Object-Oriented Programming (OOPs)

Definition:
Object-Oriented Programming (OOPs) is a programming approach where programs are designed using objects. Objects are instances of classes, and they combine data (attributes) and behavior (methods) into a single unit.

4 pillars of OOPs are

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

Benefits:
- Promotes code reusability, modularity, and ease of maintenance.
- Helps in designing scalable and flexible software systems.

Class

- **Definition**: A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects created from the class will have.
- **Key Points**:
    - It does not consume memory until an object is created.
    - It is a logical structure used to define how objects should behave.

Object

- **Definition**: An object is an instance of a class. It is a real-world entity that has specific values for the attributes and can perform actions using the methods defined in the class.
- **Key Points**:
  - Objects have a data Member (data) and method (functions).
  - They are created from classes to represent actual entities in a program.

Access Modifiers

Access modifiers are keywords in OOP that set the visibility or accessibility level of classes, methods, and attributes. They help control which parts of the code can access specific classes, methods, or variables.

1. Public

- **Definition**: Any class, method, or attribute marked as `public` can be accessed from any other class.
- **Use Case**: When you want the member to be accessible by any part of your program.

2. Private

- **Definition**: The `private` access modifier restricts access to the class itself. Methods or attributes marked as `private` can only be accessed within the class they are defined in.

- **Use Case**: When you want to hide the internal implementation details and restrict access to sensitive data.

3. Protected

- **Definition**: The `protected` access modifier allows access within the same package (in Java) or module and by subclasses. It is more restrictive than `public` but less restrictive than `private`.
- **Use Case**: When you want to allow access to a class or method within its own package and by classes that inherit from it.

4. Default (Package-Private in Java)

- **Definition**: When no access modifier is specified, it is considered `default` (also known as `package-private` in Java). It allows access only within the same package.
- **Use Case**: When you want to limit access to classes and members within the same package but not from outside.

Summary:

- **Public**: Accessible from anywhere.
- **Private**: Accessible only within the same class.
- **Protected**: Accessible within the same package and by subclasses.
- **Default (Package-Private)**: Accessible only within the same package.

Constructor

- **Definition**: A constructor is a special method in a class that is automatically called when an object of the class is created. It is used to initialize the object's attributes and set up any necessary initial state.
- **Key Points**:
  - The name of the constructor is the same as the class name.
  - It does not have a return type, not even `void`.
  - It can be overloaded, meaning you can have multiple constructors with different parameters.
- **Purpose**: To set initial values for the attributes and perform any setup operations needed when an object is created.

Destructor

- **Definition**: A destructor is a special method that is automatically called when an object is destroyed or goes out of scope. It is used to clean up resources or perform any necessary finalization.
- **Key Points**:
  - The destructor’s name is the class name preceded by a tilde (~) symbol.
  - It does not take any arguments and does not return any value.

    - Destructors are not commonly used in languages with automatic garbage collection (like Java), but they are important in languages like C++ for manual memory management.
- **Purpose**: To release resources, such as memory or file handles, that the object may have acquired during its lifetime.

Pillar of OOPs

1. Abstraction

Abstraction is a way to hide the complicated details of how something works and show only the important parts. This makes things easier to understand and use.

Key Points:

- **Purpose**: To keep things simple by showing just what's necessary and hiding the complex details.
- **How It Works**:
    - **Abstract Classes**: These are classes that can't be used directly. They can have methods that need to be filled in by other classes.
    - **Interfaces**: These are like a list of methods that don't have any code. Any class using the interface has to write the code for those methods.
- **Example**: When you use a `Car` class, you don't need to know how the engine works; you just use the `startEngine()` method to start the car.

Benefits:

- **Simplifies Code**: Hides the details and makes it easier to use.
- **Easier to Maintain**: Changes can be made without affecting other parts of the code.
- **Reusable**: The same code can be used in different situations.

