Facade Pattern

Michal Moravik, SD20w2

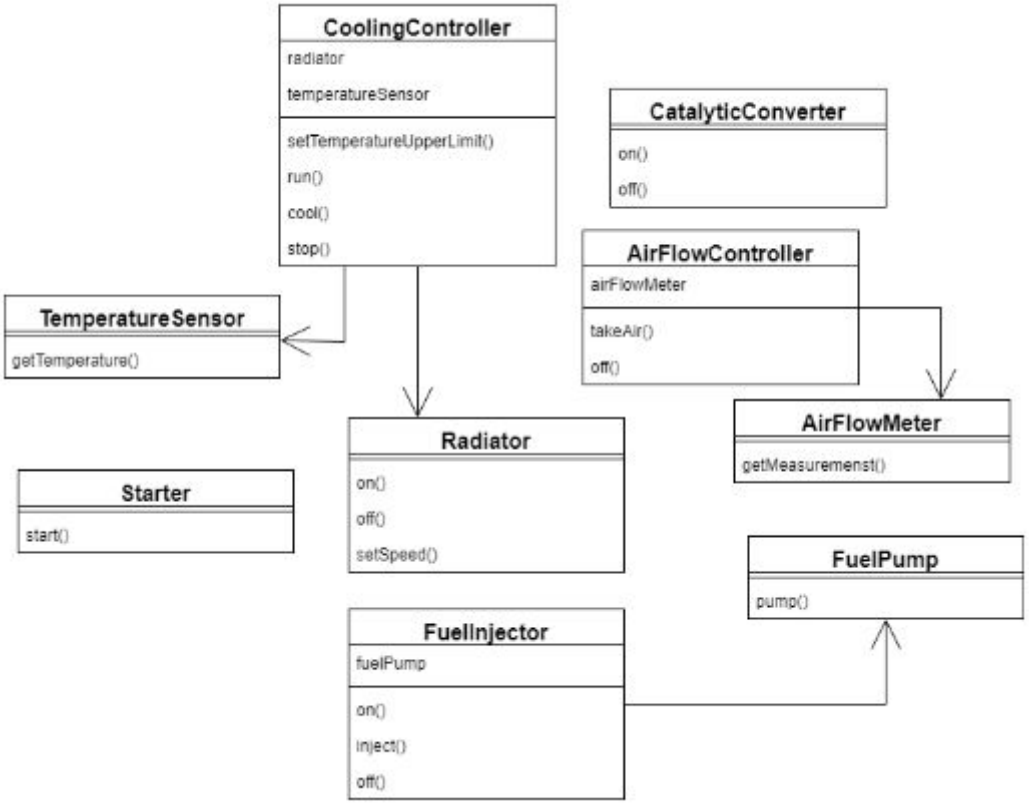
Name: Facade Pattern

Category: Structural

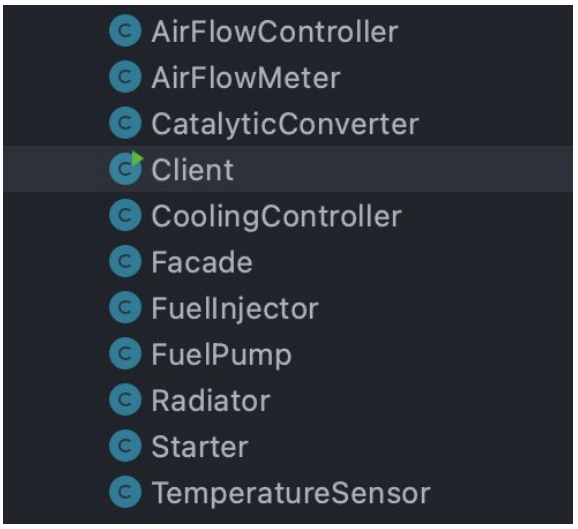
Intent: Decouples the code by providing one place, facade, where different functionalities take place. These functionalities usually include method calls from multiple different classes.

