Programming (http://www.sitepoint.com/programming/)

Interface and Inheritance in Java: Interface

(http://www.sitepoint.com/author/spanda/)

Sandeep Panda (http://www.sitepoint.com/author/spanda/)

Published January 3, 2013

Tweet (Https://twitter.com/share?text=Interface+and+Inheritance+in+Java%3A+Interface& via=sitepointdotcom)

Subscribe (Https://confirmsubscription.com/h/y/1FD5B523FA48AA2B)

This entry is part 1 of 2 in the series Interface and Inheritance In Java (http://www.sitepoint.com /series/interface-and-inheritance-in-java/)

Interface is a 100% abstract class. It contains only constants and method signatures. In other words it is a reference type similar to class. An interface can't be instantiated. It can be implemented by a class or extended by another interface.

How To Define:

An interface can be defined as the following:

```
public interface DriveCar {
void turnLeft();
void turnRight();
void moveBack();
void accelerate();
}
```

The methods declared in an interface don't have method bodies. By default all the methods in an interface are **public abstract**. Similarly all the variables we define in an interface are essentially constants because they are implicitly **public static final**. So, the following definition of interface is equivalent to the above definition.

```
public interface DriveCar {
public abstract void turnLeft();
public abstract void turnRight();
public abstract void moveBack();
public abstract void accelerate();
}
```

How To Use:

An interface defines a contract which an implementing class must adhere to. The above interface `DriveCar` defines a set of operations that must be supported by a `Car`. So, a class that actually implements the interface should implement all the methods declared in the interface.

Example:

```
class Car implements DriveCar{
void turnRight(){
//implementation code goes here
}

void turnLeft(){
//implementation code goes here
}

void moveBack(){
//implementation code goes here
}

void accelerate(){
//implementation code goes here
}

}
```

Now we can take a reference of type `DriveCar` and assign an object of `Car` to it.

Example:

```
DriveCar carDriver=new Car();
carDriver.turnLeft();
carDriver.moveBack();
//other method invocations
```

We can also code in the following way:

Example:

```
Car car=new Car();
car.turnLeft();
car.moveBack();
//other method invocations
```

Why Use Interfaces:

Interfaces act as APIs (Application Programming Interfaces). Let us take an example of an image processing company which writes various classes to provide the image processing functionalities. So, a nice approach will be creating interfaces, declaring the methods in them and making the classes implement them. In this way the software package can be delivered to the clients and the clients can invoke the methods by looking at the method signatures declared inside the interfaces. They won’t see the actual implementation of the methods. As a result the implementation part will be a secret. Later on the company may decide to re-implement the methods in another way. But the clients are concerned about the interfaces only.

Interfaces provide an alternative to multiple inheritance. Java programming language does not support multiple inheritance. But interfaces provide a good solution. Any class can implement a particular interface and importantly the interfaces are not a part of class hierarchy. So, the general rule is **extend one but implement many**. A class can extend just one class but it can implement many interfaces. So, here we have multiple types for a class. It can be of the type of its super class and all the interfaces it implements.

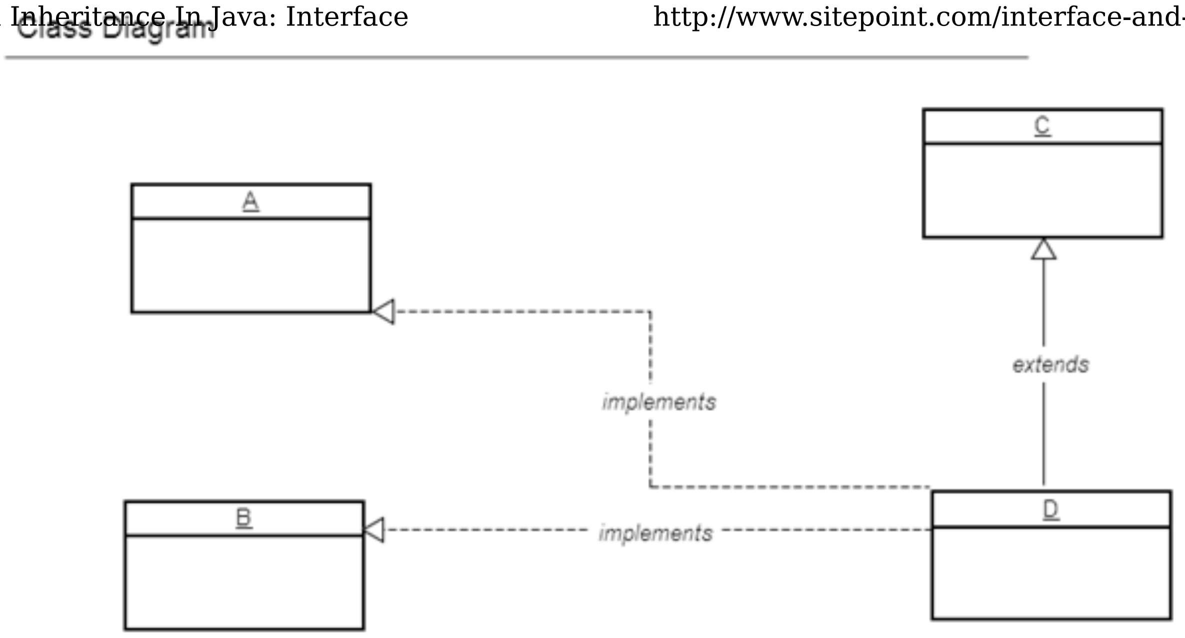
Example:

Let us say we have two interfaces A & B and two classes C & D.

```
interface A{ }
interface B{ }
class C{ }
class D extends C implements A,B { }
```

So, we can have 3 types for an object of class D as following:

```
A a=new D(); B b=new D(); C c=new D();
```



But be careful. If you use interface as reference type and assign an object of implementing class to it then you can call only those methods that are declared inside the interface. This is quite obvious because the implementing class can define methods of its own that are not a part of the contract between the interface and class. So, to call those methods you have to use the class as reference type as following:

```
D d=new D();
```

Extending an interface:

Consider the following scenario. You have an interface A and several implementing classes. It defines 2 methods.

```
interface A{
int doThis();
int doThat();
}
```

Now suppose you want to add another method to the interface A:

```
interface A{
int doThis();
int doThat();
int doThisAndThat();
}
```

If you add the third method to the interface it will break the code because the implementing classes will no more be adhering to the contract. But we can avoid the problem if we create another interface and make it extend the previous interface.

```
interface APlusPlus extends A{
int doThisAndThat();
}
```

Now your users have the option to either use the old interface or upgrade to the new interface.

Note:

Any class that implements an interface must implement the methods declared in that interface plus all the methods that are present in the super interface.

If the implementing class is abstract it may choose to implement all, some or none of the methods declared in the interface. But a concrete subclass of the abstract class must implement all the non implemented methods.

Summary:

- *Interfaces can contain only constants and method signatures, but no implementation.*
- *Interfaces cannot be instantiated. They can only be implemented by an implementing class or extended by another interface.*
- *A class that implements an interface must provide implementation to all the methods that are declared in the interface.*
- *Interfaces can be used as reference type for the object of an implementing class.*
- *An interface can be extended by another interface.*

Interface and Inheritance In Java (http://www.sitepoint.com/series/interface-and-inheritance-in-java/)

Interface and Inheritance in Java: Inheritance >> (http://www.sitepoint.com/interface-and-inheritance-in-java-inheritance/)

(http://www.sitepoint.com/author/spanda/)

Sandeep Panda (http://www.sitepoint.com/author/spanda/)

Sandeep is the Co-Founder of devmag.io (https://devmag.io). He loves startups and web technologies.

(https://twitter.com/Sandeepg33k) (http://facebook.com/sandeep.panda92)
(http://www.linkedin.com/pub/sandeep-panda/45/768/b23)
(https://plus.google.com/+SandeepPanda)

You might also like:

A Closer Look at Go Interfaces (http://www.sitepoint.com/closer-look-go-interfaces/)

Book: Jump Start Bootstrap (https://learnable.com/books/jump-start-bootstrap?utm_source=sitepoint&utm_medium=related-items&utm_content=js-bootstrap)

Symfony in Drupal 8 (http://www.sitepoint.com/symfony-drupal-8/)

Comments for this thread are now closed. ×

3 Comments SitePoint Login

Sort by Best Share Favorite

Bas • 2 years ago

Thanks for sharing your knowledge Sandeep, but I don't think these kind of general tutorials should be in Sitepoint. There is nothing new here or of any 'news' value. Also the code styling lacks indentation.

As for the examples, what is a DriveCar supposed to mean? Car implements Vehicle would have been a better choice. And the A, B, C example is even worse. If you want to explain something it's better to use practical examples.

You made a good start with explaining the usefulness of interfaces but you could have told something about design patterns that use them.

Regards,
Bas

2 • Share ›

Sandeep Panda Bas • 2 years ago

Hi Bas,

Thanks for commenting. I would like to clarify some points. First of all, this is a tutorial which is intended for beginners. Second thing is, don't you think Car extends Vehicle would have been a good choice rather than Car implements Vehicle? Since I am not explaining about Inheritance in this part I thought it would be a good idea to keep all the operations done by the car in a separate interface and make Car implement it to perform the operations. The interface specifically deals with how to drive a car and that's why I named in this way.If I were writing about inheritance here, I would have made Car extend Vehicle.

Thanks for the read and I value your suggestion. :)

Regards,
Sandeep

4 • Share ›

Swadesh Bas • 2 years ago

I am a front end guy and do have some knowledge on java. I think the article is good for the beginners like me. Well explained.

3 • Share ›