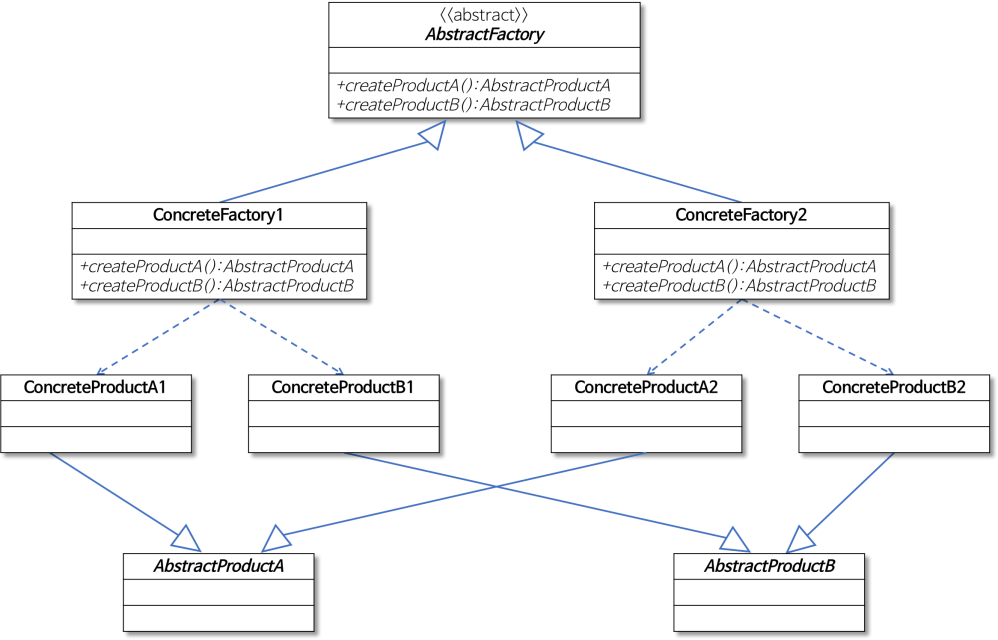
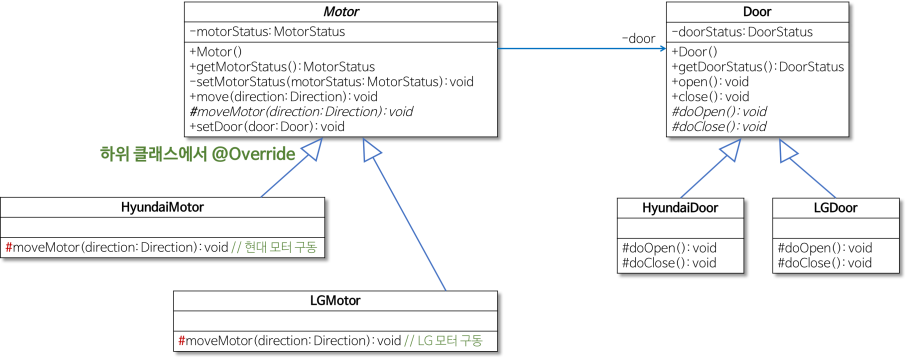
추상 팩토리 패턴

추상 팩토리 패턴이란

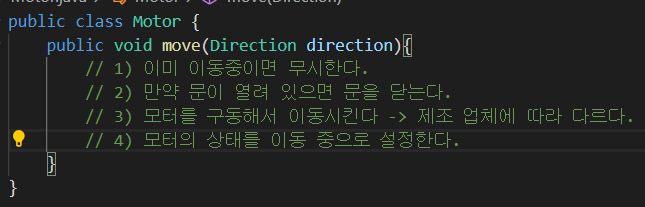
* 구체적인 클래스에 의존하지 않고 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공하는 패턴
  + 즉 관련성 있는 여러 종류의 객체를 일관된 방식으로 생성하는 경우에 유용하다.
  + 싱글턴 패턴, 팩토리 메서드 패턴을 사용한다
  + 생성 패턴의 하나.



