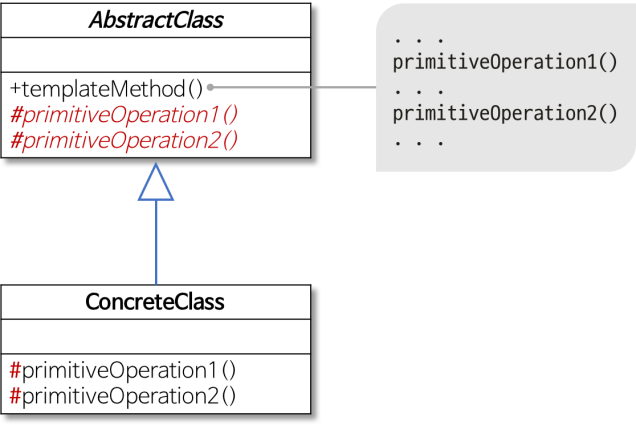
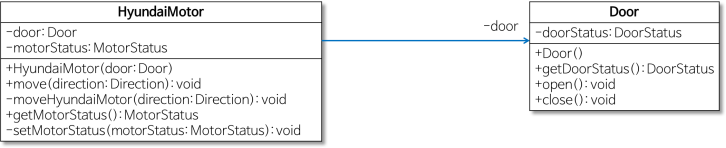
템플릿 메서드 패턴

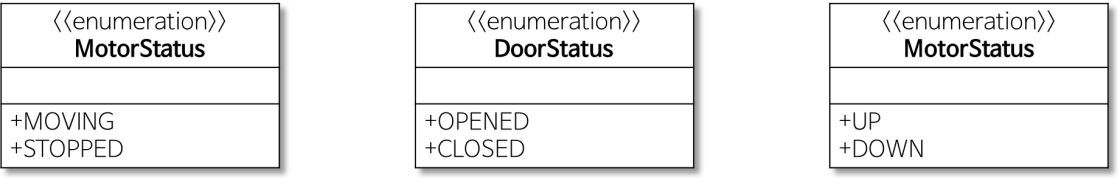
* 어떤 작업을 처리하는 일부분을 서브클래스로 캡슐화해 전체 일을 수행하는 구조는 바꾸지 않으면서 특정 단계에서는 수행하는 내역을 바꾸는 패턴
  + 즉, 전체적으로는 동일하면서 부분적으로는 다른 구문으로 구성된 메서드의 코드 중복을 최소화할 때 유용하다.
  + 다른 관점으로 보면 동일한 기능을 상위 클래스에서 정의하면서 확장 변화가 필요한 부분만 서브 클래스에서 구현할 수 있도록 한다.
  + 예를 들어 전체적인 알고리즘은 상위 클래스에서 정의하면서, 확장/변화가 필요한 부분만 서브 클래스에서 구현할 수 있도록 한다.
  + 행위 패턴 중 하나이다.

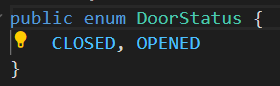
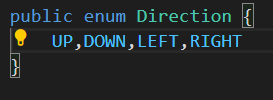


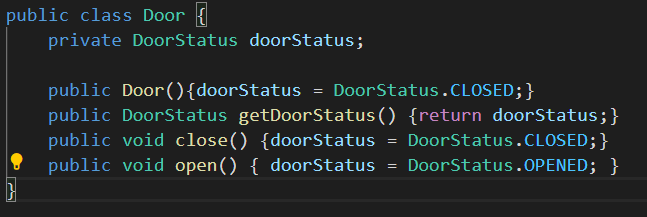
* 역할이 수행하는 과정
  + AbstractClass
    - 템플릿 메서드를 정의하는 클래스
    - 하위 클래스에 공통 알고리즘을 정의하고 하위 클래스에서 구현된 기능을 primitive메서드 또는 hook 메서드로 정의하는 클래스
  + ConcreteClass
    - 물려받은 primitive 메서드 또는 hook 메서드를 구현하는 클래스
    - 상위 클래스에 구현된 템플릿 메서드의 일반적인 알고맂므에서 하위 클래스에 적합하게 primitive 메서드나 hook 메서드를 오버라이드 하는 클래스
* 참고
  + 행위 패턴
    - 객체나 클래스 사이의 알고리즘이나 책임 분배에 관련된 패턴
    - 한 객체가 혼자 수행할 수 없는 작업을 여러 개의 객체로 어떻게 분배하는지, 또 그렇게 하면서도 객체 사이의 결합도를 최소화하는 것에 중점을 둔다.
* 예시
* 여러 회사의 모터 지원하기

