**INTERFACE**

=========

**interface** is a reference type, similar to a class, that defines a contract for ***what* a class can do, without specifying *how* it should do it**. In otherwords **interface** in Java is a completely abstract type that is used to specify a set of methods that a class must implement.

1. interface can contain

- *Abstract Methods*: Methods with no implementation.

- *Default Methods*: Methods with implementation.

- *Static Methods*: Methods that belong to the interface itself.

- *Variables*: are constants, implicitely **public static final**.

2. interface definition: use keyword **interface**

**interface** **iface\_name** {

// Abstract methods

// Default methods

// Static methods

// Constants

}

3. Usage: use keyword **implements**

*class Implementing\_class implements* ***iface\_name*** *{*

*// Implement abstract methods*

*}*

**abstract methods** : Must implement all abstract methods in implementing class

**default methods**  : inherited and available in implemeting class,

so not required override, but can override if needed.

**static methods** : NOT inherited to impemeting class.

so, do not implement static methods in implemeting class.

Can access using interface name just like any static method.

**constants** : are public, so classes can refer directly like a class variable.

*interface Animal {*

*// Abstract method - no implementation*

*void sound();*

*// default function - with implementation*

*default void walk() {*

*System.out.println("Animal is walking");*

*}*

*// static function*

*static void info() {*

*System.out.println("This is an animal.");*

*}*

*// constant - implicitely qualified as public static final*

*int LEGS = 4;*

*}*

*// implementing class...!*

*class Dog implements Animal {*

*// Must implement this method, else gives compilation error*

*public void sound() {*

*System.out.println("Dog barks");*

*}*

*public int getLegs() {*

*// inherited, hence use directly..!*

*return LEGS;*

*}*

*}*

*public class Main {*

*public static void main(String[] args) {*

*// Calling the static method from the interface*

*Animal.info();*

*// Calling the default method from the interface*

*Dog myDog = new Dog();*

*myDog.walk();*

*// Can access constant using interface name*

*int noOfLegs = Animal.LEGS;*

*}*

*}*

4. Any number of classes can implement an interface.

5. interface can extend another interface

6. interface can NOT implement another interface

7. interface can NOT extend another class

9. **One class can implement any number of interfaces**.

– This helps to achieve **multiple inheritance** in Java.

Eg, Multiple Inheritance in java

*interface Animal {*

*void eat();*

*}*

*interface Bird {*

*void fly();*

*}*

*class Bat implements Animal, Bird {*

*// Implementing abstract method from Animal interface*

*public void eat() {*

*System.out.println("Bat eats insects.");*

*}*

*// Implementing abstract method from Bird interface*

*public void fly() {*

*System.out.println("Bat can fly.");*

*}*

*}*

*public class Main {*

*public static void main(String[] args) {*

*Bat bat = new Bat();*

*bat.eat(); // From Animal interface*

*bat.fly(); // From Bird interface*

*}*

*}*

**Practical Scenario**

In an application managing different types of vehicles (e.g., cars, bikes, trucks), an interface Vehicle can be created that all vehicle types implement. This allows consistent method names like start() and stop() across different classes.

*interface Vehicle {*

*void start();*

*void stop();*

*}*

*class Car implements Vehicle {*

*public void start() {*

*System.out.println("Car started");*

*}*

*public void stop() {*

*System.out.println("Car stopped");*

*}*

*}*

*class Bike implements Vehicle {*

*public void start() {*

*System.out.println("Bike started");*

*}*

*public void stop() {*

*System.out.println("Bike stopped");*

*}*

*}*

**Question**

Show how the following inheritance hierarchy can be implemented in Java.

\_\_\_\_\_

|\_A\_\_|

\_\_\_\_/ \\_\_\_\_\_

|\_B\_\_| |\_C\_\_|

\\_\_\_\_\_/

|\_D\_\_|

D inheriting from B & C , indicating multiple inheritance, not supported in java.

Hence B can be a Class and C can be an Interface,

or vice-versa or both are interfaces.

Let B be a class and C be an Interface.

B and C inherit from A, where C is an interface.

Interface C can not inherit a class.

Hence A must be another Interface, so that C can extend Interface A

*interface A {*

*void f1();*

*}*

*class B implements A {*

*//implement A interface function*

*public void f1() {*

*System.out.println("Class B: f1");*

*}*

*}*

*interface C extends A {*

*void f2();*

*}*

*class D extends B implements C {*

*//implement C interface function*

*public void f2() {*

*System.out.println("Class D: f2");*

*}*

*}*

*public class ITest {*

*public static void main(String[] args) {*

*D d = new D();*

*d.f1(); // extended function from B*

*d.f2(); // implemented function from C*

*}*

*}*