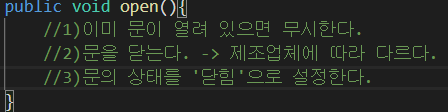
* 역할이 수행하는 작업
  + AbsctractFactory
    - 실제 팩토리 클래스의 공통 인터페이스
  + ConcreteFactory
    - 구체적인 팩토리 클래스로 AbstractFactory 클래스의 추상 메서드를 오버라이드 함으로써 구체적인 제품을 생성한다
  + AbstractProduct
    - 제품의 공통 인터페이스
  + ConcreteProduct
    - 구체적인 팩토리 클래스에서 생성되는 구체적인 제품
* 참고
  + 생성패턴
    - 객체 생성에 관련된 패턴
    - 객체의 생성과 조합을 캡슐화해 특정 객체가 생성되거나 변경되어도 프로그램 구조에 영향을 크게 받지 않도록 유연함을 제공한다.
* 예시
* 엘리베이터 부품 업체 변경하기



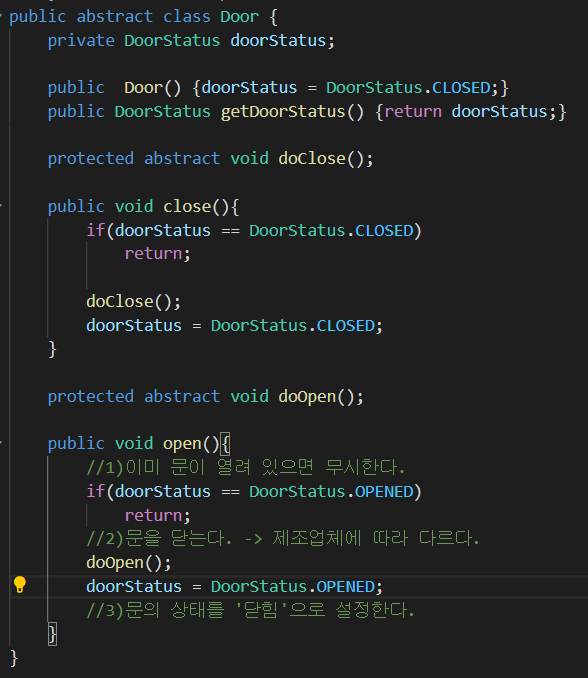
* 여러 제조업체 부품을 사용하더라도 같은 동작을 지원하게 하는 것이 바람직하다. 즉, 엘리베이터 프로그램의 변경을 최소화해야 한다.
  + 추상클래스로 Motor Door를 정의
    - 제조업체가 다른 경우 구체적인 제어 방식은 다르지만 엘리베이터 입장에서는 모터를 구동해 엘리베이터를 이동시킨다는 면에서는 동일하다
    - 그러므로 추상 클래스로 Motor를 정의하고 여러 제조업체의 모터를 하위 클래스로 정의할 수 있다.
  + Motor클래스 – 연관관계 -> door
    - Motor클래스는 이동하기 전에 문을 닫아야 한다.
    - Door 객체의 getDoorStatus() 메서드를 호출하기 위해서 연관관계가 필요하다.
  + Motor클래스의 핵심 기능인 이동은 move()메서드로 정의

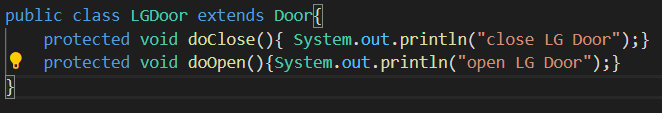


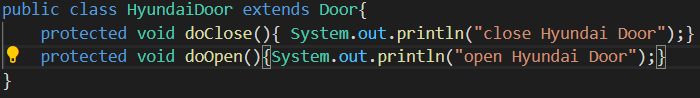
* + - 4단계의 동작 모두 LGMotor, HyundaiMotor 클래스의 공통 기능이고, 3)단계 부분만 달라진다.
    - 템플릿 메서드 패턴
      * 즉, 일반적인 흐름에서는 동일하지만, 특정 부분만 다른 동작을 하는 경우에는 일반적인 기능을 상위 클래스에 템플릿 메서드로서 설계할 수 있다.
      * 특정 부분에 대해서는 하위 클래스에서 오버라이드 할 수 있도록 한다.
    - Door클래스의 open(), close() 메서드 정의



