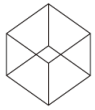
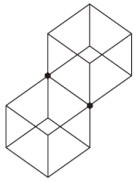
13. 추상 팩토리 패턴



JAVA 객체 지향 디자인 패턴

UML과 GoF 디자인 패턴 핵심 10가지로 배우는

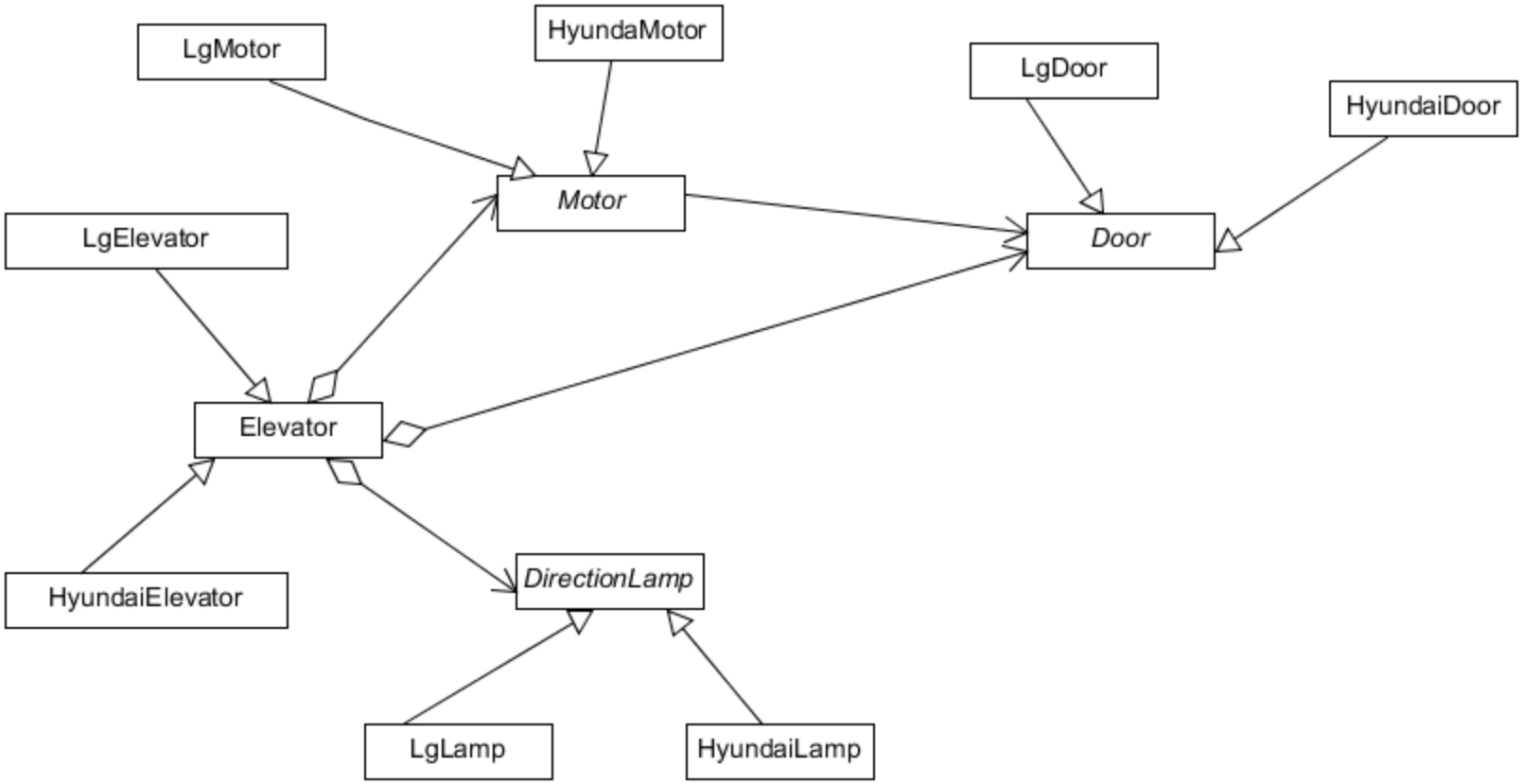


학습목표

학습목표

- 관련된 여러 클래스의 객체를 생성하는 코드의 캡슐화 방법 이해하기
- 추상 팩토리 패턴을 이용한 관련 객체의 생성 방법 이해하기
- 사례 연구를 통한 추상 팩토리 패턴의 핵심 특징 이해하기

엘리베이터 만들기


LgMotor
HyundaMotor
LgDoor
HyundaiDoor
Motor
Door
LgElevator
Elevator
DirectionLamp
HyundaiElevator
LgLamp
HyundaiLamp

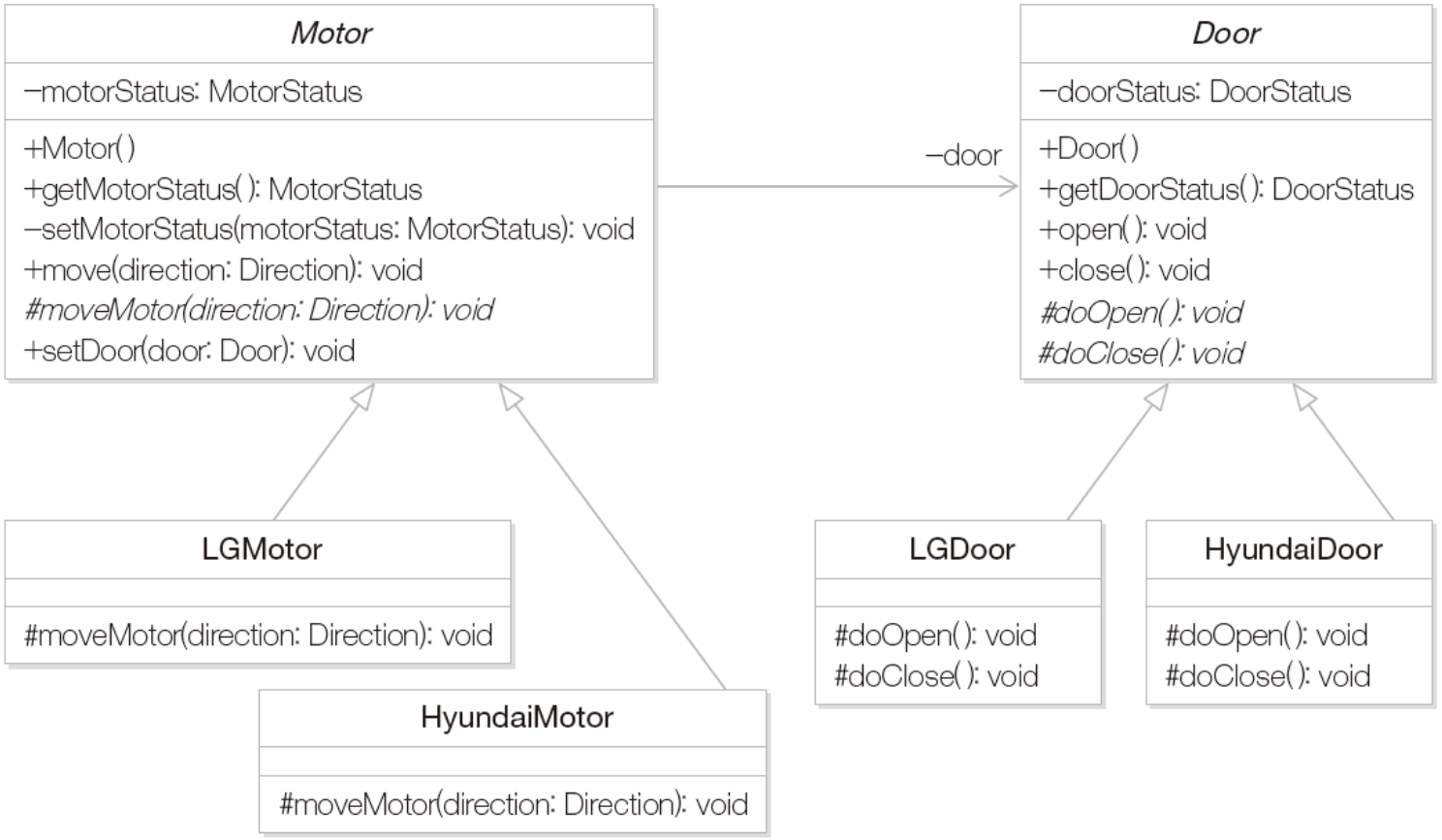
소스코드

```
public class Elevator {
    private Motor motor;
    private Door door;
    private DirectionLamp lamp;

    public void setLamp(DirectionLamp lamp) {
        this.lamp = lamp;
    }
    public void setMotor(Motor motor) {
        this.motor = motor;
    }

    public void setDoor(Door door) {
        this.door = door;
    }

    public void move(Direction direction) {
        motor.move(direction);
        lamp.doLight(direction);
    }
}
```

