Java Modifiers

Modifiers

By now, you are quite familiar with the `public` keyword that appears in almost all of our examples:

```
public class Main
```

The `public` keyword is an **access modifier**, meaning that it is used to set the access level for classes, attributes, methods and constructors.

We divide modifiers into two groups:

- **Access Modifiers** - controls the access level
- **Non-Access Modifiers** - do not control access level, but provides other functionality

Access Modifiers

For **classes**, you can use either `public` or *default*:

Modifier	Description	Try it
`public`	The class is accessible by any other class	Try it »
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter	Try it »

For **attributes, methods and constructors**, you can use the one of the following:

Modifier	Description	Try it
`public`	The code is accessible for all classes	Try it »
`private`	The code is only accessible within the declared class	Try it »
default	The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter	Try it »
`protected`	The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter	Try it »

Non-Access Modifiers

For **classes**, you can use either `final` or `abstract` :

Modifier	Description	Try it
`final`	The class cannot be inherited by other classes (You will learn more about inheritance in the Inheritance chapter)	Try it »
`abstract`	The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters)	Try it »

For **attributes and methods**, you can use the one of the following:

Modifier	Description
`final`	Attributes and methods cannot be overridden/modified
`static`	Attributes and methods belongs to the class, rather than an object
`abstract`	Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example **abstract void run();**. The body is provided by the subclass (inherited from). You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters
`transient`	Attributes and methods are skipped when serializing the object containing them
`synchronized`	Methods can only be accessed by one thread at a time
`volatile`	The value of an attribute is not cached thread-locally, and is always read from the "main memory"

Final

If you don't want the ability to override existing attribute values, declare attributes as `final`:

Example

```
public class Main {



  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 50; // will generate an error: cannot assign a value to a final va
    myObj.PI = 25; // will generate an error: cannot assign a value to a final v
    System.out.println(myObj.x);
  }
}
```

Try it Yourself »

Static

A `static` method means that it can be accessed without creating an object of the class, unlike `public`:

Example

An example to demonstrate the differences between `static` and `public` methods:

```
public class Main {
  // Static method

    System.out.println("Static methods can be called without creating objects");
  }

  // Public method

    System.out.println("Public methods must be called by creating objects");
  }
  // Main method
  public static void main(String[ ] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would output an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method
  }
}
```

Try it Yourself »

Abstract

An `abstract` method belongs to an `abstract` class, and it does not have a body. The body is provided by the subclass:

Example

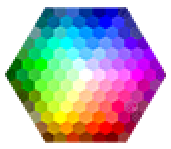
```
// Code from filename: Main.java
// abstract class

  public String fname = "John";
  public int age = 24;

}

// Subclass (inherit from Main)
class Student extends Main {
  public int graduationYear = 2018;

    System.out.println("Studying all day long");
  }
}
// End code from filename: Main.java
// Code from filename: Second.java
class Second {
  public static void main(String[] args) {
    // create an object of the Student class (which inherits attributes and meth
    Student myObj = new Student();

    System.out.println("Name: " + myObj.fname);
    System.out.println("Age: " + myObj.age);
    System.out.println("Graduation Year: " + myObj.graduationYear);
    myObj.study(); // call abstract method
  }
}
```

Try it Yourself »

❮ Previous

Next ❯

COLOR PICKER



Get certified
by completing
a Java
course today!

Get started

CODE GAME

Play Game

ADVERTISEMENT

ADVERTISEMENT

ADVERTISEMENT

Report Error

Spaces

Pro

Buy Certificate

Top Tutorials

HTML Tutorial
CSS Tutorial
JavaScript Tutorial
How To Tutorial
SQL Tutorial
Python Tutorial
W3.CSS Tutorial
Bootstrap Tutorial

PHP Tutorial
Java Tutorial
C++ Tutorial
jQuery Tutorial

Top References

HTML Reference
CSS Reference
JavaScript Reference
SQL Reference
Python Reference
W3.CSS Reference
Bootstrap Reference
PHP Reference
HTML Colors
Java Reference
Angular Reference
jQuery Reference

Top Examples

HTML Examples
CSS Examples
JavaScript Examples
How To Examples
SQL Examples
Python Examples
W3.CSS Examples
Bootstrap Examples
PHP Examples
Java Examples
XML Examples
jQuery Examples

Get Certified

HTML Certificate
CSS Certificate
JavaScript Certificate
Front End Certificate
SQL Certificate
Python Certificate
PHP Certificate
jQuery Certificate
Java Certificate
C++ Certificate
C# Certificate
XML Certificate

FORUM | ABOUT