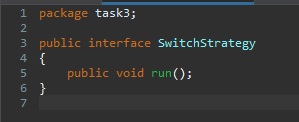
**Strategy pattern**

The strategy pattern allows a classes behaviour or algorithm to be changed at runtime, this type of pattern comes under being a behaviour pattern.

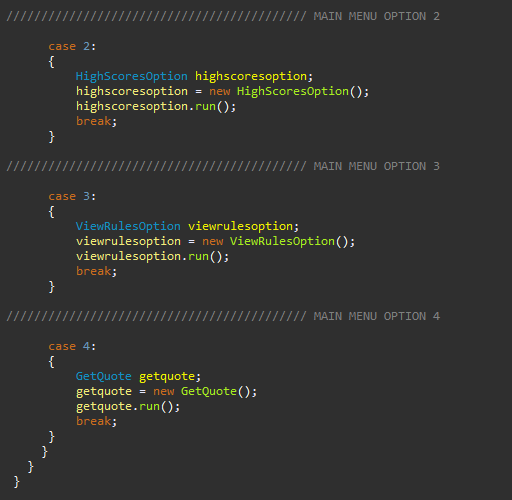
This pattern provides users a way of changing the behaviour of a class without extending it However as shown in figure 3 the class was still extended to capture needed variables.

Using the concepts of the strategy pattern, code was implemented to break up the large switch statements within the Main.java class and have the Main.java class instead reference separate classes under the control of a public interface called “SwitchStrategy”.



**Figure 1:** SwitchStrategy interface.

Having the interface with only a single method called “run()”, this allowed me to dissect and move code within the Main.java run() method and extract them to separate classes. This helped in allowing me to start to be able to tidy up some bad smells within the code (Large class/method/Large Switch statements) as well as make the code relating to the core functionality of the Abominodo game be easily accessed viewable by future developers looking to add functionality within the game.



**Figure 2:** Main.java main menu method, showing calls to methods now in new classes.



**Figure 3:** new HighScoreOption class with code moved from main.java to here making it more accessible and easier to identify.

Using this pattern meant that multiple algorithms could be defined (like the one shown in figure 3). And then passed as a parameter for the code to run off (shown in figure 2) all under the name of the run() function shown in figure 1.