Encapsulation

Encapsulation is the concept of wrapping data (variables) and the methods (functions) that work on the data into a single unit, usually a class. It also means restricting direct access to some of an object's components, which helps prevent accidental changes and makes the code more secure.

Key Points:

- **Purpose**: To keep the data safe and hidden from outside interference and misuse.

- **How It Works**:
  - **Access Modifiers**: `private`, `protected`, and `public` keywords control how data is accessed. Variables are often marked `private` to hide them, and `public` methods (getters and setters) are used to access or modify the data safely.

- **Example**: A `BankAccount` class might have a `balance` variable set to `private` so that it can only be changed using a `public` method like `deposit()` or `withdraw()`.

Benefits:

- **Security**: Protects the internal state of an object from being changed in unexpected ways.
- **Control**: You can control how the data is accessed and modified through methods.
- **Simplicity**: Helps keep code organized by bundling data and the methods that use it.

Polymorphism

Polymorphism is the ability of a single function, method, or operator to work in different ways based on the context. In simple terms, it allows one interface to be used for different types of actions, making code more flexible and reusable.

Key Points:

- **Types of Polymorphism**:
  - **Compile-Time Polymorphism (Static Binding)**: This is achieved through **method overloading** and **operator overloading**. Method overloading means having multiple methods in the same class with the same name but different parameters.
  - **Run-Time Polymorphism (Dynamic Binding)**: This is achieved through **method overriding**. A subclass provides a specific implementation of a method that is already defined in its superclass.

- **Example**:
  - **Compile-Time**: A class might have multiple `add()` methods that take different numbers or types of arguments.

- **Run-Time**: A `Animal` class has a method `speak()`, and subclasses like `Dog` and `Cat` override `speak()` with their own versions, so calling `speak()` on an `Animal` object will run the appropriate version based on the actual object type (Dog or Cat).

Benefits:

- **Reusability**: The same method can be used for different types of objects, which reduces code duplication.
- **Flexibility**: Code can be written to handle different data types and objects using a single interface.
- **Scalability**: Easy to extend and add new functionality without changing existing code.

Summary:

Polymorphism is like a remote control that can operate different devices (TV, air conditioner, music system). The same button can work differently depending on which device it is controlling.

Method Overriding

Definition:
Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its parent class. The method in the subclass has the same name, return type, and parameters as the method in the parent class. This is a way to achieve **run-time polymorphism**.

Method Overloading

Definition:
Method overloading occurs when multiple methods in the same class have the same name but different parameters (different type, number, or both). The return type can be different but is not enough for method overloading by itself.

Methods with the same name must have a different parameter list (different type, number of parameters, or both).

It is a way to achieve **compile-time polymorphism**.

Inheritance

Inheritance is a fundamental concept in OOP where one class (called a subclass or child class) inherits properties and methods from another class (called a superclass or parent class). It allows the child class to use the code and behavior defined in the parent class, promoting code reusability and establishing a natural hierarchy between classes.

Key Points:

- **Reusability**: Inheritance allows a class to use the properties and methods of another class without rewriting the code.
- **Hierarchy**: It establishes a parent-child relationship between classes.
- **Access to Members**: The child class can access public and protected members of the parent class. Private members of the parent class are not accessible directly by the child class.
- **Types of Inheritance**:
    - **Single Inheritance**: A class inherits from one parent class.
    - **Multiple Inheritance**: A class inherits from more than one parent class (not supported directly in Java but possible in C++).
    - **Multilevel Inheritance**: A class is derived from a child class, making a chain of inheritance.

    - **Hierarchical Inheritance**: Multiple classes inherit from one parent class.
    - **Hybrid Inheritance**: A combination of two or more types of inheritance (can lead to complexity like the diamond problem in C++).

Benefits:

- **Code Reusability**: You don't need to write the same code multiple times; you can use the code from the parent class.
- **Organized Code**: Helps keep code structured and organized by grouping similar concepts together.
- **Extensibility**: Makes it easier to add new functionality to existing code without changing the parent class.

