Software Engineering

Assignment 1 – Matthew Henderson 2031944H

Design 1 – Reduced Coupling

The system in figure 1 seeks to absolutely reduce coupling. Content coupling has been reduced with encapsulation through the use of private variables, and getter and setter methods. Common coupling has been avoided by the omission of public static (global) variables.

By using basic data types as method arguments, stamp coupling has been reduced as much as possible, with only one method taking an object as an argument. Although there is some data coupling, this is unavoidable. In general, data coupling has been kept to minimum and is only used where absolutely necessary.

To minimise routine coupling, changeSize, changeFont and changeColour have been combined into an encapsulated method named updateText. This means that to change the code, it need only be modified in the encapsulated method. Type use coupling has been reduced by changing the templates and fonts data types in UserInterface from an ArrayList to a List, allowing more scope for future change. The system does not import any external libraries, therefore does not exhibit import coupling. However, since the files are saved to the hard drive, external coupling is unavoidable.

Design 2 – Increased Cohesion

Figure 2 shows a design that maximises cohesion. The overall functional cohesion of the system was increased by separating classes out so that they only perform related function, for example the EventListener class only responds to user events. This is especially apparent for the text elements which were were split into Colour, TextSize and Font classes. This is functionally much more cohesive than design 1 which has all these elements stored in the UserInterface class.

Communication cohesion has been achieved through a FileHandler class which handles all interaction with the hard drive. Communication cohesion gives the benefit of finding this related code in the same place. Furthermore, if a user wishes to save a file, it must first be created. Both the createFile and saveFile methods are found in the FileHandler class, therefore the FileHandler also exhibits procedural cohesion.

This design exhibits layer cohesion. This means that none of the lower level classes can access classes in the higher level of the architecture. In this example there is no need for utility, sequential or temporal cohesion.

Design 3 – A balanced Design

Finally, figure 3 provides a balanced solution to the problem. For this solution the model-view-controller architecture has been used. This is much less coupled than the EventListener class in design 1 and much more cohesive than not having any sort of controller like design 2. Therefore, this design pattern offers an ideal trade-off. The model class eliminates the need for content coupling displayed in design 2, whilst still allowing the system to be more functionally cohesive than design 1.

The FileHandler class from design 2 was kept in this model. This added communication cohesion to the model without greatly increasing coupling. However, the File class in design 2 was removed as this was deemed to add too much coupling for only a small gain in functional cohesion.

The text elements have been split into Text and Font classes, which is a compromise between design 1 and design 2. It was thought that this split would increase the flexibility of the system. The Model class and Template class are highly coupled, however this was deemed acceptable for the overall cohesion of the system.