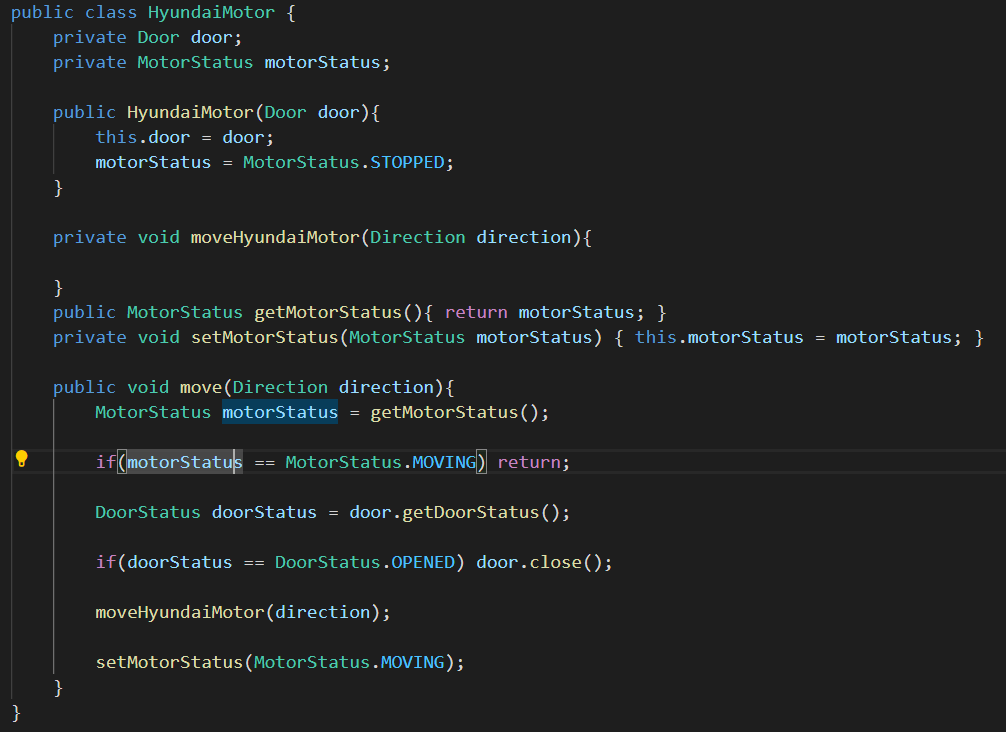


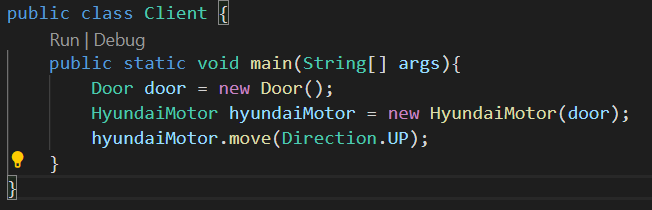
* + 엘리베이터 제어 시스템에서 모터를 구동시키는 기능
    - 예를 들어 현대 모터를 이용하는 제어시스템이라면 HyundaiMotor클래스에 move메서드를 정의할 수 있다.
    - HyundaiMoter Class – 연관관계 -> door클래스
      * Move 메서드를 실행할 때 안전을 위해 door가 닫혀 있는지 확인하기 위해 연관관계를 정의한다.
    - Enumeration Interface
      * 모터의 상태(정지 중, 이동 중)
      * 문의 상태 (닫혀 있는중, 열려 있는중)
      * 이동방향 (위로 아래로)