13.1 엘리베이터 부품 업체의 변경하기

❖ LG와 현대 업체의 모터와 문을 지원하는 클래스

그림 13-1 LG와 현대의 모터와 문



템플릿 메서드 패턴의 적용

```
class Motor {
   public void move(Direction direction) {
      // 1) 이미 이동 중이면 무시한다.
      // 2) 만약 문이 열려 있으면 문을 닫는다.
      // 3) 모터를 구동해서 이동시킨다. ➔ 이 부분만 LG, 현대에서 달라짐
      // 4) 모터의 상태를 이동중으로 설정한다.
   }
}
```

소스코드

```java
public abstract class Motor {
    private Door door;
    private MotorStatus motorStatus;
    public void setDoor(Door door) {
        this.door = door;
        this.motorStatus = MotorStatus.STOPPED;
    }

    public void move(Direction direction) {
        MotorStatus motorStatus = getMotorStatus();
        if (motorStatus == MotorStatus.MOVING) return;
        DoorStatus doorStatus = door.getDoorStatus();
        if (doorStatus == DoorStatus.OPENED) door.close();
        moveMotor(direction);
        setMotorStatus(MotorStatus.MOVING);
    }

    private void setMotorStatus(MotorStatus motorStatus) {
        this.motorStatus = motorStatus;
    }

    protected abstract void moveMotor(Direction direction);

    private MotorStatus getMotorStatus() {
        return motorStatus;
    }
}
```

```java
public enum Direction {
    UP, DOWN
}
```

```java
public enum MotorStatus {
    MOVING, STOPPED
}
```

```java
public enum DoorStatus {
    OPENED, CLOSED
}
```

소스 코드

```
public class LgMotor extends Motor {
    @Override
    protected void moveMotor(Direction direction) {
        System.out.println("Lg motor is Moving " + direction);
    }
}
```

```
public class HyundaiMotor extends Motor {
    @Override
    protected void moveMotor(Direction direction) {
        System.out.println("Hyundai motor is Moving " + direction);
    }
}
```

템플릿 메서드 패턴의 적용

```
class Door {
   public void open() {
      // 1) 이미 문이 열려있으면 무시한다.
      // 2) 문을 연다. ➔ 이 부분만 LG, 현대에서 달라짐
      // 3) 문의 상태를 열림으로 설정한다.
   }
}
```

소스코드

```
public abstract class Door {
    private DoorStatus doorStatus;
    public Door() {
        this.doorStatus = DoorStatus.OPENED;
    }

    public DoorStatus getDoorStatus() {
        return doorStatus;
    }

    public void close() {
        if (doorStatus == DoorStatus.CLOSED) return;
        doClose();
        doorStatus = DoorStatus.CLOSED;
    }

    protected abstract void doClose();

    public void open() {
        if (doorStatus == DoorStatus.OPENED) return;
        doOpen();
        doorStatus = DoorStatus.OPENED;
    }

    protected abstract void doOpen();
}
```

소스 코드

```java
public class LgDoor extends Door {
    @Override
    protected void doClose() {
        System.out.println("Close Lg Door");
    }

    @Override
    protected void doOpen() {
        System.out.println("Open Lg Door");
    }
}
```

```java
public class HyundaiDoor extends Door {
    @Override
    protected void doClose() {
        System.out.println("Close Lg Door");
    }

    @Override
    protected void doOpen() {
        System.out.println("Open Lg Door");
    }
}
```

템플릿 메서드 패턴의 적용

```
class DirectionLamp {
   public void light(Direction direction) {
      // 1) 램프의 상태가 미미 이동방향으로 설정되어 있으면 무시
      // 2) 램프를 이동방향으로 설정➔이 부분만 LG, 현대에서 달라짐
      //  3) 램프의 상태를 이동방향으로 설정한다.
   }
}
```

소스코드

```
public abstract class DirectionLamp {
    private Direction lampStatus;
    public void light(Direction direction) {
        if (lampStatus == getLampStatus()) return;
        doLight(lampStatus);
        setLampStatus(lampStatus);
    }
    public Direction getLampStatus() {
        return lampStatus;
    }

    public void setLampStatus(Direction lampStatus) {
        this.lampStatus = lampStatus;
    }

    protected abstract void doLight(Direction lampStatus);


}
```

소스 코드

```java
public class HyundaiLamp extends DirectionLamp {
    @Override
    protected void doLight(Direction direction) {
        System.out.println("Hyundai Lamp "+direction);
    }
}
```

```java
public class LgLamp extends DirectionLamp {
    @Override
    protected void doLight(Direction direction) {
        System.out.println("Lg Lamp "+direction);
    }
}
```

소스코드

```
public class Elevator {
    private Motor motor;
    private Door door;
    private DirectionLamp lamp;

    public void setLamp(DirectionLamp lamp) {
        this.lamp = lamp;
    }
    public void setMotor(Motor motor) {
        this.motor = motor;
    }

    public void setDoor(Door door) {
        this.door = door;
    }

    public void move(Direction direction) {
        motor.move(direction);
        lamp.doLight(direction);
    }
}
```

소스코드

```
public class ElevatorCreator {
    public static Elevator assembleElevator() {
        Elevator elevator = new LgElevator();
        Motor motor = new LgMotor();
        elevator.setMotor(motor);
        Door door = new LgDoor();
        elevator.setDoor(door);
        motor.setDoor(door);
        DirectionLamp lamp = new LgLamp();
        elevator.setLamp(lamp);
        return elevator;
    }

    public static void main(String[] args) {
        Elevator elevator = assembleElevator();
        elevator.move(Direction.UP);
    }
}
```

