# In Java, what's the difference between public, default, protected, and private?

Are there clear rules on when to use each of these when making classes and interfaces and dealing with inheritance?

java private public protected access-modifiers

edited Sep 13 '12 at 7:39
Mechanical snail
9,388 2 35 69

asked Oct 18 '08 at 19:53
intrepion
4,656 3 13 16

24 `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. – tennenrishin Feb 13 '13 at 9:56

9 @Tennenrishin — No ; contrary to C++, in Java `protected` makes the method also accessible from the whole package. This stupidity in Java's visiblity model breaks the goal of `protected`. – Nicolas Barbulesco Aug 21 '13 at 9:51

1 @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. – tennenrishin Mar 14 '14 at 10:59

2 @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 '14 at 13:45

3 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 '14 at 13:53

## 16 Answers

This Java tutorial may be of some use to you.

```
Modifier    | Class | Package | Subclass | World
————————————+———————+—————————+——————————+———————
public      |  y    |    y    |    y     |   y
————————————+———————+—————————+——————————+———————
protected   |  y    |    y    |    y     |   n
————————————+———————+—————————+——————————+———————
no modifier |  y    |    y    |    n     |   n
————————————+———————+—————————+——————————+———————
private     |  y    |    n    |    n     |   n

y: accessible
n: not accessible
```

edited Jul 29 '14 at 10:52
emaillenin
7,083 7 30 71

answered Oct 18 '08 at 19:57
David Segonds
28.8k 5 30 58

303 It is probably worth pointing out that in the case of no modifier, whether or not the subclass can see it's superclass's methods/fields depends on the location of the subclass. If the subclass is in another package, then the answer is it can't. If the subclass is in the same package then it CAN access the superclass methods/fields. – Edd Feb 9 '12 at 16:17

3 If you are trying to access a protected method or instance variable in the same class but on an object thats not you as in the case of .equals(Klass var) would it work? – Jordan Medlock Feb 6 '13 at 15:17

2 Default fields are visible in subclasses if the subclasses are at the same package that their parent class is. – Maksim Dmitriev Aug 14 '13 at 8:24

7 In versions of Java and the JDK up to 1.0.1, you could use private and protected together to create yet another form of protection that would restrict access to methods or variables solely to subclasses of a given class. – Vitalii Fedorenko Aug 22 '13 at 1:37

4 FYI, `Subclass` column - subclasses declared outside of the package. – Zhomart Jun 19 '14 at 17:18

(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

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 member.

### 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 worry that someone was using that code you just overhauled.

So, 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 absolutely 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 own 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

Personally, I stick with just "private" and "public". Many OO languages just have that. "Protected" can be handy, but it's really 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.

edited Oct 15 '13 at 19:17 answered Oct 18 '08 at 21:17
Schwern
**36.4k** 7 53 148

6 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. – phresnel Jun 7 '11 at 10:13

10 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 '11 at 12:34

@RhysvanderWaerden I did not know that! I've edited that in. – Schwern Oct 2 '11 at 17:48

2 @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 '13 at 17:36

5 @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 '14 at 20:37

---

Easy rule. Start with declaring everything private. And then progress towards public as the needs arises and design warrant 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 to much dependencies on the actual representation rather than it's observable behavior.

As a general rule I try to avoid overriding method implementations by sub-classing, it's to 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.

edited Feb 6 '09 at 17:18 answered Oct 18 '08 at 20:00

John Nilsson
**5,208** 6 20 33

---

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()); // OK
        Example.leakPrivateClass().secretMethod(); // error
    }
}

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 defined in an inaccessible
class or interface
        Example.leakPrivateClass().secretMethod(); // error
                                  ^
1 error
```

Some related questions:

- Java - Method accessibility inside package-private class?

answered Sep 13 '12 at 7:38
Mechanical snail
9,388 2 35 69

