### Abstraction in Java

* **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
* Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.
* Abstraction lets you focus on what the [object](https://www.javatpoint.com/object-and-class-in-java) does instead of how it does it.

### Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

### Abstract class in Java

* A class which is declared with the abstract keyword is known as an abstract class in Java.
* It can have abstract and non-abstract methods (method with the body).
* It needs to be extended and its method implemented.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

**Example of abstract class**

**abstract** **class** A{}

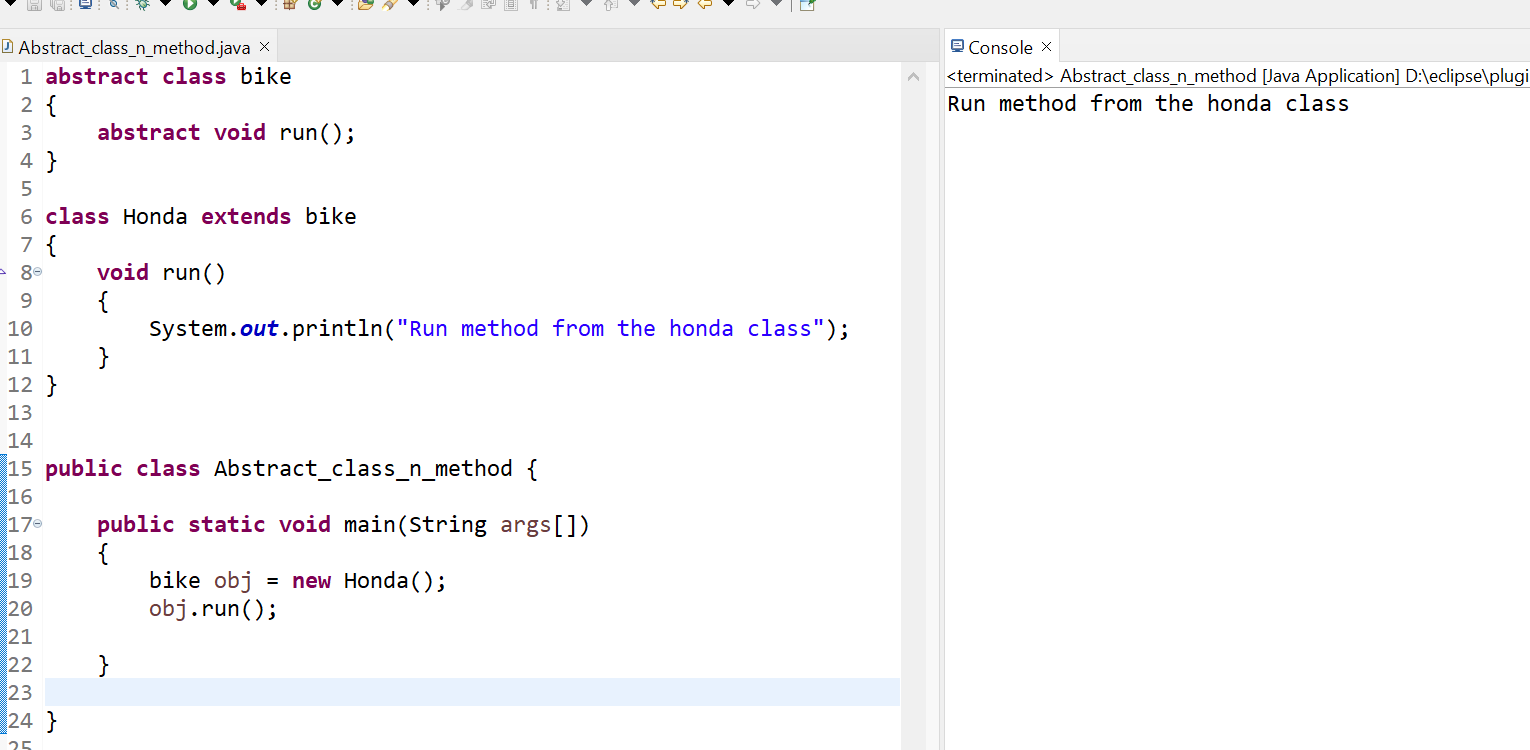
### Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