문제점

- New 연산자 사용으로 인한 OCP 위배
  - Lg 엘리베이터에서 현대 엘리베이터로 변경할 때 기존 코드를 변경

```
public class ElevatorCreator {
    public static Elevator assembleElevator() {
        Elevator elevator = new HyundaiElevator();
        Motor motor = new HyundaiMotor();
        elevator.setMotor(motor);
        Door door = new HyundaiDoor();
        elevator.setDoor(door);
        motor.setDoor(door);
        DirectionLamp lamp = new HyundaiLamp();
        elevator.setLamp(lamp);
        return elevator;
    }

    public static void main(String[] args) {
        Elevator elevator = assembleElevator();
        elevator.move(Direction.UP);
    }
}
```

팩토리 메서드 패턴의 적용: ElevatorFactory

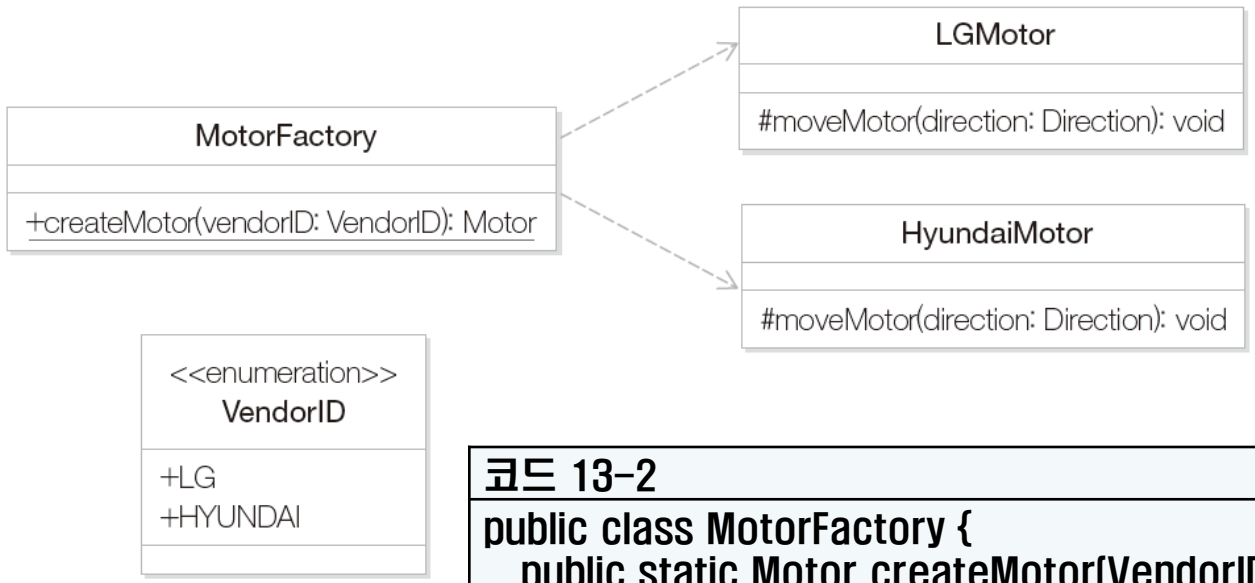
코드

```
public class ElevatorFactory {
   public static Elevator createElevator(VendorID vendorID) {
      Elevator elevator= null ;
      switch ( vendorID ) {
      case LG : elevator = new LgElevator() ; break ;
      case HYUNDAI : elevator = new HyundaiElevator() ; break
;
      }
      return elevator;
   }
}
```

public enum VendorID { LG, HYUNDAI }

팩토리 메서드 패턴의 적용: MotorFactory

그림 13-2 모터 객체 생성을 위한 MotorFactory 클래스



코드 13-2

```
public class MotorFactory {
   public static Motor createMotor(VendorID vendorID) {
      Motor motor = null ;
      switch ( vendorID ) {
      case LG : motor = new LGMotor() ; break ;
      case HYUNDAI : motor = new HyundaiMotor() ; break ;
      }
      return motor ;
   }
}
```

팩토리 메서드 패턴의 적용: DoorFactory

코드 13-3

```
public class DoorFactory {
   public static Door createDoor(VendorID vendorID) {
      Door door = null ;
      switch ( vendorID ) {
      case LG : door = new LGDoor() ; break ;
      case HYUNDAI : door = new HyundaiDoor() ; break ;
      }
      return door ;
   }
}
```

팩토리 메서드 패턴의 적용: LampFactory

코드

```
public class LampFactory {
   public static Lamp createLamp(VendorID vendorID) {
      DirectionLamp lamp = null ;
      switch ( vendorID ) {
      case LG : lamp = new LgLamp() ; break ;
      case HYUNDAI : lamp = new HyundaLamp() ; break ;
      }
      return lamp;
   }
}
```

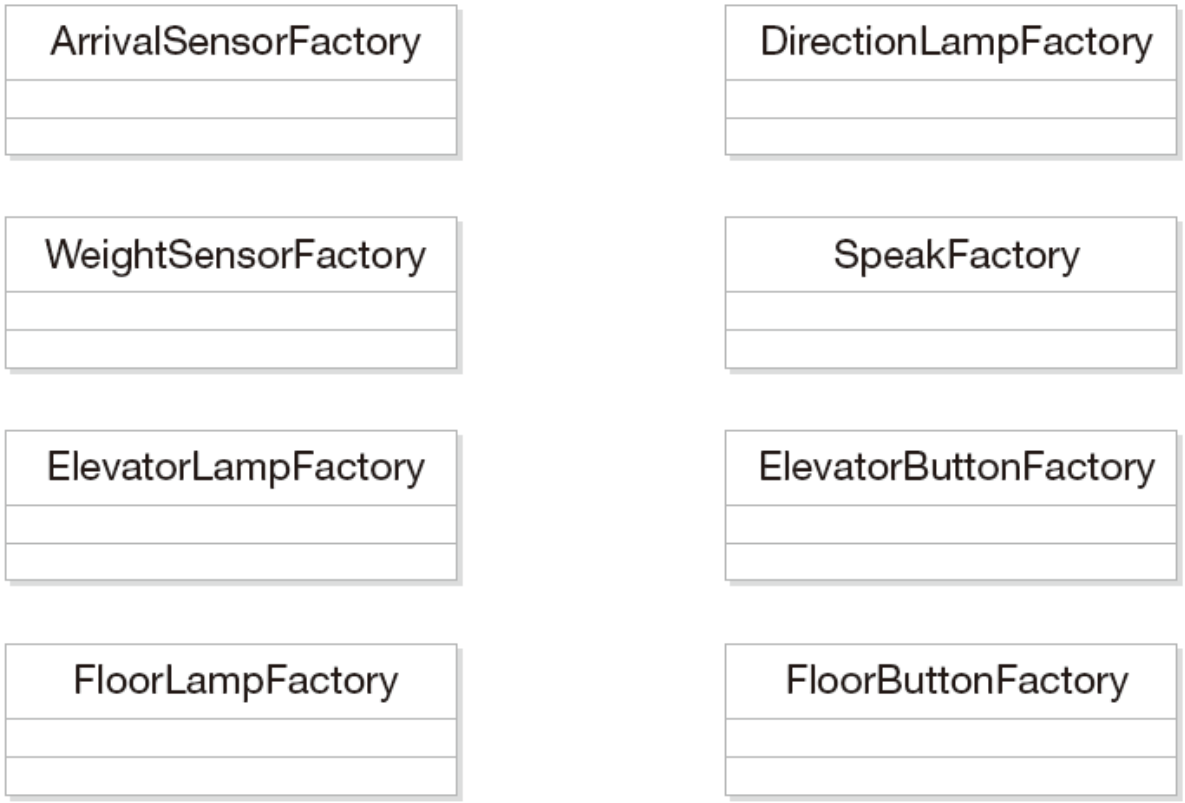
클라이언트 코드

```
public class ElevatorCreator {
    public static Elevator assembleElevator() {
        Elevator elevator = new LgElevator();
        Motor motor = new LgMotor();
        elevator.setMotor(motor);
        Door door = new LgDoor();
        elevator.setDoor(door);
        motor.setDoor(door);
        DirectionLamp lamp = new LgLamp();
        elevator.setLamp(lamp);
        return elevator;
    }

    public static void main(String[] args) {
        Elevator elevator = assembleElevator();
        elevator.move(Direction.UP);
    }
}
```

