

What is the difference between public, protected, package-private and private in Java?

Asked 16 years, 2 months ago Modified 7 months ago

Viewed 2.7m times



3714



In Java, are there clear rules on when to use each of access modifiers, namely the default (package private), `public`, `protected` and `private`, while making `class` and `interface` and dealing with inheritance?

java

private

public

protected

access-modifiers

Share

Improve this question

Follow

edited Sep 11, 2018 at 14:54



Steve Chambers

39.3k ● 29 ● 172 ● 220

asked Oct 18, 2008 at 19:53



intrepion

38.8k ● 4 ● 25 ● 22

192 `private` hides from other classes within the package.
`public` exposes to classes outside the package.
`protected` is a version of `public` restricted only to subclasses. – [Museful](#) Feb 13, 2013 at 9:56 ✎

- 110 @Tennenrishin — No ; contrary to C++, in Java `protected` makes the method also accessible from the whole package. This stupidity in Java's visibility model breaks the goal of `protected` . — [TheMaskedCucumber](#) Aug 21, 2013 at 9:51
-
- 41 @Nicolas It is accessible from the whole package, with or without `protected` . As an access *modifier*, all that `protected` does is to expose to subclasses outside the package. — [Museful](#) Mar 14, 2014 at 10:59 ✎
-
- 18 @tennenrishin - well, that is what Nicolas said... and you are just repeating it now. What you originally said was that `protected` - and I quote - 'is a version of public restricted only to subclasses' which is not true by your own admission since `protected` also allows access through the whole package (ergo, it does not **restrict** access to subclasses.) — [luis.espinal](#) Apr 7, 2014 at 13:45 ✎
-
- 14 I also agree with Nicolas in that the `protected` access mode in Java is idiotic. What happened is that Java conflated horizontal (lattice) and vertical access restriction qualifiers. Default scope is a horizontal/lattice restriction with the lattice being the package. Public is another horizontal restriction where the lattice is the whole world. Private and (C++) `protected` are vertical. It would have been better if we had a cross-cut access, say, `protected-package` for the rare cases where we actually needed it, leaving `protected` to be equivalent to the C++ version of `protected`. — [luis.espinal](#) Apr 7, 2014 at 13:53
-

32 Answers

Sorted by:

Highest score (default)



1

2

Next

[The official tutorial](#) may be of some use to you.



6409



	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	
<code>public</code>	✓	✓	✓	✓	
<code>protected</code>	✓	✓	✓	✓	
<code>no modifier</code>	✓	✓	✓	✗	
<code>private</code>	✓	✗	✗	✗	

✓: accessible

✗: not accessible

Share Improve this answer

edited May 23 at 20:16

Follow



Bolpat

1,695 ● 1 ● 14 ● 28

answered Oct 18, 2008 at 19:57



David Segonds


84.7k ● 10 ● 49 ● 66


19 *World is within your project.* I should explain further. Libraries are within your project, and if you're creating a library, they would expose these public classes and methods as well. So, saying just within your project is a bit off. "Everything that uses it" is a better description. – [adprocas](#) May 3, 2018 at 13:02

7 For example, if I have `MyClass` and I'm doing `AnotherClass extends MyClass` I will have access to all protected and public methods and properties from within `AnotherClass`. If I do `MyClass myClass = new MyClass();` in `AnotherClass` somewhere - let's say the

constructor - I will only have access to the public methods if it is in a different package. Note that if I do `= new MyClass() { @Override protected void protectedMethod() { //some logic } };` it appears that I can access protected methods, but this kind of the same as extending it, but inline instead. – [adprocas](#) May 4, 2018 at 12:25

15 Unfortunately, this answer is a gross oversimplification. Reality is a bit more complicated, especially when you consider `protected` (which is actually quite a difficult access modifier to fully understand - most people who think they know what `protected` means really don't). Also, as Bohemian pointed out, it doesn't answer the question - it says *nothing* about when to use each access modifier. In my opinion, this answer isn't *quite* bad enough to downvote, but close. But over 4000 upvotes? How did this happen? – [Dawood ibn Kareem](#) Jul 11, 2018 at 6:26

22 @niks protected members CAN be accessed by subclasses from a different package. Isn't that the point? Otherwise what's the difference between default and protected? – [Jack](#) Oct 15, 2020 at 12:30 

8 @Jack Yeah, niks' comment is wrong despite the many upvotes stating otherwise. The Java Tutorials link in the answer clearly says that protected members can also be accessed within subclasses from a different package. It seems that he/she meant "package-level" instead of "protected", or was referring to a different edit. – [Piovezan](#) Feb 5, 2021 at 2:57 



547

(Caveat: I am not a Java programmer, I am a Perl programmer. Perl has no formal protections which is perhaps why I understand the problem so well :))



Private



Like you'd think, only the **class** in which it is declared can see it.

Package Private

It can only be seen and used by the **package** in which it was declared. This is the default in Java (which some see as a mistake).

Protected

Package Private + can be seen by subclasses or package members.

Public

Everyone can see it.

Published

Visible outside the code I control. (While not Java syntax, it is important for this discussion).

C++ defines an additional level called "friend" and the less you know about that the better.

When should you use what? The whole idea is encapsulation to hide information. As much as possible

you want to hide the detail of how something is done from your users. Why? Because then you can change them later and not break anybody's code. This lets you optimize, refactor, redesign, and fix bugs without worrying that someone was using that code you just overhauled.

So, the rule of thumb is to make things only as visible as they have to be. Start with private and only add more visibility as needed. Only make public that which is necessary for the user to know, every detail you make public cramps your ability to redesign the system.