**Example of abstract method**

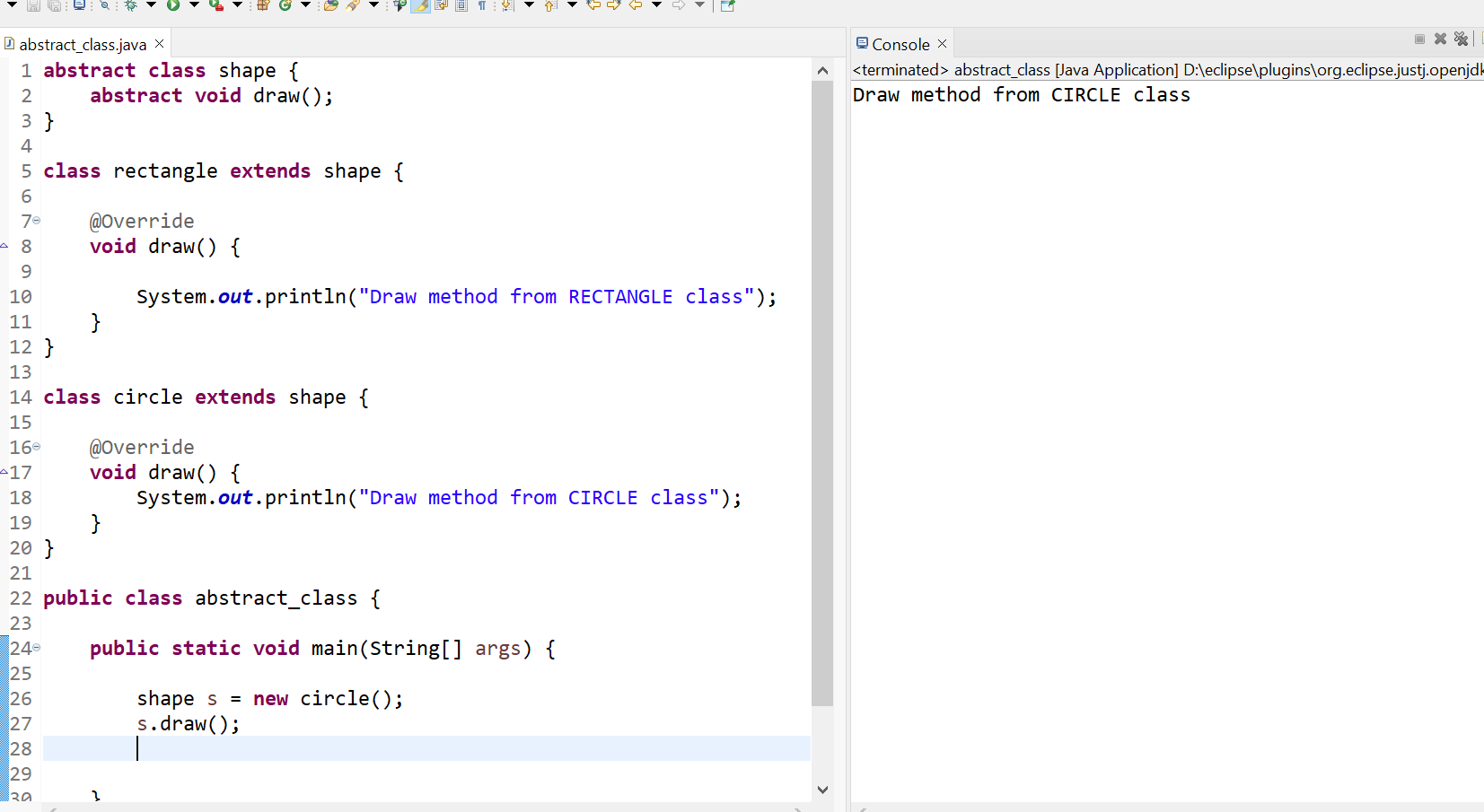
**abstract** **void** printStatus();//no method body and abstract

### Example of Abstract class that has an abstract method

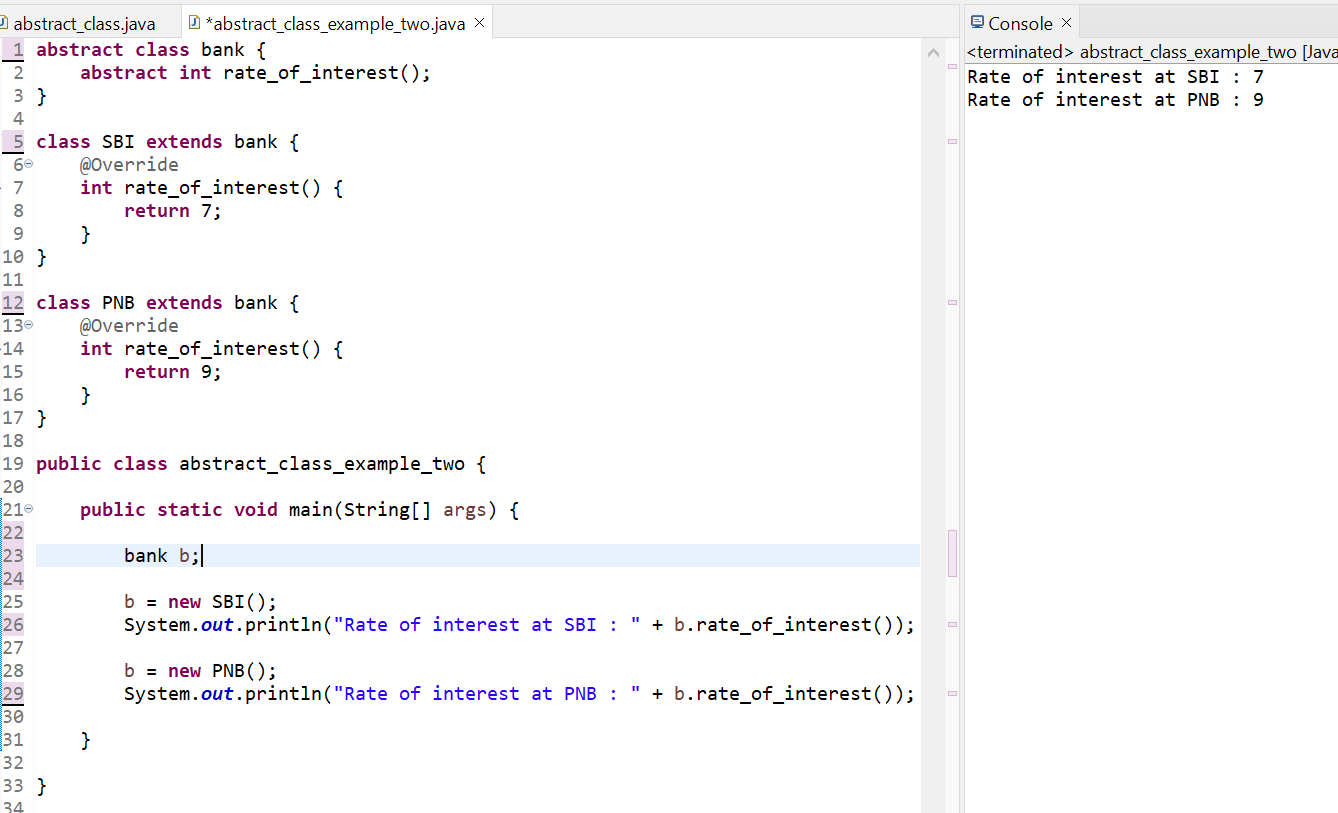


### Understanding the real scenario of Abstract class

* In this example, Shape is the abstract class, and its implementation is provided by the Rectangle and Circle classes.
* Mostly, we don't know about the implementation class (which is hidden to the end user), and an object of the implementation class is provided by the **factory method**.
* A **factory method** is a method that returns the instance of the class.
* In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

