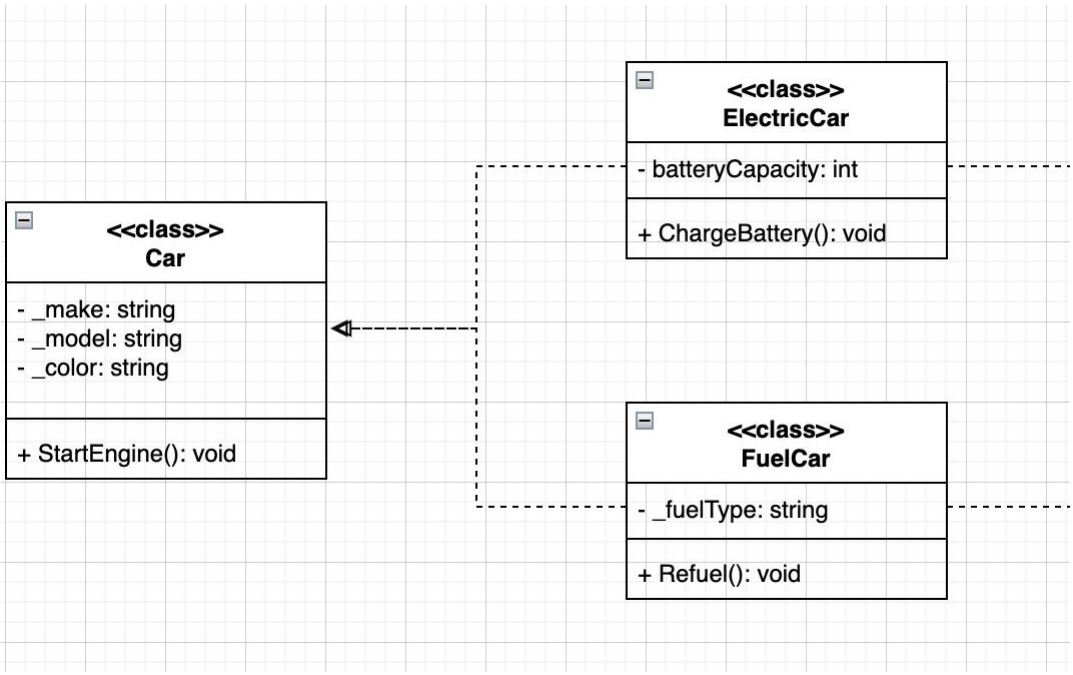
In this task, I will use class diagram to demostrate my understanding of the core concepts: Inheritance, Polymorphism, Delegates and Lambdas.

Inheritance

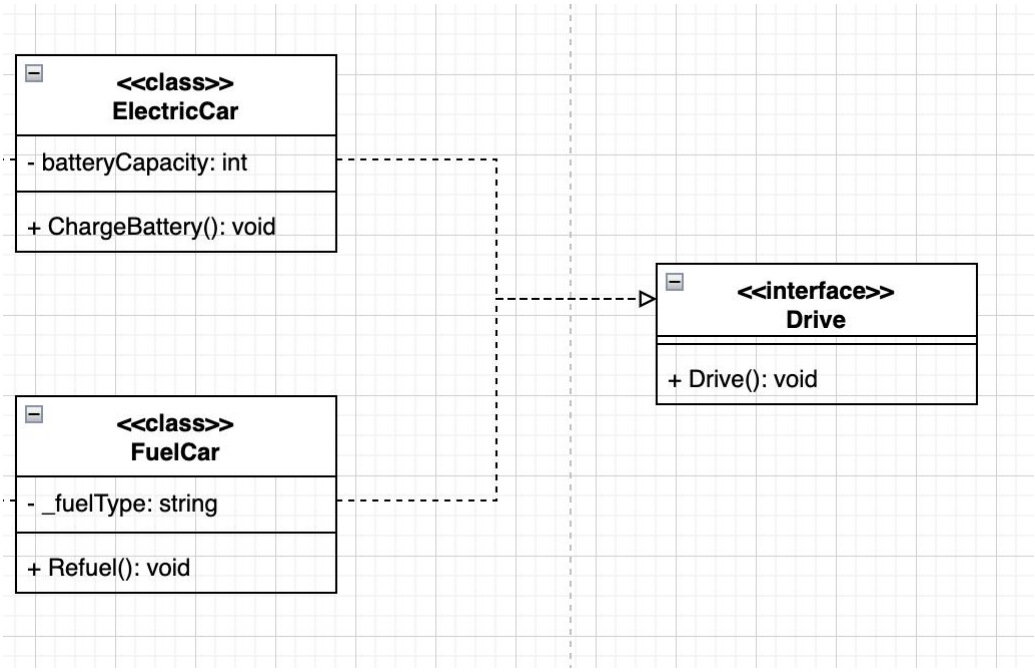
Inheritance allows a class to inherit fields and methods from another class or interfaces.