If you want users to be able to customize behaviors, rather than making internals public so they can override them, it's often a better idea to shove those guts into an object and make that interface public. That way they can simply plug in a new object. For example, if you were writing a CD player and wanted the "go find info about this CD" bit customizable, rather than make those methods public you'd put all that functionality into its object and make just your object getter/setter public. In this way being stingy about exposing your guts encourages good composition and separation of concerns

I stick with just "private" and "public". Many OO languages just have that. "Protected" can be handy, but it's a cheat. Once an interface is more than private it's outside of your control and you have to go looking in other people's code to find uses.

This is where the idea of "published" comes in. Changing an interface (refactoring it) requires that you find all the code which is using it and change that, too. If the interface is private, well no problem. If it's protected you have to go find all your subclasses. If it's public you have to go find all the code which uses your code. Sometimes this is possible, for example, if you're working on corporate code that's for internal use only it doesn't matter if an interface is public. You can grab all the code out of the corporate repository. But if an interface is "published", if there is code using it outside your control, then you're hosed. You must support that interface or risk breaking code. Even protected interfaces can be considered published (which is why I don't bother with protected).

Many languages find the hierarchical nature of public/protected/private to be too limiting and not in line with reality. To that end, there is the concept of a [trait class](#), but that's another show.

Share Improve this answer

Follow

edited Dec 20, 2021 at 0:53



R.Kabir

11 ● 2

answered Oct 18, 2008 at 21:17



Schwern


164k ● 27 ● 218 ● 362


27 friends -> "The less you know about it the better" ---> It gives selective visibility, which is still superior to package privacy. In C++, it has its uses, because not all functions can be

member functions, and friends is better than public'ing. Of course there is a danger of misuse by evil minds.

– [Sebastian Mach](#) Jun 7, 2011 at 10:13 

- 33 It should also be noted that "protected" in C++ has a different meaning - a protected method is effectively private, but can still be called from an inheriting class. (As opposed to Java where it can be called by any class within the same package.) – [Rhys van der Waerden](#) Oct 2, 2011 at 12:34
-

- 9 @RhysvanderWaerden C# is the same as C++ in this aspect. I find it pretty odd that Java doesn't allow to declare a member that's accessible to the subclass but not the entire package. It's sort of upside down to me - a package is broader scope than a child class! – [Konrad Morawski](#) Oct 15, 2013 at 17:36 
-

- 16 @KonradMorawski IMHO package is smaller scope than subclass. If you haven't declared your class final, users should be able to subclass it - so java protected is part of your published interface. OTOH, packages are implicitly developed by a single organization: e.g. com.mycompany.mypackage. If your code declares itself in my package, you implicitly declare yourself part of my organization, so we should be communicating. Thus, package publishes to a smaller/easier to reach audience (people in my company) than subclass (people who extend my object) and so counts as lower visibility. – [Eponymous](#) May 22, 2014 at 20:37 
-

- 3 `friend` is good for defining special relationships between classes. It allows superior encapsulation in many cases when used correctly. For example it can be used by a privileged factory class to inject internal dependencies into a constructed type. It has a bad name because people who don't care about correctly maintaining a well designed object model can abuse it to ease their workload. – [Dennis](#) Dec 8, 2014 at 10:05
-



Here's a better version of the table, that also includes a column for modules.

546



	Different class but same package	Different package but subclass	Unrelated class but same module	Different module and p1 not exported
<pre>package p1; public class A { private int i; int j; protected int k; public int l; }</pre>	<pre>package p1; class B {</pre>	<pre>package p2; class C extends A {</pre>	<pre>package p2; class D {</pre>	<pre>package x; class E {</pre>
	Accessible	Accessible	Inaccessible	Inaccessible

Explanations

- A **private** member (**i**) is *only* accessible within the same class as it is declared.
- A member with **no access modifier** (**j**) is only accessible within classes in the same package.
- A **protected** member (**k**) is accessible within all classes in the same package *and* within subclasses in other packages.
- A **public** member (**l**) is accessible to all classes (unless it resides in a [module](#) that does not export the package it is declared in).

Which modifier to choose?

Access modifiers is a tool to help you to prevent accidentally breaking encapsulation^(*). Ask yourself if you

intend the member to be something that's internal to the class, package, class hierarchy or not internal at all, and choose access level accordingly.

Examples:

- A field `long internalCounter` should probably be private since it's mutable and an implementation detail.
- A class that should only be instantiated in a factory class (in the same package) should have a package restricted constructor, since it shouldn't be possible to call it directly from outside the package.
- An internal `void beforeRender()` method called right before rendering and used as a hook in subclasses should be protected.
- A `void saveGame(File dst)` method which is called from the GUI code should be public.

(*) [What is Encapsulation exactly?](#)

Share Improve this answer

edited Aug 17, 2022 at 18:30

Follow

answered Nov 10, 2015 at 10:27



aioobe

421k ● 114 ● 827 ● 838

27 Just saying: there are a lot of people who have problems with distinguishing red/green coloring. Tables using

red/green (or yellow/orange/...) coloring schemes are rarely "better" at anything ;-)) – [GhostCat](#) Oct 11, 2018 at 10:50 ✎

10 @GhostCat, I disagree. I think red/green aligns intuitively with "works"/"does not work" for many people, i.e. it is better than many alternatives. – [aioobe](#) Nov 14, 2018 at 14:10

16 colourblindawareness.org/colour-blindness/... ... *The 8% of colour blind men can be divided approximately into 1% deuteranopes, 1% protanopes, 1% protanomalous and 5% **deuteranomalous**.* And as I am one of those 50% of that 5%, rest assured: red/green sucks. – [GhostCat](#) Nov 14, 2018 at 14:13 ✎

7 @GhostCat Ok.. that's a larger part of the population than I expected. I uploaded the image in this [color blindness simulator](#) and tested all different modes. Even in monochromacy/achromatopsia mode the color difference is reasonable. Can you see the difference or is the simulator off? (I'm still of the opinion that red/green is very intuitive for color seeing people.) – [aioobe](#) Nov 14, 2018 at 14:32 ✎