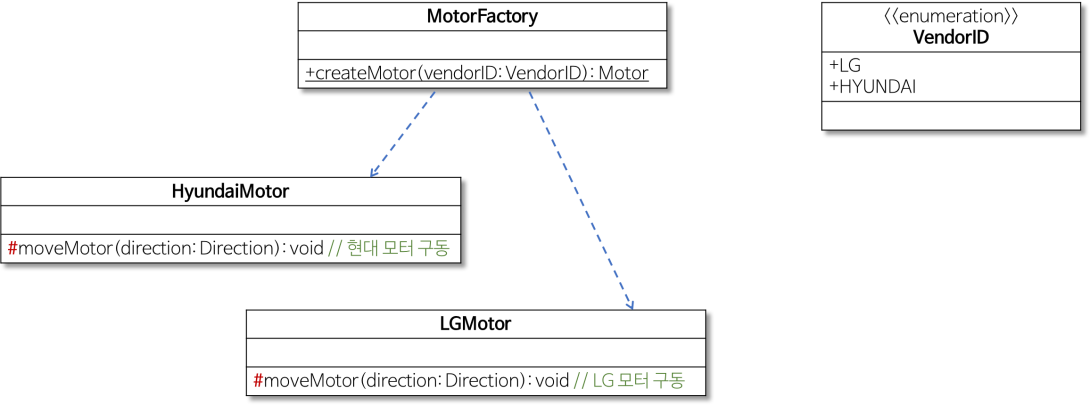
* + - 나머지 동작은 공통으로 필요한 기능이고, 2)단계 부분만 달라진다.
    - 즉 move() 메서드와 같이 템플릿 메서드 패턴을 적용할 수 있다.
* 템플릿 메서드 패턴을 적용한 예제 코드
  + 전체적으로 동일하면서 부분적으로는 다른 구문으로 구성된 메서드의 코드 중복을 최소화할 수 있다.

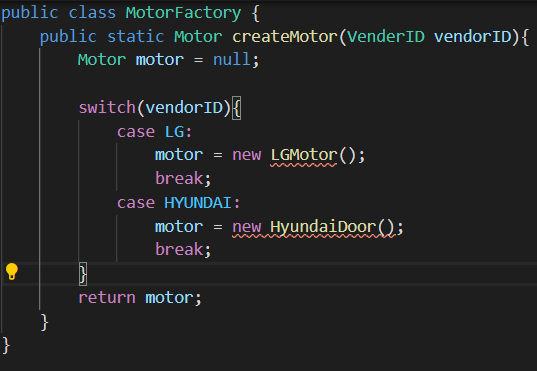




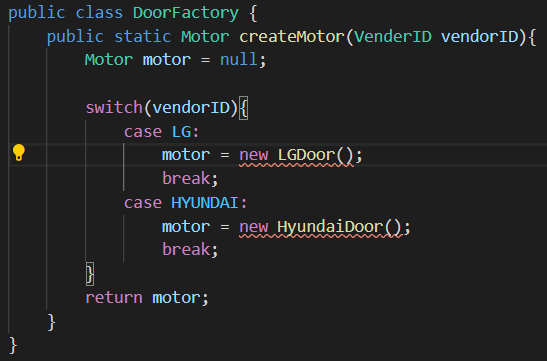


* 엘리베이터 입장에서는 특정 제조 업체의 모터와 문을 제어하는 클래스가 필요하다.
  + 예를 들어 LGMotor. LGDoor 객체가 필요하다
  + 팩토리 메서드 패턴 이용
* 팩토리 메서드 패턴을 적용한 예제 코드
  + 객체 생성 처리를 서브 클래스로 분리하여 캡슐화함으로써 객체 생성의 변화에 대비할 수 있다.

