




H2K Infosys is  business based in Atlanta, Georgia – United States  
Providing Online IT training services world wide.

[www.H2KINFOSYS.com](http://www.H2KINFOSYS.com)

USA - +1-(770)-777-1269, UK - (020) 3371 7615  
[Training@H2KInfosys.com](mailto:Training@H2KInfosys.com) / [H2KInfosys@Gmail.com](mailto:H2KInfosys@Gmail.com)

# **H2K INFOSYS PROVIDES WORLD CLASS SERVICES IN**

**IT Trainings  
with Real time  
Project Work  
for  
Corporates  
&  
Individuals**

**Special IT  
Training  
for MS  
Students in  
US**

**Software Design,  
Development,  
QA – Manual,  
Automation &  
Performance  
Testing,  
Maintenance.**

**IT Staff  
Augmentation  
  
Job Placement  
Assistance  
  
Tech Support**

- **Interfaces are “contract” to be followed by implementing class.**

In the Java programming language, an interface is a reference type, similar to a class, that can contain only constants, method signatures, and nested types.

There are no method bodies.

Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces.

Lets learn two words “**extends**” and “**implements**”

- When a Class wants to inherit properties and methods of another class, it can do that with ‘extends’ keyword.

*public class TestClass **extends** ParentTestClass*

*This means, inheritance can be achieved with the help of ‘**extends**’.*

Let’s take an example. PreferredCustomer class.

- **Interfaces are “contract” to be followed by implementing class.**

Interface are defined with help of 'interface' keyword.

```
interface Reachable {  
    // Define constants here  
    // Define methods here  
}
```

- **Constants / Variables in Interface are always “public static final”:**

public – accessible to everyone

static – don't need an instance (which is anyway not possible)

final – you cannot assign a different value at runtime.

## **Methods in Interfaces are just definitions without bodies.**

There will be no {} either.

The methods are always public and abstract. No other modifier is even allowed.

public – accessible to everyone

abstract – has no method bodies.

- **Interfaces are “contract” to be followed by implementing class.**

```
interface Reachable {  
    // Example of Constant - public static final by default  
    String COUNTRY_CODE = "US";  
    // Method declaration - public abstract by default  
    boolean validateAddress(String address) throws AddressNotFoundException;  
    // Don't worry about AddressNotFoundException - We gonna cover it in Exception Handling  
}
```

To use interface, we will need a class which will implement this interface. Means, will follow Reachable contract. To do so, we will use keyword '**implements**' e.g.:

```
class Customer implements Reachable{ }
```

Though above class looks valid, you will get a compile time error: The type Customer must implement the inherited abstract method **Reachable.validateAddress(String)**

```
class Customer implements Reachable{  
    public boolean validateAddress(String address)  
        throws AddressNotFoundException {  
        return false; // Actual Validation Logic goes here  
    }  
}
```

- **If you define a reference variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface.**

Reachable goodCustomer = new Customer (); // As Customer class implements Reachable.

- **Interface can extend another interface.**

What is wrong with the following interface?

```
public interface SomethingIsWrong {  
    void aMethod(int aValue){  
        System.out.println("Hi Mom");  
    }  
}
```

Is the following interface valid?

```
public interface Marker {  
}
```



A class that is derived from another class is called a subclass (also a derived class, extended class, or child class). The class from which the subclass is derived is called a superclass (also a base class or a parent class)..

Excepting Object, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, **every class is implicitly a subclass of Object**.

A subclass inherits all of the **public and protected** members of its parent, no matter what package the subclass is in.

A subclass **does not inherit the private** members of its parent class. However, if the superclass has public or protected methods for accessing its private fields, these can also be used by the subclass.

You can write a subclass constructor that invokes the constructor of the superclass, either **implicitly** or by using the keyword **super**.

**Interesting:** You can instantiate Subclass with Super Class declaration.

Let's create PreferredCustomer class which is Subclass of Customer class.

So now you can do this: `Customer pCustomer = new PreferredCustomer();`

With that rule anything in Java can be: `Object obj = new AnyClassNameHere();`

- **An** instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method. In this case, method implementation in Super class is **hidden**.
- Let's understand the rules of Overriding method.
  1. A method is said to be overridden if a class which extends another class defines method with the same name and arguments list.
  2. The method defined in the Super class should **be visible in** the derived class. If this is not so, the method in the derived class will not be considered overridden version but will be treated as a normal method.
  3. The method name and arguments list **should be same** for overriding and overridden methods. But the return type can be **co-variant**. This means that if the return type of the method in super class is Map, then the return type of the same method can be HashMap.
  4. The access modifier in the overriding method (sub class) should not be more limiting than that of the overridden method (Super class). This means that if the access modifier for base class method is protected then the access modifier for the derived class method should not be default or private but can be protected, public. The order of increasing visibility of various modifier is: private, default, protected, public
  5. The exceptions specified in the derived class method should be either **same or sub-class** of them. Thus if the base class method specifies the exception as IOException in the throws clause then the derived class method can specify the exception as FileNotFoundException, IOException but not Exception.



- If a subclass defines a class method with the same signature as a class method in the superclass, the method in the subclass hides the one in the superclass.
- Within a class, a field that has the same name as a field in the superclass hides the superclass's field, even if their types are different.
- Within the subclass, the field in the superclass cannot be referenced by its simple name. Instead, the field must be accessed through super, which is covered in the next section.
- Hiding fields is not recommended in corporate programming. As it makes code difficult to read.

- If your method overrides one of its superclass's methods, you can invoke the overridden method through the use of the keyword `super`.

```
public void validateAddress() {  
    super.validateAddress();  
    System.out.println("Subclass Address Validation Logic");  
}
```

**super()** can be used to call Superclass constructor like below:

```
public Subclass() {  
    super();  
}
```

- If a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass.
- If the super class does not have a no-argument constructor, you will get a compile-time error. Object does have such a constructor, so if Object is the only superclass, there is no problem.

- What is Superclass don't want to subclass to override certain methods?
- What if A class don't want a Subclass to be created?

**final** is at your rescue.

- You use the final keyword in a method declaration to indicate that the method cannot be overridden by subclasses.
- If the super class does not have a no-argument constructor, you will get a compile-time error. Object does have such a constructor, so if Object is the only superclass, there is no problem.
- A class that is declared final cannot be sub-classed. This is particularly useful, for example, when creating an immutable class like the String class.

# **Thank you**

**Your feedback is highly important to improve our course material and teaching methodologies. Please email your suggestions.**

**USA +1-(770)-777-1269  
Training@H2KInfosys.com**

**UK (020) 3371 7615  
H2KInfosys@Gmail.com**

**H2K Infosys is e-Verified business based in Atlanta, Georgia – United States**

H2K Infosys acknowledges the proprietary rights of the trademarks and product names of other companies mentioned in this document.



*H2K Infosys, LLC (hereinafter “H2K”) acknowledges the proprietary rights of the trademarks and product names of other companies mentioned in any of the training material including but not limited to the handouts, written material, videos, power point presentations, etc. All such training materials are provided to H2K students for learning purposes only. H2K students shall not use such materials for their private gain nor can they sell any such materials to a third party. Some of the examples provided in any such training materials may not be owned by H2K and as such H2K does not claim any proprietary rights for the same. H2K does not guarantee nor is it responsible for such products and projects. H2K acknowledges that any such information or product that has been lawfully received from any third party source is free from restriction and without any breach or violation of law whatsoever.*