5 I can see the difference, but I am also able to pass half of the color blindness tests that we have to do in Germany for the drivers licence ;-)) ... but I think such a simulator is "good enough". – [GhostCat](#) Nov 14, 2018 at 14:40



231



	highest precedence <-----> lowes		
* _____	+	+	+
\ xCanBeSeenBy	this	any class	this sub
\ _____	class	in same	in anoth
\	nonsubbed	package	package
Modifier of x \			
* _____	+	+	+
public	✓	✓	✓
protected	✓	✓	✓

	+		+		+
package-private (no modifier)		✓		✓	✗
private		✓		✗	✗

Share Improve this answer

edited Oct 18, 2023 at 18:07

Follow



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Jan 9, 2013 at 21:42



Abdull

27.7k ● 27 ● 134 ● 180

- 2 It is worth putting in words - "Protected modifier makes the object available across other packages, whereas default/no-modifier restricts access to the same package" – [vanguard69](#) Aug 15, 2016 at 16:53
- 3 @vanguard69, the `protected` modifier makes the marked *thing* (class, method, or field) available to some other class in some other package **only iff** said other class is a subclass of the class where that `protected` - marked *thing* is declared. – [Abdull](#) Aug 15, 2016 at 18:14



178

Easy rule. Start with declaring everything private. And then progress towards the public as the needs arise and design warrants it.



When exposing members ask yourself if you are exposing representation choices or abstraction choices. The first is something you want to avoid as it will





introduce too many dependencies on the actual representation rather than on its observable behavior.

As a general rule I try to avoid overriding method implementations by subclassing; it's too easy to screw up the logic. Declare abstract protected methods if you intend for it to be overridden.

Also, use the `@Override` annotation when overriding to keep things from breaking when you refactor.

Share Improve this answer

Follow

edited Jul 24, 2018 at 14:45



kvantour

26.4k ● 4 ● 52 ● 79

answered Oct 18, 2008 at 20:00



John Nilsson

17.3k ● 8 ● 35 ● 42

4 @RuchirBaronia, "world" = all code in the application, regardless where it is. – Andrejs Feb 26, 2016 at 19:12

Although the info in this answer may be useful, the question was not if we should start by declaring everything private and then exposing things, if we need to do that. – nbro Apr 5, 2022 at 10:39



123



It's actually a bit more complicated than a simple grid shows. The grid tells you whether an access is allowed, but what exactly constitutes an access? Also, access levels interact with nested classes and inheritance in complex ways.

The "default" access (specified by the absence of a keyword) is also called [package-private](#). Exception: in an interface, no modifier means public access; modifiers other than public are forbidden. Enum constants are always public.

Summary

Is an access to a member with this access specifier allowed?

- Member is `private`: Only if member is defined within the same class as calling code.
- Member is package private: Only if the calling code is within the member's immediately enclosing package.
- Member is `protected`: Same package, or if member is defined in a superclass of the class containing the calling code.
- Member is `public`: Yes.

What access specifiers apply to

Local variables and formal parameters cannot take access specifiers. Since they are inherently inaccessible to the outside according to scoping rules, they are effectively private.

For classes in the top scope, only `public` and package-private are permitted. This design choice is presumably because `protected` and `private` would be redundant at the package level (there is no inheritance of packages).

All the access specifiers are possible on class members (constructors, methods and static member functions, nested classes).

Related: [Java Class Accessibility](#)

Order

The access specifiers can be strictly ordered

`public > protected > package-private > private`

meaning that `public` provides the most access, `private` the least. Any reference possible on a private member is also valid for a package-private member; any reference to a package-private member is valid on a protected member, and so on. (Giving access to protected members to other classes in the same package was considered a mistake.)

Notes

- [A class's methods are allowed to access private members of other objects of the same class.](#) More precisely, a method of class C can access private members of C on objects of any subclass of C. Java doesn't support restricting access by instance, only by class. (Compare with Scala, which does support it using `private[this]`.)
- You need access to a constructor to construct an object. Thus if all constructors are private, the class can only be constructed by code living within the class (typically static factory methods or static variable initializers). Similarly for package-private or protected constructors.
 - Only having private constructors also means that the class cannot be subclassed externally, since Java requires a subclass's constructors to implicitly or explicitly call a superclass constructor. (It can, however, contain a nested class that subclasses it.)

Inner classes

You also have to consider *nested* scopes, such as inner classes. An example of the complexity is that inner classes have members, which themselves can take access modifiers. So you can have a private inner class with a public member; can the member be accessed? (See below.) The general rule is to look at scope and

think recursively to see whether you can access each level.

However, this is quite complicated, and for full details, [consult the Java Language Specification](#). (Yes, there have been compiler bugs in the past.)

For a taste of how these interact, consider this example. It is possible to "leak" private inner classes; this is usually a warning:

```
class Test {
    public static void main(final String ... args) {
        System.out.println(Example.leakPrivateClass())
        Example.leakPrivateClass().secretMethod(); //
    }
}

class Example {
    private static class NestedClass {
        public void secretMethod() {
            System.out.println("Hello");
        }
    }
    public static NestedClass leakPrivateClass() {
        return new NestedClass();
    }
}
```

Compiler output:

```
Test.java:4: secretMethod() in Example.NestedClass is
inaccessible class or interface
    Example.leakPrivateClass().secretMethod(); //
                                ^
1 error
```

Some related questions:

- [Java - Method accessibility inside package-private class?](#)

Share Improve this answer

edited May 23, 2017 at 12:34

Follow



Community Bot

1 • 1

answered Sep 13, 2012 at 7:38



Mechanical snail

30.6k • 14 • 90 • 113

-
- 3 *"modifiers other than public are forbidden"* — as of Java 9, this is no longer the case: interfaces can also have private methods. — [MC Emperor](#) Aug 25, 2018 at 20:34
-



99



As a rule of thumb:

- `private`: class scope.
- `default` (or `package-private`): package scope.
- `protected`: `package scope + child` (like package, but we can subclass it from different packages). The protected modifier always keeps the "parent-child" relationship.
- `public`: everywhere.