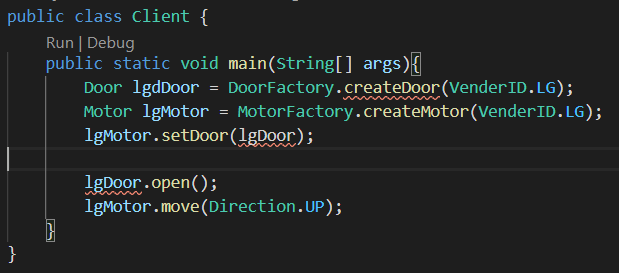


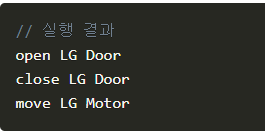
* MotorFactory 클래스의 createMotor()메서드는 인자로 주어진 VendorID에 따라 LGMotor객체 또는 HyundaiMotor 객체를 생성한다.



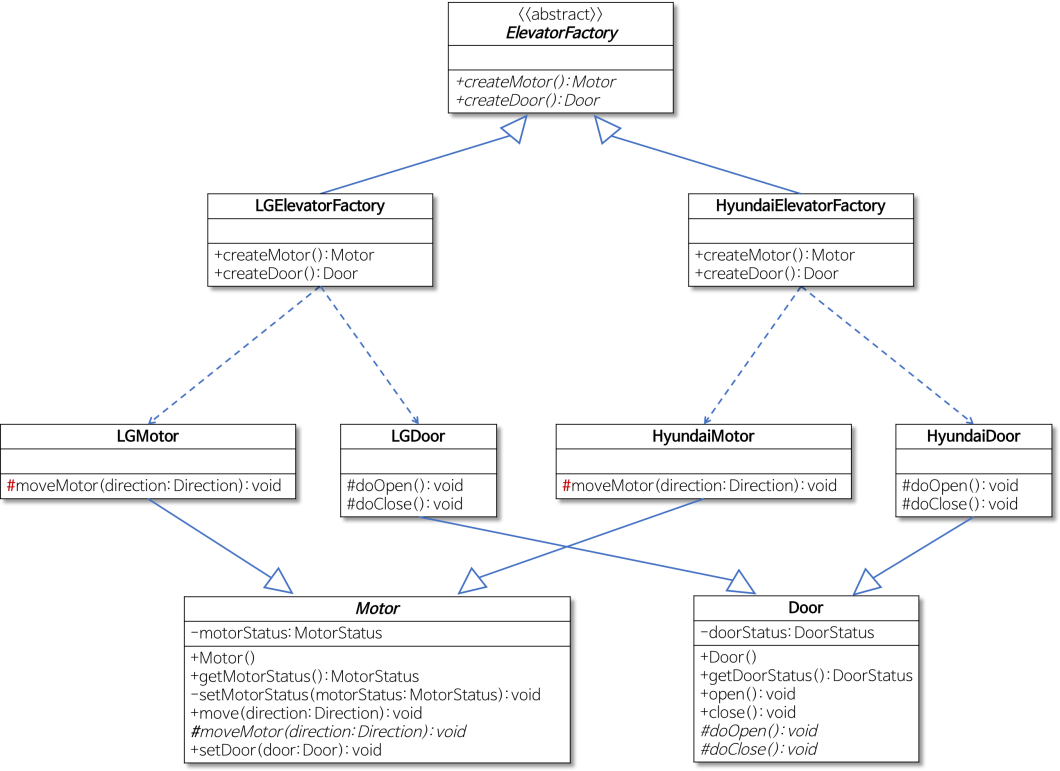
* DoorFactory클래스의 createDoor() 메서드는 인자로 주어진 VendorID에 따라 LGDoor 객체 또는 HyundaiDoor 객체를 생성한다.



