class can implement more than one interface.
- **Multilevel Inheritance:** A class can inherit from a derived class making it a base class for a new class, **for example,** a Child inherits behavior from his father, and the father has inherited characteristics from his father.
- **Hierarchical Inheritance:** One class is inherited by multiple subclasses.
- **Hybrid Inheritance:** This is a combination of single and multiple inheritances.

Q #10) What is Interface?

Answer: Interface is similar to the class where it can have methods and variables, but its methods do not have a body, just a signature known as the abstract method. Variables declared in the interface can have public, static, and final by default. Interface is used in Java for abstraction and multiple inheritances, where the class can implement multiple interfaces.

Q #11) Can you explain the advantages of Abstraction and Inheritance?

Answer: Abstraction reveals only essential details to the user and ignores or hides irrelevant or complex details. In other words, data abstraction exposes the interface and hides implementation details. Java performs abstraction with the help of interfaces and abstract classes. Advantage of abstraction is that it makes simple in viewing things by reducing or hiding the complexity of implementation.

Duplication of code is avoided, and it increases code reusability. Only essential details are revealed to the user and improves the security of the application.

Inheritance is where child class inherits functionality (behavior) of the parent class. We need not write code once written in parent class for functionality again in the child class and thus making it easier to reuse the code. The code becomes readable as well. Inheritance is used where there "is a" relation. **Example:** Hyundai **is a** car OR MS Word **is a** software.

Q #12) What is the difference between extends and implements?

Answer: Both extends and implements keyword are used for inheritance but in different ways.

The differences between Extends and Implements keywords in Java are explained below:

Extends	**Implements**
A class can extend another class (child extending parent by inheriting his characteristics). Interface as well inherit (using keyword extends) another interface.	A class can implement an interface
Sub class extending super class may not override all of the super class methods	Class implementing interface has to implement all the methods of the interface.
Class can only extend a single super class.	Class can implement any number of interfaces.
Interface can extend more than one interfaces.	Interface cannot implement any other interface.
Syntax: **class Child extends class Parent**	**Syntax:** **class Hybrid implements Rose**

Q #13) What are different access modifiers in Java?

Answer: Access modifiers in Java controls access scope of class, constructor, variable, method, or data member. **Various types of access modifiers are as follows:**

- **Default access modifier** is without any access specifier data members, class and methods, and are accessible within the same package.
- **Private access modifiers** are marked with the keyword private, and are accessible only within class, and not even accessible by class from the same package.
- **Protected access modifiers** can be accessible within the same package or subclasses from different packages.
- **Public access modifiers** are accessible from everywhere.

Q #14) Explain the difference between abstract class and method?

Answer: Following are some differences between abstract class and abstract method in Java:

Abstract Class	Abstract Method
Object cannot be created from the abstract class.	Abstract method has a signature but does not have a body.
Sub class created or inherit abstract class to access members of abstract class.	It is compulsory to override abstract methods of super class in their sub class.
Abstract class can contain abstract methods or non abstract methods.	Class containing abstract method should be made abstract class.

Q #15) What are the differences between method and constructor?

Answer: Following are the differences between constructors and methods in Java:

Constructors	Methods
Constructors name should match with that of Class.	Methods should not have same name as Class name.
They are used to create, initialize and allocate memory to the object.	Methods are used to execute certain statements written inside them.
Constructors are implicitly invoked by the system whenever objects are created.	Methods are invoked when it is called.
They are invoked using new keyword while creating an instance of the class (object).	Methods are invoked during program execution.
Constructor does not have return type.	Method has a return type.
Constructor cannot be inherited by the subclass.	Methods can be inherited by a sub class.

Q #16) What is a constructor in Java?

Answer: Constructor is a method without a return type and has its name the same as the class name. When we create an object, a default constructor allocates memory for an object during the compilation of Java code. Constructors are used to initializing objects and set initial values for object attributes.

Q #17) How many types of constructors can be used in Java? Please explain.

Answer: There are basically three types of constructors in Java.

These are:

1. **Default constructor:** This constructor is without any parameter and invokes every time you create an instance of a class (object). If a class is an Employee, then the syntax of the default constructor will be Employee().
2. **No-arg constructor:** As the name implies, a constructor without any argument is called a no-arg constructor.
3. **Parameterized constructor:** Constructor with a number of parameters is called a parameterized constructor. You are required to provide arguments, i.e. initial values with respect to the data type of parameters in that constructor.

Q #18) Why new keyword is used in Java?

Answer: When we create an instance of class, i.e. objects, we use the Java keyword **new**. It allocates memory in the heap area where JVM reserve space for an object. Internally, it invokes the default constructor as well.

Syntax:

```
Class_name obj = new Class_name();
```

Q #19) When do you use the super keyword?