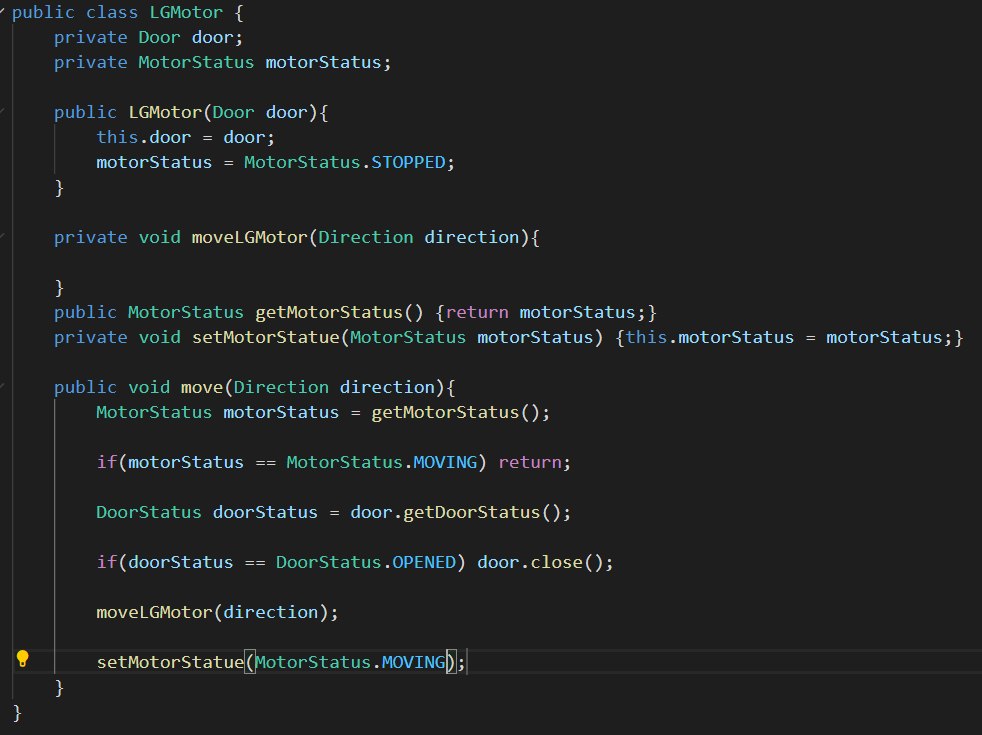




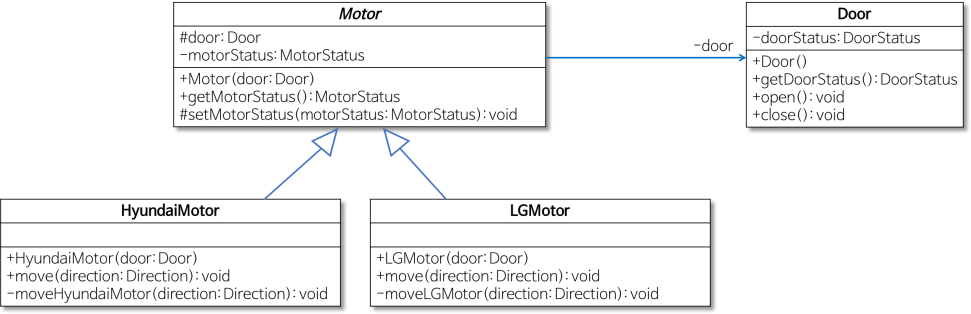




* + HyundaiMotor 클래스의 move메서드는 우선 getMotorStatus 메서드를 호출해 모터의 상태를 조회한다.
    - 모터가 이미 동작중이면 move 메서드의 실행을 종료한다.
  + Door클래스의 getDoorStatus 메서드를 호출해 문의 상태를 조회한다.
    - 문이 열려 있는 상태면 Door클래스의 close 메서드를 호출해 문을 닫는다
    - 그리고 moveHyundaiMotor 메서드를 호출해 모터를 구동시킨다.
    - setMotorStatus를 호출해 모터의 상태를 MOVING으로 기록한다
* 문제점
  + 다른 회사의 모터를 제어해야 하는 경우
    - HyundaoMotor 클래스는 현대 모터를 구동시킨다. 만약 LG모터를 구동시키면?



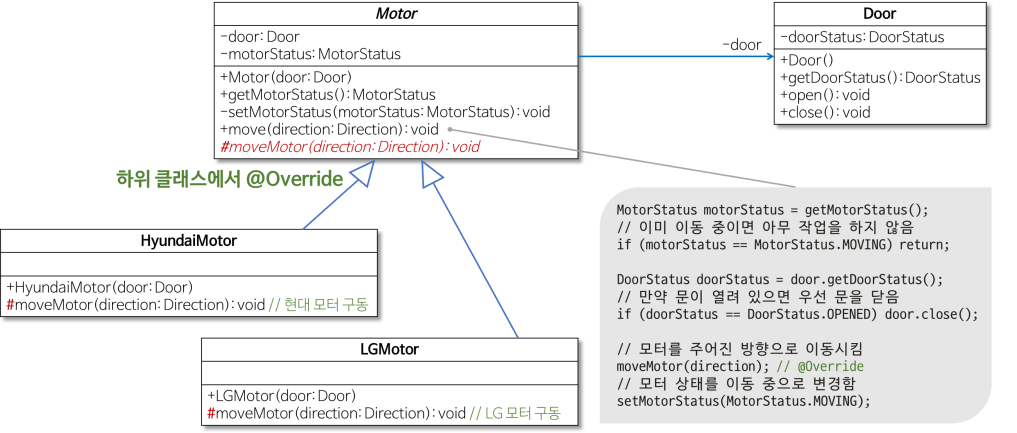
* + HyundaiMotor와 LGMotor에는 여러 개 메서드가 동일하게 구현되어 있다.
    - 즉, 2개의 클래스는 많은 중복 코드를 가진다
      * Door클래스의 연관관계
      * motorStatus필드
      * getMotorStatus, setMotorStatus클래스
    - 중복 코드는 유지보수성을 약화 시키므로 바람직하지 않다.
* 해결책
* 방법1
  + 2개 이상의 클래스가 유사한 기능을 제공하면서 중복된 코드가 있느 경우에는 상속을 이용해서 코드 중복은 문제를 피할 수 있다.