As a result, if we divide access right into three rights:

- **(D)irect** (invoke from a method inside the same class, or via "this" syntax).

- **(R)eference** (invoke a method using a reference to the class, or via "dot" syntax).
- **(I)nheritance** (via subclassing).

then we have this simple table:

	Same Package	Different Packages
<code>private</code>	D	
<code>package-private</code> (no modifier)	D R I	
<code>protected</code>	D R I	I
<code>public</code>	D R I	R I

Share Improve this answer

edited Jun 25, 2019 at 15:55

Follow

answered Dec 18, 2012 at 18:01



Hoa Nguyen

14.5k ● 11 ● 46 ● 46



In very short

58



- `public`: accessible from everywhere.
- `protected`: accessible by the classes of the same package and the subclasses residing in any package.



- default (no modifier specified): accessible by the classes of the same package.
- `private`: accessible within the same class only.

Share Improve this answer

edited Sep 24, 2016 at 7:05

Follow

answered Oct 27, 2012 at 17:49



Ravi

31.4k ● 43 ● 122 ● 180



56



The most misunderstood access modifier in Java is `protected`. We know that it's similar to the default modifier with one exception in which subclasses can see it. But how? Here is an example which hopefully clarifies the confusion:



- Assume that we have 2 classes; `Father` and `Son`, each in its own package:

```
package fatherpackage;
```

```
public class Father  
{  
  
}
```

```
package sonpackage;
```

```
public class Son extends Father  
{
```

```
}
```

- Let's add a protected method `foo()` to `Father`.

```
package fatherpackage;

public class Father
{
    protected void foo(){}
}
```

- The method `foo()` can be called in 4 contexts:
 1. Inside a class that is located in the same package where `foo()` is defined (`fatherpackage`):

```
package fatherpackage;

public class SomeClass
{
    public void someMethod(Father f, Son s)
    {
        f.foo();
        s.foo();
    }
}
```

2. Inside a subclass, on the current instance via `this` or `super`:

```
package sonpackage;

public class Son extends Father
{
    public void sonMethod()
    {
        this.foo();
        super.foo();
    }
}
```

```
    }  
}
```

3. On an reference whose type is the same class:

```
package fatherpackage;  
  
public class Father  
{  
    public void fatherMethod(Father f)  
    {  
        f.foo(); // valid even if foo() is pri  
    }  
}
```

```
package sonpackage;  
  
public class Son extends Father  
{  
    public void sonMethod(Son s)  
    {  
        s.foo();  
    }  
}
```

4. On an reference whose type is the parent class and it is *inside* the package where `foo()` is defined (`fatherpackage`) [This can be included inside context no. 1]:

```
package fatherpackage;  
  
public class Son extends Father  
{  
    public void sonMethod(Father f)  
    {  
        f.foo();  
    }  
}
```

```
}  
}
```

- The following situations are not valid.
 1. On an reference whose type is the parent class and it is *outside* the package where `foo()` is defined (`fatherpackage`):

```
package sonpackage;  
  
public class Son extends Father  
{  
    public void sonMethod(Father f)  
    {  
        f.foo(); // compilation error  
    }  
}
```

2. A non-subclass inside a package of a subclass (A subclass inherits the protected members from its parent, and it makes them private to non-subclasses):

```
package sonpackage;  
  
public class SomeClass  
{  
    public void someMethod(Son s) throws Exception  
    {  
        s.foo(); // compilation error  
    }  
}
```

Share Improve this answer

edited Sep 14, 2017 at 19:35

Follow

answered Nov 15, 2013 at 20:06



Eng.Fouad

117k ● 74 ● 323 ● 426

`Object#clone()` is an example of a `protected` member.
– Eng.Fouad Nov 15, 2013 at 20:08

1 What is the difference between doing `super.foo()` and the first invalid situation `f.foo()`? – cst1992 Oct 28, 2017 at 9:18

7 @cst1992 It's confusing but see the Java Language Specification 6.6.2: "A protected member or constructor of an object may be accessed from outside the package in which it is declared only by code that is responsible for the implementation of that object". With `super.foo()` the reference "super" is "directly responsible for the implementation" but the reference "f" is not. Why? Because you can be 100% certain that "super" is of type `Father`, but not for "f"; at run-time it could be some other sub-type of `Father`. See docs.oracle.com/javase/specs/jls/se9/html/... – skomisa Jan 30, 2018 at 17:55

2 It's refreshing to read an answer from someone who understands `protected`. Unfortunately, all the other answers on this page that define `protected` get it a little bit wrong. – Dawood ibn Kareem Jul 11, 2018 at 6:20

@MarcoSulla skomisa also quoted from "the official Oracle site" but his quote is from the *spec*, whereas your quote is from a *tutorial*. The tutorial omits the detail that access to protected members from subclasses outside the package depends on the reference. – Jim Balter Nov 17, 2023 at 20:02

Private



30



- Methods, Variables and Constructors

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.

- Class and Interface

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Note

Variables that are declared private can be accessed outside the class if public getter methods are present in the class. Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

Protected

- Class and Interface

The protected access modifier cannot be applied to class and interfaces.

Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

Note

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

Public

A class, method, constructor, interface etc declared public can be accessed from any other class.

Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

- *Different Packages*

However if the public class we are trying to access is in a different package, then the public class still need to be imported.

Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Default -No keyword:

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

- *Within the same Packages*

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

Note

We cannot Override the Static fields.if you try to override it does not show any error but it doesnot work what we except.