엘리베이터

❖ 부품 별로 팩토리를 구현해야 함

그림 13-3 각 부품별 Factory 클래스를 추가한 클래스 다이어그램



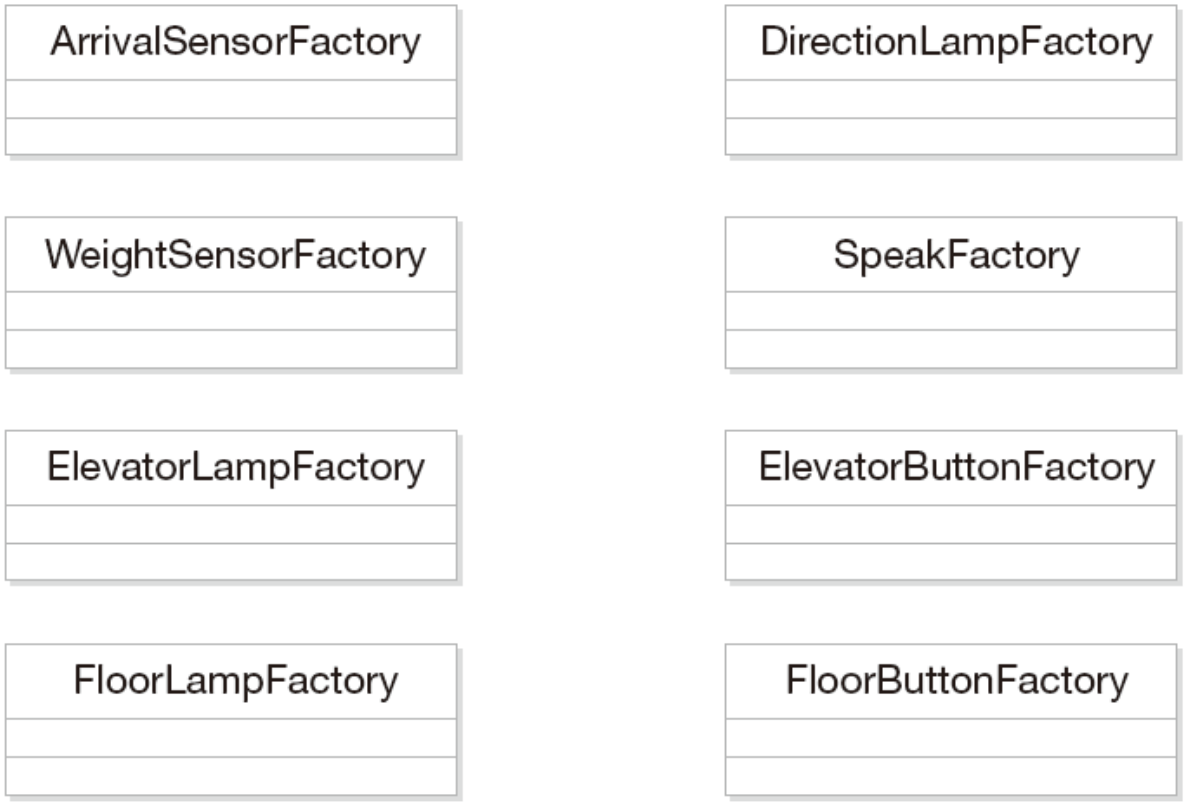
13.2 문제점

- 현재 프로그램은 LG의 부품(LGMotor와 LGDoor)를 사용하고 있다. 만약 다른 제조업체의 부품을 사용해야 한다면? 예를 들어 LG 부품 대신에 현대의 부품(HyundaiMotor와 HyundaiDoor)를 사용해야 한다면?
- 게다가 새로운 제조업체의 부품을 사용해야 한다면? 예를 들어 삼성에서 엘리베이터 부품을 생산하기 시작해서 SamsungMotor와 SamsungDoor 클래스를 사용해야 한다면?

팩토리 메서드 패턴을 이용한 현대 부품 사용

❖ 부품 별로 팩토리를 구현해야 함

그림 13-3 각 부품별 Factory 클래스를 추가한 클래스 다이어그램



팩토리 메서드 패턴을 이용

코드 13-6

```
public class ElevatorCreator {
   public static Elevator assembleElevator(VendorId id){
      Door hyundaiDoor = DoorFactory.createDoor(id) ;
      Motor hyundaiMotor = MotorFactory.createMotor(id) ;
      hyundaiMotor.setDoor(hyundaiDoor) ;
      ArrivalSensor hyundaiArrivalSensor =
         ArrivalSensorFactory.createArrivalSensor (id) ;
      WeightSensor hyundaiWeightSensor =
         WeightSensorFactory.createWeightSensor (id) ;
      ElevatorLamp hyundaiElevatorLamp =
         ElevatorLampFactory.createElevatorLamp (id) ;
      FloorLamp hyundaiFloorLamp = FloorLampFactory.createFloorLamp (id) ;
      DirectionLamp hyundaiDirectionLamp =
         DirectionLampFactory.createDirectionLamp (id) ;
      Speaker hyundaiSpeaker = SpeakerFactory.createSpeaker (id) ;
      ElevatorButton hyundaiElevatorButton =
         ElevatorButtonFactory.createElevatorButton (id) ;
      FloorButton hyundaiFloorButton =
         FloorButtonFactory.createElevatorFloorButton (id) ;
      ...

      }
}
```