Class inheritance



The Car class is the base class with properties like make and model, and a method StartEngine().
The FuelCar class inherits from the Car class and adds new features like fuelType and the method Refuel().
The ElectricCar class inherits from Car and adds batteryCapacity and ChargeBattery().

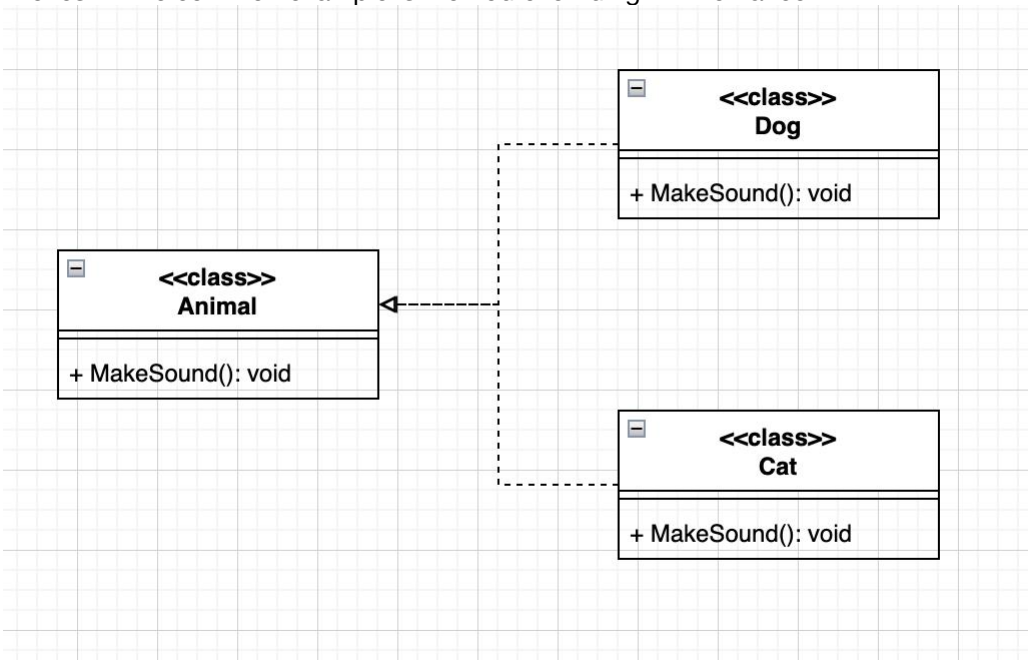
Interface Inheritance(Implementation)



Both FuelCar and ElectricCar have the ability to drive, but the drive function is not responsible for actually constructing the car; it is a method or external function used to control the car's movement. So we need to create Drive Interface, and let ElectricCar and FuelCar to inherite from it.

Polymorphism

Polymorphism allows the same method to behave differently based on the object that invokes it. The common example is method overriding in inheritance.



When we call MakeSound() on an Animal object, it might bark or meow, depending on whether the object is a Dog or a Cat. This demonstrates runtime polymorphism, where the actual method invoked is determined at runtime.

```
Animal dog= new Dog();
dog.MakeSound(); // Output: Bark!

Animal cat= new Cat();
cat.MakeSound(); // Output: Meow!
```

Delegates and Lambdas

Delegates are types that represent references to methods, and Lambdas are a shorthand for anonymous methods in C#.

```
using System;
using SplashKitSDK;

namespace _delegate_test
{
    delegate void Procedure();
    public class Program
    {
        public static void Method1()
        {
            Console.WriteLine("Method 1");
        }

        public static void Method2()
        {
            Console.WriteLine("Method 2");
        }

        public void Method3()
        {
            Console.WriteLine("Method 3");
        }

        public static void Main()
        {
            Procedure someProcs = null;

            someProcs += new Procedure(Program.Method1);
            someProcs += new Procedure(Method2);  // Example with omitted class name

            Program demo = new Program();

            someProcs += new Procedure(demo.Method3);
            someProcs();
        }
    }
}
```

Just like creating a slot, when we use it in a specific situation, we fill the slot with a specific method and then utilize it.

Lambda Expressions simplify the syntax for writing inline methods.

```
Action<int, int> sum = (x, y) => Console.WriteLine(x + y); // Lambda used for a method
sum(3, 4); // Output: 7
```

The delegate Action<int, int> points to a method signature that takes two integers and returns void.
The lambda expression (x, y) => Console.WriteLine(x + y) is an inline function that calculates the sum of x and y.