Java Interface

In Java,

- An interface is a collection of abstract methods (methods without a body) and constant variables.
- It provides a way to achieve abstraction, define a contract for classes, and support multiple inheritance of behaviour.

Here are some key aspects of interfaces in Java and why they are required:

1. Abstraction:
    - Interfaces allow us to declare a set of method signatures without providing any implementation details. This promotes abstraction by defining what a class should do without specifying how it should do it.
    - Abstract methods in interfaces represent behaviours that implementing classes must provide.
2. Multiple Inheritance of Behaviour:
    - Java supports multiple inheritance through interfaces. A class can implement multiple interfaces, inheriting behaviour from each interface.
    - This is particularly useful when a class needs to exhibit characteristics of multiple types without being constrained to a specific class hierarchy.
3. Contract Specification:
    - Interfaces act as contracts, defining a set of methods that implementing classes must provide. Any class that implements an interface is obligated to implement all the abstract methods declared by that interface.
    - This ensures a consistent API and helps in designing modular and maintainable code.
4. Loose Coupling:
    - Interfaces allow for loose coupling between classes. A class can be designed to interact with another class through interfaces without needing to know the concrete implementation details of the other class.
    - This promotes flexibility and easier maintenance, as changes to the implementation of a class do not affect the classes that interact with it through interfaces.
5. Code Reusability:
    - Interfaces promote code reusability by providing a common set of methods that can be implemented by multiple classes. This helps in avoiding code duplication and encourages modular design.
6. Constants:
    - Interfaces can contain constant variables (implicitly public, static, and final). These constants are accessible by classes that implement the interface and can be used for configuration or other purposes.

Common Interview Question :

What is an interface and explain with an example?

Answer :

- An interface is a specification of method prototypes. It means that the methods are incomplete in nature in an interface.
- All the methods of the interface are public and abstract by default.

Example : Consider a scenario where we are writing a program to connect different database, retrieve data and process it. Then close the database.

Consider there are two different databases.

We can use the concept of class and create two different classes for each database and write the functionality.

For example,

```
class OracleDBOperation
{
    void connectToDatabse()
    {

    }
    void disconnectToDatabase()
    {

    }
}
```

```
class HanaDBOperation
{
    void connectToDatabse()
    {

    }
    void disconnectToDatabase()
    {

    }
}
```

This approach has a limitation because each individual class is meant for specific database operation. Consider, if there are many different databases then we should create different classes.

The concept of Interface helps us to solve this problem.

```
interface DBOperationInterface
{
    void connectToDatabase();
    void disConnectToDatabase();

}
```

```java
class OracleDBOperation implements DBOperationInterface
{

    @Override
    public void connectToDatabase() {
        // TODO Auto-generated method stub
        System.out.println("Connect to Oracle Database and process the data");
    }

    @Override
    public void disConnectToDatabase() {
        // TODO Auto-generated method stub
        System.out.println("Disconnect from Oracle Database");
    }

}
```

```java
class HanaDBOperation implements DBOperationInterface{

    @Override
    public void connectToDatabase() {
        // TODO Auto-generated method stub
        System.out.println("Connect to SAP HANA Database and process the data");
    }

    @Override
    public void disConnectToDatabase() {
        // TODO Auto-generated method stub
        System.out.println("Disconnect from SAP HANA Database");
    }

}
```

Now, different third party will provide implementation class and implement the interface methods specific to their database requirement.

Why the methods of an interface are public and abstract by default?

Answer :

- Interface methods are public because it should be available to third party vendors to provide implementation.
- Interface methods are abstract because implementation is left for the third-party vendors.

Let's deep dive into the interface methods and variables :

In Java, interface methods are public and abstract by default. This is by design, and it serves specific purposes related to the nature and use of interfaces.

Public by Default:

- Interfaces are meant to define contracts that specify what a class implementing the interface must provide.
- Making interface methods public ensures that the methods are accessible from outside the interface, allowing implementing classes to adhere to the contract and providing a clear, consistent API.

Abstract by Default:

- Interfaces cannot be instantiated on their own; they are meant to be implemented by classes. As a result, interface methods do not have a default implementation.
- Making interface methods abstract enforces the requirement that implementing classes must provide their own concrete implementation for each method declared in the interface.

Implicitly Static and Final Variables:

- In addition to methods being public and abstract by default, interface variables (fields) are implicitly public, static, and final.
- This is to ensure that interface constants can be accessed using the interface name and to support the creation of constants within interfaces.

Multiple Inheritance using Interface in Java :

We know that in multiple inheritance, sub classes are derived from multiple super classes. If two of the super classes have same names for their members like variables and methods, then which member is inherited to the sub class is the main confusion in multiple inheritance.

This is the reason for java does not supports multiple inheritance.

```
class A {
    void method() {
        System.out.println("Method of class A");
    }
}

class B extends A {
    void method() {
        System.out.println("Method of class B");
    }
}

class C extends A {
    void method() {
        System.out.println("Method of class C");
    }
}

class D extends B, C { // Hypothetical syntax for multiple inheritance
}

public class Example {
    public static void main(String[] args) {
        D d = new D();
        d.method();
    }
}
```

Which implementation of method() should be called?
Is it the one from class B or class C?

Below is the example of multiple inheritance using interface in java –

```
interface Father
{
    float height = 5.8f;
    void height();
}
interface Mother
{
    float height = 5.2f;
    void height();
}
class Child implements Father, Mother
{

    @Override
    public void height() {
        float heightOfChild = (Father.height + Mother.height)/2;
        System.out.println("Height of Child is " + heightOfChild);
    }

}
public class MultiPleInheritance {
    public static void main(String args[])
    {
        Child ch = new Child();
        ch.height();
    }
}
```

Difference between Abstract Class and Interface :

Abstract Class	Interface
A class with one or more abstract methods.	A collection of abstract methods and constant fields.
Can have constructors.	Cannot have constructors.
Can have instance variables (fields).	Can only have constants (public, static, final fields).
Can have different access modifiers (public, protected, private).	Have only public access modifier.
Can have abstract methods (methods without a body).	All methods are implicitly abstract.
Can have concrete (implemented) methods.	Cannot have concrete methods.
Supports single inheritance (extends one class).	Supports multiple inheritance (implements multiple interfaces).
Used when there are some common features shared by all the objects.	Used when all the features are implemented differently in different objects.
When an abstract class is written, it is the duty of programmer to provide a sub class to it.	An interface is written when the programmer wants to leave the implementation to the third-party vendors.

Let's summarize few important points of "interface" in java –

- An interface is a specification of method prototypes.
- Only method names are written in the interface without method body.
- Interface methods are by default abstract and public.
- Variables of the interface are public, static, and final by default.
- Variables of the interface are constants.
- Interface supports only public access modifier.
- All the methods of interface should be implemented in its implementation class.
- Objects are not created for the interface.
- An interface can extend another interface.
- An interface cannot implement another interface.
- It is possible to write a class within an interface.