HowToDoInJava

Interface vs Abstract Class in Java

Last Updated: January 29, 2022 By: Lokesh Gupta Java Object Oriented Programming Java OOP

Abstract classes and interfaces are the two main building blocks of most Java APIs. In this article, We will touch down the most glaring differences between interfaces and abstract classes in Java.

Table of Contents

I will recommend you to read about abstraction first, because it the main force behind abstract classes and interfaces.

1. Abstract classes in java

In simplest words, **an abstract class is which is declared abstract using keyword `abstract`**. It may or may not contain any abstract method. JVM identifies abstract class as **incomplete class**, which has not defined its complete behavior. Declaring a class `abstract` enforces only one thing: you can not create an instance of this class, and that's it.

So why even you bother to create a class which can not be instantiated at all? The answer is in its usage in solving some critical design issues. We will come to that part later in this article.

1.1. Syntax of abstract class

```
abstract class TestAbstractClass
{
    public abstract void abstractMethod();
    public void normalMethod()
    {
        //method body
    }
}
```

Here, our `TestAbstractClass` has two methods, one is abstract and second one is normal method. An abstract method. Having an abstract method in your class will force you to declare your class as abstract itself.

1.2. Abstract method

An **abstract method**, is a method which is not implemented in place. An abstract method adds the incompleteness to class, thus compiler wants to declare whole class abstract.

The only way to use an abstract class in your application is to extend this class. Its subclasses if not declared `abstract` again, can be instantiated. The feature that subclass inherits the behavior of the superclass and superclass can hold the reference of subclass increases the importance of abstract classes many folds.

2. Interfaces in java

Interfaces are yet another basic building block of most Java APIs. You name it e.g. collection, I/O or SWT, you can see them in action everywhere.

> Interface define contracts, which implementing classes need to honor.

These contracts are essentially unimplemented methods. Java already has a keyword for unimplemented methods i.e. *abstract*. Java has the provision where any class can implement any interface, so all the methods declared in interfaces need to be public only.

2.1. Syntax of interface

```
public interface TestInterface
{
    void implementMe();
}
```

For above interface, any implementing class needs to override `implementMe()` method.

2.2. Abstract class implementing interface

There is only one scenario when you implement an interface and do not override the method i.e. declare the implementing class itself `abstract`.

```
Abstract class

public abstract class TestMain implements TestInterface
{
    //No need to override implement Me
}
```

Otherwise, you must implement the method `implementMe()` in you class without any other exception.

```
Non-abstract class
```

```
public class TestMain implements TestInterface
{
    @Override
    public void implementMe() {
        // TODO Auto-generated method stub
    }
}
```

3) Abstract classes vs Interfaces

Let's note down **differences between abstract classes and interfaces** for quick review:

1. Interfaces have all methods inherently ***public* and *abstract***. You can not override this behavior by trying to reduce accessibility of methods. You can not even declare the static methods. Only public and abstract.
   On the other side, abstract classes are flexible in declaring the methods. You can define abstract methods with protected accessibility also. Additionally, you can define static methods as well, provided they are not abstract. Non-abstract static methods are allowed.

2. Interfaces can't have fully defined methods. By definition, interfaces are meant to provide the only contract.
   Abstract classes can have non-abstract methods without any limitation. You can use any keyword with non-abstract methods as you will do in any other class.

3. Any class which want to use abstract class can extend abstract class using keyword `extends`, whereas for implementing interfaces keyword used is `implements`.
   A class can extend only one class but can implement any number of interfaces. This property is often referred as simulation of **multiple inheritance** in java.

4. Interface is absolutely `abstract` and cannot be instantiated; A Java abstract class also cannot be instantiated, but can be invoked if a main() exists.

Next, a question may come if we have abstract methods and main class both, we may try to call the abstract method from `main()`. But this attempt will fail, as `main()` method is always static and abstract methods can never be static, so you can never access any non-static method inside the static method.

