**Kim Mir – 1824409**

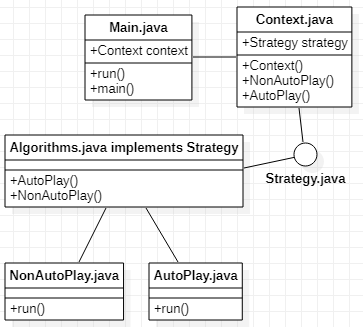
**Strategy Pattern**

The strategy design pattern is a behavioural design pattern that lets you to define different algorithms of the same type by putting them into separate classes.

In the context of the task, main.java needs to have its role split into different classes instead of containing both types of algorithms. Those classes would be main.java, Context.java, Algorithms.java, Strategy.java, NonAutoPlay.java, and AutoPlay.java.

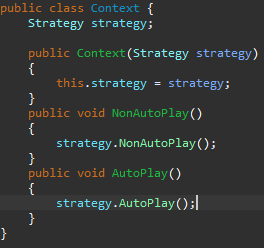
The Context class will be the field in main.java that refers to the algorithm that needs to be executed. The Context class’s parameter is the Algorithms.java class. Context.java uses the Strategy.java class to execute the required algorithm. It does this by having functions named NonAutoPlay() and AutoPlay(). These functions use Strategy.java. Strategy.java, an interface has functions that link to Algorithms.java as it is linked to Context.java as well as Algorithms.java implements Strategy.java. Algorithms.java can create AutoPlay.java or NonAutoPlay.java classes and execute the algorithm requested.

Here is a UML example.

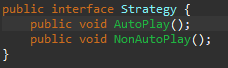




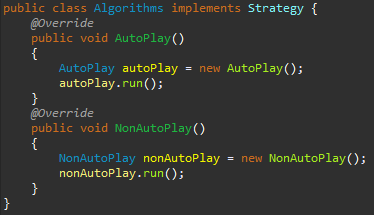
Above is an image of main.java. As you can see at the top, A context field with a parameter of Algorithms() is there. This starts the process as it is used after the user decides which type of algorithm to execute.



Above is the Context.java class. This is what is executed from Main.java. The strategy interface is declared and used for this.



Above is the Strategy interface. This is implemented into Algorithms, the parameter for the Context class.



The AutoPlay() and NonAutoPlay() functions are what start the AutoPlay.java and NonAutoPlay.java functions. This is the final step as the classes named start the game.