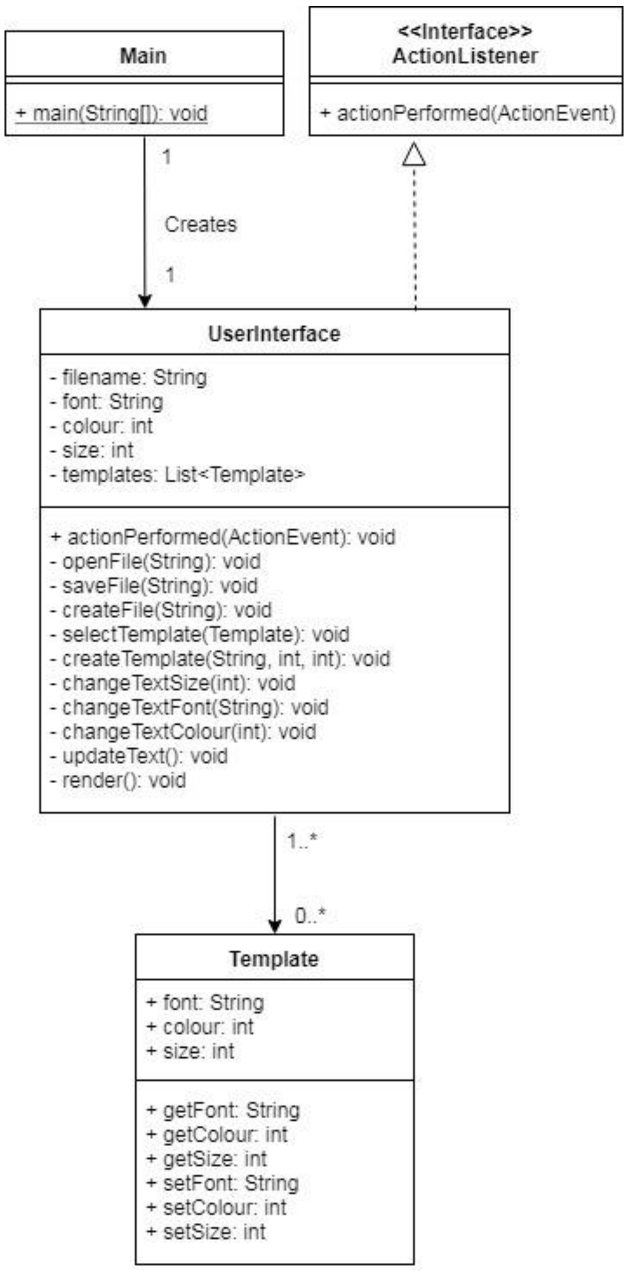

Figure 1. Reduced Coupling

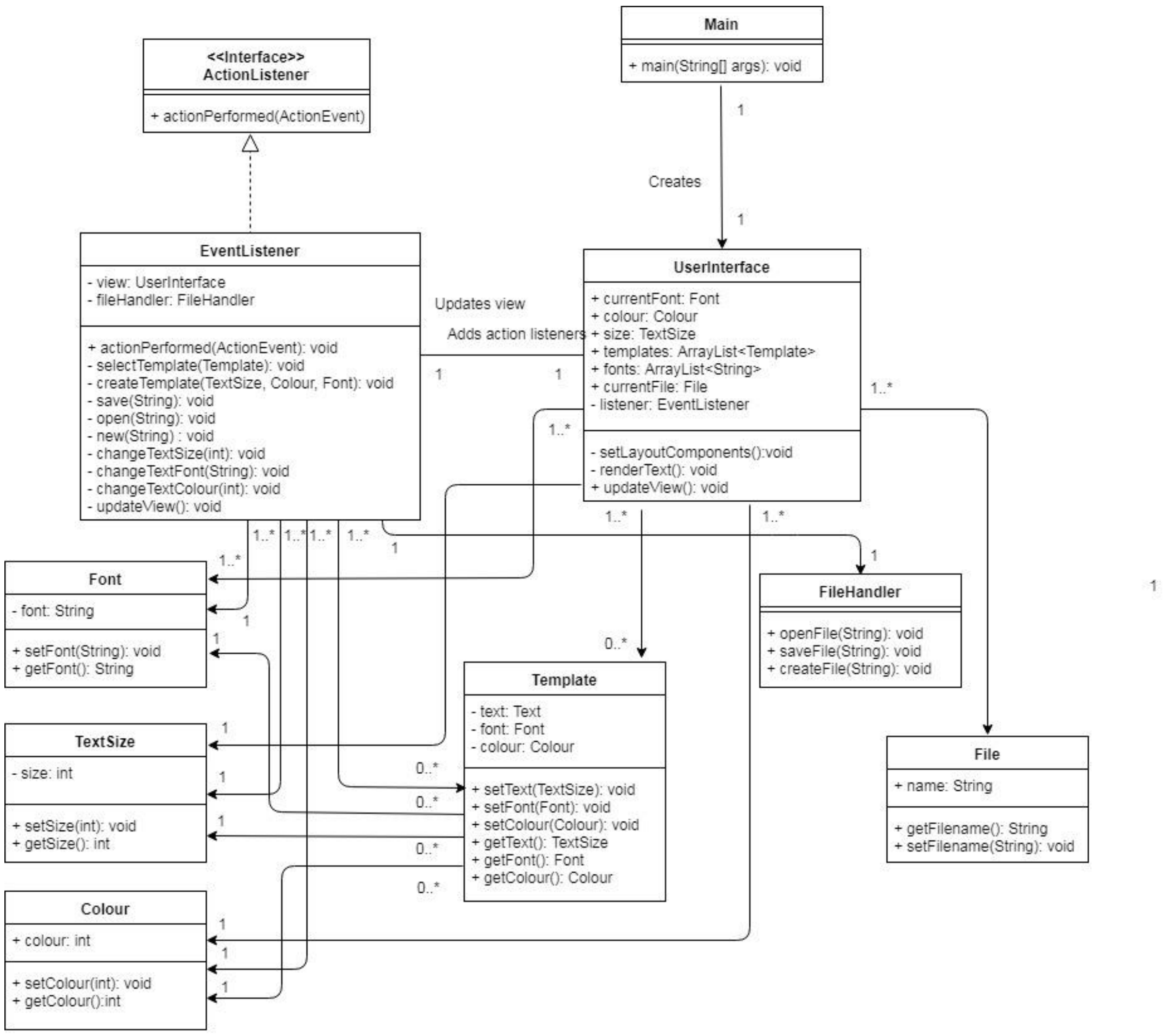

