Rules for Class Member Access Levels

Source	http://www.artima.com/objectsandjava/webuscript/PackagesAccess1.html

Rules of Thumb for Class Member Access Levels

The most important rule of thumb concerning the use of access control modifiers is to keep data private unless you have a good reason not to. Keeping data private is the best way to maximize the robustness and ease of modification of your classes. If you keep data private, other classes can access a class's fields only through its methods. This enables the designer of a class to keep control over the manner in which the class's fields are manipulated. If fields are not private, other classes can change the fields directly, possibly in unpredictable and improper ways. Keeping data private also enables a class designer to more easily change the algorithms and data structures used by a class. Given that other classes can only manipulate a class's private fields indirectly, through the class's methods, other classes will depend only upon the external interface to the private fields provided by the methods. You can change the private fields of a class and modify the code of the methods that manipulate those fields. As long as you don't alter the signature and return type of the methods, the other classes that depended on the previous version of the class will still link properly. Making fields private is the fundamental technique for hiding the implementation of Java classes.

As mentioned in an earlier chapter, one other reason to make data private is because you synchronize access to data by multiple threads through methods. This justification for keeping data private will be discussed in Chapter 17.

As a general rule, the only good non-private field is a final one. Given that final fields cannot be changed after they are initialized, non-private final fields do not run the risk of improper manipulation by other classes. Other classes can use the field, but not change it.

A common use of non-private final fields is to define names to represent a set of valid values that may be passed to (or returned from) a method. As mentioned in Chapter 5, such fields are called constants and are declared static as well as non-private and final. A Java programmer will create constants in this manner in situations where a C++ programmer would have used an enumerated type or declared a "const" member variable.

Rules of thumb such as the ones outlined above are called rules of thumb for a reason: They are not absolute laws. Java allows you to declare fields in classes with any kind of access level, and you may very well encounter situations in which declaring a field private is too restrictive. One potential justification for non-private fields is simple trust. In some situations you may have absolute trust of certain other classes. For example, perhaps you are designing a small set of types that must work together closely to solve

a particular problem. It may make sense to put all of these types in their own package, and allow them direct access to some of each other's fields. Although this would create interdependencies between the internal implementations of the classes, you may deem the level of interdependency to be acceptable. If later you change the internal implementation of one of the classes, you'll have to update the other classes that relied on the original implementation. As long as you don't grant access to the fields to classes outside the package, any repercussions of the implementation change will remain inside the package.

Nevertheless, the general rule of thumb in designing packages is to treat the types that share the same package with as much suspicion as types from different packages. If you don't trust classes from other packages to directly manipulate your class's fields, neither should you let classes from the same package directly manipulate them. Keep in mind that you usually can't prevent another programmer from adding new classes to your package, even if you only deliver class files to that programmer. If you leave all your fields with package access, a programmer using your package can easily gain access to those fields by creating a class and declaring it as a member of your package. Therefore, it is best to keep data private, except sometimes when the data is final, so that irrespective of what package classes are defined in, all classes must go through methods to manipulate each other's fields.

The methods you define in public classes should have whatever level of access control matches their role in your program. You should exploit the full range of access levels provided by Java on the methods of your public classes, assigning to each method the most restrictive access level it can reasonably have.

You can use the same rule of thumb to design classes that have package access. You must keep in mind, however, that for package-access classes, fields and methods declared public won't be accessible outside the package. Fields and methods declared protected won't be accessible to subclasses in other packages, because there won't be any subclasses in other packages. Only classes within the same package will be able to subclass the package-access class. Still, you should probably keep the same mindset when designing package-access classes as you do when designing public classes, because at some later time you may turn a package-access class into a public class.

~~~ End of Article ~~~