**ACCESS MODIFIERS**

* *private* these fields are completely hidden and “encapsulated” within the single class hierarchy in which they are declared. This completely isolates the field to a very specific area of the program. If a field is only needed within a single class hierarchy, then it is best to use this access modifier. However, if child classes do need access to these fields, then it is best to use the following access modifier (see below).
* *protected* allows all child classes to “see” and use the direct field of its parent class. This makes sense because, often times, like in a family, child classes need direct access to resources from their parents. This direct access provides a simpler way to access the needed fields. The reason why child classes may need to access these fields is because of their relationship to the parents – they are part of one class hierarchy because they share a similar “purpose” or “goal,” making it necessary for them to inherit certain resources from other classes.
* *public* just as the word suggests, this access modifier makes the field visible and usable by all. This does not enforce encapsulation, which goes against OOP. The purpose of encapsulation is to simplify the user interface/experience by providing only the necessary details. It stays with the principle of KISS – Keep it simple, stupid. Simplicity allows for a more efficient way to do things. In other words, public should not be used on fields (since they are the inner details of a class). They are best used for methods, which need to be accessed by outer parts of the program in order to utilize the fields and other inner workings of a class hierarchy.
* *Abstract/override* this keyword set allows for abstract methods to be declared in the base class and forces child classes to create the implementation for it. This is a form of polymorphism. It is useful when a class hierarchy knows that all child classes share common methods but each class has its own version of it. By forcing override in each class, you are ensuring that the common methods are implemented in all child classes. The abstract base class keeps track of all these methods by only declaring them (not implementing them).
* *Properties* properties provide an alternative way to code fields. The standard way of coding properties uses “get” and “set” followed by implementation that allows access to the fields created. The auto property auto-generates fields – it’s the simplest way to code. Access modifiers do still stand true to their definitions. If you want access to a field that is protected or private, you must use a public method.

**CONSTRUCTOR CALLING SEQUENCE**

* When creating a new base class object, the default and/or master constructors are invoked.
* When creating a new child class object, and if the base class constructor is linked to the child class constructor (using “base” keyword), then the base constructor is invoked first, followed by the child class constructor.
* Whether the default or master constructor is called depends on your parameters.