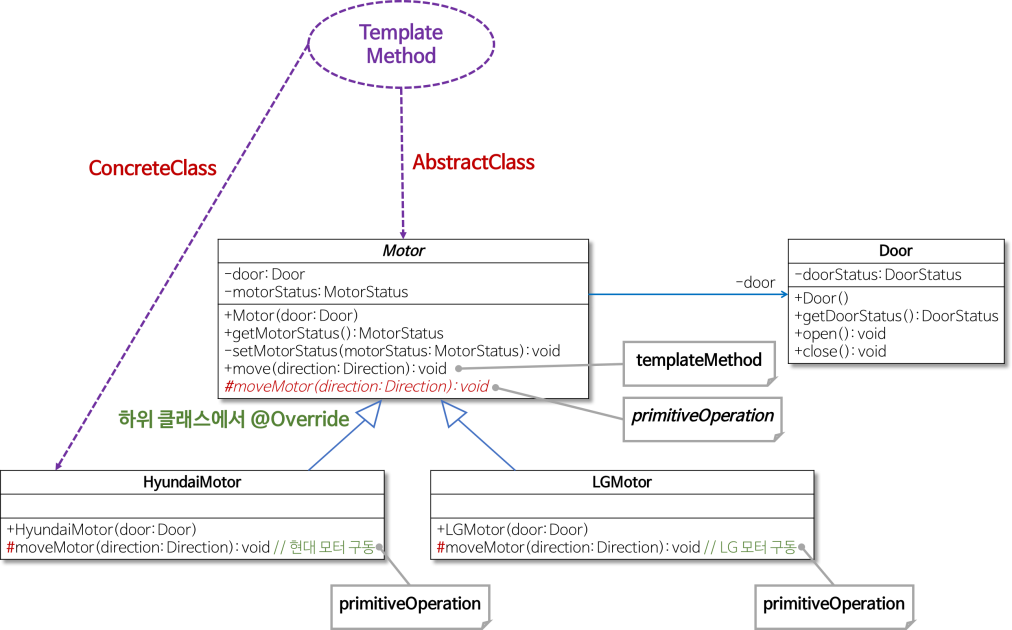
* + Motor클래스를 상위 클래스로 정의함으로써 중복 코드는 제거할 수 있다.
    - Door 클래스와의 연관관계
    - motorStatus필드
    - getMotorStatus, setMotorStatus 메서드
  + 그러나 hyundaiMotor와 LGMotor의 move 메서드는 대부분이 비슷하다.
    - 즉 move 메서드에는 여전히 코드 중복 문제가 있다.
* 방법 2

위의 move 메서드와 같이 부분적으로 중복되는 경우에도 상속을 활용해 코드 중복을 피할 수 있다.

* + move메서드에서 moveHyundaiMotor 메서드와 moveLGMotor 메서드를 호출하는 부분만 다르다.
  + 또한 moveHyundaiMotor메서드와 moveLGMotor 메서드는 기능면에서는 동일하다.



* + Move 메소드를 상위 motor 클래스로 이동시킨다.
  + MovehyundaiMotor 메서드와 moveLGMotor메서드의 호출 부분을 하위클래스에서 오버라이드 한다.
  + Motor 클래스의 move메소드는 hyundaiMotor와 LGMotor에서 동일한 기능을 구현하면서, 각 하위 클래스에서 구체적으로 정의할 필요가 있는 부분, 즉 moveMotor메서드 부분만 각 하위 클래스에서 오버라이드 되도록 한다.
  + 이렇게 Template Method 패턴을 이용하면 전체적으로는 동일하면서 부분적으로는 다른 구문으로 구성된 메서드의 코드 중복을 최소화 할 수 있다.



* + “AbstractClass” : Motor클래스
  + “ConcreteClass” : HyundaiMotor클래스와 LGMotor클래스
  + “템플릿 메서드” : Motor클래스의 move메서드
  + primitive또는 hook메서드 : move메서드에서 호출되면서 하위 클래스에서 오버라이드 될 필요가 있는 moveMotor클래스