Sam Pesce

IS 217

Design Patterns

1. A design pattern is a standard way of solving common problems that a programmer will encounter. The problems covered are so common that design patterns are used in all programs; they can be thought of as paradigms for successful programming. Design patterns also make it easier for developers to communicate, in that it provides a standard that all can follow and universal terminology. Design patterns were developed over many years and are the accumulation of other people’s work; we are able to build on proven methods of programming architecture rather than having to develop them from scratch.
3. The facade pattern: This pattern is basically a way of expressing abstraction. It means that the user who is accessing a given method or class can do many things that are encapsulated into one entity, but the user isn’t concerned with how the tasks are performed. A facade is created which hides a much more complex underlying framework; all of that code was written by someone else, but is accessible via something simpler and more intuitive for the purpose of usability by anyone. This pattern is important because it falls under “not wanting to recreate the wheel;” Why waste time writing what someone else has already created?
4. The factory pattern: This pattern provides a way of creating objects with methods and attributes from something that is passed into the factory method/class, and is usually something simple that gets built on. The factory pattern is a good way of creating few to many similar objects, that can differ based on what is passed into the factory. This pattern is important because it’s useful to encapsulate data and functionality into one object, rather than having the code scattered. A factory also represents a reliable means of creating objects that can be used over and over again, so you don’t have to rewrite code.
5. The observer pattern: The observer pattern observers or subscribes to keep in touch with a certain object or subject that can broadcast notifications to all observers. The observers are dependent on the subject in some way. Observers can subscribe and unsubscribe to a subject, and the subject would then remove that observer from its list. This pattern has a lot of practical uses, and many times an object will need to be aware of when a specific event happened to react accordingly, such as with event handlers/event bindings.

3.) The two patterns that I chose to make work together were the factory and facade patterns. This seemed like a logical choice to me because often times in a factory, the operations for creating the object are already abstracted, so I thought that I could add more complexity to the factory by accessing exterior methods and classes without the user knowing. In my example, I made a “div creator” which is a class that creates divs with custom text and size supplied via a two dimensional array. The divs themselves are created through a loop which iterates as many times as the array is long. Within the loop though, there is a separate style class instantiated unknown to the user, and this can be used to change style aspects on the page. There is also a content modifying function that is called unknown to the user and it changes the size of the text supplied based on a value from the original array. There are also methods in the factory class which can access methods in the style class, for a greater feel of encapsulation; this was demonstrated by changing the color of a div to blue using the divCreator class rather than the style class.