새로운 제조 업체의 지원

❖ 각 팩토리에서 새로운 제조 업체 부품을 생성하도록 수정이 필요함

코드 13-7

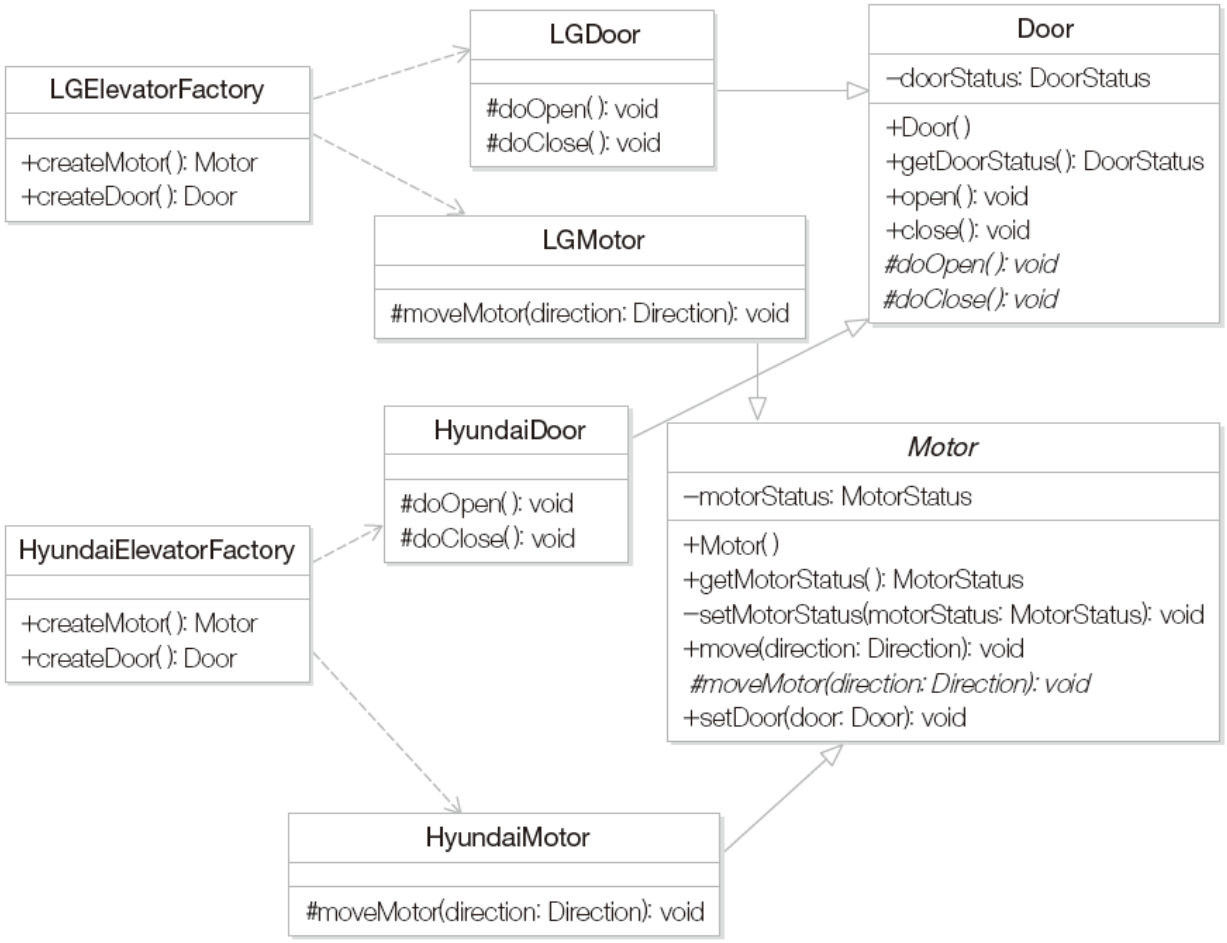
```
public class DoorFactory {
   public static Door createDoor(VendorID vendorID) {
      Door door = null ;
      switch ( vendorID ) {
      case LG : door = new LGDoor() ; break ;
      case HYUNDAI : door = new HyundaiDoor() ; break ;
      case SAMSUNG : door = new SamsungDoor() ; break ;
      }
      return door ;
   }
}
```

삼성 부품을 지원하도록 팩토리 메서드를 수정함 ➔ OCP를 위반함

13.3. 해결책

❖ 부품이 아니라 제조업체 별로 팩토리를 정의함

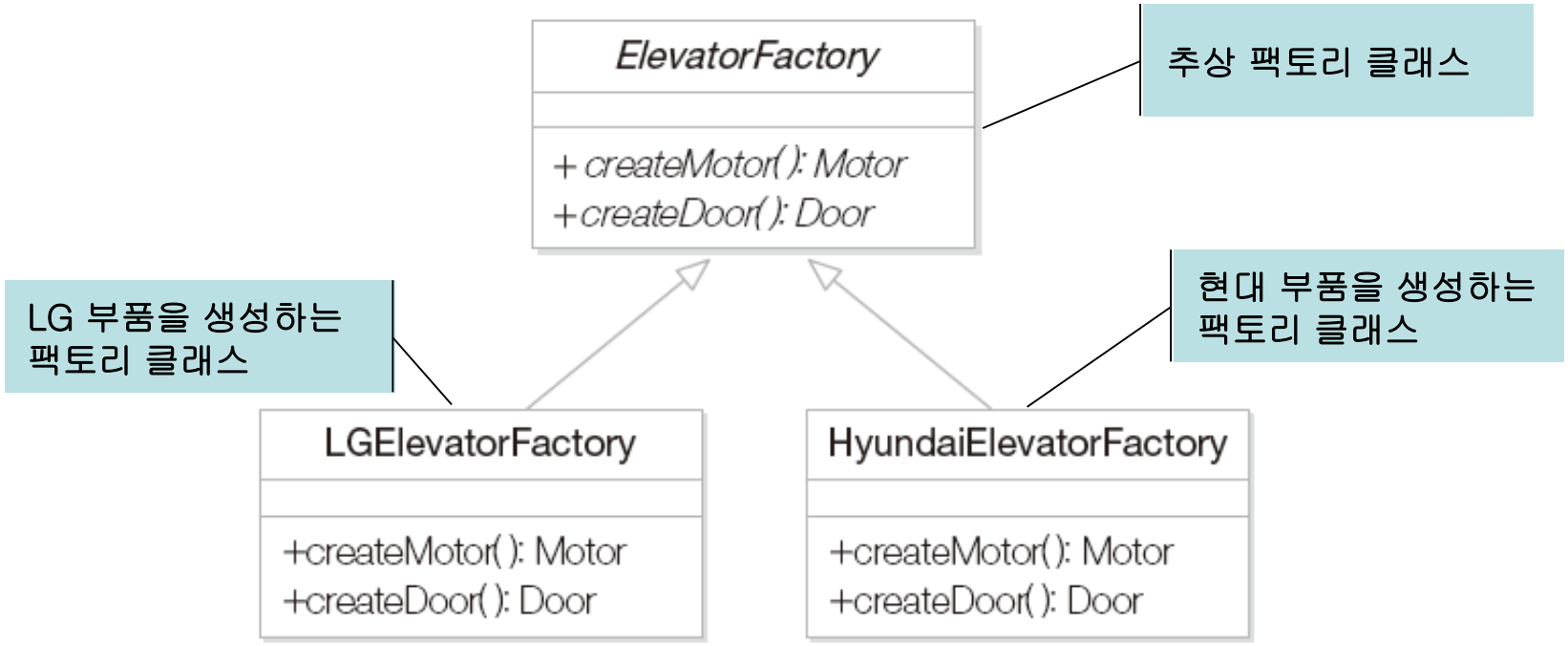
그림 13-4 LGElevatorFactory와 HyundaiElevatorFactory 클래스를 이용한 설계