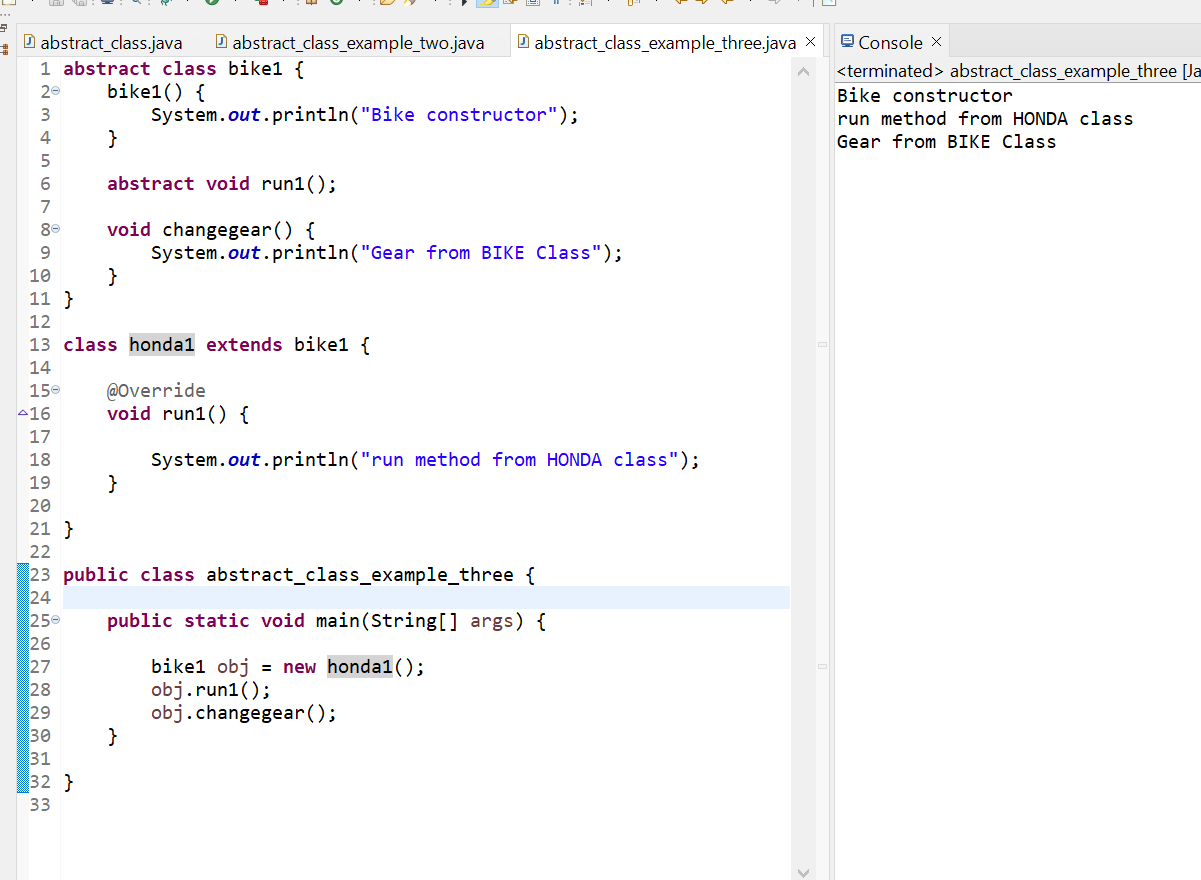


### Another example of Abstract class in java



### Abstract class having constructor, data member and methods

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.



**NOTE :**

1. **If there is an abstract method in a class, that class must be abstract.**
2. **If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.**

# Interface in Java

* An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
* The interface in Java is *a mechanism to achieve* [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).
* In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.
* Java Interface also **represents the IS-A relationship**.
* It cannot be instantiated just like the abstract class.
* Since Java 8, we can have **default and static methods** in an interface.
* Since Java 9, we can have **private methods** in an interface.

## Why use Java interface?

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.

## How to declare an interface?

* An interface is declared by using the interface keyword.
* It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
* A class that implements an interface must implement all the methods declared in the interface.
* Syntax ;

**interface** <interface\_name>{

// declare constant fields

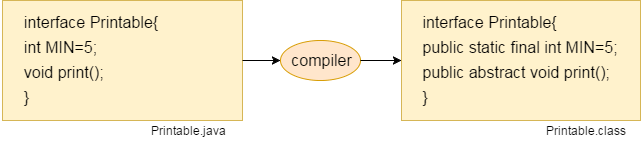
// declare methods that abstract

// by default.

}

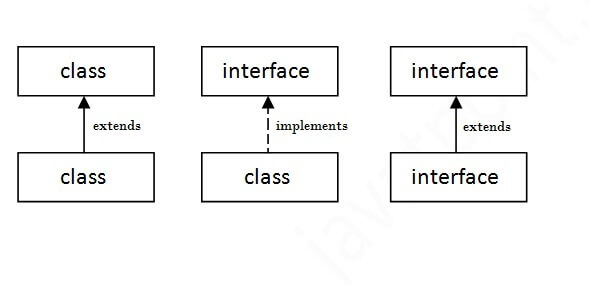
**NOTE** :

* The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.
* In other words, Interface fields are public, static and final by default, and the methods are public and abstract.

****

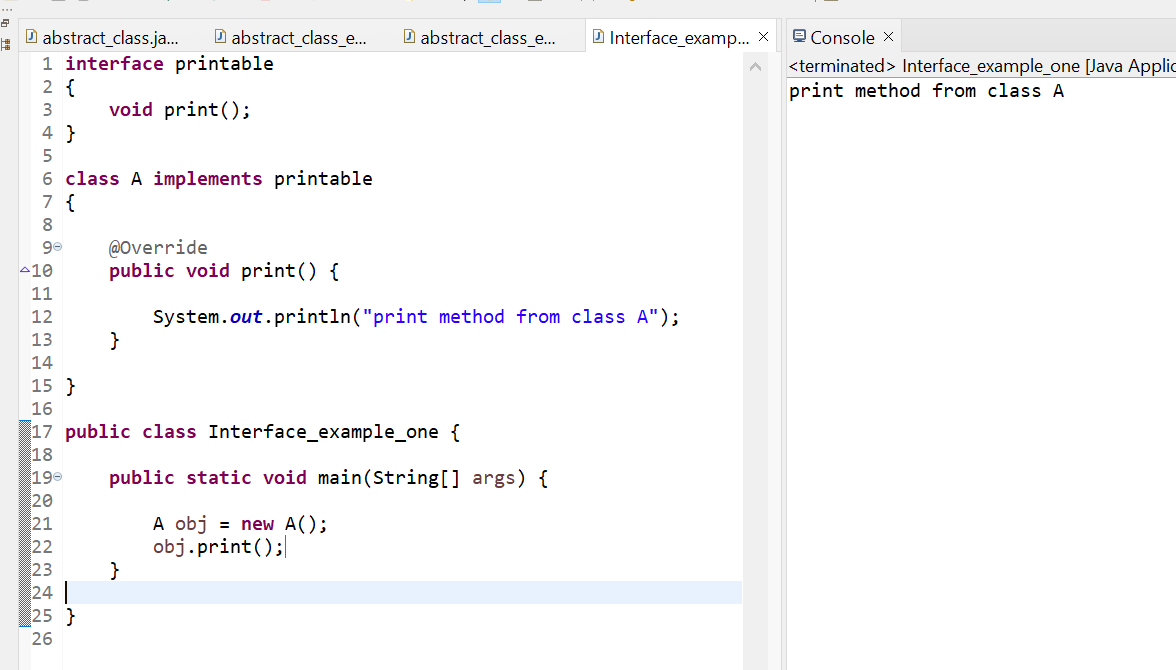
#### The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



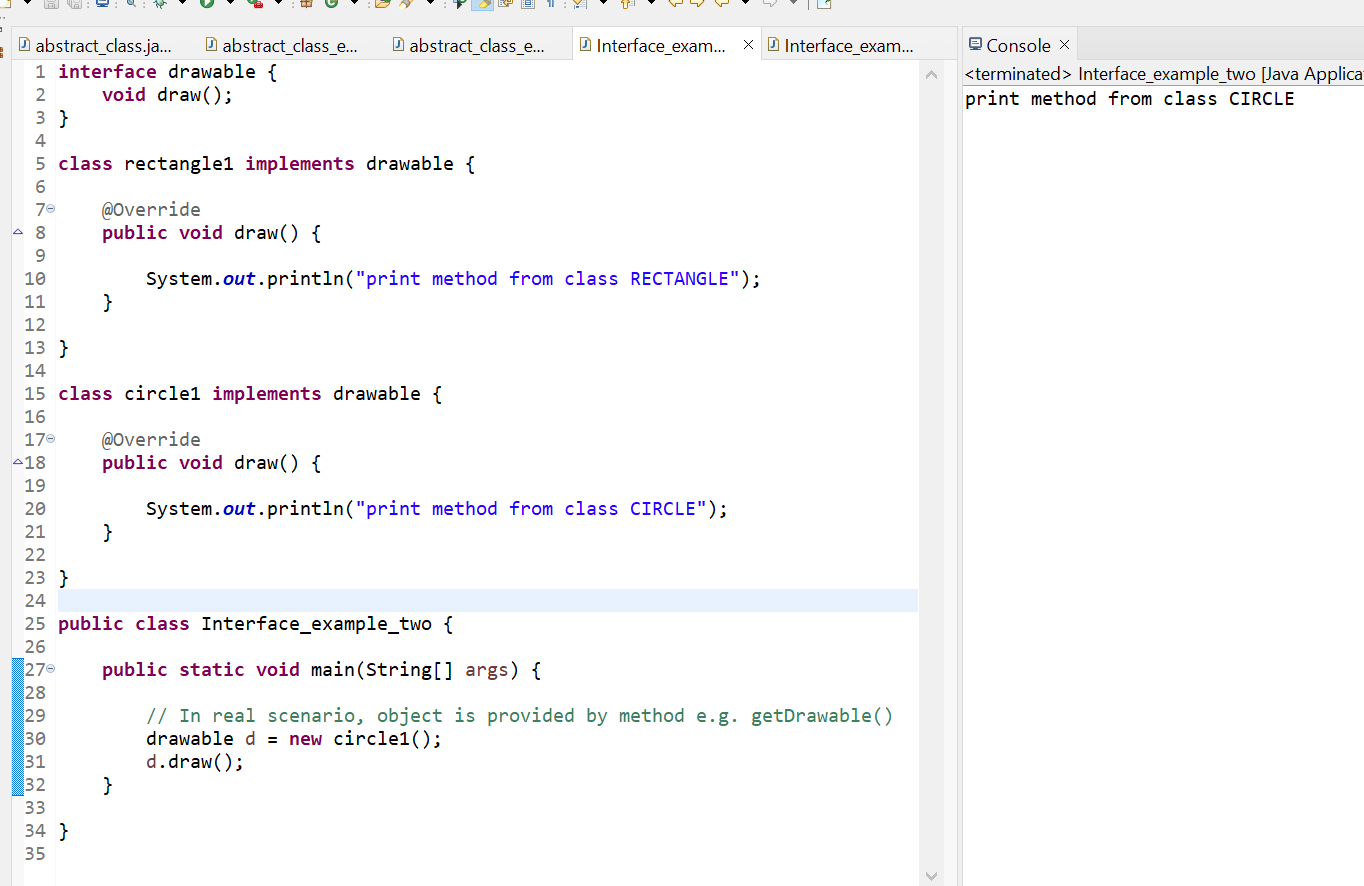
## Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A class.



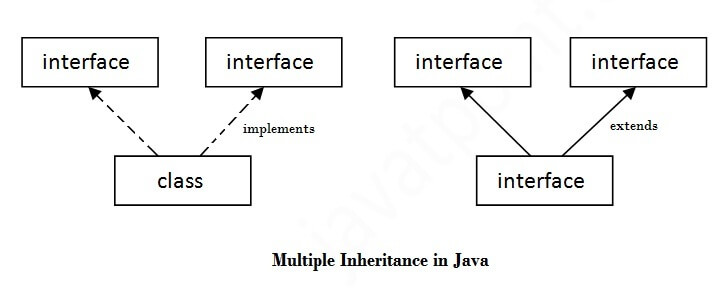
## Java Interface Example: Drawable

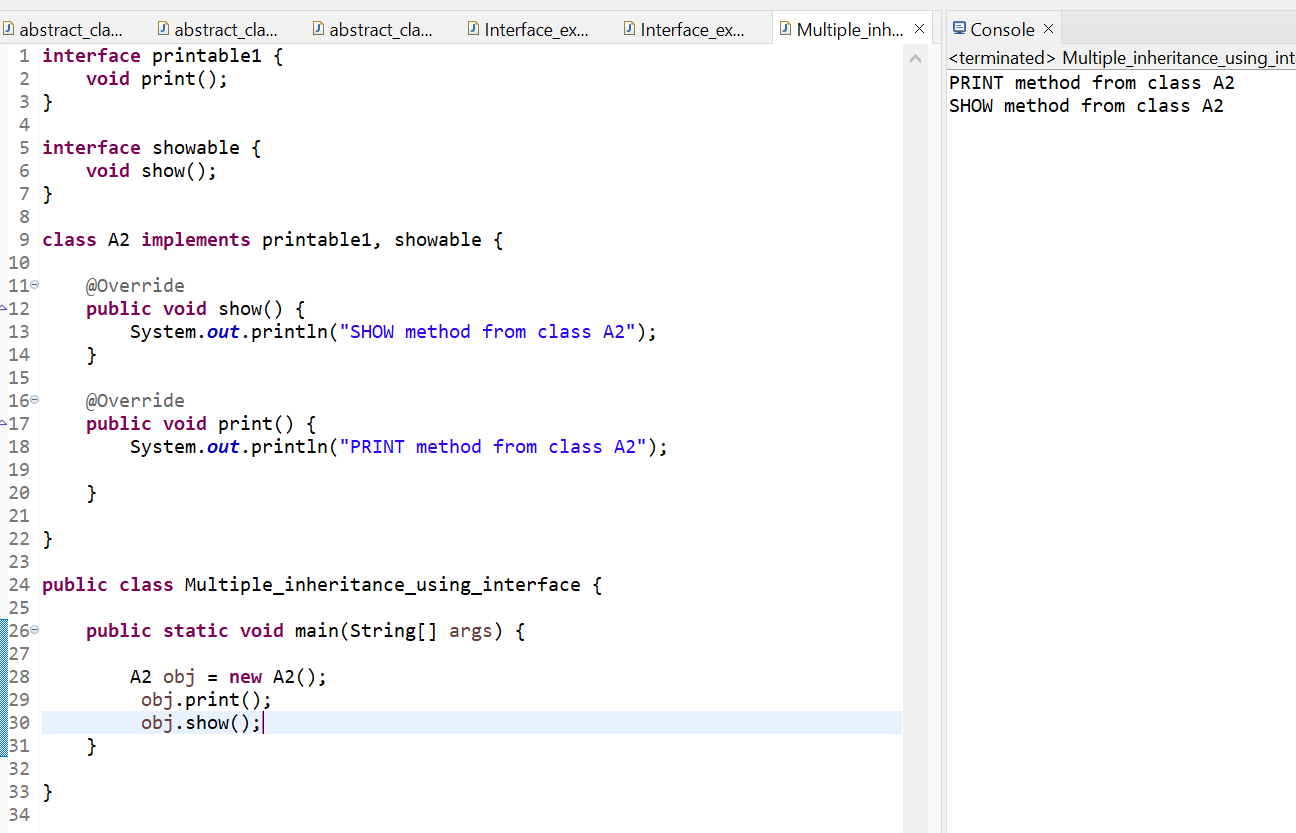
In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface



## Multiple inheritance in Java by interface

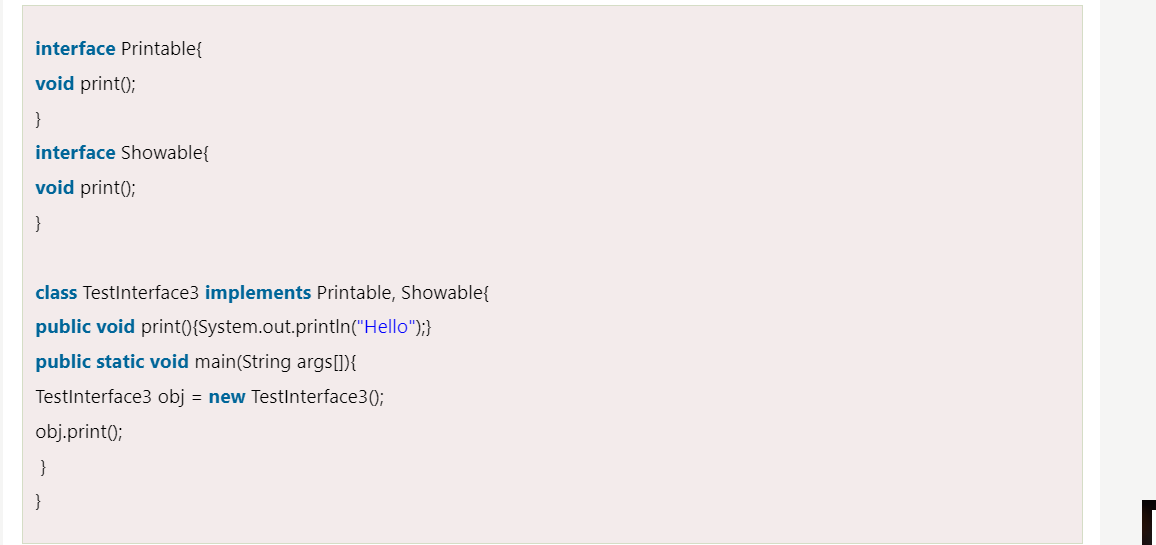
If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

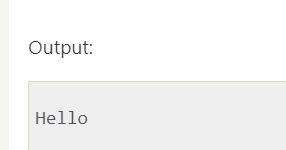




## Q) Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

As we have explained in the inheritance chapter, multiple inheritance is not supported in the case of [class](https://www.javatpoint.com/object-and-class-in-java) because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.Example :

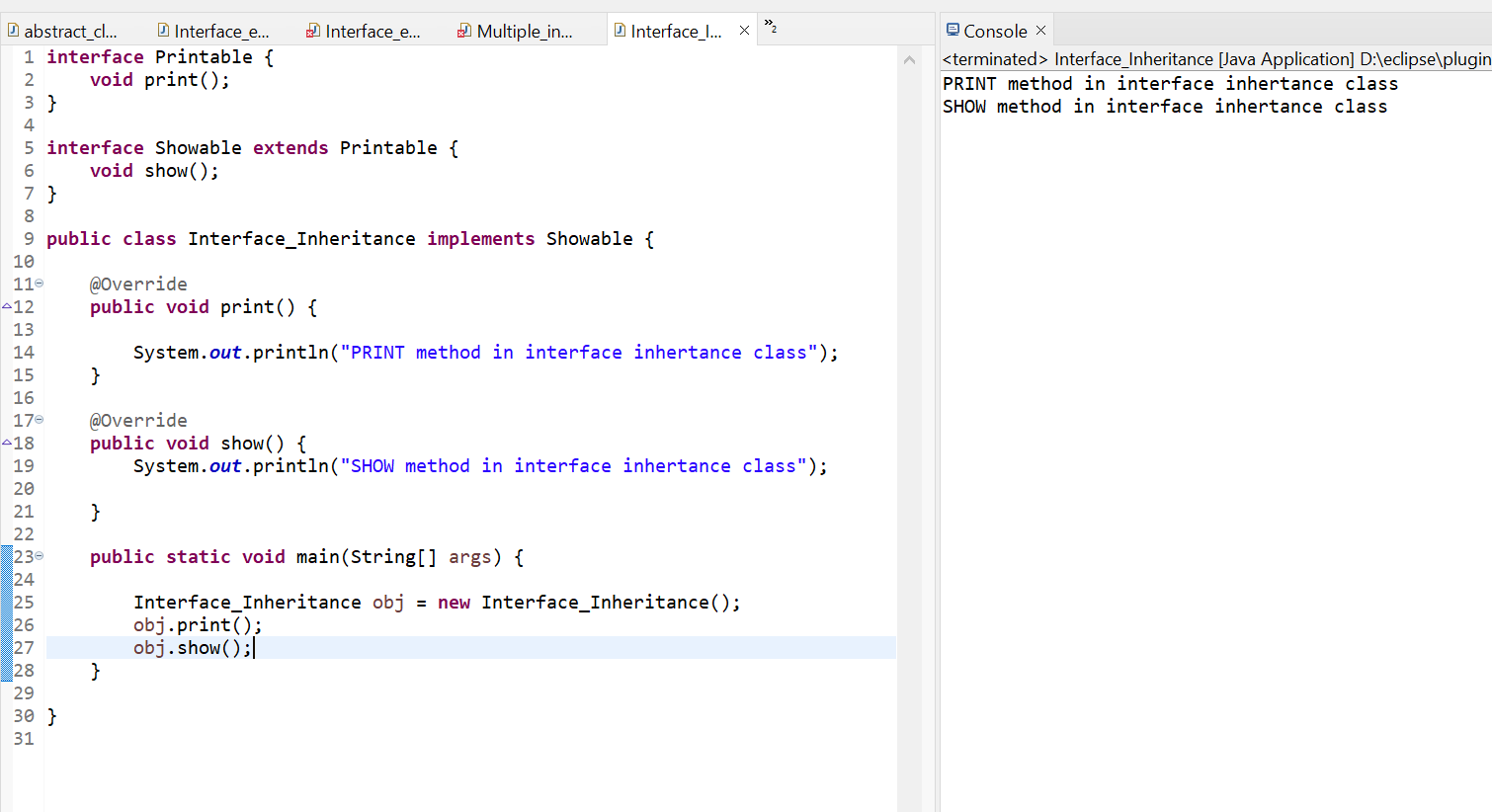




As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

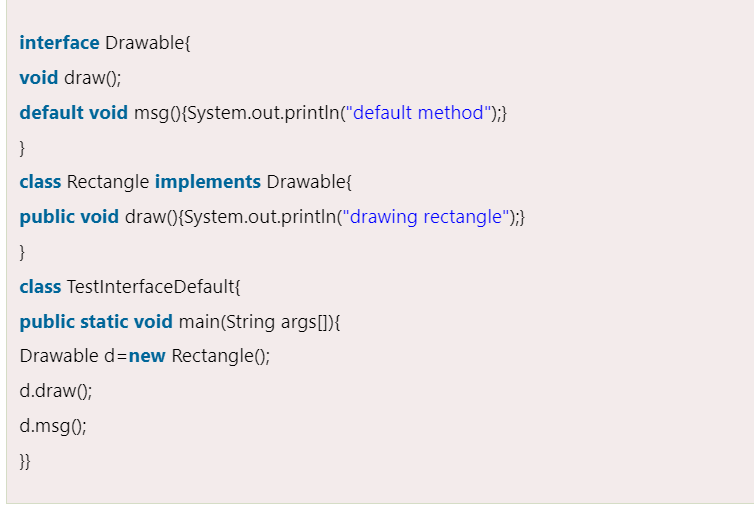
## Interface inheritance

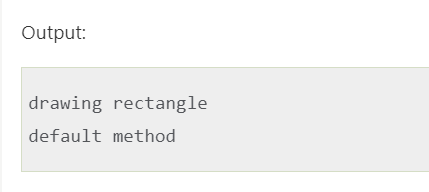
A class implements an interface, but one interface extends another interface.



## Java 8 Default Method in Interface

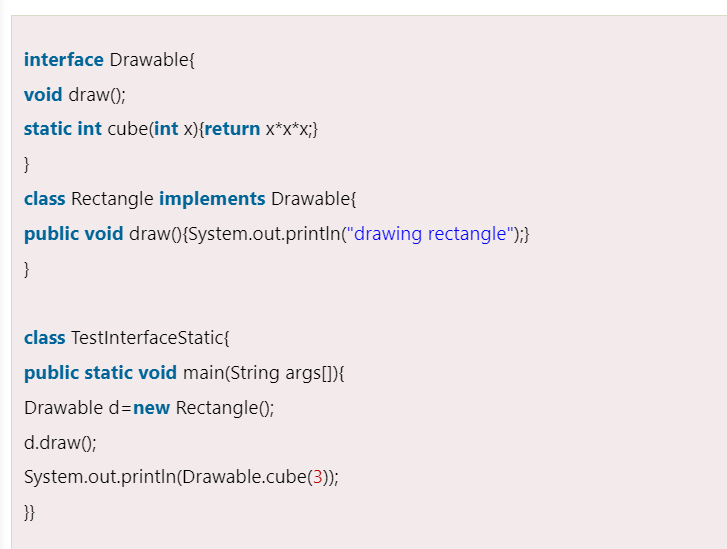
Since Java 8, we can have method body in interface. But we need to make it default method. Let's see an example:

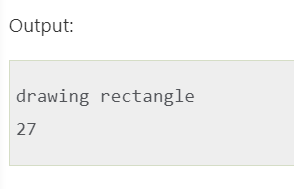


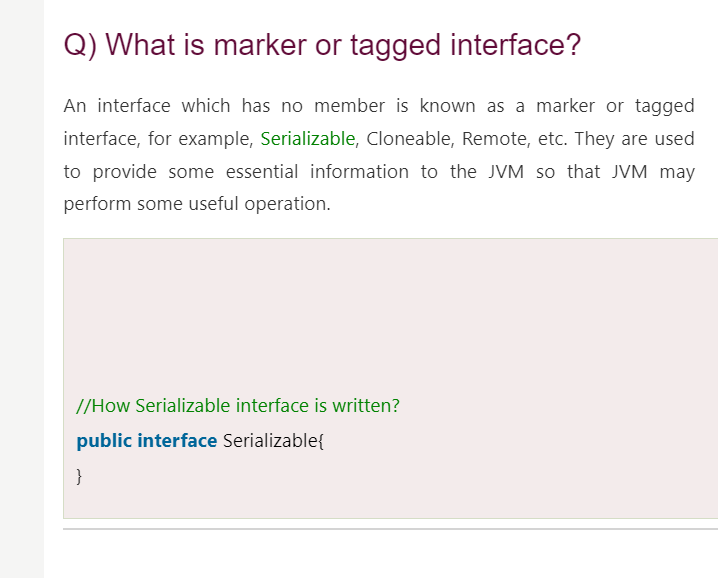


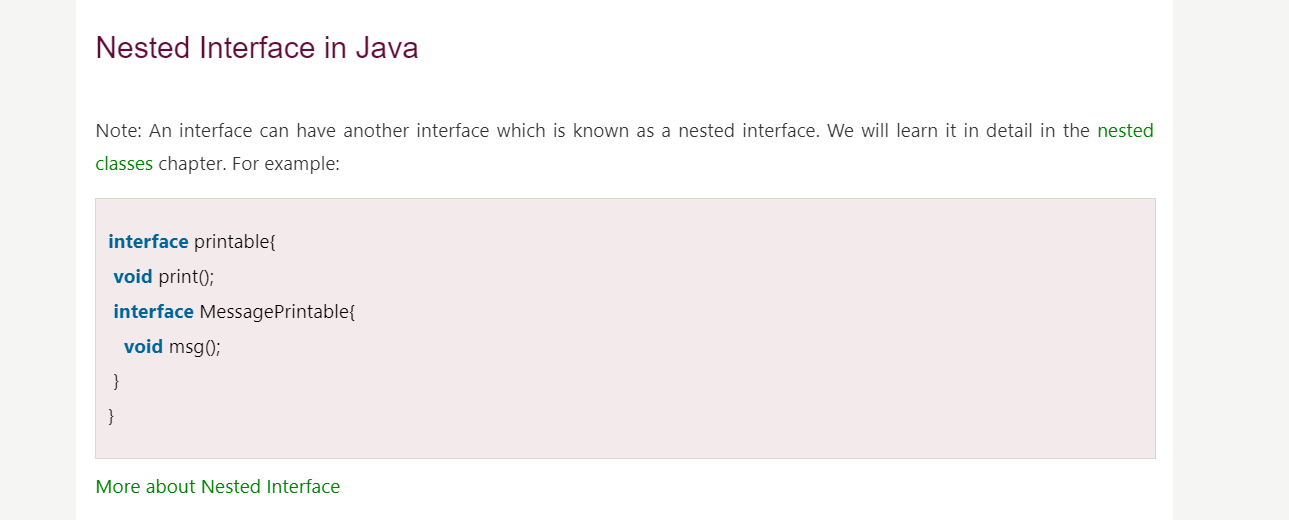
## Java 8 Static Method in Interface

Since Java 8, we can have static method in interface. Let's see an example:





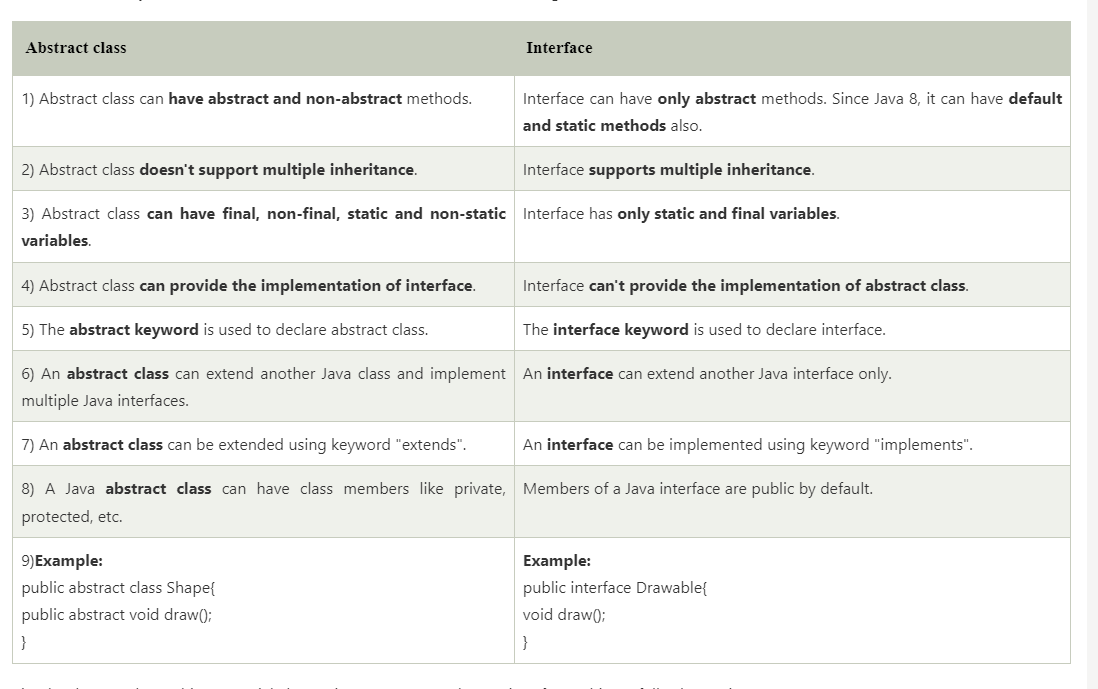




# Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

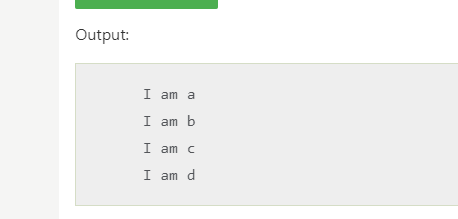
But there are many differences between abstract class and interface that are given below.



### Example of abstract class and interface in Java

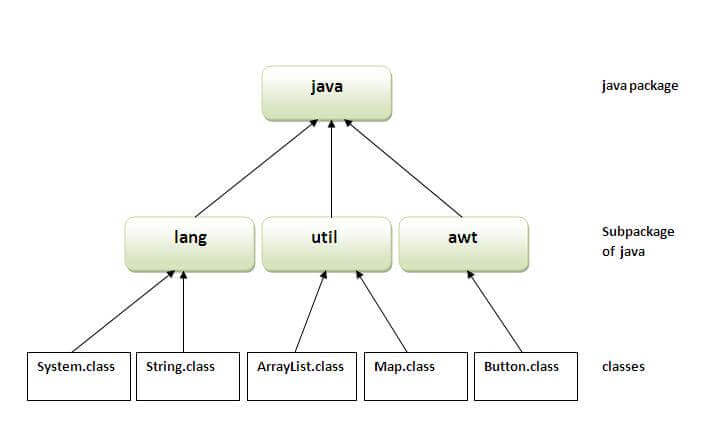
Let's see a simple example where we are using interface and abstract class both.





# Java Package

* A **java package** is a group of similar types of classes, interfaces and sub-packages.
* Package in java can be categorized in two form, built-in package and user-defined package.
* There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.



## Advantage of Java Package

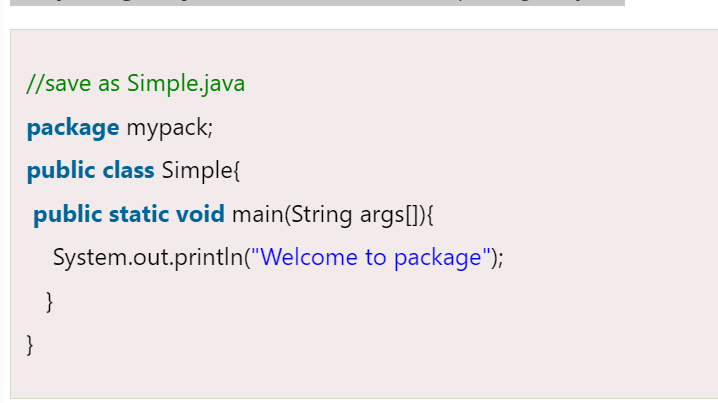
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

## Simple example of java package

The **package keyword** is used to create a package in java.



## How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

javac -d directory javafilename

For **example :**

**javac -d . Simple.java**

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

## How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

| **To Compile:** javac -d . Simple.java |
| --- |
| **To Run:** java mypack.Simple |

Output:Welcome to package

| The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder. How to access package from another package? There are three ways to access the package from outside the package.   1. import package.\*; 2. import package.classname; 3. fully qualified name.  1) Using packagename.\* If you use package.\* then all the classes and interfaces of this package will be accessible but not subpackages.  The import keyword is used to make the classes and interface of another package accessible to the current package.   2) Using packagename.classname If you import package.classname then only declared class of this package will be accessible.   3) Using fully qualified name  * If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface. * It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.     **NOTE :** If you import a package, subpackages will not be imported.  **If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.**   Subpackage in java  * Package inside the package is called the subpackage. It should be created to categorize the package further. * Let's take an example, Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.  The standard of defining package is domain.company.package e.g. com.javatpoint.bean or org.sssit.dao.  How to send the class file to another directory or drive? There is a scenario, I want to put the class file of A.java source file in classes folder of c: drive. For example:     To Compile: **e:\sources> javac -d c:\classes Simple.java** To Run:  | To run this program from e:\source directory, you need to set classpath of the directory where the class file resides. | | --- | | **e:\sources> set classpath=c:\classes;.;** | | **e:\sources> java mypack.Simple** |  Another way to run this program by -classpath switch of java: The -classpath switch can be used with javac and java tool.  To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:  **e:\sources> java -classpath c:\classes mypack.Simple**  Output:Welcome to package How to put two public classes in a package?  | If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same. For example: | | --- |  1. //save as A.java 3. **package** javatpoint; 4. **public** **class** A{} 5. //save as B.java 7. **package** javatpoint; 8. **public** **class** B{} |
| --- | --- | --- | --- | --- |