Strategy Design Pattern

The read function in the filer will need different versions for each file format. The strategy design pattern will work well, as it will encapsulate the read method and make it interchangeable. The different file types often require different algorithms to read them depending on how they are formatted and saved. When the Strategy Design pattern is implemented into my solution, I will implement sub-classes of a Plain TXT filer and a CSV file reader.

Builder Design Pattern

The reasoning behind implementing the builder design pattern is to make it easier to expand the application, at the moment there is a specific amount of attributes that the person will have. So creating a builder class then sub-classing into a person builder will be used to create “persons” that is then used to match the data.

Evaluation report

* The design pattern helps encapsulate the algorithms and make it easier for the client to use. It will reduce the complexity for the client as they will not need to read or look at the algorithm to figure out which one is used for a specific file type.
* By implementing this design pattern, it allows for more functionality to be easily added later on, if more file formats need or want to be added a new sub class of filer can be added without the need to change the filer class or the other sub-classes. This will significantly reduce the time needed to expand the application and the amount of code that will need to be re-written in the application. Where the application is now all that will be required to add a file type would be a slight two/three-line modification in the controller class and then a new sub-class of filer.
* By using this design pattern, it also reduces the amount of duplicate code within the application, the reason for this is that there will be certain parts of the read method that will stay the same throughout each file type. These parts can be separated from the method itself and therefore not needed to be repeated every time the method changes. This will also be able to give the application the chance to re-use the classes that have the same method of reading files and therefore reduce the amount of classes and the amount of repeated code if the method of reading the file is the same.
* When using this pattern, it reduces multiple bad smells. Single Responsibility, each sub-class of the filer will be used for its specific purpose and its specific file type. Large Class, when sub-classing the filer it will then reduce the amount of code in the Filer class. As well as the bad smell that would occur if the read method had various if statements to account for the file types. Obviously, that wasn’t happening in the current version but it will help eliminate a bad smell further down the development track.
* This also includes making it easier for future designers to know which class is doing what and if need the methods need to be changed (if a file format changes how it saves in the future). Alongside expanding the application if any file type is being read or interpreted incorrectly a future designer will know exactly which class to look at to find the source of the problem, therefore, reducing the amount of time spent debugging the program if errors are encountered within the filer.