Aaron Islas

CSE210

Explaining Programming with Classes

Starting with the first principle of Abstraction in programming, to me, it means that we can break down the parts of the entire project/program we’re looking to do, and by sectioning them into categories we can decide which parts of the program need to be hidden or shown to the end user. That way the end user also doesn’t have to fully know how the system works, only that it does and it will function as needed when used. One example of using this was in our week 2’s Journal Program where we created separate classes that would generate a prompt and which wasn’t supplied to the user through the menu, instead the user simply had to choose to get the prompt and then it would return one if chosen so they could begin writing. The user didn’t see or have access to the prompt generator, only the feature of showing that prompt. That also let us secure the generator so if we wanted to make any adjustments in the future to it alone, we would not have to touch the main menu program that the user interfaces with, instead we could focus only on the class that contained the generator itself.

With the second principle of programming called Encapsulation, I define it as the way we can keep the variables and methods of a class hidden from other parts of the program which allows us to ensure that we can keep each class contained and unable to be overwritten or accessed by another part of the program. It also allows us to better organize each class making sure that it only has what it needs to provide to the main program with no other filler. One place I used this was through week 3’s project of the Scripture Memorizer. Inside my class called “ScriptureCollection.cs” I had to make sure that the class itself was public so that the main program could find it, but then I made sure to private the scripture list that the main program could call only to generate the scripture when needed. It would not appear otherwise. That allowed me to keep the list itself safe, and if I wanted to, I could then edit the list and it would not affect the main program itself. In essence, I could change the ingredients without them being shown outside of the container.

The third principle of programming with classes I learned was Inheritance. This idea is simply taking a parent class which will carry any variables and functions/methods that are common across our planned child classes which can taken and iterate those similar to save time and not to have to enter them into each of the child classes directly. One place I was able to show that was using our week 5’s Mindfulness activity. The main class I used was called “Activity” which contained the activity names, the time spinner which was used to show time elapsing and the end completion notifications. Those would be then taken and returned by all 3 child activities so that I only had to edit and change it in the main parent class “Activity” this saved me time so that I didn’t have to put the same functions/methods into all the other classes since they could all use the same information found in the parent class.

The fourth principle of programming with classes was Polymorphism. Similar to the inheritance idea, it allows us to use an abstract method which we can leave blank, and this way if a child class needs to use a function with the same information as another which is found in the parent class it could do so in a different context. This is done so that the child class doesn’t have to completely make a separate function/method instead it can take and change the function/method to do what it needs in a specific context. One place I used this was for our week 6’s program Eternal Quest. On the options to earn points depending on the user input there is multiple ways to earn them. They can choose to either do a scripture report activity, or write their own. The call I created through the menu allows them to re-use the information and the GoalTracker.cs class will ensure that whichever activity they choose, their progress and points will be saved. That allows us to give multiple uses to a single function, but depending on user input it can change the context and what it does.