Related Answers

- [Overriding static methods in java](#)

References links

http://docs.oracle.com/javase/tutorial/java/javaOO/access_control.html

http://www.tutorialspoint.com/java/java_access_modifiers.htm

Share Improve this answer

Follow

edited May 23, 2017 at 11:47



Community Bot

1 • 1

answered Jan 22, 2014 at 13:08

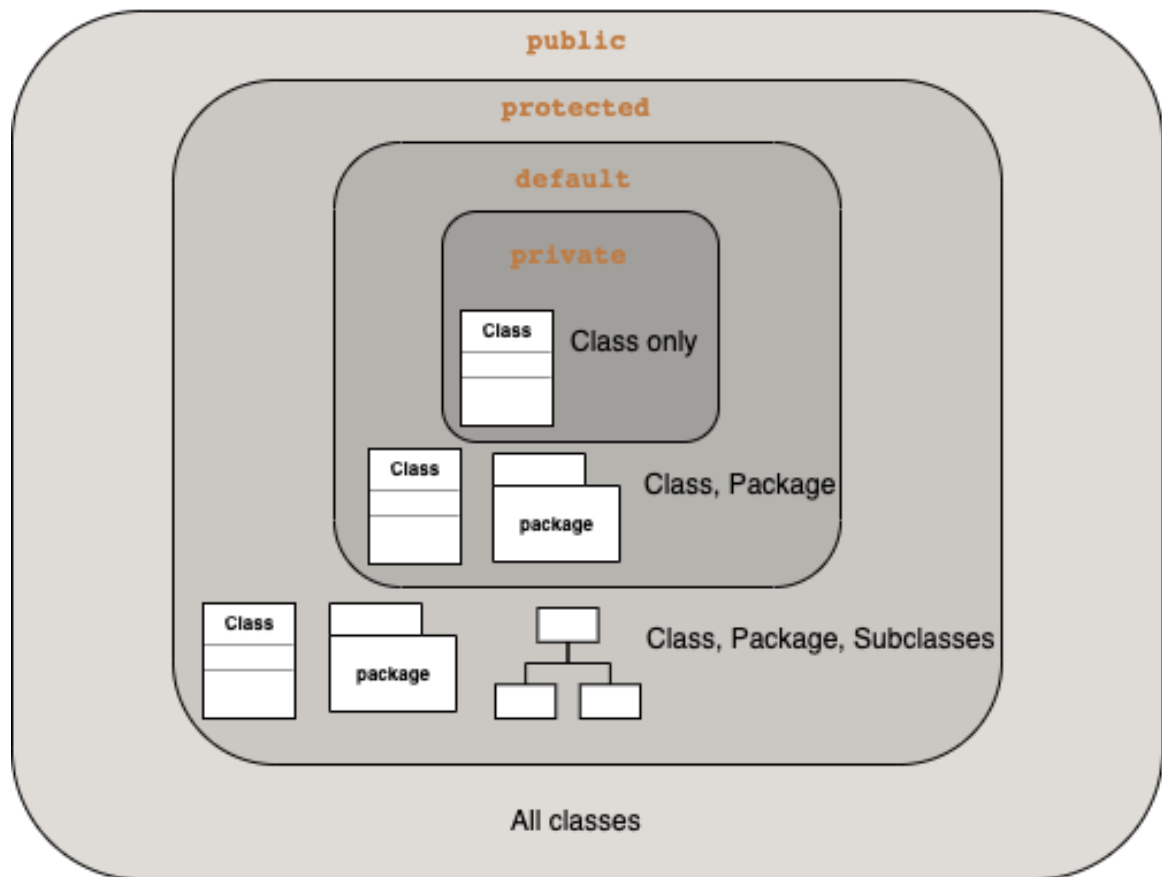


Nambi

12k ● 4 ● 39 ● 50

Java access modifiers

24



Access modifier can be applicable for `class`, `field` [\[About\]](#), `method`. Try to access, subclass or override this.

- Access to `field` or `method` is through a `class`.
- Inheritance and **Open Closed Principle** [\[About\]](#)
 - Successor `class` (subclass) access modifier can be **any**.
 - Successor `method` (override) access modifier should be the **same or expand it**

Top level class(first level scope) can be `public` and `default`. Nested class [\[About\]](#) can have any of them

package is not applying for package hierarchy

[\[Swift access modifiers\]](#)

Share Improve this answer

edited Dec 29, 2023 at 14:30

Follow

answered Dec 6, 2019 at 20:42



yoAlex5

34k ● 10 ● 223 ● 239



22

The difference can be found in the links already provided but which one to use usually comes down to the "Principle of Least Knowledge". Only allow the least visibility that is needed.



Share Improve this answer

answered Oct 18, 2008 at 20:00



Follow



Joe Phillips

51k ● 33 ● 108 ● 165



21

Private: Limited access to class only

Default (no modifier): Limited access to class and package



Protected: Limited access to class, package and subclasses (both inside and outside package)



Public: Accessible to class, package (all), and subclasses... In short, everywhere.

Share Improve this answer

edited Nov 11, 2017 at 22:26

Follow



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Jun 18, 2014 at 7:13



samkit shah

657 ● 7 ● 18



Access modifiers are there to restrict access at several levels.

17



Public: It is basically as simple as you can access from any class whether that is in same package or not.



To access if you are in same package you can access directly, but if you are in another package then you can create an object of the class.



Default: It is accessible in the same package from any of the class of package.

To access you can create an object of the class. But you can not access this variable outside of the package.

Protected: you can access variables in same package as well as subclass in any other package. so basically it is **default + Inherited** behavior.

To access protected field defined in base class you can create object of child class.

Private: it can be access in same class.

In non-static methods you can access directly because of **this** reference (also in constructors) but to access in static methods you need to create object of the class.

Share Improve this answer

Follow

edited Nov 11, 2017 at 22:28



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Dec 17, 2014 at 6:29



Prashant

2,604 ● 2 ● 22 ● 26



Access modifiers in Java.

