[0550] Access Modifiers - the Keys to the Castle

As I mentioned on the last page, you should think of Encapsulation as a medieval castle - the member variables are the "gold" in the castle's more secure area (its "Keep") and the Getters and Setters are the guards at each of the castle's gates that you have to negotiate with in order to get what you need.

Given that metaphor, C# access modifiers are the castle's walls and moat. They control how much unauthorized access (if any) is allowed.

C# has five access modifiers. In reality, you usually only care about two, but here are all five of them (along with their medieval equivalents):

- **public** - public is the "no castle" approach to data security. i.e., there is none and you deserve to be robbed for failing to protect the important stuff. Your castle guards - the Getters and Setters - need to be public of course, but not much else!
- **private** - these are the strong walls that keep thieves out. No code outside of your class can access private things. PRIVATE SHOULD BE YOUR DEFAULT ACCESS LEVEL. ONLY MAKE SOMETHING NON-PRIVATE WHEN THERE IS A VERY GOOD REASON TO!
- **protected** - Protected is misnamed. Technically, protected items can only be used by the class that defines it and any of its subclasses. The GOTCHA here is that, unless you are careful, anything can subclass your class and thus gain access to your "protected" member variables. That would be like someone gaining access to our castle's keep simply because they built a bigger castle that enclosed our castle. Favor private over protected.
- **internal** - internal items can be accessed by anything in the same assembly as your class. That's even less protected than protected! It would be like someone gaining access to our castle's keep simply because they built a castle in the same country as ours.
- **protected internal** - Completely nuts and extremely rarely used.

In general, ***always use private access unless there is a good reason not to.***

For completeness sake, in addition to class members, public and internal can also be applied to classes while the other ones cannot.

Default Access Modifiers

When you don't specify access modifiers explicitly, C# usually does "the right thing." A class without an access modifier will become an "internal" class. Anything inside a class that doesn't explicitly include an access modifier will be "private." That said, don't rely on default behaviors! Always add "private" to the front of private things just to be sure.