Java Modifiers

‹ Previous Next ›

Modifiers

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

```
public class MyClass
```

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 in the Inheritance chapter)	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 in the Inheritance chapter
`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 variables as `final` :

Example

```
public class MyClass {
  final int x = 10;
  final double PI = 3.14;

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

Static

A static method means that it belongs to the class, and cannot be accessed from outside of the class, unlike `public` :

Example

```
public class MyClass {

  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating
objects");
```

```
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating
objects");
  }

}
```

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: Person.java
// abstract class
abstract class Person {
  public String fname = "John";
  public int age = 24;
  public abstract void study(); // abstract method
}

// Subclass (inherit from Person)
class Student extends Person {
  public int graduationYear = 2018;
  public void study() { // the body of the abstract method is provided
here
    System.out.println("Studying all day long");
  }
}
// End code from filename: Person.java

// Code from filename: MyClass.java
class MyClass {
  public static void main(String[] args) {
```

```
    // create an object of the Student class (which inherits attributes
and methods from Person)
    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
  }
}
```

Run example »

‹ Previous

Next ›