17



Java access modifiers are used to provide access control in Java.

1. Default:



Accessible to the classes in the same package only.



For example,

```
// Saved in file A.java
package pack;

class A{
    void msg(){System.out.println("Hello");}
}

// Saved in file B.java
package mypack;
import pack.*;

class B{
```

```
public static void main(String args[]){
    A obj = new A(); // Compile Time Error
    obj.msg(); // Compile Time Error
}
}
```

This access is more restricted than public and protected, but less restricted than private.

2. Public

Can be accessed from anywhere. (Global Access)

For example,

```
// Saved in file A.java

package pack;
public class A{
    public void msg(){System.out.println("Hello");}
}

// Saved in file B.java

package mypack;
import pack.*;

class B{
    public static void main(String args[]){
        A obj = new A();
        obj.msg();
    }
}
```

Output:Hello

3. Private

Accessible only inside the same class.

If you try to access private members on one class in another will throw compile error. For example,

```
class A{
    private int data = 40;
    private void msg(){System.out.println("Hello java");
}

public class Simple{
    public static void main(String args[]){
        A obj = new A();
        System.out.println(obj.data); // Compile Time Error
        obj.msg(); // Compile Time Error
    }
}
```

4. Protected

Accessible only to the classes in the same package and to the subclasses

For example,

```
// Saved in file A.java
package pack;
public class A{
    protected void msg(){System.out.println("Hello");}
}

// Saved in file B.java
package mypack;
import pack.*;

class B extends A{
    public static void main(String args[]){
        B obj = new B();
    }
}
```

```
obj.msg();
}
}
```

Output: Hello

Most Restrictive ← → Least Restrictive				
Access Modifiers ->	private	Default/no-access	protected	public
Inside class	Y	Y	Y	Y
Same Package Class	N	Y	Y	Y
Same Package Sub-Class	N	Y	Y	Y
Other Package Class	N	N	N	Y
Other Package Sub-Class	N	N	Y	Y

Same rules apply for inner classes too, they are also treated as outer class properties

Share Improve this answer

edited Jun 20, 2020 at 9:12

Follow



Community Bot

1 ● 1

answered Aug 7, 2016 at 14:16



Aftab

2,943 ● 1 ● 35 ● 43



Visible to the package. The default. No modifiers are needed.

14



Visible to the class only (**private**).

Visible to the world (**public**).



Visible to the package and all subclasses (**protected**).



Variables and methods can be declared without any modifiers that are called. Default examples:

```
String name = "john";

public int age(){
    return age;
}
```

Private access modifier - private:

Methods, variables and constructors that are declared private can only be accessed within the declared class itself. The private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Variables that are declared private can be accessed outside the class if public getter methods are present in the class.

Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

Examples:

```
Public class Details{

    private String name;

    public void setName(String n){
        this.name = n;
    }
}
```

```
    public String getName(){  
        return this.name;  
    }  
}
```

Public access modifier - public:

A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java universe.

However, if the public class we are trying to access is in a different package, then the public class still need to be imported.

Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Example:

```
public void cal(){  
  
}
```

Protected access modifier - protected:

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in another package or any class within the package of the protected members' class.

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

```
class Van{  
    protected boolean speed(){  
    }  
}  
  
class Car{  
    boolean speed(){  
    }  
}
```

Share Improve this answer

Follow

edited Nov 11, 2017 at 22:32



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered May 26, 2015 at 14:54



amila isura

1,070 ● 1 ● 17 ● 26



13

- **public** - accessible from anywhere in the application.
- **default** - accessible from package.



- **protected** - accessible from package and sub-classes in other package. as well
- **private** - accessible from its class only.



Share Improve this answer

edited Mar 16, 2016 at 15:31

Follow



Alex Weitz

3,359 ● 5 ● 38 ● 61

answered Aug 29, 2015 at 7:15



Shailendra Singh

483 ● 5 ● 9



[This page writes well about the protected & default access modifier](#)

11



.... Protected: Protected access modifier is the a little tricky and you can say is a superset of the default access modifier. Protected members are same as the default members as far as the access in the same package is concerned. The difference is that, the protected members are also accessible to the subclasses of the class in which the member is declared which are outside the package in which the parent class is present.

But these protected members are “accessible outside the package only through inheritance“. i.e you can access a protected member of a class in its subclass present in some other package directly as if the member is present in the subclass itself. But that protected member will not be accessible in the subclass outside the package by using parent class’s reference.

Share Improve this answer

Follow

edited Nov 11, 2017 at 22:23



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Mar 15, 2012 at 11:53



dameng

528 ● 7 ● 13

Just to add this "Once the child gets access to the parent class's protected member, it becomes private (or rather I would say a special private member which can be inherited by the subclasses of the subclass) member of the subclass."

– [Anand](#) Oct 27, 2012 at 18:55



10



David's answer provides the meaning of each access modifier. As for when to use each, I'd suggest making public all classes and the methods of each class that are meant for external use (its API), and everything else private.



Over time you'll develop a sense for when to make some classes package-private and when to declare certain methods protected for use in subclasses.

Share Improve this answer

Follow

edited Nov 11, 2017 at 22:22



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Oct 19, 2008 at 3:18



Dov Wasserman

2,672 ● 17 ● 14



6



Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier*	✓	✓	✗	✗
private	✓	✗	✗	✗



This image will make you understand easily about the basic differences between public, private, protected and default access modifiers. The default modifier takes place automatically when you don't declare any access modifiers in your code.

Share Improve this answer

answered Sep 12, 2020 at 7:33

Follow



Mushfiqur Rahman Abir

808 ● 1 ● 11 ● 18

2 For more clarity, `no modifier == package private | package protected` – [Ahmed Nabil](#) Dec 18, 2020 at 20:56



5



Public Protected Default and private are access modifiers.

They are meant for encapsulation, or hiding and showing contents of the class.