```
                | highest precedence <---------> lowest precedence
*———————————————+———————————————————+———————————+———————————————+———————
 \ xCanBeSeenBy | this              | any class | this subclass | any
  \             | class             | in same   | in another    | class
   \——————————\ | nonsubbed         | package   | package       |
Modifier of x \ |                   |           |               |
————————————————*———————————————————+———————————+———————————————+———————
public          |         ✔         |     ✔     |     ✔         |   ✔
————————————————+———————————————————+———————————+———————————————+———————
protected       |         ✔         |     ✔     |     ✔         |   ✘
————————————————+———————————————————+———————————+———————————————+———————
package-private |                   |           |               |
(no modifier)   |         ✔         |     ✔     |     ✘         |   ✘
————————————————+———————————————————+———————————+———————————————+———————
private         |         ✔         |     ✘     |     ✘         |    ✘
```

edited Feb 13 '13 at 14:02 answered Jan 9 '13 at 21:42
Abdull
2,239 2 22 46

6 Keeping tradition of ASCII art :) – Malachiasz Mar 14 '14 at 15:22

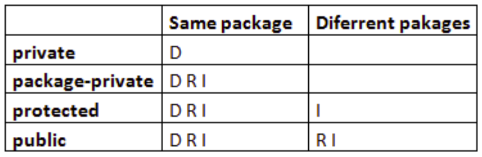
As a rule of thumb:

- **private**: class scope.
- **default** (or *package-private*): package scope
- **protected**: package scope **+ child** (like pakage, but we can subclass it from different pakages). 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)
- **(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 | Diferrent pakages |
|---|---|---|
| **private** | D | |
| **package-private** | D R I | |
| **protected** | D R I | I |
| **public** | D R I | R I |

answered Dec 18 '12 at 18:01
nxhoaf
669 12 21

In very short

- `Public` are accessible from everywhere.
- `Protected` are accessible by the classes of the same package and the subclasses residing in any package.
- `Default` are accessible by the classes of the same package.
- `private` are accessible within the same class only.

edited Aug 19 '13 at 16:19 answered Oct 27 '12 at 17:49
jWeaver
4,538 10 39 75

## Private

- 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/accesscontrol.html http://www.tutorialspoint.com/java/java_access_modifiers.htm

edited May 20 '14 at 4:42

answered Jan 22 '14 at 13:08
Nambi Narayanan
7,149 2 8 28

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 3 classes; `Grandfather`, `Father` and `Son`:

```
package grandfatherpackage;

public class Grandfather
{

}

-------------------------------------------

package fatherpackage;

public class Father extends Grandfather
{

}

-------------------------------------------

package sonpackage;

public class Son extends Father
{

}
```

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

```
package grandfatherpackage;

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

- The method `foo()` can be called in 3 contexts:

  1. Inside a class that is located in the same package where `foo()` is defined, which is `grandfatherpackage`:

```
package grandfatherpackage;

public class SomeClass
{
    public void someMethod() throws Exception
    {
        Father f = new Father();
        f.foo();

        Son s = new Son();
        s.foo();
    }
}
```

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

```
package fatherpackage;

public class Father extends Grandfather
{
    public void fatherMethod()
    {
        this.foo();
    }
}

-------------------------------------------

package sonpackage;
```

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

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

```
package fatherpackage;

public class Father extends Grandfather
{
    public void fatherMethod()
    {
        Father f = new Father();
        f.foo();
    }
}

-------------------------------------------

package sonpackage;

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

- Regarding the third situation, it won't compile if the reference is of type of the parent class:

```
package fatherpackage;

public class Father extends Grandfather
{
    public void fatherMethod()
    {
        Grandfather f = new Grandfather();
        g.foo(); // compilation error

        Grandfather g = new Father();
        g.foo(); // compilation error
    }
}

-------------------------------------------

package sonpackage;

public class Son extends Father
{
    public void sonMethod()
    {
        Grandfather g = new Grandfather();
        g.foo(); // compilation error

        Grandfather s = new Son();
        s.foo(); // compilation error

        Father f = new Father();
        f.foo(); // compilation error

        Father ff = new Son();
        ff.foo(); // compilation error
    }
}
```

answered Nov 15 '13 at 20:06
Eng.Fouad
**48.6k** 28 127 217

`Object#clone()` is an example of a `protected` member. – Eng.Fouad Nov 15 '13 at 20:08

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.

answered Oct 18 '08 at 20:00
Joe Philllips
**15.4k** 14 54 105

**Private** : Limited Access to Class only

**Default(No Modifier)** : Limited Access to Class and Package

**Protected**: Limited Access to Class,Pacakge and Subclasses(Inside and Outside Package both)

**Public**: Accessible to Class,Package(All),Subclasses...In short everywhere

answered Jun 18 '14 at 7:13
samkit shah
**317** 2 12

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 (it's API), and everything else private. You'll develop over time a sense for when to make some classes package-private and when to declare certain methods protected for use in subclasses.

answered Oct 19 '08 at 3:18
Dov Wasserman
**1,736** 9 12

this page writes well about the protected & default access modifier

.... 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. ....

answered Mar 15 '12 at 11:53
dameng
**78** 1 6

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 '12 at 18:55

Here is a complete explanation on how protected and default (package) access work in Java:

Access to protected and default (package) members in Java

Package access is specially tricky, and you might want to pay special attention to the details.

Hope it helps.

answered Aug 3 '12 at 0:49
svpino
**267** 1 10

Access Modifiers are there to restrict access at several level.

**Public :** it is basically as simple as you can access from any class either that is in same package or not.

To access if you are in same package you can access directly but if you are in other package then you can create object of class.

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

to access you can create object of 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.

answered Dec 17 '14 at 6:29
Prashant
**384** 2 9

1 Good short answer ! Clear and Concise – Naga R Jan 4 at 7:00

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.

answered Jul 30 '14 at 3:51
richa_v
**21** 4

**protected** by Mysticial Mar 28 '13 at 2:18

Thank you for your interest in this question. Because it has attracted low-quality answers, posting an answer now requires 10 reputation on this site.

Would you like to answer one of these unanswered questions instead?