Objectives

Click to Continue

This session will help you to understand :

- What are Interfaces?
- When to go for an interface?
- Usage of interfaces
- Difference between interface and abstract classes.

What is an Interface

Click to Continue

USB is an interface in a computer through which any of the **USB** compliant device can be connected.

In software world **Interface** is nothing but the protocol or the contract with which software components interact.

Interface is a class where **all methods** are **abstract**, the class should be declared as an ***Interface***.

This defines the **contract** for classes to follow. These cannot be instantiated.

Interface Analogy

Click to Continue

Real World Scenario: I In Aadhar card application the center defines the rules for applying aadhar card, and these rules will be followed by all regional offices in the states.

Centre: Defines the rule that during aadhar card registration the citizen needs to provide a photo identity.

Interface Analogy

Click to Continue

Real World Scenario: I In Aadhar card application the center defines the rules for applying aadhar card, and these rules will be followed by all regional offices in the states.

Centre: Defines the rule that during aadhar card registration the citizen needs to provide a photo identity.

Kolkatta: Adheres to the process and collects PAN card as photo identity.

Interface Analogy

Click to Continue

Real World Scenario: I In Aadhar card application the center defines the rules for applying aadhar card, and these rules will be followed by all regional offices in the states.

Mumbai,
- Adheres to the process and it collects Passport

Centre: Defines the rule that during aadhar card registration the citizen needs to provide a photo identity.

Kolkatta: Adheres to the process and collects PAN card as photo identity.

Interface Analogy

Click to Continue

Similarly **Centre** can define **multiple rules**, which represents the **multiple abstract methods** in the **interface**. All methods needs to be implemented by the sub classes.

Develop an Interface

Click to Continue

Interface creation is done by replacing the '**class**' keyword with the '**interface**' keyword

- To create a Automobile interface, instead of

public class Automobile

public interface Automobile

Step 1: Let us create the Interface?

Use the interface keyword.

```
public interface Automobile {

    public void honk();
    public void applyBrake();

}
```

Abstract Methods

Step 2 – Class Implementing Interface.

Click to Continue

Create a class which implements the interface using the '**implements**' keyword

Uses implements keyword to implement interface.

```
public class Car implements Automobile {

    @Override
    public void honk() {
        // TODO Auto-generated method stub

    }

    @Override
    public void applyBrake() {
        // TODO Auto-generated method stub

    }

}
```

The abstract method are implemented

Interface & Class

Click to Continue

Salient Points of interfaces:

- All the methods in interface are abstract.
- You cannot create an instance of an interface. The below code throws an error.

  IRBIBank bank = new IRBIBank(); throw an compilation error.

- An interface can be implemented by other classes or extended by other interfaces.

Salient Points of classes implementing interfaces:

- Class can implement any number of interfaces.
- Classes should implement all the methods of the interface(s) else an error is thrown at compile time.
- Implementing class can also extend a class or abstract class.

Why Interfaces?

To implement **multiple inheritance**,

A class can **implement multiple interfaces** while it can **extend** only **one class**.

```
public class Car implements Automobile, RoadAutomobile {
```

Here the class **Car** implements two interfaces **Automobile** and **RoadAutomobile.**

Class **Car** should implement all the methods declared in both the interfaces.

Java **does not support multiple inheritance** of classes. It can be indirectly achieved my **implementing multiple interfaces**.

Try It Out - Interfaces

Click to Continue

Let us simulate a banking Application: RBI bank dictates the process and rules of other banks. Assume ABC is a bank primarily focusing on Saving account and XYZ is another bank which provides home loan and saving account services is a subsidiary of ABC bank.

1. Create an interface **IRBIBank** with the below methods:

   openAccountProcess();

   closeAccountProcess();

   checkPaymentProcess();

2. Create a class **ABCBank.java** which implements **IRBIBank** . Add SOP for the method implementation :

   Example: System.out.println("Open Saving account")

Interface

```
package com.demo.Interface;

public interface IRBIBank {
    public void openAccountProcess();

    public void closeAccountProcess();

    public void checkPaymentProcess();
}
```

Class

```
package com.demo.Interface;

public class ABCBank implements IRBIBank {

    @Override
    public void openAccountProcess() {
        System.out.println("Open Savings Account...");
    }

    @Override
    public void closeAccountProcess() {
        System.out.println("Close Savings Account...");

    }

    @Override
    public void checkPaymentProcess() {
        System.out.println("Check Payment Process...");

    }

}
```

Interfaces & Abstract Classes

Click to Continue

Interface	Abstract Classes
All methods are abstract.	Abstract classes can have both concrete and abstract methods.
Constants that are defined in interfaces are by default static final.	Fields declared in abstract classes values can be changed.
A class can implement any number of interfaces.	A class can extend only one abstract class.
Interface can be 'implemented'.	Abstract classes needs to be extended.

Interface Usage

Always declare the reference variable with the interface as mentioned below.

Illustration: Assume class ***Car*** implements the ***IAutoMobile*** interface.

You should not declare using the class

```
Car carObject = new Car();
```

You should declare using the interface

```
IAutoMobile carObject = new Car();
```

Try It Out – Interface reference

Click to Continue

Let us reuse the banking interfaces created in the previous slide.

- Develop a class ***BankManager*** with a main method.
- In the main method create object of ***ABCBank*** and invoke the ***openAccountProcess*** and ***closeAccountProcess***

Don't forget to declare using the interface

Interface

```
package com.interfaces;

public interface IRBIBank {
    public void openAccountProcess();
    public void closeAccountProcess();
}
```

Class

```
package com.interfaces;

public class BankAccountManager {

    public static void main(String args[])
    {
        IRBIBank abcBank = new ABCBank();
        abcBank.closeAccountProcess();
    }

}
```

A problem statement

Click to Continue

In the bank scenario, if there are 200 bank classes which has implemented RBI Bank interface.

Assume RBI banks comes up with a new process for transferring loan which is applicable for all new banks opened from today.

Problem: Now if we add a new method **transferLoanProcess** in **RBIBank** interface. All the existing 200 Bank classes needs to change and implement the new method **transferLoanProcess**.

How can we avoid this change?

Solution is Interface Inheritance

Create a new interface **IRBISubDivision** which will extend **IRBIBank**. Now any classes which needs to adopt the new process (**transferLoanProcess**) will implement **IRBISubDivision**.

All the existing 200 Banks will not have any problems as they will extend **IRBIBank** which will not have method **transferLoanProcess**.

```
package com.interfaces;

public interface IRBISubDivision extends IRBIBank {

    public void tranferLoanProcess();

}
```

Interface extending another interface

```
package com.demo.Interface;

public class ABCBank implements IRBIBank {

    @Override
    public void openAccountProcess() {
        System.out.println("Open Savings Account...");
    }

    @Override
    public void closeAccountProcess() {
        System.out.println("Close Savings Account...");

    }

    @Override
    public void checkPaymentProcess() {
        System.out.println("Check Payment Process...");
```

ABCBank which is a existing bank need not implement the method transferLoanProcess as it implements IRBIBank

All new banks origination will have to implement **IRBIDivision**. So it should implement the **IRBIBank** and **IRBIDivision** interfaces methods.