1. Class can be public or default
2. Class members can be public, protected, default or private.

Private is not accessible outside the class Default is accessible only in the package. Protected in package as well as any class which extends it. Public is open for all.

Normally, member variables are defined private, but member methods are public.

Share Improve this answer

answered Jul 30, 2014 at 3:51

Follow



richa_v

119 ● 1 ● 7

Default is not an access modifier, and two of the others are misspelt. – [user207421](#) Sep 24, 2016 at 3:13



Note: *This is just a **supplement** for the accepted answer.*

5

This is related to Java [Access Modifiers](#).



From [Java Access Modifiers](#):



A Java access modifier specifies which classes can access a given class and its fields, constructors and methods. Access modifiers can be specified separately for a class, its constructors, fields and methods. Java access modifiers are also sometimes referred to in daily speech as Java access specifiers, but the correct name is Java access modifiers. Classes, fields, constructors and methods can have one of four different Java access modifiers:

- List item
- private
- default (package)
- protected
- public

From [Controlling Access to Members of a Class](#) tutorials:

Access level modifiers determine whether other classes can use a particular field or invoke a particular method. There are two levels of access control:

- At the top level—public, or package-private (no explicit modifier).
- At the member level—public, private, protected, or package-private (no explicit modifier).

A class may be declared with the modifier public, in which case that class is visible to all classes everywhere. If a class has no modifier (the default, also known as package-private), it is visible only within its own package

The following table shows the access to members permitted by each modifier.

Modifier	Class	Package	Subclass	World
<code>public</code>	Y	Y	Y	Y
<code>protected</code>	Y	Y	Y	N
no modifier	Y	Y	N	N
<code>private</code>	Y	N	N	N

The first data column indicates whether the class itself has access to the member defined by the access level. As you can see, a class always has access to its own members. The second column indicates whether classes in the same package as the class (regardless of their parentage) have access to the member. The third column indicates whether subclasses of the class declared outside this package have access to the member. The fourth column indicates whether all classes have access to the member.

Access levels affect you in two ways. First, when you use classes that come from another source, such as the classes in the Java platform, access levels determine which members of those classes your own classes can use. Second, when you write a class, you need to decide what access level every member variable and every method in your class should have.



1 what exactly is the supplement, and why is it not an edit to the existing post? – [sehe](#) Nov 17, 2016 at 15:39

the supplement is Access Modifiers. Why not an edit? To keep the accepted answer unaltered for historical sake and to give my answer. – [ישו אוהב אותך](#) Nov 18, 2016 at 2:25 ✎



4



Often times I've realized that remembering the basic concepts of any language can be made possible by creating real-world analogies. Here is my analogy for understanding access modifiers in Java:

Let's assume that you're a student at a university and you have a friend who's coming to visit you over the weekend. Suppose there exists a big statue of the university's founder in the middle of the campus.

- When you bring him to the campus, the first thing that you and your friend see is this statue. This means that anyone who walks in the campus can look at the statue without the university's permission. This makes the statue as **PUBLIC**.
- Next, you want to take your friend to your dorm, but for that you need to register him as a visitor. This means that he gets an access pass (which is the same as yours) to get into various buildings on campus. This would make his access card as **PROTECTED**.

- Your friend wants to login to the campus WiFi but doesn't have the any credentials to do so. The only way he can get online is if you share your login with him. (Remember, every student who goes to the university also possesses these login credentials). This would make your login credentials as **NO MODIFIER**.
- Finally, your friend wants to read your progress report for the semester which is posted on the website. However, every student has their own personal login to access this section of the campus website. This would make these credentials as **PRIVATE**.

Hope this helps!

Share Improve this answer

answered Apr 6, 2017 at 4:09

Follow



Greedy Coder

87 ● 2 ● 9

It's remarkable that this nonsense got upvotes. – [Jim Balter](#)

Nov 17, 2023 at 20:06



3

When you are thinking of access modifiers just think of it in this way (applies to both **variables** and **methods**):



`public` --> accessible from every where

`private` --> accessible only within the same class where it is declared





Now the confusion arises when it comes to `default` and `protected`

`default` --> No access modifier keyword is present. This means it is available strictly within the package of the class. **Nowhere** outside that package it can be accessed.

`protected` --> Slightly less stricter than `default` and apart from the same package classes it can be accessed by sub classes outside the *package* it is declared.

Share Improve this answer

edited Jun 27, 2017 at 2:38

Follow

answered Jun 27, 2017 at 2:33



Pritam Banerjee

18.9k ● 10 ● 96 ● 111



It is all about **encapsulation** (or as Joe Phillips stated, *least knowledge*).

2



Start with the most restrictive (private) and see if you need less restrictive modifiers later on.



We all use method and member modifiers like private, public, ... but one thing too few developers do is use packages to **organize** code logically.

For example: You may put sensitive security methods in a 'security' package. Then put a public class which accesses some of the security related code in this

package but keep other security classes **package private**. Thus other developers will only be able to use the publicly available class from outside of this package (unless they change the modifier). This is not a security feature, but will **guide** usage.

```
Outside world -> Package (SecurityEntryClass ---> Pack
```

Another thing is that classes which depend a lot on each other may end up in the same package and could eventually be refactored or merged if the dependency is too strong.

If on the contrary you set everything as **public** it will not be clear what should or should not be accessed, which may lead to writing a lot of javadoc (which does not enforce anything via the compiler...).