Questions & Answers

1. What is OOP?

Answer: OOP (Object-Oriented Programming) is a programming paradigm that uses objects and classes to design and build software. It is based on four main principles: Encapsulation, Abstraction, Inheritance, and Polymorphism.

2. What is a class?

Answer: A class is a blueprint or template for creating objects. It defines attributes (variables) and methods (functions) that the created objects will have.

3. What is an object?

Answer: An object is an instance of a class. It is a real-world entity with state (attributes) and behavior (methods).

4. What is encapsulation?

Answer: Encapsulation is the concept of bundling data (attributes) and methods that operate on the data into a single unit (class) and restricting direct access to some of the object's components to protect its integrity.

5. What is abstraction?

Answer: Abstraction is hiding the complex implementation details and showing only the essential features. It allows you to focus on what an object does rather than how it does it.

6. What is inheritance?

Answer: Inheritance is the mechanism where one class (subclass/child class) inherits properties and behaviors from another class (superclass/parent class). It promotes code reusability and helps in creating a hierarchical relationship between classes.

7. What is polymorphism?

Answer: Polymorphism is the ability of a method or function to operate in different ways depending on the context. It can be achieved through method overloading (compile-time) and method overriding (run-time).

8. What is method overloading?

Answer: Method overloading occurs when multiple methods in the same class have the same name but different parameters (different type or number). It is a form of compile-time polymorphism.

9. What is method overriding?

Answer: Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its parent class. It is a form of run-time polymorphism.

10. What are access modifiers?

Answer: Access modifiers are keywords that define the visibility of classes, methods, and variables. Common access modifiers are:

- `public`: Accessible from any class.
- `private`: Accessible only within the class.
- `protected`: Accessible within the package and by subclasses.
- Default (package-private): Accessible only within the same package.

11. What is a constructor?

Answer: A constructor is a special method in a class that is used to initialize objects. It is called automatically when an object is created and has the same name as the class.

12. What is a destructor?

Answer: A destructor is a special method used to clean up resources when an object is no longer needed. It is called automatically when an object is destroyed. (Note: In Java, garbage collection is used, so explicit destructors are not common.)

13. What is an interface?

Answer: An interface is a collection of abstract methods (methods without an implementation). A class that implements an interface must provide the behavior for all the methods defined in that interface.

14. What is an abstract class?

Answer: An abstract class is a class that cannot be instantiated and may have abstract methods (methods without an implementation) as well as concrete methods (methods with implementation). It serves as a blueprint for other classes.

15. What is the difference between an abstract class and an interface?

Answer:

- **Abstract Class**: Can have both abstract and concrete methods; can have constructors and instance variables.
- **Interface**: Can only have abstract methods (until Java 8, which allows default and static methods); cannot have constructors or instance variables.

16. What is the significance of the `super` keyword?

Answer: The `super` keyword is used to refer to the immediate parent class's methods and constructors. It is used to call the parent class's methods and to access parent class properties.

17. What is the `this` keyword?

Answer: The `this` keyword refers to the current instance of the class. It is used to differentiate instance variables from local variables when they have the same name and to call the current class's methods.

18. What is dependency injection?

Answer: Dependency injection is a design pattern that allows an object to receive its dependencies from an external source instead of creating them. This makes the code more modular, testable, and easier to manage.

19. What is a class variable?

Answer: A class variable is a variable that is shared among all instances of a class. It is declared using the `static` keyword.

20. What is an instance variable?

Answer: An instance variable is a variable that is unique to each instance of a class. It is used to hold the state of an object.

These questions and answers should help you prepare for your viva and give you a solid understanding of key OOP concepts.

21. What is the `super` keyword used for?

Answer: The `super` keyword is used to refer to the immediate parent class. It is used to call parent class methods and access parent class variables from a subclass.

22 .What is the difference between `==` and `equals()` in Java?

Answer:

- **`==`**: Checks if two references point to the same object in memory (reference comparison).
- **`equals()`**: Checks if the content of two objects is the same (content comparison). The `equals()` method must be overridden in custom classes for proper content comparison.

Java FAQs

1. Explain the concept of exception handling and why it is important in programming.

Answer: Exception handling is a mechanism in programming that allows a program to handle runtime errors gracefully without crashing. It uses `try`, `catch`, `throw`, `throws`, and `finally` blocks to catch exceptions and take appropriate action. Exception handling is important because it improves the robustness and reliability of a program by catching errors and maintaining program flow, preventing unexpected crashes and facilitating better error reporting.

2. Why did you choose to create a custom exception instead of using an existing Java exception?

Answer: Creating a custom exception is useful when you want to represent a specific type of error in your program that isn't adequately represented by the existing Java exceptions. Custom exceptions can provide more clarity and context, making the code easier to understand and maintain. It allows you to define unique error handling logic that is specific to the application domain.

3. Suppose the program needs to ignore specific numbers (e.g., zeroes) while reading from the file. How would you implement this feature?

Answer: You could use a `try-catch` block while reading the file and include a condition to skip processing when a zero is encountered. For example, if reading numbers from a file, you could use an `if` condition to check if the number is zero and continue to the next iteration without processing it.

4. What is the difference between checked and unchecked exceptions, and where does PositiveNumberException fall?

Answer:

- **Checked exceptions** are exceptions that are checked at compile-time. The programmer must handle these exceptions explicitly using `try-catch` blocks or declare them using `throws` in the method signature. Examples include `IOException` and `SQLException`.
- **Unchecked exceptions** are exceptions that are not checked at compile-time, meaning they do not need to be declared or handled explicitly. They typically inherit from `RuntimeException`, such as `NullPointerException` or `ArrayIndexOutOfBoundsException`.

PositiveNumberException would be considered a **checked exception** if it is meant to be handled explicitly by the programmer. If it's designed as an error that should be handled during the runtime without needing explicit handling, it could be an unchecked exception.

5. Describe how the `throws` keyword is used to declare that a method might throw an exception.

Answer: The `throws` keyword is used in a method signature to indicate that the method may throw one or more exceptions. This informs the compiler and the caller that the method might throw an exception that needs to be handled. For example:

```
public void readFile() throws IOException {
    // code that might throw IOException
}
```

This means the `readFile` method might throw an `IOException`, and any method calling `readFile` must handle or propagate this exception.

6. What is the difference between an abstract class and a concrete class?

Answer:

- **Abstract class**: A class that cannot be instantiated and may have abstract methods (methods without implementation) as well as concrete methods. It is used to provide a common base for other classes.
- **Concrete class**: A class that can be instantiated and has complete implementation for all its methods. It can extend an abstract class and provide implementations for its abstract methods.

7. What is the difference between a static and an instance variable?

Answer:

- **Static variable**: A variable that belongs to the class rather than to any specific instance. It is shared across all instances of the class and is accessed using the class name.
- **Instance variable**: A variable that belongs to a specific instance of a class. Each object has its own copy of the instance variables, and they are accessed through the object reference.

8. What is the advantage of an abstract class in Java?

Answer: Abstract classes provide a way to create a base class with shared code and enforce a contract for subclasses to follow (by defining abstract methods). They allow a combination of fully implemented methods and abstract methods, facilitating code reusability and enforcing a common structure in subclasses.

9. How does a LinkedList differ from an ArrayList in terms of performance for insertion and deletion operations?

Answer:

- **LinkedList**: Provides better performance for insertion and deletion operations, especially when adding or removing elements from the middle of the list, as it only requires updating the references between nodes (O(1) for insertion/deletion at the start or end).
- **ArrayList**: Has better performance for random access (O(1)) but slower for insertion and deletion, as it requires shifting elements to maintain the array structure (O(n) for insertion/deletion).

10. Discuss the concept of uniqueness in HashSet and how hashing ensures that each element is stored only once.

Answer: A `HashSet` is a collection that does not allow duplicate elements. It uses a hashing mechanism to ensure that each element is unique. When an element is added to a `HashSet`, its `hashCode()` is calculated and used to determine its position in the underlying hash table. If another element with the same hash code is inserted, the `HashSet` checks for equality using the `equals()` method to prevent duplicates.

11. How does the HashSet handle collisions when two different elements have the same hash code?

Answer: HashSet handles collisions using techniques such as chaining (linked lists) or open addressing. In Java, `HashSet` uses separate chaining, where each bucket in the hash table points to a linked list of elements that have the same hash code. When a collision occurs, the new element is added to the linked list of that bucket.

12. Can a HashSet maintain the order of elements? If not, which Java collection would you use if the order is important?

Answer: No, a `HashSet` does not maintain the order of its elements. If you need to maintain the order of elements, you should use `LinkedHashSet`, which maintains the order of insertion, or `TreeSet`, which sorts the elements in natural order or according to a specified comparator.

13. Describe scenarios where LinkedLists would be preferred, such as implementing queues, stacks, or maintaining a dynamic list of tasks.

Answer: `LinkedList` is preferred when:

- You need fast insertion and deletion operations, especially from the beginning or middle of the list.
- You are implementing data structures such as queues and stacks where elements are frequently added or removed.
- You need a dynamically sized list, as `LinkedList` can grow or shrink without the need for resizing, unlike `ArrayList`.

14. What is the main method in Java?

Answer: The `main` method is the entry point of a Java program. It is where the execution of a Java program begins. It has the following signature:

```
public static void main(String[] args)
```

- `public`: Makes the method accessible from anywhere.
- `static`: Allows the method to be called without creating an instance of the class.
- `void`: The method does not return any value.

- `String[] args`: A parameter that accepts command-line arguments as an array of `String`.

15. What is a package in Java?

Answer: A package is a namespace that organizes a set of related classes and interfaces. It helps in avoiding naming conflicts and provides access control. For example, `java.util` and `java.io` are built-in Java packages.

16. What are the main features of Java?

Answer:

- **Object-Oriented**: Supports classes and objects for code organization and reusability.
- **Platform-Independent**: Java code is compiled to bytecode, which can run on any platform using the Java Virtual Machine (JVM).
- **Simple**: Java is designed to be easy to learn and use.
- **Secure**: Provides built-in security features such as a security manager and access control.
- **Multithreaded**: Supports multithreading for concurrent execution.
- **Distributed**: Has built-in support for distributed computing, including network communication.
- **Robust**: Provides strong memory management with garbage collection.

17. What is the difference between JDK, JRE, and JVM?

Answer:

- **JDK (Java Development Kit)**: A software development kit that includes the JRE and development tools (e.g., compiler `javac`, debugger).
- **JRE (Java Runtime Environment)**: Provides the libraries, Java Virtual Machine (JVM), and other components needed to run Java applications.
- **JVM (Java Virtual Machine)**: An abstract machine that runs Java bytecode and provides features like garbage collection and memory management.

18. What are the predefined packages in Java?

Answer: Predefined packages in Java are standard libraries that provide various functionalities. Some commonly used packages include:

- **`java.lang`**: Contains fundamental classes like `String`, `Math`, `Object`, etc.
- **`java.util`**: Provides utility classes, such as `ArrayList`, `HashMap`, `Date`, etc.
- **`java.io`**: Used for input and output through data streams, serialization, etc.
- **`java.net`**: Provides classes for networking functionality.
- **`java.sql`**: Used for database connectivity.
- **`java.awt` and `javax.swing`**: Used for building graphical user interfaces.