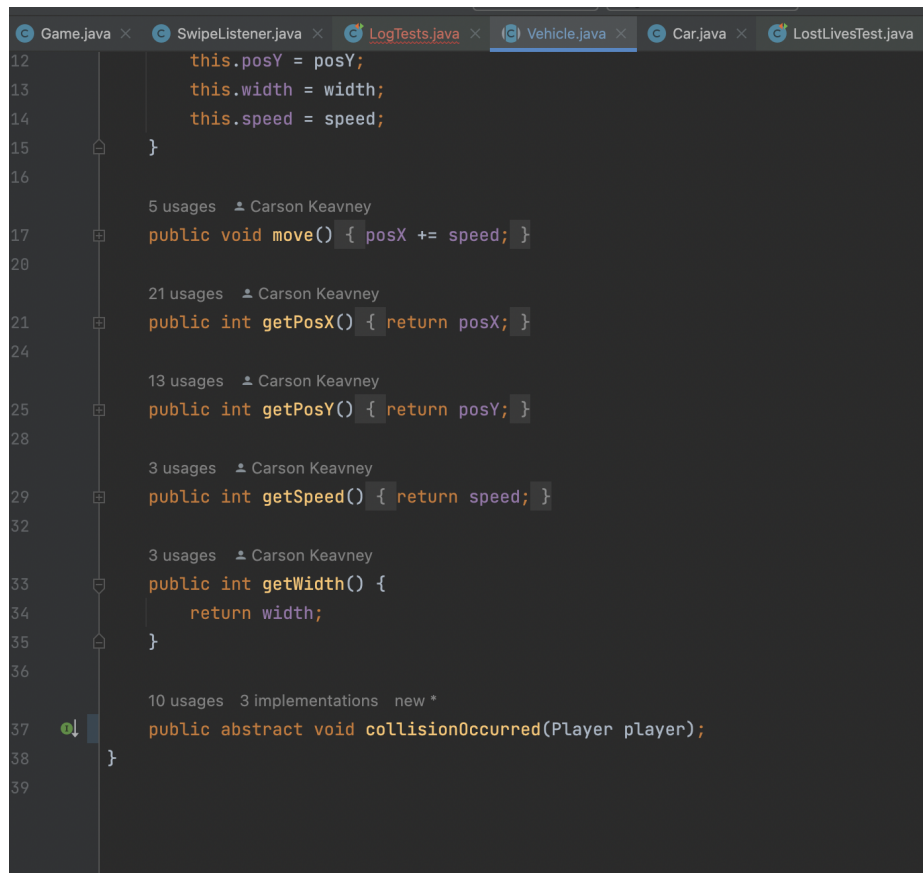


## Implementing Behavioral Design Pattern: Template Method

The Template Method is a behavior design pattern where the code base implements a unique abstract class to lay the scaffolding for an algorithm in a base class and allows its subclasses to alter specific algorithm steps without changing its overall structure. For the sake of our implementation, we chose to implement an abstract Vehicle class with abstract methods that would be overridden by the needs of its respective subclasses which happened to be our game-playing vehicles which were Car, Racecar, and Truck.



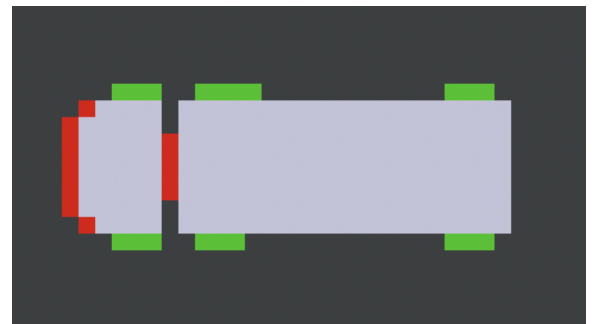
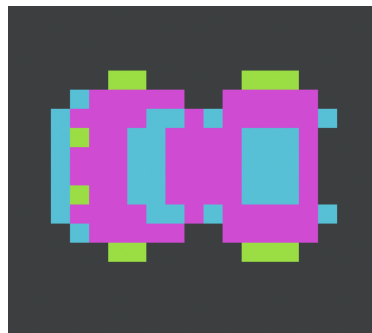
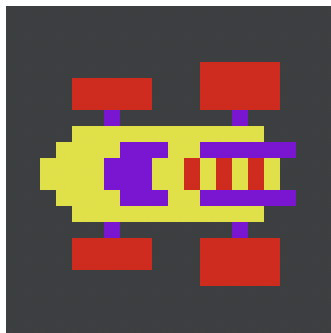
```
Game.java x SwipeListener.java x LogTests.java x Vehicle.java x Car.java x LostLivesTest.java x
12         this.posY = posY;
13         this.width = width;
14         this.speed = speed;
15     }
16
17     5 usages Carson Keavney
18     public void move() { posX += speed; }
19
20
21     21 usages Carson Keavney
22     public int getPosX() { return posX; }
23
24
25     13 usages Carson Keavney
26     public int getPosY() { return posY; }
27
28
29     3 usages Carson Keavney
30     public int getSpeed() { return speed; }
31
32
33     3 usages Carson Keavney
34     public int getWidth() {
35         return width;
36     }
37
38     10 usages 3 implementations new *
39     public abstract void collisionOccurred(Player player);
40 }
```

```

1  package edu.frogger.game;
2
3  public class Car extends Vehicle {
4      public Car(int posX, int posY) { super(posX, posY, width: 1, speed: -10); }
5
6      @Override
7      public void collisionOccurred(Player player) {
8          player.setLives(player.getLives() - 1);
9          player.resetPos();
10     }
11 }
12
13

```

The code screenshots above demonstrate this concept where we start with instantiating the abstract class and methods in our Vehicle class which provides a framework for attributes and methods needed by all the vehicle subclasses. For example, the abstract collisionOccured() method is implemented unique to the game conditions set for the Car class. That same abstract method is implemented uniquely again in Racecar and Truck classes as well, showcasing the useability of the template method and why this kind of behavior is beneficial to us when we have classes responsible for similar game attributes. The concrete subclasses inherit and implement what is already laid out in the abstract framework class and make it more customizable without hindering the effectiveness or efficiency of the program.



Whether a Car, Racecar, or Truck, the Vehicle template method design pattern was crucial to our system's code reusability, flexibility, and simplicity which are ultimately the overall purpose for using this behavioral strategy in the first place. We attribute our successes with the flow of our game system to this foundational and valuable design tool and recognize the significance of such strategy when our game has as many concurrent moving pieces as seen below.