13.3. 해결책

❖ 제조업체의 팩토리 클래스의 공통 추상 팩토리 클래스를 정의

그림 13-5 LGElevatorFactory와 HyundaiElevatorFactory 클래스의 일반화



13.3. 해결책: 소스 코드

코드 13-8

```
public abstract class ElevatorFactory {
   public abstract Motor createMotor() ;
   public abstract Door createDoor() ;
  public abstract Lamp createLamp() ;

}
```

```
public class LGElevatorFactory extends ElevatorFactory {
   public Motor createMotor() {
      return new LGMotor() ;
   }
   public Door createDoor() {
      return new LGDoor() ;
   }
   public Lamp createLamp() {
      return new LGLamp() ;
   }
}
```

```
public class HyundaiElevatorFactory extends ElevatorFactory {
   public Motor createMotor() {
      return new HyundaiMotor() ;
   }
   public Door createDoor() {
      return new HyundaiDoor() ;
   }
 public Lamp createLamp() {
      return new Hyundaiamp() ;
   }
}
```

Factory 객체 이용: 소스코드

```
public class ElevatorCreator {
    public static Elevator assembleElevator(ElevatorFactory factory) {
        Elevator elevator = factory.createElevator();
        Motor motor = factory.createMotor();
        elevator.setMotor(motor);
        Door door = factory.createDoor();
        elevator.setDoor(door);
        motor.setDoor(door);
        DirectionLamp lamp = factory.createLamp();
        elevator.setLamp(lamp);
        return elevator;
    }

}
```

13.3. 해결책: 소스 코드

코드 13-9

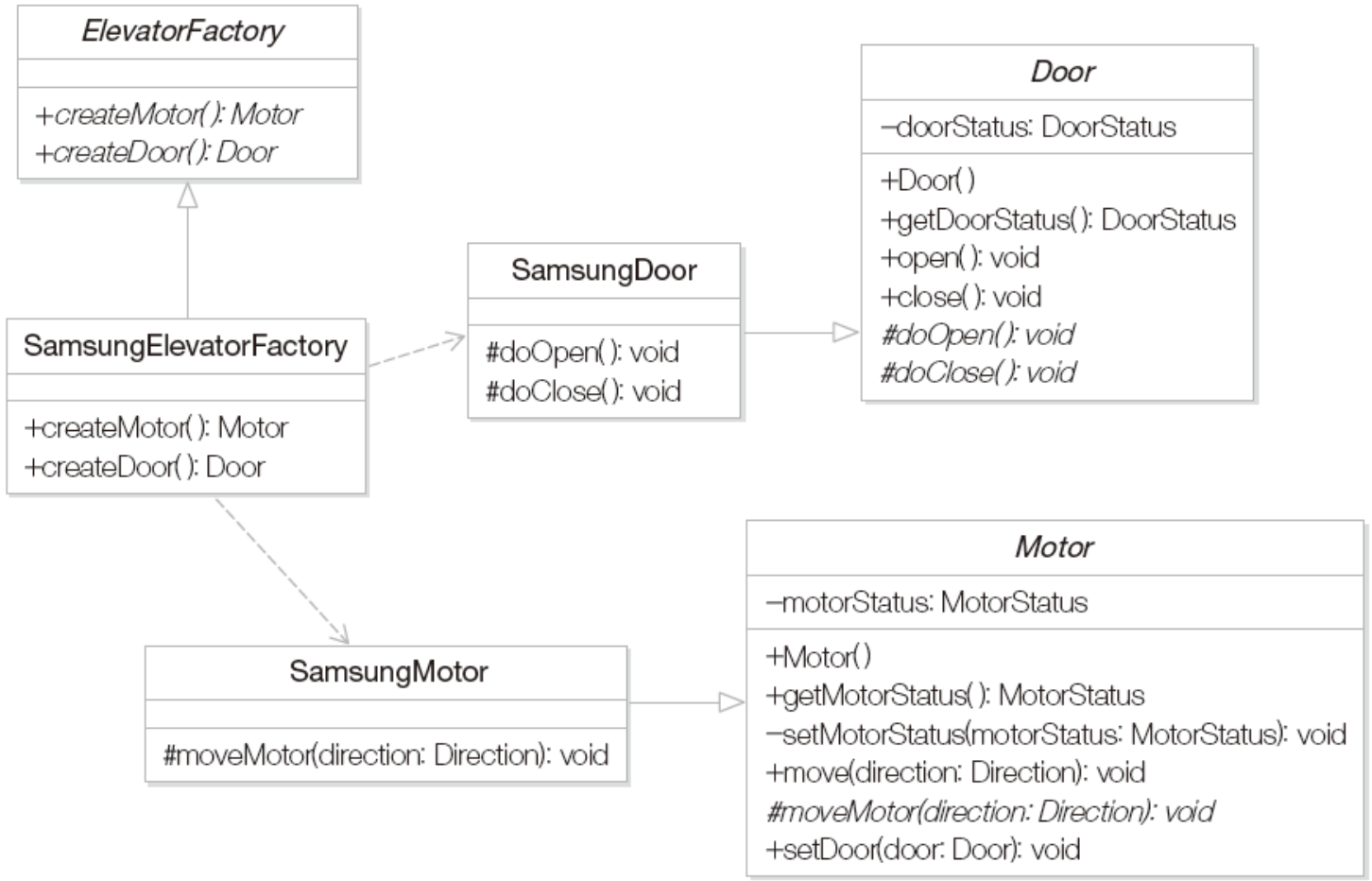
```
public class Client {
   public static void main(String[] args) {
      ElevatorFactory factory = null ;
      String vendorName = args[0] ;
      if ( vendorName.equalsIgnoreCase("LG") )
         factory = new LGElevatorFactory() ;
      else
         factory = new HyundaiElevatorFactory() ;
      Elevator elevator = assembleElevator(factory);

      elevator.move(Direction.UP);
   }
}
```

프로그램 인자	LG	Hyundai
실행 결과	open LG Door close LG Door move LG Motor	open Hyundai Door close Hyundai Door move Hyundai Motor

새로운 제조 업체의 지원

❖ 삼성 부품의 지원

그림 13-6 SamsungFactory 클래스를 이용한 삼성 부품의 객체 생성



소스 코드

코드 13-11

```
public class Client {
    public static void main(String[] args) {
        ElevatorFactory factory = null ;
        String vendorName = args[0] ;
        if ( vendorName.equalsIgnoreCase("LG") )
            factory = new LGElevatorFactory() ;
        else if ( vendorName.equalsIgnoreCase("Samsung") )
            factory = new SamsungElevatorFactory() ;
        else
            factory = new HyundaiElevatorFactory() ;
    }
     Elevator elevator = assembleElevator(factory);

      elevator.move(Direction.UP);

}
```

