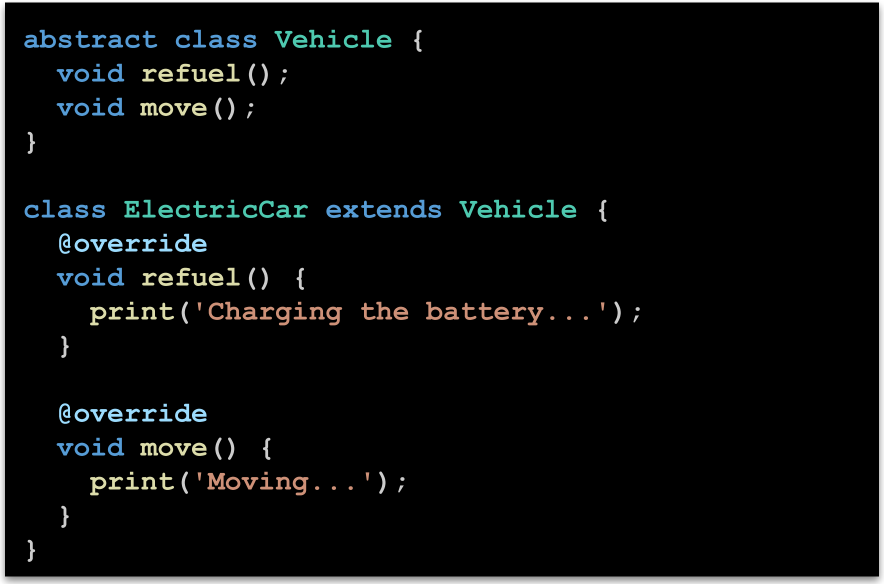
Design Patterns - Hallmarks of Good Architecture

SOLID Principles of **Object-Oriented Programming**

Practice Session

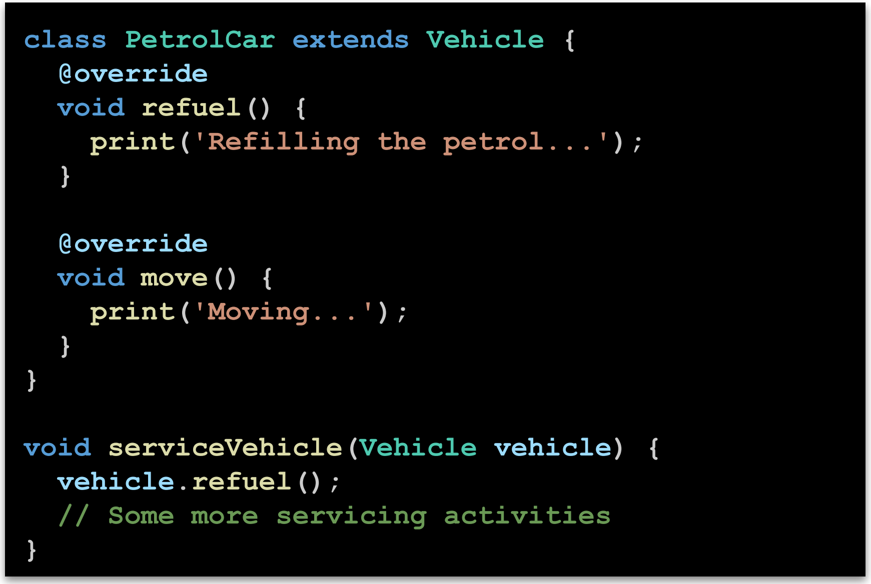
Liskov Substitution Principle (LSP)



```
abstract class Vehicle {
  void refuel();
  void move();
}

class ElectricCar extends Vehicle {
  @override
  void refuel() {
    print('Charging the battery...');
  }

  @override
  void move() {
    print('Moving...');
  }
}
```



```
class PetrolCar extends Vehicle {
  @override
  void refuel() {
    print('Refilling the petrol...');
  }

  @override
  void move() {
    print('Moving...');
  }
}

void serviceVehicle(Vehicle vehicle) {
  vehicle.refuel();
  // Some more servicing activities
}
```

Liskov Substitution Principle (LSP)

Hints:

1. Identify methods that aren't applicable to all subclasses.
2. Consider splitting the superclass into more specific subclasses or interfaces.
3. Ensure each subclass can be used interchangeably with the superclass without causing any issues.

Design Patterns - Hallmarks of Good Architecture

```
abstract class Vehicle {
  void move();
}

abstract class FuelVehicle extends Vehicle {
  void refuel();
}

abstract class ElectricVehicle extends Vehicle {
  void charge();
}

class ElectricCar extends ElectricVehicle {
  @override
  void charge() {
    print('Charging the battery...');
  }

  @override
  void move() {
    print('Moving...');
  }
}
```

```
class PetrolCar extends FuelVehicle {
  @override
  void refuel() {
    print('Refilling the petrol...');
  }

  @override
  void move() {
    print('Moving...');
  }
}

void serviceFuelVehicle(FuelVehicle vehicle) {
  vehicle.refuel();
  // Some more servicing activities
}

void serviceElectricVehicle(ElectricVehicle
vehicle) {
  vehicle.charge();
  // Some more servicing activities
}
```

Design Patterns - Hallmarks of Good Architecture

Liskov Substitution Principle (LSP)

1. In the refactored solution, we separated `FuelVehicle` and `ElectricVehicle` as two different abstractions, both extending `Vehicle`.
2. This allows us to create **separate service methods** for each type of vehicle, ensuring that we don't attempt to perform an action that doesn't make sense for a particular type of vehicle.
3. The original code violated the **Liskov Substitution Principle** because `ElectricCar`, as a subclass of `Vehicle`, wasn't truly substitutable for `Vehicle` in all situations.
4. Specifically, the refuel method didn't make sense for `ElectricCar`.