

Core Java

Lesson 07 :Interfaces



Lesson Objectives

- After completing this lesson, participants will be able to:
 - Understand concept of Interfaces
 - Declaring interfaces
 - Extending interfaces
 - Default and static methods in interface
 - Differentiate between abstract classes and interfaces
 - Loose coupling using interfaces
 - Interface-Rules
 - Abstract Classes and Interfaces
 - Runtime Polymorphism





Interface

- Special kind of class which consist of only the constants and the method signatures.
- Approach also known as “programming by contract”.
- It’s essentially a collection of constants and abstract methods.
- It is used via the keyword “implements”. Thus, a class can be declared as follows:

```
class MyClass implements MyInterface{  
    ...  
}
```



What is Interface?

- A Java interface definition looks like a class definition that has only abstract methods, although the abstract keyword need not appear in the definition

```
public interface Testable {  
    void method1();  
    void method2(int i, String s);  
    int x=10;  
}
```

note no
implementation for
the methods, public
by default

Static final variable



Declaring and Using Interfaces

```
public interface SimpleCalc {
```

```
    int add(int a, int b);
```

abstract method

```
    int i = 10;
```

By default is public, static and final

```
}
```

//Interfaces are to be implemented.

```
class Calc implements SimpleCalc {
```

```
    int add(int a, int b){
```

```
        return a + b;
```

```
    }
```

```
}
```



Demo

- Execute the program Interface Implementation.java





7.3: Extending Interfaces

Extending Interfaces

- An interface can extend another interface in the same way that a class can extend another class.
- The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.
- The following Sports interface is extended by Hockey and Football interfaces.

```
// Filename: Sports.java
public interface Sports {
    public void setHomeTeam(String name);
    public void setVisitingTeam(String name);
}
```



Extending Interfaces

```
// Filename: Football.java
public interface Football extends Sports {
    public void homeTeamScored(int points);
    public void visitingTeamScored(int points);
    public void endOfQuarter(int quarter);
}
```

class that implements Football needs to define the three methods from Football and the two methods from Sports.

```
// Filename: Hockey.java
public interface Hockey extends Sports {
    public void homeGoalScored();
    public void visitingGoalScored();
    public void endOfPeriod(int period);
    public void overtimePeriod(int ot);
}
```

class that implements Hockey needs to implement all six methods.



Default Methods

- Starting from Java SE 8, interfaces can define default methods
- A default method in an interface is a method with implementation
- Use “default ” keyword in method signature to make it default.

```
interface xyz {  
    default return-type method-name(argument-  
list) {  
        -----  
        -----  
    }  
}
```

- A class which implements the interface doesn't need to implement default methods



Static Methods

- Along with the default methods an Interface can also have static methods
- The syntax of static method is similar to default method, where static keyword will replace default

```
interface xyz {  
    static return-type method-name(argument-list)  
    {  
        -----  
        -----  
    }  
}
```



7.4: Default and static

Demo

- Interface with default and static methods





7.5: Loose coupling using interface

Loose coupling using Interface

➤ Consider the below example.

- In below example we have two classes A and B

```
package com.capgemini.loosecoupling;
public class A {

    void display(InterfaceClass obji)
    {
        obji.display();

        System.out.println(obji.getVar())
    ;
    }
}
```



Loose coupling using Interface

```
package com.capgemini.loosecoupling;

public class B implements InterfaceClass{

    private String var="variable Interface";

    public String getVar() {
        return var;
    }

    public void setVar(String var) {
        this.var = var;
    }

    @Override
    public void display() {
        // TODO Auto-generated method stub
        System.out.println("Display Method Called");
    }

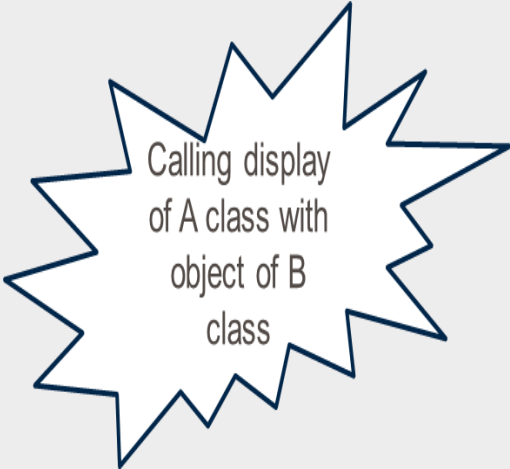
}
```



Loose coupling using Interface

```
package com.capgemini.loosecoupling;  
  
public interface InterfaceClass {  
    void display();  
    String getVar();  
}
```

```
package interface_package.loose_coupling;  
  
public class MainClass {  
    public static void main(String[] args) {  
        // TODO Auto-generated method  
        stub  
  
        A obja=new A();  
        B objb=new B();  
        obja.display(objb);  
    }  
}
```



Calling display
of A class with
object of B
class



Loose coupling using Interface

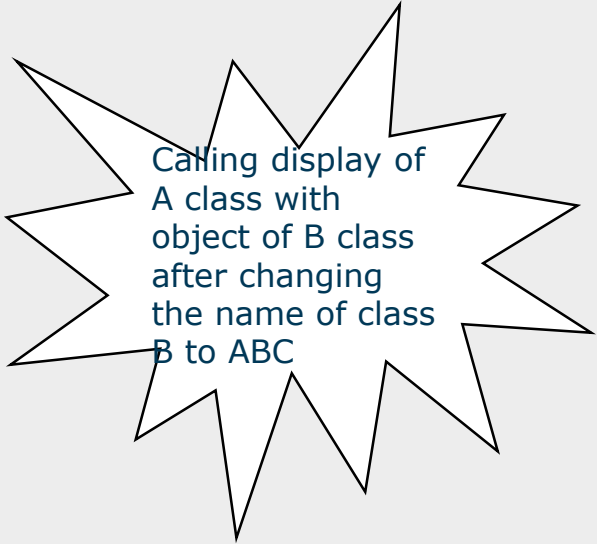
- Loose Coupling: suppose in future we have to change the name of Class B to ABC then we do not have to change its name in display method of class B, just make the object of new (ABC class) and pass it to the display method in MainClass. We do not have to change anything in Class A

```
package interface_package.loose_coupling;
public class MainClass {

    public static void main(String[] args) {
        // TODO Auto-generated method

        stub

        A obja=new A();
        ABC objb=new ABC();
        obja.display(objb);
    }
}
```



Calling display of
A class with
object of B class
after changing
the name of class
B to ABC



Interface - Rules

- Methods other than default and static in an interface are always public and abstract.
- Static methods in interface are always public .
- Data members in a interface are always public, static and final.
- Interfaces can extend other interfaces.
- A class can inherit from a single base class, but can implement multiple interfaces.



Abstract Classes and Interfaces

Abstract classes	Interfaces
Abstract classes are used only when there is a “is-a” type of relationship between the classes.	Interfaces can be implemented by classes that are not related to one another.
You cannot extend more than one abstract class.	You can extend more than one interface.
Abstract class can contain abstract as well as implemented methods.	Interfaces contain only abstract, default and static methods.
With abstract classes, you grab away each class’s individuality.	With Interfaces, you merely extend each class’s functionality.



Runtime Polymorphism

- Runtime polymorphism enables a method can do different things based on the object used for invoking method at runtime
- Runtime polymorphism is implemented by doing method overriding

```
class Parent {  
    public String sayHello() {  
        return "Hello from  
Parent";  
    }  
}  
  
class Child extends Parent {  
    public String sayHello() {  
        return "Hello from  
Child";  
    }  
}
```

```
Parent object = new  
Child();  
object.sayHello();
```



Hello from
Child



Accessing Implementations through Interface Reference

```
class sample implements TestInterface {  
    // Implement Callback's interface  
    public void interfacemethod() {  
        System.out.println("From interface method"); }  
    public void noninterfacemethod() {  
        System.out.println("From interface method"); }  
}
```

```
class Test {  
    public static void main(String args[]) {  
        TestInterface t = new sample();  
        t.interfacemethod()    //valid  
        t.noninterfacemethod() //invalid }  
}
```



7.8: Runtime Polymorphism

Demo

➤ Runtime polymorphism



Lab

➤ Lab 6: Interfaces





Summary

➤ In this lesson, you have learnt about:

- Interfaces
- default methods
- static methods on Interface
- Runtime Polymorphism





Review Question

- Question 1: All variables in an interface are :
 - **Option 1:** Constant instance variables
 - **Option 2:** Static and final
 - **Option 3:** Constant instance variables
- Question 2: Will this code throw a compilation error?

```
interface sample
{
    int x;
}
```

- **Option 1:** True
- **Option 2:** False