Figure 2. Increased Cohesion

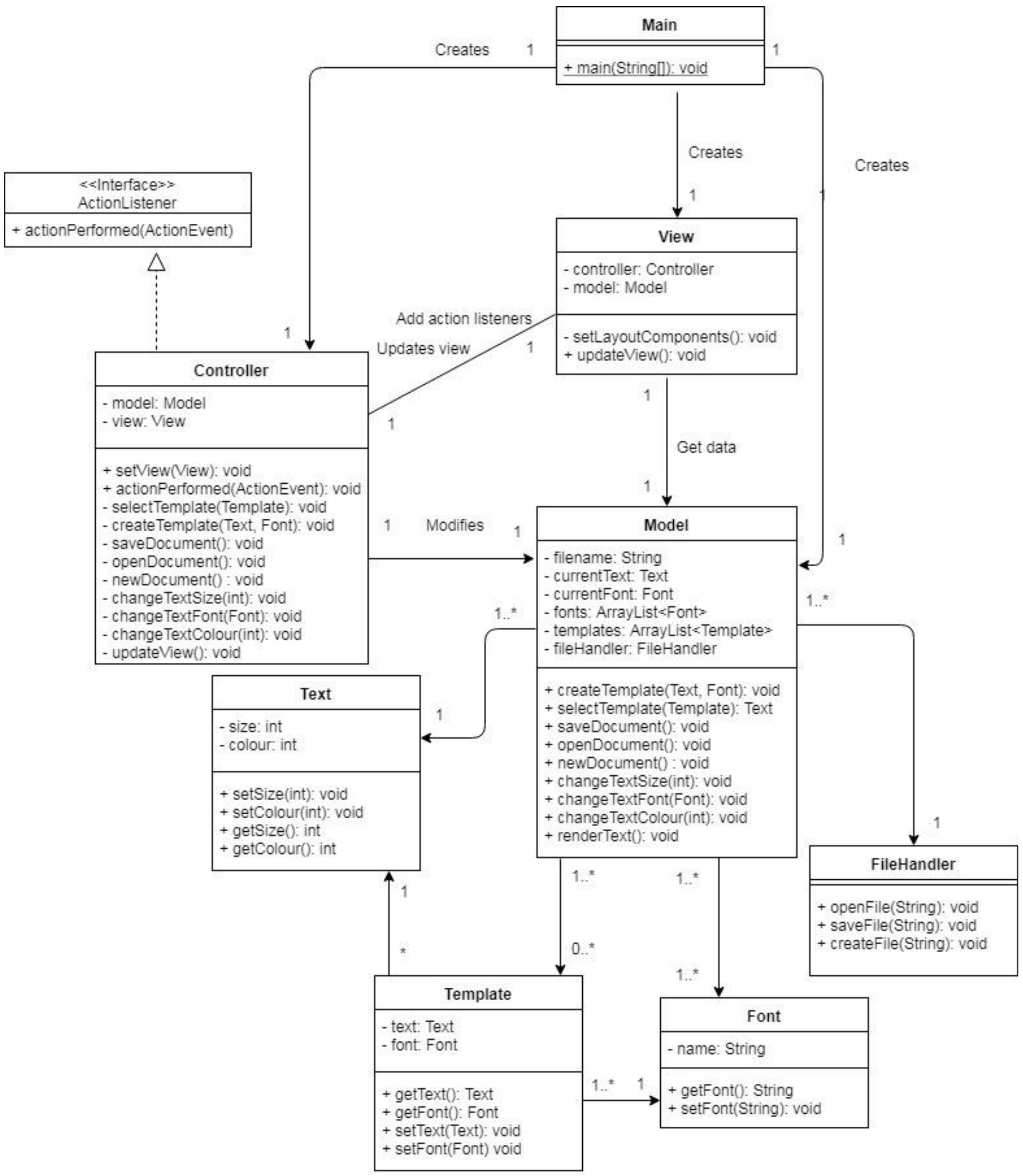

Figure 3. Balanced Design