Interfaces

Agenda

Introduction to Interfaces

What is an Interface?

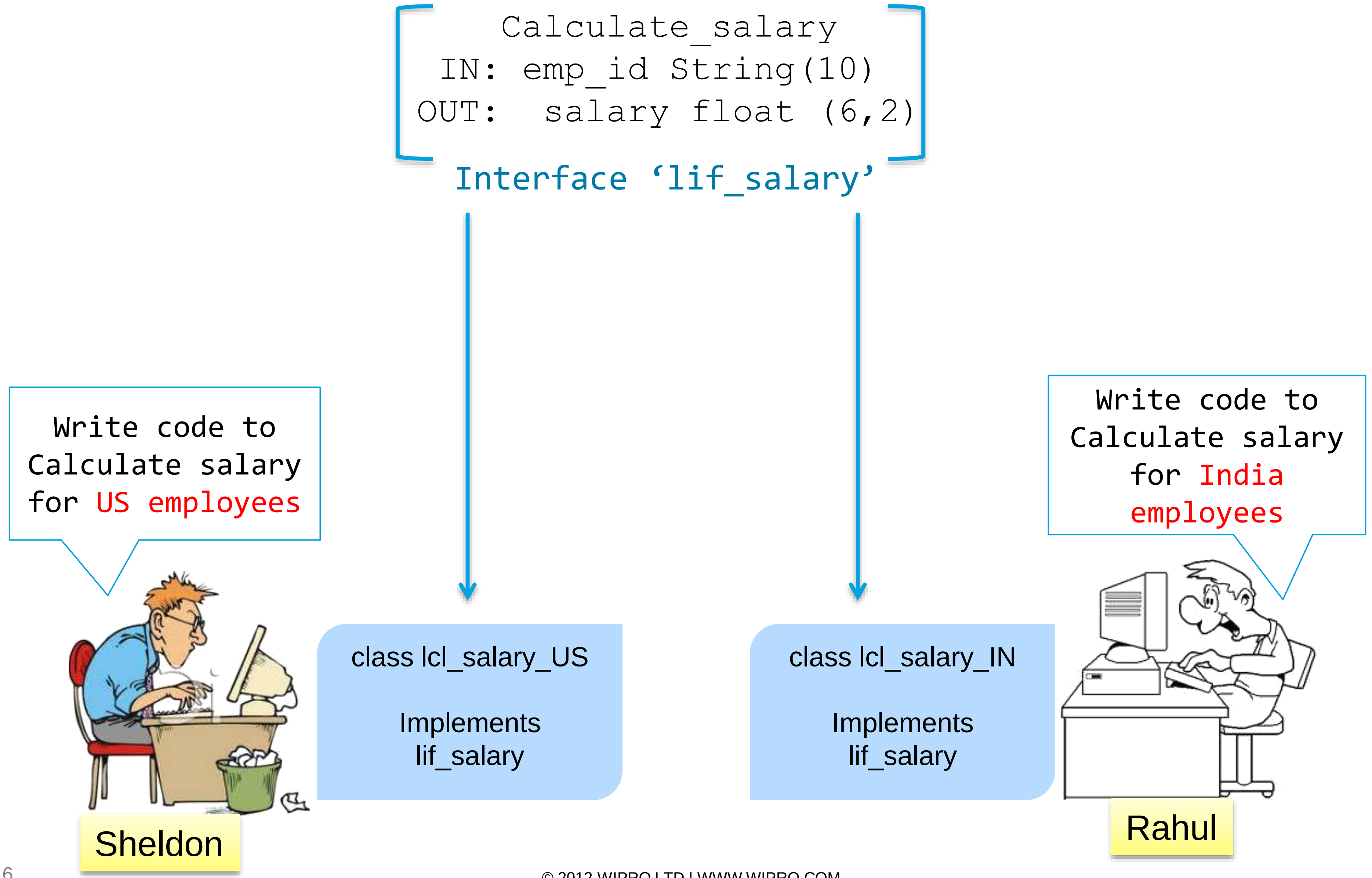
An interface is a named collection of method declarations (without implementations)

- An interface can also include constant declarations
- An interface is syntactically similar to an abstract class
- An interface is a collection of abstract methods and final variables
- A class implements an interface using the **implements** clause

What is an Interface? (Contd.).

