**Remarks**

While I have not yet turned in a project which can run due to the *NotImplementedExceptions* littered about my code, I have a good reason for it. It took a little over a week and a half to successfully research and implement practice programs (which I can submit if you wish) that tackled different aspects of WPF development. I have no regrets concerning the time I invested in these skills as I think they will benefit this program’s (and future ones) development in the long run. My only regret is not having the time to turn in a more finished product. Anyway, onto the rubric.

**Rubric**

In my program you’ll notice that I have most of the bare bones implemented. From there, it won’t take too long to get a functional program up and running (honestly a day or two of dedicated work would be enough). In the **Model** folder I have well over 4 class objects. In most of them, I only need to implement the getters and setters in each property (given the custom functionality of each getter/setter I’ll actually need more fields than I originally thought, an oversight on my part).

Complexity definitely exists, even in this bare-bones stage of my application. This is most evident in my **ViewModel/MainViewModel.cs** & **ViewModel/ViewModelBase.cs** classes (I have commented within these classes to demonstrate my understanding of them).

I have invited you to my GitHub repo, so Version Control can be checked off the list.

I have submitted my diagram file; in there you will see how each part of the MVVM interact with one another. While, the diagram is mostly done, it does not fully represent the changes that need to be made once the code is generated. I tried to keep each diagram as clean as possible, unfortunately the **ModelStructure** became a bit messy after I split my previously proposed **CharacterComponent** class into separate interfaces (after some thought it made sense to adhere to some SOLID principles, that being that my interfaces have focused goals).

Many of the classes within Model inherit from a few classes. Even **MainViewModel.cs** inherits from **ViewModelBase.cs**. In order to keep my workspace clean, I moved files into different folders (that being **Model**, **View**, and **ViewModel**). Adhering to this organization I also created “sub-namespaces” encapsulating classes within appropriate parts of the MVVM Model. In Model the classes **XPCumulative.cs** & **XPNonCumulative.cs** contain virtual methods. While these don’t technically utilize the abstract keyword, they fit my purposes better and virtualization behaves similar enough to abstraction where I think it might count.

In terms of advanced topics I think my application checks most of the boxes. As noted previously, I have organized my classes into folders, I have implemented interfaces (and used multiple inheritance as well) and I have used SOLID principles (at least regarding my interfaces). In terms of things that weren’t on the list I liked to give a few honorable mentions to the skills I learned and implemented (or will be implementing) in this project: **Data Contracts** (you can see these in the classes in my Model folder, they will be important for serialization later), **Data Binding** (check my **MainWindow** XAML) and **Data Templates** (also in **MainWindow**, this will generate custom controls for each Character instance once I actually replace the *NotImplementedExceptions* in my **Model** classes).