프로그램 인자	Samsung
실행 결과	open Samsung Door close Samsung Door move Samsung Motor

패턴의 추가적인 적용

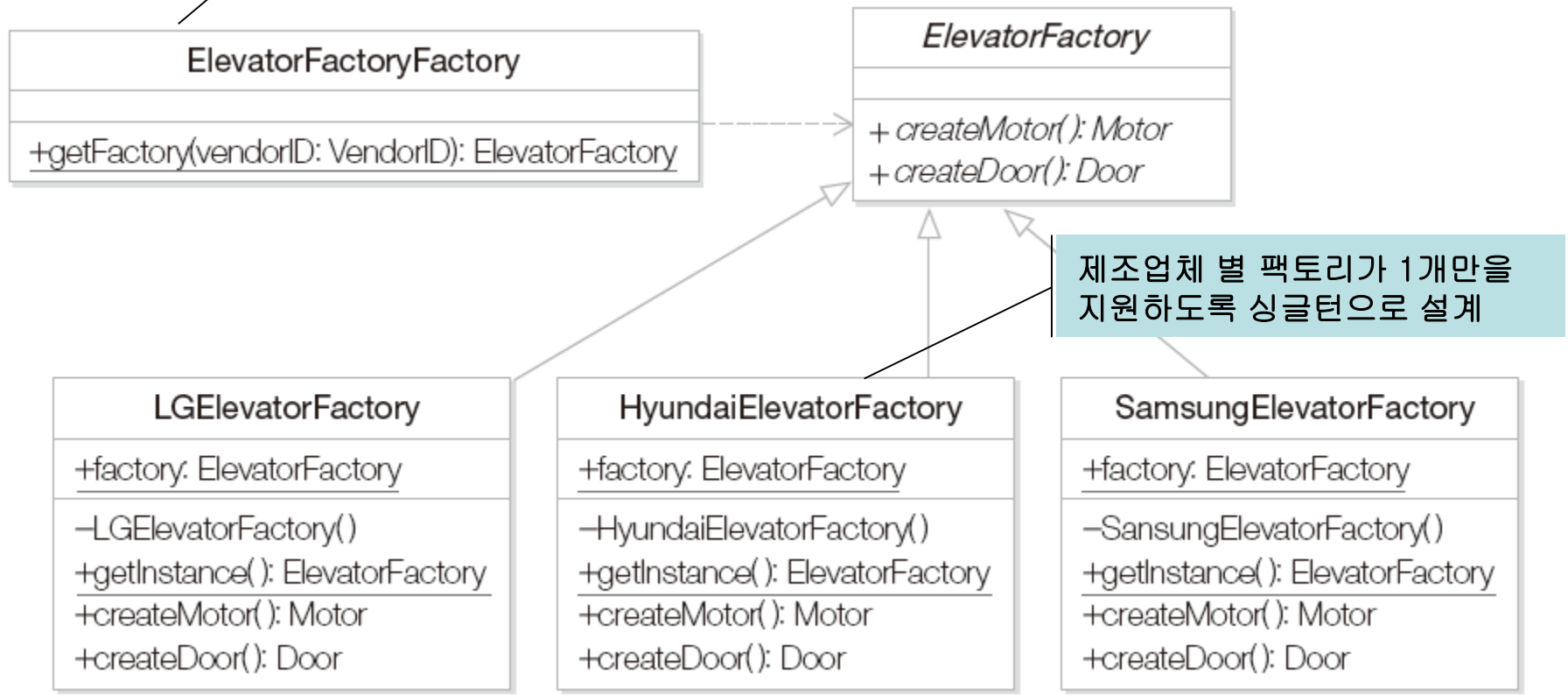
- 제조업체 별로 Factory 클래스를 생성하는 부분을 팩토리 메소드 패턴을 적용하여 설계
- 제조업체 별 팩토리는 1개만 필요하다면 ➔ 싱글턴 패턴을 적용

```
if ( vendorName.equalsIgnoreCase("LG") )
   factory = new LGElevatorFactory() ;
else if ( vendorName.equalsIgnoreCase("Samsung") )
   factory = new SamsungElevatorFactory() ;
else
   factory = new HyundaiElevatorFactory() ;
```

패턴의 추가적인 적용

제조업체 별 팩토리를 생성하는 팩토리 클래스

그림 13-7 팩토리 메서드와 싱글턴 패턴을 적용한 제조 업체별 Factory 클래스 다이어그램



패턴의 추가적인 적용: 소스 코드

코드 13-12

```
public class ElevatorFactoryFactory {
    public static ElevatorFactory getFactory(VendorID vendorID) {  // 팩토리 메서드
        ElevatorFactory factory = null ;
        switch ( vendorID ) {
        case LG: factory = LGElevatorFactory.getInstance() ; break ;
        case HYUNDAI : factory = HyundaiElevatorFactory.getInstance() ; break ;
        case SAMSUNG : factory = SamsungElevatorFactory.getInstance() ; break ;
        }
        return factory ;
    }
}
```

```
public class LGElevatorFactory extends ElevatorFactory {  // 싱글턴을 적용한 LG 팩토리
    private static ElevatorFactory factory ;
    private LGElevatorFactory() {}
    public static ElevatorFactory getInstance() {
        if ( factory == null ) factory = new LGElevatorFactory() ;
        return factory ;
    }
    public Motor createMotor() { return new LGMotor() ; }
    public Door createDoor() { return new LGDoor() ; }
}
```

패턴의 추가적인 적용: 소스 코드

코드 13-13

```
public class Client {
   public static void main(String[] args) {
      String vendorName = args[0] ;
      VendorID vendorID ;
      if ( vendorName.equalsIgnoreCase("LG")) vendorID = VendorID.LG ;
      else if ( vendorName.equalsIgnoreCase("Samsung"))
         vendorID = VendorID.SAMSUNG ;
      else vendorID = VendorID.HYUNDAI ;

      ElevatorFactory factory = ElevatorFactoryFactory.getFactory(vendorID) ;

      Elevator elevator = assembleElevator(factory);
      elevator.move(Direction.UP);
   }
}
```