Answer: Super is a Java keyword used to identify or refer parent (base) class.

- We can use super to access super class constructor and call methods of the super class.
- When method names are the same in super class and sub class, to refer super class, the **super** keyword is used.
- To access the same name data members of parent class when they are present in parent and child class.
- **Super** can be used to make an explicit call to no-arg and parameterized constructors of the parent class.
- Parent class method access can be done using **super**, when child class has method overridden.

Q #20) When do you use this keyword?

Answer: this keyword in Java refers to the current object in the constructor or in the method.

- When class attributes and parameterized constructors both have the same name, **this** keyword is used.
- Keywords **this** invokes the current class constructor, method of the current class, return the object of the current class, pass an argument in the constructor, and method call.

Q #21) What is the difference between Runtime and compile-time polymorphism?

Answer: Both runtime and compile-time polymorphism are two different types of polymorphism. **Their differences are explained below:**

Compile Time Polymorphism	Runtime Polymorphism
Call is resolved by a compiler in compile-time polymorphism.	Call is not resolved by the compiler in runtime polymorphism.
It is also known as static binding and method overloading.	It is also known as dynamic, late, and method overriding.
Same name methods with different parameters or methods with the same signature and different return types are compile-time polymorphism.	Same name method with the same parameters or signature associated in different classes are called method overriding.
It is achieved by function and operator overloading.	It can be achieved by pointers and virtual functions.
As all the things are executed at compile time. compile-time polymorphism is less flexible.	As things execute at run time, runtime polymorphism is more flexible.

Q #22) What object-oriented features are used in Java?

Answer: A concept of using an object in Java programming language benefits by the use of object-oriented concepts like encapsulation for binding together the state and behavior of an object, secures data access with access specifiers, features like abstraction in information hiding, inheritance to extend state, and behavior of base classes to child classes, compile-time and runtime polymorphism for method overloading and method overriding, respectively.

Q #23) What is method overloading?

Answer: When two or more methods with the same name have either a different number of parameters or different types of parameters, these methods may have or may not have different return types, then they are overloaded methods, and the feature is method overloading. Method overloading is also called compile-time polymorphism.

Q #24) What is method overriding?

Answer: When a method of sub class (derived, child class) has the same name, parameters (signature), and same return type as the method in its super class (base, parent class) then the method in the subclass is said to be overridden the method in the superclass. This feature is also known as runtime polymorphism.

Q #25) Explain constructor overloading.

Answer: More than one constructor having different parameters so that different tasks can be carried out with each constructor is known as constructor overloading. With constructor overloading, objects can be created in different ways. Various Collection classes in Java API are examples of constructor overloading.

Q #26) What types of arguments can be used in Java?

Answer: For Java methods and functions, parameter data can be sent and received in different ways. If methodB() is called from methodA(), methodA() is a caller function and methodB() is called function, arguments sent by methodA() is actual arguments and parameters of methodB() is called formal arguments.

- **Call By Value:** Changes made to formal parameter (parameters of methodB()) do not get sent back to the caller (methodA()), This method is called **call by value**. Java supports the call by value.
- **Call by Reference:** Changes made to formal parameter (parameters of methodB()) are sent back to the caller (parameters of methodB()).
- Any changes in formal parameters (parameters of methodB()) are reflected in actual parameters (arguments sent by methodA()). This is called call by reference.

Q #27) Differentiate between static and dynamic binding?

Answer: The differences between Static and Dynamic binding are explained in the below table.

Static Binding	Dynamic Binding
Static binding in Java use type of fields and class to as a resolution.	Dynamic binding in Java uses object for resolving binding.
Method Overloading is an example of static binding.	Method overriding is an example of dynamic binding.
Static binding gets resolved at compile time.	Dynamic binding gets resolved at run time.
Methods and variables using static binding are private, final and static types.	Virtual methods use dynamic binding.

Q #28) Can you explain base class, subclass, and superclass?

Answer: Base class, sub class, and super class in Java are explained as follows:

- Base class or parent class is a super class and is a class from which sub class or child class is derived.
- Sub class is a class that inherits attributes (properties) and methods (behavior) from the base class.

Q #29) Is Operator overloading supported in Java?

Answer: Operator overloading is not supported by Java as,

- It makes the interpreter put more effort to understand the actual functionality of the operator making code complex and difficult to compile.
- Operator overloading makes programs more error-prone.
- However, the feature of operator overloading can be achieved in method overloading in a simple, clear, and error-free way.

Q #30) When the finalize method is used?

Answer: finalize method is called just before the object is about to be garbage collected. This method overrides to minimize memory leaks, undertake cleanup activities by removing system resources.

Q #31) Explain about Tokens.

Answer: Tokens in the Java program are the smallest elements that the compiler recognizes. Identifiers, keywords, literals, operators, and separators are examples of tokens.