Motivation: We want to encapsulate multiple parts of the code which together form one functionality of the system. This can tremendously help when decoupling the system while it increases readability of the code.

UML:



Implementation:



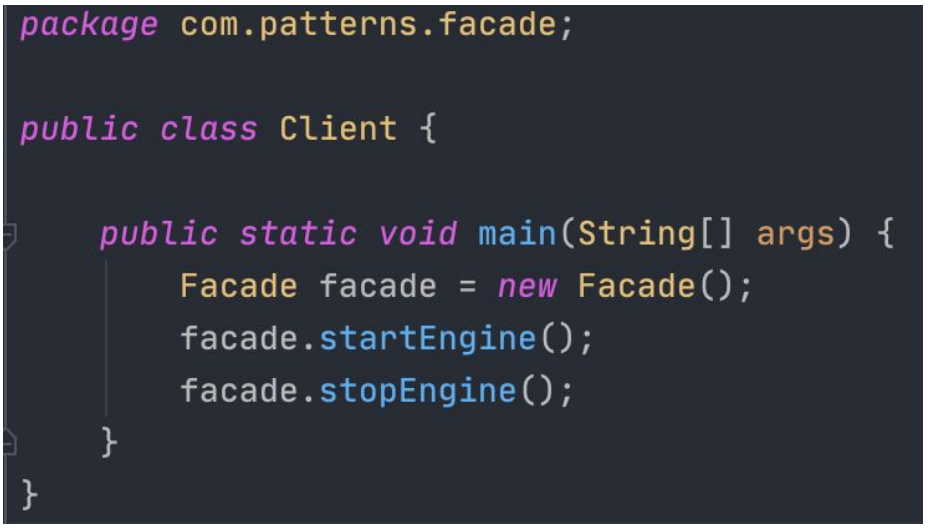

```
package com.patterns.facade;

public class Facade {
    AirFlowMeter airFlowMeter;
    AirFlowController airFlowController;
    FuelPump fuelPump;
    FuelInjector fuelInjector;
    Starter starter;
    Radiator radiator;
    TemperatureSensor temperatureSensor;
    CatalyticConverter catalyticConverter;
    CoolingController coolingController;

    public Facade() {
        this.airFlowMeter = new AirFlowMeter(25);
        this.airFlowController = new AirFlowController(airFlowMeter);
        this.fuelPump = new FuelPump();
        this.fuelInjector = new FuelInjector(fuelPump);
        this.starter = new Starter();
        this.radiator = new Radiator(15);
        this.temperatureSensor = new TemperatureSensor(50);
        this.coolingController = new CoolingController(radiator, temperatureSensor);
        this.catalyticConverter = new CatalyticConverter();
    }

    public void startEngine() {
        this.airFlowController.takeAir();
        this.fuelInjector.on();
        this.fuelInjector.inject();
        this.starter.start();
        this.coolingController.setTemperatureUpperLimit();
        this.coolingController.run();
        this.catalyticConverter.on();
    }

    public void stopEngine() {
        this.fuelInjector.off();
        this.catalyticConverter.off();
        this.coolingController.setTemperatureUpperLimit();
        this.coolingController.stop();
        this.airFlowController.off();
    }
}
```



```
package com.patterns.facade;

public class Client {

    public static void main(String[] args) {
        Facade facade = new Facade();
        facade.startEngine();
        facade.stopEngine();
    }
}
```

In the above screenshots, we can see that Facade class encapsulate multiple function calls to create one specific functionality which is then called by the client. We, therefore, make the client more readable file.

Consequences:
Pros:

- Decouples code - allows you to have exactly the required coupling between two entities
- Increasing readability of the code
- No need to change the client's code

Cons:

- ...

Known uses:

- Simply everywhere where people wanted to decouple a code and encapsulate one functionality composed of multiple function calls.

Related patterns:

- Factory pattern - can implement conditional creation in the facade