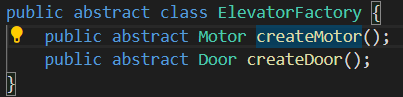
* + 제조업체에 따라 문에 해당하는 구체적인 클래스를 생성해야 한다.
  + LGDoor와 LGMotor객체를 활용하므로 move()메서드를 호출하기 전 open()메서드에 의해 문이 열려있는 상태다.
  + 그러므로 move()메서드는 문을 먼저 닫고 이동을 시작한다.
  + 이때 LGDoor와 LGMotor객체가 모두 사용한다.

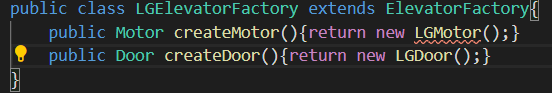


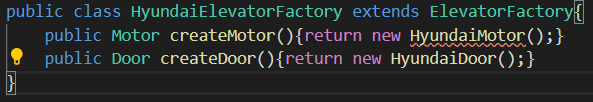
* 문제점
  + 다른 제조업체의 부품을 사용해야 하는 경우
    - LG의 부품 대신 현대의 부품(HyundaiMotor, HyundaiDoor 클래스)을 사용해야 한다면?
    - 총 10개의 부품을 사용해야 한다면 각 Factory 클래스를 구현하고 이들의 Factory객체를 각각 생성해야 한다.
    - 즉 부품의 수가 많으면 특정 업체별 부품을 생성하는 코드의 길이가 길어지고 복잡해진다.
  + 새로운 제조업채의 부품을 지원해야 하는 경우
    - 삼성의 부품(SamsumgMotor, SamsumDoor클래스)을 지원해야 한다면
      * DoorFactory 클래스 뿐만 아니라 나머지 9개의 부품과 연관된 Factory 클래스에서도 마찬가지로 삼성의 부품을 생성하도록 변경해야 한다.
      * 또한 위와 마찬가지로 특정 업체별 부품을 생성하는 코드에서 삼성의 부품을 생성하도록 모두 변경해야 한다.
    - 즉 결과적으로 기존의 팩토리 메서드 패턴을 이용한 객체 생성은 관련있는 여러 개의 객체를 일관성 있는 방식으로 생성하는 경우에 많은 코드 변경이 발생하게 된다는 것이다.
  + 해결책
    - 여러 종류의 객체를 생성할 때는 객체들 사이의 관련성이 있는 경우라면 각 종류별로 별도의 Factory 클래스를 사용하는 대신 관련 객체들을 일관성 있게 생성하는 Factory클래스를 사용하는 것이 편리할 수 있다.