Share Improve this answer

edited Jul 31, 2018 at 11:27

Follow

answered Feb 14, 2018 at 9:55



Christophe Roussy

16.9k ● 5 ● 92 ● 85



My two cents :)

2

private:



class -> a top level class cannot be private. inner classes can be private which are accessible from same class.



instance variable -> accessible only in the class. Cannot access outside the class.



package-private:

class -> a top level class can be package-private. It can only be accessible from same package. Not from sub package, not from outside package.

instance variable -> accessible from same package. Not from sub package, not from outside package.

protected:

class -> a top level class cannot be protected.

instance variable -> Only accessible in same package or subpackage. Can only be access outside the package while extending class.

public:

class -> accessible from package/subpackage/another package

instance variable -> accessible from package/subpackage/another package

Here is detailed answer

<https://github.com/junto06/java-4-beginners/blob/master/basics/access-modifier.md>

Share Improve this answer

edited Oct 22, 2019 at 16:25

Follow

answered Oct 16, 2019 at 3:10



Mudassar

1,566 ● 4 ● 21 ● 31



1



1. **public:** Use for wide accessibility across your codebase and in different packages. Suitable for classes and methods you want to expose universally.
2. **protected:** Provides access within the same package and in subclasses, even in different packages. Useful for balancing encapsulation with subclass access.
3. **Default (Package Private):** No explicit modifier; accessible within the same package. Offers encapsulation while allowing access within a package.
4. **private:** Restricts access to within the class. Ideal for hiding implementation details from other classes.

Inheritance:

Classes:

1. **public:** Subclassable by any class.

2. **protected**: Subclassable within the same package and in different packages.
 3. **Default**: Subclassable within the same package.
 4. **private**: Not subclassable.
-

Methods and Fields:

1. **public**: Accessible everywhere.
2. **protected**: Accessible within the same package and in subclasses.
3. **Default**: Accessible within the same package.
4. **private**: Accessible only within the same class.

Subclasses can't have more restrictive access than their superclasses for overridden methods or fields.

Share Improve this answer

answered Aug 24, 2023 at 12:31

Follow



Mian Asad Ali

65 ● 4 ● 13



1



For beginners, considering this example can be helpful;

Consider I have developed `MyClass` in the `foo` package and it has a fantastic method named `print` which you are interested in calling it (it could be a `method` or `property`):

```
package foo; // I am in foo
public class MyClass {
```

```

        private void print() { //This is private
            System.out.println("I can print!");
        }
    }
}

```

You have developed `YourClass` in `bar` package, and you are interested to use `MyClass#print`

```

package bar; \\You are not in same package as me
import foo.MyClass;
public class YourClass {
    void test() {
        MyClass myClass = new MyClass();
        myClass.print();
    }
}

```

Your code is not compile and you get error `The method print() is undefined for the type MyClass`

You come to me:

- You: I want to use your method but it is `private`. Can you make it `public`?
- Me : No I don't want `others` use it
- You: I am your friend at least let me `not others` use it.
- Me : Ok I will remove the `private` keyword. My Access modifier will be `default` or `private package`. As you are my friend you must be in same package as me. So you `must` come to my package `foo`. I mean exact package not even a sub package.

Then `MyClass` will be

```
package foo;
public class MyClass {
    void print() { // No access modifier means default
        System.out.println("I can print!");
    }
}
```

`YourClass` will be:

```
package foo; // You come to my package
public class YourClass {
    void test() {
        MyClass myClass = new MyClass();
        myClass.print();
    }
}
```

Now consider this: Again you come to me

- You: My boss told me I can not change my package (in the actual world, you can not change your package to use other classes methods)
- Me : There is another way, if you `extend` me and I make `print()` `protected`, then you can use it whether you change your package or not. (So subclassing will always give you access to my method).

Here is `MyClass`

```
package foo;
protected class MyClass { // It is now protected
    protected void print() {
        System.out.println("I can print!");
    }
}
```

Here is `YourClass`:

```
package bar; // You are on your own package
import foo.MyClass;

public class YourClass extends MyClass {
    void test() {
        // You initiate yourself! But as you extend me
        YourClass yourClass = new YourClass();
        yourClass.print();
    }
}
```

You may have noticed that by making a method protected *all* other classes can use it by `extending` it, and you can not easily control how it can be used. This was solved in [Java 17](#) by introducing `sealed` and `permits` words. So you can define `which` classes `can` extend you. By something like `public sealed class MyClass permits YourClass` See [What are sealed classes in Java 17?](#) for more information.

Share Improve this answer

Follow

edited Oct 18, 2023 at 21:38



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Jan 16, 2023 at 14:25



It is the scope of a Class's Object variables or methods simply put.

1



Use **Public** if you want your Class variables to be accessible by even other packages/folders!



Use **Protected** if okay you don't want other packages/folders accessing your Class's variables.



Use **No Modifier** if you don't want other packages/folders accessing your Class's variables and you don't want your Class's subclasses to access them, but you still want them accessible in the package. (Also called "Package-Private")

Use **Private** if you don't want other packages/folders accessing your Class's variables and you don't want your Class's subclasses to access them, and you don't want them accessible anywhere else in the package.

Modifier	Class	Package	Subclasses	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

answered Jan 27 at 14:52

[Ahmedakhtar11](#)

1,448 ● 18 ● 8



0



- public

If a class member is declared to be public then it can be accessed from anywhere



- protected

If a class member is declared with keyword protected, it can be accessed from the same class members, outside class members within the same package, and inherited class members. If a class member is protected then it can NOT be accessed from an outside package class unless the outside packaged class is inherited i.e. extends the other package superclass. But a protected class member is always available to the same package classes it does NOT matter whether the same package class is inherited or NOT

- default

In Java default is NOT an access modifier keyword. If a class member is declared without any access modifier keyword, it is considered a default member. The default class member is always available to the

same package class members. But outside package class members can NOT access default class members even if outside classes are subclasses unlike protected members

- private

If a class member is declared with keyword protected then in this case it is available ONLY to the same class members

Share Improve this answer

Follow

edited May 3, 2023 at 17:32



Ahmad Adibzad

657 ● 3 ● 10 ● 17

answered Sep 19, 2019 at 22:34



Vipul Verma

123 ● 1 ● 7

1

2

Next



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.