4. When to use abstract class and when to use interfaces

Always remember that **choice between the interface or abstract class** is not either/or scenario, where choosing anyone without proper analysis would yield the same results. A choice must be made very intelligently after understanding the problem at hand. Let us try to put some intelligence here.

4.1. Partial behavior with abstract classes

Abstract classes let you define some behaviors; it makes them excellent candidates inside of application frameworks.

Lets take example of HttpServlet. It is the main class you must inherit if you are developing a web application using Servlets technology. As we know, each servlet has a definite life cycle phases, i.e. initialization, service, and destruction. What if each servlet we create, we have to write the same piece of code regarding initialization and destruction again and again. Surely, it will be a big pain.

JDK designers solved this by making `HttpServlet` abstract class. It has all basic code already written for initialization of a servlet and destruction of it also. You only need to override certain methods where you write your application processing related code, that's all. Make sense, right !!

Can you add above feature using interface? No, even if you can, the design will be like a hell for most innocent programmers.

4.2. Contract only interfaces

Now, let's look at the usage of interfaces.**An interface only provide contracts and it is the responsibility of implementing classes to implement each and every single contract provided to it**.

An interface is the best fit for cases where you want to **define only the characteristics of class**, and you want to force all implementing entities to implement those characteristics.

4.3. Example

I would like to take an example of `Map` interface in the collections framework. It provides only rules, how a map should behave in practice. e.g. it should store the key-value pair, the value should be accessible using keys etc. These rules are in form of abstract methods in the interface.

All implementing classes (such as HashMap, HashTable, TreeMap or WeakHashMap) implements all methods differently and thus exhibit different features from rest.

Also, interfaces can be used in defining the separation of responsibilities. For example, `HashMap` implements 3 interfaces: `Map`, Serializable and Cloneable. Each interface defines separate responsibilities and thus an implementing class choose what it want to implement, and so will provide that much limited functionality.

5. Java 8 default methods in interfaces

With Java 8, now you can define methods in interfaces. These are called default methods. Default methods enable you to add new functionality to the interfaces of your libraries and ensure binary compatibility with code written for older versions of those interfaces.

As the name implies, default methods in Java 8 simply default. If you do not override them, they are the methods which will be invoked by caller classes.

```
Default methods
```

```
public interface Moveable {
    default void move(){
        System.out.println(&quot;I am moving&quot;);
    }
}
```

In above example, `Moveable` interface defines a method `move()` and provided a default implementation as well. If any class implements this interface then it need not to implement it's own version of `move()` method. It can directly call `instance.move()`.

```
Animal.java

public class Animal implements Moveable{
    public static void main(String[] args){
        Animal tiger = new Animal();
        tiger.move();    //I am moving
    }
}
```

And if class willingly wants to customize the behavior then it can provide its own custom implementation and override the method. Now it's own custom method will be called.

```
Animal.java

public class Animal implements Moveable{

    public void move(){
        System.out.println(&quot;I am running&quot;);
    }

    public static void main(String[] args){
        Animal tiger = new Animal();
        tiger.move();    //I am running
    }
}
```

5.1) Difference between abstract class and interface in Java 8

If you see we are now able to provide a partial implementation with interfaces as well, just like abstract classes. So essentially the line between interfaces and abstract classes has become very thin. They provide almost the same capabilities now.

Now, only one big difference remains that **you cannot extend multiple classes whereas you can implement multiple interfaces**. Apart from this difference, you can achieve any possible functionality from interfaces which abstract classes can make possible, and vice-versa is also true.

Hope you found enough information on **interfaces and abstract classes in java**.

Happy learning !!

Recommended Reading:

1. Java abstract keyword – abstract classes and methods
2. Abstract Factory Pattern Explained
3. [Solved]: javax.xml.bind.JAXBException: class java.util.ArrayList nor any of its super class is known to this context
4. Java Cloneable interface – Is it broken?
5. Private Methods in Interface – Java 9
6. Java Comparable Interface
7. Java Comparator Interface
8. Java Iterator interface example

9. Java ListIterator interface

0. Java Spliterator interface

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*

16 thoughts on “Interface vs Abstract Class in Java”

Test

April 12, 2022 at 2:30 am

Abstract classes can hold instance variables, while interfaces cannot.