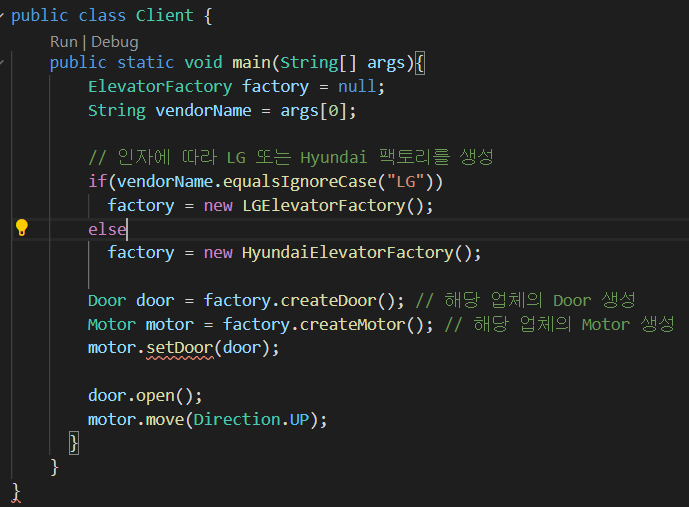


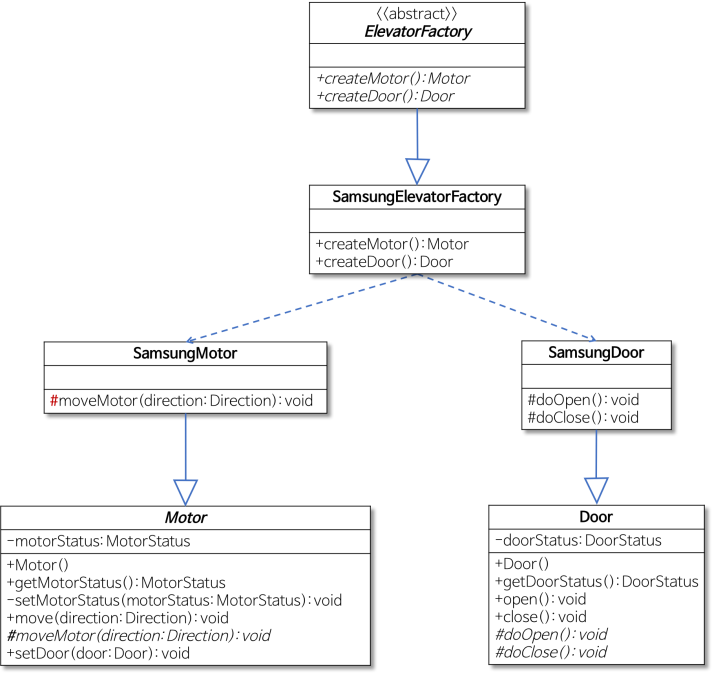
* + - 예를 들어 MotorFactory, DoorFactory 클래스와 같이 부품별로 Factory클래스를 만드는 대신LGElevatorFactory나 HyundaiElevatorFactory 클래스와 같이 제조 업체별로 Factory클래스를 만들 수 있다.
    - LGElevatorFactory : LGMotor LGDoor 객체를 생성하는 팩토리 클래스
    - HyundaiElevatorFactory : HyundaiMotor, HyundaiDoor 객체를 생성하는 팩토리 클래스

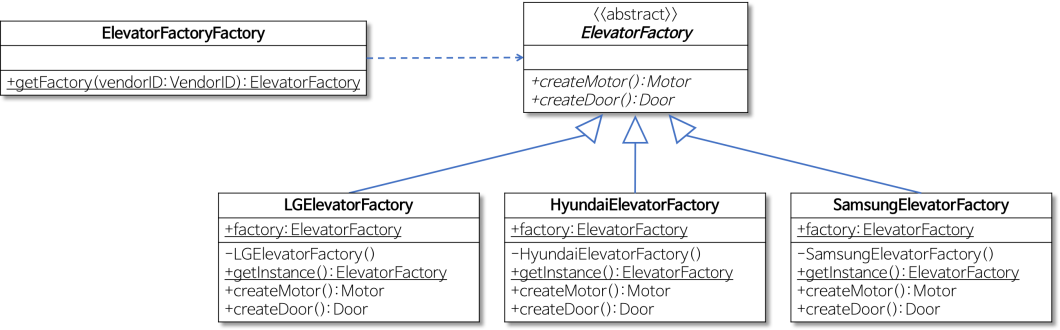








* 인자로 주어진 업체 이름에 따라 적절한 부품 객체를 생성한다.
  + 즉 문제점 1과 같이 다른 제조 업체의 부품으로 변경하는 경우에도 Client코드를 변경할 필요가 없다.
* 제조 업체별로 Factory클래스를 정의했으므로 제조 업체별 부품 객체를 간단히 생성할 수 있다.
  + 즉, 문제점 2와 같이 새로운 제조 업체의 부품을 지원하는 경우에도 해당 제조 업체의 부품으로 생성하는 Factory클래스만 새롭게 만들면 된다.
* 
* 추가 보완 해결책(정확한 추상 팩토리 패턴 적용)