- An interface defines a protocol of behavior
- An interface lays the specification of what a class is supposed to do
- How the behavior is implemented is the responsibility of each implementing class
- Any class that implements an interface adheres to the protocol defined by the interface, and in the process, implements the specification laid down by the interface

Interface: Example


Calculate_salary
IN: emp_id String(10)
OUT: salary float (6,2)
Interface 'lif_salary'
Write code to Calculate salary for US employees
Write code to Calculate salary for India employees
class lcl_salary_US Implements lif_salary
class lcl_salary_IN Implements lif_salary
Sheldon
Rahul

Why interfaces are required ?

- Interfaces allow you to implement common behaviors in different classes that are not related to each other

- Interfaces are used to describe behaviors that are not specific to any particular kind of object, but common to several kind of objects

Why interfaces are required ? (Contd.).

- Defining an interface has the advantage that an interface definition stands apart from any class or class hierarchy
- This makes it possible for any number of independent classes to implement the interface
- Thus, an interface is a means of specifying a consistent specification, the implementation of which can be different across many independent and unrelated classes to suit the respective needs of such classes
- Interfaces reduce coupling between components in your software

Why interfaces are required ? (Contd.).

- Java does not support multiple inheritance
- This is a constraint in class design, as a class cannot achieve the functionality of two or more classes at a time
- Interfaces help us make up for this loss as a class can implement more than one interface at a time
- Thus, interfaces enable you to create richer classes and at the same time the classes need not be related

Interface members

- All the methods that are declared within an interface are always, by default, public and abstract

- Any variable declared within an interface is always, by default, public static and final

What will you choose..?

What is the behavior which is common among the entities depicted in the pictures above?

Yes..You are right. All of them can fly.

Requirement : You have to develop 3 classes, Bird, Superman and Aircraft with the condition that all these classes must have a method called fly().

What is the mechanism, using which you can ensure that the method fly() is implemented in all these classes?

An Abstract class or An Interface?

Defining an Interface

- An interface is syntactically similar to a class
- It’s general form is:

```
public interface FirstInterface {
  int addMethod(int x, int y);
  float divMethod(int m, int n);
  void display();
    int VAR1 = 10;
    float VAR2 = 20.65;
  }
```

Implementing Interfaces

- A class implements an interface
- A class can implement more than one interface by giving a comma- separated list of interfaces

```
class MyClass implements FirstInterface{
 public int  addMethod(int a, int b){
   return(a+b);
 }

 public float divMethod(int i, int j){
   return(i/j);
 }

 public void display(){
   System.out.println(“Variable 1 :” +VAR1);
   System.out.println(“Variable 2 :” +VAR2);
 }
}
```

Quiz

Will the following code compile successfully ?

```
interface I1 {
     private int a=100;
     protected void m1();
}

class A1 implements I1 {
     public void m1() {
     System.out.println("In m1 method");
     }
}
```

It will throw compilation errors.. Why?

Quiz (Contd.).

Will the following code compile successfully ?

```
interface I1 {
     static int a=100;
     static void m1();
}

class A1 implements I1 {
     public void m1() {
     System.out.println(“In m1 method”);
     }
}
```

It will throw compilation error.. Why?

Applying Interfaces

Applying Interfaces

- Software development is a process where constant changes are likely to happen
- There can be changes in requirement, changes in design, changes in implementation
- Interfaces support change
- Programming through interfaces helps create software solutions that are reusable, extensible, and maintainable

Applying Interfaces (Contd.).

```
interface IntDemo{
  void display();
}
class classOne implements IntDemo{
  void add(int x, int y){
    System.out.println("The sum is :" +(x+y));
  }
  public void display(){
    System.out.println("Welcome to Interfaces");
  }
}
```

Applying Interfaces (Contd.).

```
class classTwo implements IntDemo{
  void multiply(int i,int j, int k) {
    System.out.println("The result:" +(i*j*k) );
  }
  public void display(){
    System.out.println("Welcome to Java ");
  }
}
class DemoClass{
  public static void main(String args[]) {
    classOne c1= new classOne();
    c1.add(10,20);
    c1.display();
    classTwo c2 = new classTwo();
    c2.multiply(5,10,15);
    c2.display();
  }
}
```

Interface References

- When you create objects, you refer them through the class references. For example :
  - ClassOne c1= new classOne(); /* Here, c1 refers to the object of the class classOne. */
- You can also make the interface variable refer to the objects of the class that implements the interface
- The exact method will be invoked at run time
- It helps us achieve run-time polymorphism

Interface References (Contd.).

```
interface IntDemo{
 void display();
}
class classOne implements IntDemo{
 void add(int x, int y){
  System.out.println("The sum is :" +(x+y));
 }
 public void display(){
  System.out.println("Class one display method
 ");
 }
}
```

Interface References (Contd.).

```
class classTwo implements IntDemo {
 void multiply(int i,int j, int k){
   System.out.println("The result:" +(i*j*k) );
 }
 public void display(){
   System.out.println("Class two display method"
 );
 }
}
class DemoClass{
 public static void main(String args[]){
   IntDemo c1= new classOne();
   c1.display();
   c1 = new classTwo();
   c1.display();
 }
}
```

Extending Interfaces

- Just as classes can be inherited, interfaces can also be inherited
- One interface can extend one or more interfaces using the keyword **extends**
- When you implement an interface that extends another interface, you should provide implementation for all the methods declared within the interface hierarchy

Marker Interface

- An Interface with no method declared in it, is known as Marker Interface

- Marker Interface is provided as a handle by java interpreter to mark a class, so that it can provide special behavior to it at runtime

- Examples of Marker Interfaces :
  - java.lang.Cloneable
  - java.io.Serializable
  - java.rmi.Remote

Quiz

Will the following code compile successfully ?

```
interface I1 {
      int a=100;
      void m1();
}

class A1 extends I1 {
      public void m1() {
      System.out.println("In m1 method");
      }
}
```

It will throw compilation error.. Why?

Quiz (Contd.).

Will the following code compile successfully ?

```
interface I1 {
     int a=100;
     void m1();
}

interface A1 implements I1 {
     public void m2();
}
```

It will throw compilation error.. Why?

Quiz (Contd.).

Will the following code compile successfully ?

```
interface I1 {
     int a=100;
     void m1();
}

interface A1 extends I1 {
     public void m2();
}

class Aimp implements I1 {
     public void m1() {
     System.out.println("In m1 method");
     }
}
```

This code will compile successfully..!

Abstract Classes v/s Interfaces

Abstract Classes	Interfaces
Abstract classes can have non-final non-static variables.	Variables declared within an interface are always static and final.
Abstract Classes can have abstract methods as well as concrete methods.	Interfaces can have only method declarations(abstract methods). You cannot define a concrete method.
You can declare any member of an abstract class as private, default, protected or public. Members can also be static.	Interface members are by default public. You cannot have private or protected members. Interface methods cannot be static.
Abstract class is extended by another class using "extends" keyword.	An interface is "implemented" by a java class using "implements" keyword .

Contd..

Abstract Classes v/s Interfaces (Contd.).

Abstract Classes	Interfaces
An abstract class can extend another class and it can implement one or more interfaces.	An interface can extend one or more interfaces but cannot extend a class. It cannot implement an interface.
An abstract class can have constructors defined within it.	You cannot define constructors within an interface.
An abstract class cannot be instantiated using “new” Keyword	An interface cannot be instantiated.
You can execute(invoke) an abstract class, provided it has public static void main(String[] args) method declared within it.	You cannot execute an interface

Assignment

Summary

- Introduction to interfaces
- Creating interfaces
- Implementing interfaces
- Difference between interfaces and abstract classes

Thank You