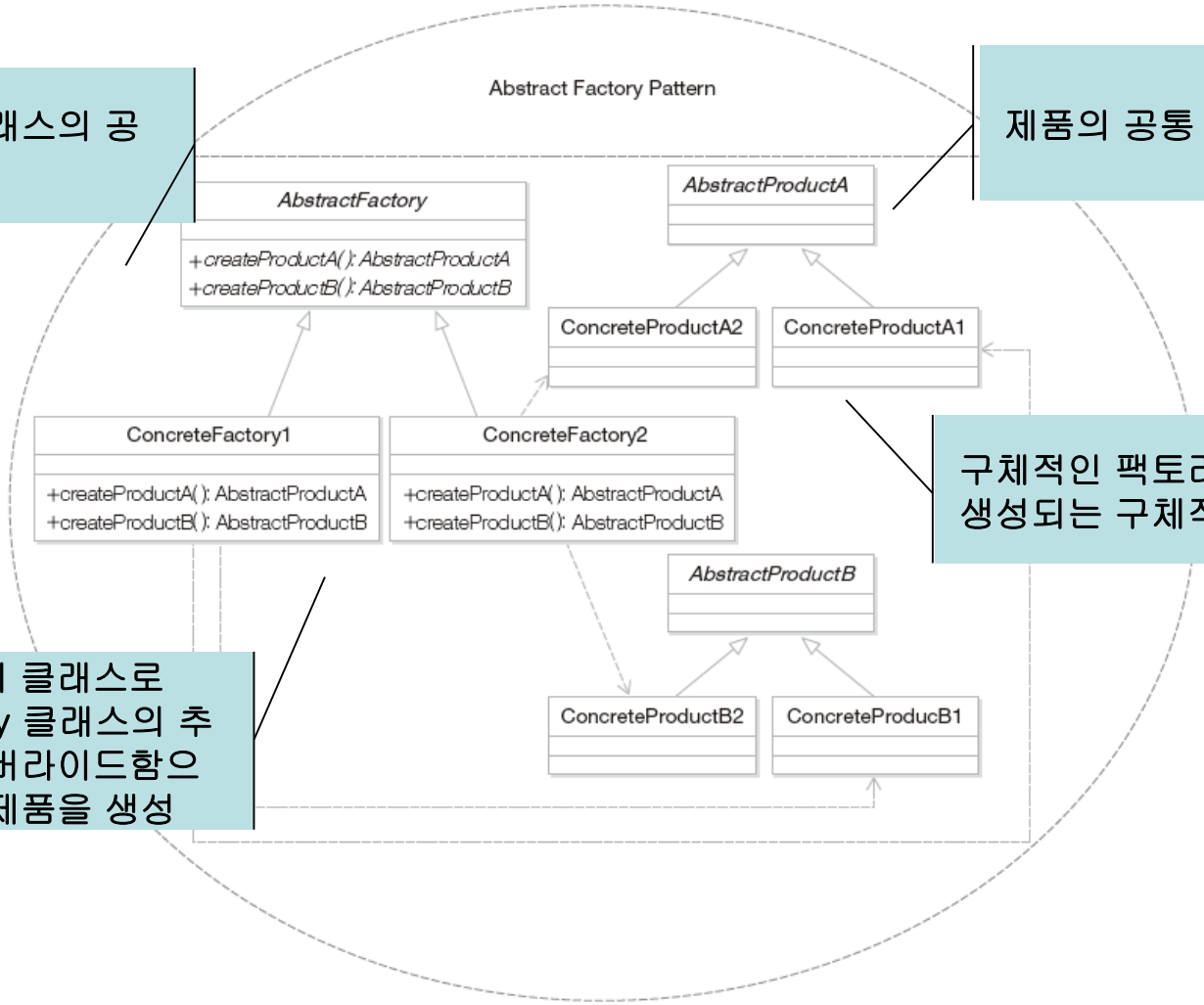
13.4 추상 팩토리 패턴

- 관련성이 있는 여러 종류의 객체를 일관된 방식으로 생성하는 경우에 유용

> 추상 팩토리 패턴은 관련성이 있는 여러 종류의 객체들을 일관된 방식으로 생성할 때 유용하다.

13.4 추상 팩토리 패턴

그림 13-8 추상 팩토리 패턴의 컬레보레이션



실제 팩토리 클래스의 공통 인터페이스

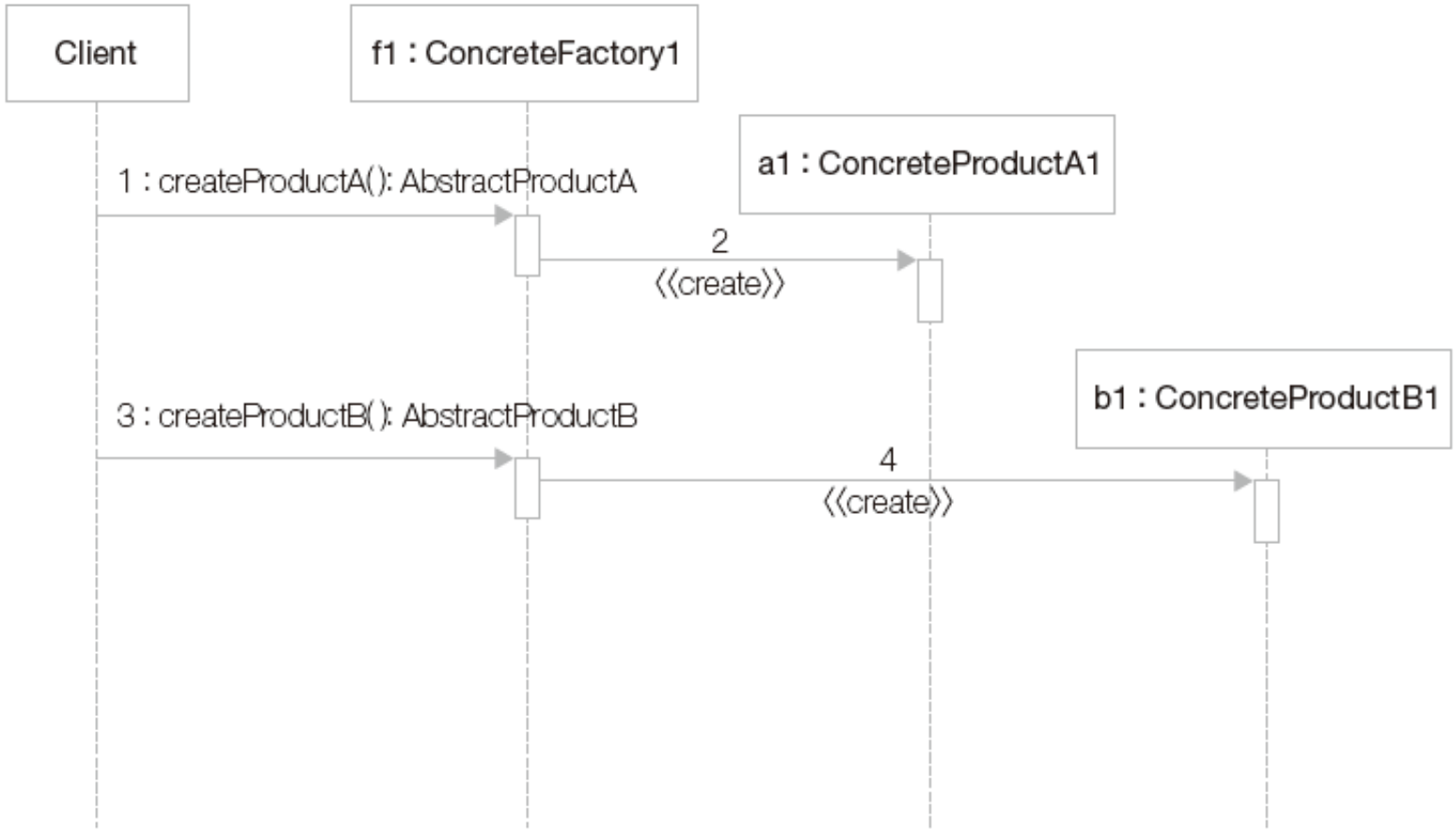
제품의 공통 인터페이스

구체적인 팩토리 클래스에서 생성되는 구체적인 제품

구체적인 팩토리 클래스로 AbstractFactory 클래스의 추상 메서드를 오버라이드함으로써 구체적인 제품을 생성

13.4 추상 팩토리 패턴

그림 13-9 추상 팩토리 패턴의 순차 다이어그램



추상 팩토리 패턴의 적용

그림 13-10 추상 팩토리 패턴을 엘리베이터 부품 업체 예제에 적용한 경우