Reply

anjali

June 25, 2018 at 11:01 am

Abstract classes can have a constructor because the constructor does not define a method signature that must be matched inherited by child classes. It’s the only method that doesn’t require that same signature on child implementations

Reply

arun singh

May 6, 2018 at 12:36 am

thanks,
great article

Reply

Dushyant Gupta

December 6, 2016 at 9:08 pm

Hi Lokesh,

Apart from the "simulated multiple inheritance" difference between interface and abstract class, what else is the diff. between them. I have been asked this question like:
"When you can achieve everything using abstract class that you can using an interface, then why interface is used"

I gave the answer of multiple inheritance concept, but the interviewers are always looking for something new. What could it be?

Reply

Lokesh Gupta

December 7, 2016 at 12:58 pm

Abstract classes are never a substitute of interfaces. Both have their own use cases. Abstract class can provide a class "partial behavior" which was impossible using interfaces in java 7. Java 8 default methods give you enough flexibility and blur the line between both constructs.
Even after above change there is a major difference in how other members are defined and accessed in both constructs i.e. abstract classes allow non-static and non-final fields and allow methods to be public, private, or protected while interfaces' fields are inherently public, static, and final, and all interface methods are inherently public. This can be deciding factor in software design process.

Reply

Sandip

April 25, 2015 at 10:53 am

Hi Lokesh,
Thanks for such nice articles.Can u please explain with real life example that when to use interface and abstract class.

Reply

struggler

June 11, 2014 at 7:19 am

Excellent article. Really helpful.

About static methods in abstract classes, you said

“Here also, static is prohibited.”

Is that correct?

Check this link:

https://stackoverflow.com/questions/17407203/can-we-use-static-method-in-an-abstract-class

Reply

Lokesh Gupta

June 11, 2014 at 7:26 am

You are right. I will add more test to remove the confusion. You can't declare a static method to be abstract; but you can create non-abstract static methods.

Reply

kheem singh

May 16, 2014 at 4:18 am

hi lokesh,,
i ve found your articles really interesting, please explore the concept reflection in java.

Reply

Lokesh Gupta

May 19, 2014 at 7:12 am

Sure, I will.

Reply

HIMANSU NAYAK

May 4, 2014 at 2:46 am

Hi Lokesh,

5.) Also Abstract class can have constructor which definitely not possible in interface.

There are also things like nested Abstract class, Nested Interface, Defining Class inside Interface and Interface inside Class but never ever have seen this implemented anywhere in the real time application.

Reply

Lokesh Gupta

May 4, 2014 at 2:59 am

Good point.

Reply

harivenkat

April 25, 2014 at 7:35 am

"Can you add above feature using inheritance? No,"

Why not? Can you explain on this?

I believe even inheritance can be used to support HttpServlet feature. Can you
Whether the design is efficient or not is a different question. Thanks

Reply

Lokesh Gupta

April 25, 2014 at 8:08 am

My bad. It should be written as Can you add above feature using interface?

I am assuming that you meant interface as well when you asked this question. HTTPServlet is backbone of our web applications. If you write using interface then you will leave everything in hand on poor developers "WILLINGLY". It will prove disaster for most of applications today we are seeing around, because in these methods init(), and destroy() you interact with heart of the application and server infrastructure. You simply cannot leave the implementation on hand of poor developers. They should focus only on what they are best at i.e. application logic.

You are right that nothing is impossible but my point again is "It's not either/or based decision". It should be done very intelligently.

Please correct me If i am wrong.

Reply

harivenkat

April 25, 2014 at 8:14 am

Thanks for the response Lokesh. I agree with you.

Please do modify the article as well.

Reply

Java Experience (@javaexper)

February 8, 2013 at 8:54 am

Additionally, as shown in my example tutorial, abstract classes can have non-abstract methods.

Reply

Leave a Comment

Name *

Email *

Website

☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search …

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial