

UNIT III – INHERITANCE AND POLYMORPHISM

INTERFACES AND MULTIPLE INHERITANCE THROUGH INTERFACES:

INTERFACES:

- An interface in Java is a collection of abstract methods that defines a contract which implementing classes must follow.
- An interface specifies what a class must do but not how it should do it.
- A class uses the implements keyword to provide code for all the methods declared in the interface.

RULES OF INTERFACES:

- All methods in an interface are:
Public, Abstract (no body, unless default or static)
- Variables in an interface are always:
public, static, final (constants).
- A class uses the implements keyword to use an interface.
- A class can implement multiple interfaces.
- Interfaces themselves can extend other interfaces.

TYPES OF METHODS IN INTERFACES:

- Before Java 8
Only abstract methods (no body).
- From Java 8
 - ✓ Default methods → have a body (used to add new methods to interfaces without breaking old code).
 - ✓ Static methods → can be called without an object of implementation class.(c)
- From Java 9
Private methods → used inside interfaces to avoid code duplication.

EXAMPLE PROGRAMS:

```
interface Animal          // Define an interface
{
    void sound(); // abstract method
    void eat();   // abstract method
}
```

```
// Dog implements Animal interface
class Dog implements Animal
{
    public void sound()
    {
        System.out.println("Dog says: Woof Woof!");
    }
    public void eat()
    {
        System.out.println("Dog eats bones.");
    }
}

class Cat implements Animal // Cat implements Animal interface
{
    public void sound()
    {
        System.out.println("Cat says: Meow Meow!");
    }
    public void eat()
    {
        System.out.println("Cat drinks milk.");
    }
}

class Cow implements Animal // Cow implements Animal interface
{
    public void sound()
    {
        System.out.println("Cow says: Moo Moo!");
    }
    public void eat()
    {
```

```
        System.out.println("Cow eats grass.");
    }
}
// Main class
public class AnimalInterfaceDemo
{
    public static void main(String[] args)
    {
        Animal a1 = new Dog();
        Animal a2 = new Cat();
        Animal a3 = new Cow();
        a1.sound();a1.eat();
        a2.sound();a2.eat();
        a3.sound();a3.eat();
    }
}
```

Output:

```
D:\Santhi\Java Programs>javac AnimalInterfaceDemo.java
D:\Santhi\Java Programs>java AnimalInterfaceDemo
Dog says: Woof Woof!
Dog eats bones.
Cat says: Meow Meow!
Cat drinks milk.
Cow says: Moo Moo!
Cow eats grass.
D:\Santhi\Java Programs>_
```

\\Both interfaces define the same default method

interface A

```
{
    default void show()
    {
        System.out.println("Default show from A");
    }
}
```

interface B

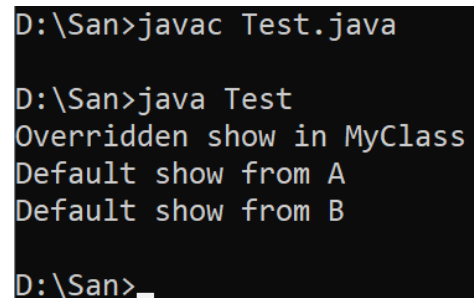
```
{
    default void show()
    {
        System.out.println("Default show from B");
    }
}
```

class MyClass implements A, B

```
{
    public void show()          // Must override to resolve conflict
    {
        System.out.println("Overridden show in MyClass");
        // Optionally call specific interface methods:
        A.super.show(); // Calls A's version
        B.super.show(); // Calls B's version
    }
}
```

public class Test

```
{
    public static void main(String[] args)
    {
        MyClass obj = new MyClass();
        obj.show();
    }
}
```



```
D:\San>javac Test.java
D:\San>java Test
Overridden show in MyClass
Default show from A
Default show from B
D:\San>_
```

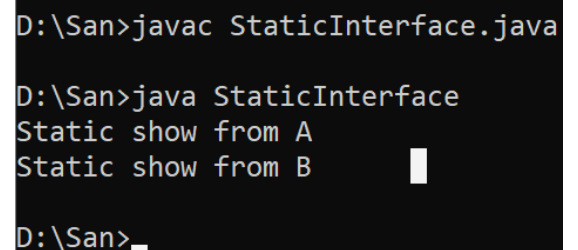
Multiple Interfaces with Same Static Method

```
interface A
{
    static void show()
    {
        System.out.println("Static show from A");
    }
}

interface B
{
    static void show()
    {
        System.out.println("Static show from B");
    }
}

class MyClass implements A, B
{
    // No conflict, since static methods are not inherited
}

public class StaticInterface
{
    public static void main(String[] args)
    {
        A.show();
        B.show();
    }
}
```



```
D:\San>javac StaticInterface.java
D:\San>java StaticInterface
Static show from A
Static show from B
D:\San>
```

Static Methods:

- Not inherited → must be called using InterfaceName.method().
- Cannot be overridden in implementing classes.
- If two interfaces have same static method → no conflict (call by interface name).
- Static methods are mostly used for utility/helper functions inside interfaces.

Comparison Between Abstract Class and Interfaces:

Feature	Abstract Class	Interface
Keyword	abstract	interface
Methods	Can have abstract + concrete methods	Before Java 8: only abstract methods; From Java 8: abstract + default + static methods
Variables	Can have instance variables (normal)	All variables are public static final (constants)
Constructors	Can have constructors	Cannot have constructors
Access Modifiers for methods	Can be public, protected, private	All methods are public (by default)
Inheritance	A class can extend only one abstract class (single inheritance)	A class can implement multiple interfaces (multiple inheritance)
Use case	When classes share a common base with some default behavior	When you need to define a contract for multiple unrelated classes
Example in real life	Vehicle as abstract class with start() and stop() methods	Drivable, Flyable, Swimmable interfaces implemented by different classes