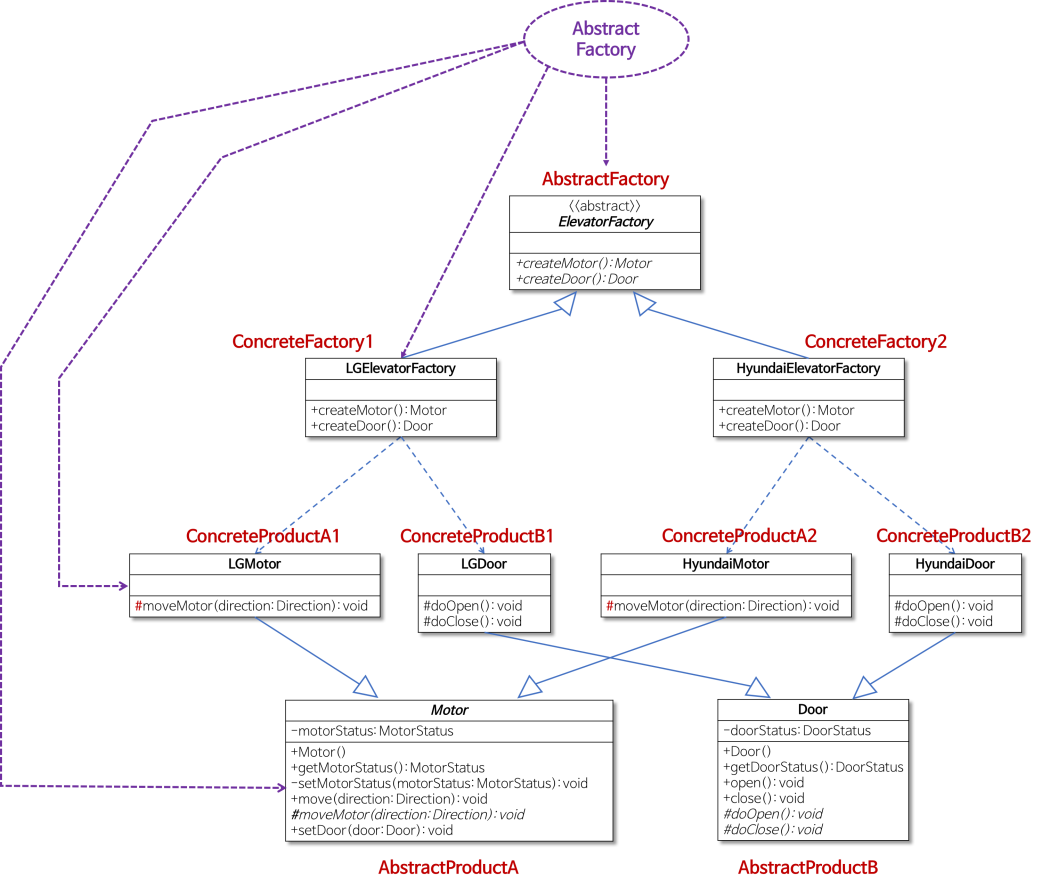


과정1

* 팩토리 메서드 패턴 : 제조 업체별 Factory 객체를 생성하는 방식을 캡슐화 한다.
* ElevatorFactory클래스 : vendorID에 따라 해당 제조 업체의 Factory객체를 생성
* ElevatorFacotry클래스의 getFactory() 메서드: 팩토리 메서드

과정2

* 싱글턴 패턴 : 제조 업체별 Factory객체는 각각 1개만 있으면 된다
  + 3개의 제조 업체별 Factory클래스를 싱글턴으로 설계
  + 추상 팩토리 패턴을 이용한 방법을 사용하는Client
  + 추상 팩토리 패턴 최종 클래스 다이어그램



* AbstractFactory : ElebatorFactory클래스
* ConcreteFactory : LGElevatorFactory 클래스와 HyundaiElevatorFactory클래스
* AbstractProductA : Motor클래스
* ConcreteProductA : LGMotor클래스와 HyundaiMotor클래스
* AbstractProductB : Door 클래스
* ConcreteProductB : LGDoor 클래스와 HyundaiDoor 클래스