KB금융그룹 | 국민의 평생 금융파트너

데이터 베이스를
좀더 상위 버전으로 래핑하여 사용할 수 있게끔함

mybatis hibernate 프레임 워크들은 데이터 베이스와 연결하여 동기화, 관리 등을 해줌

이때 팩토리 메서드가 활용된다

2025년 상반기 K-디지털 트레이닝

Factory Method

- 하위 클래스에서 인스턴스를 만든다

[KB] IT's Your Life

KB 국민은행

Factory Method 패턴

- Template Method 패턴을 인스턴스 생성 장면에 적용한 것
  - 인스턴스를 생성하는 공장을 Template Method 패턴으로 구성한 것

- 인스턴스 생성 방법을 상위 클래스에서 결정하되, 구체적인 클래스 이름까지는 결정하지 않음
- 구체적인 살은 모두 하위 클래스에서 정의

→ 인스턴스 생성을 위한 뼈대(프레임워크)와 실제 인스턴스를 생성하는 클래스로 나누어 생각

Factory Method 패턴

예제 프로그램

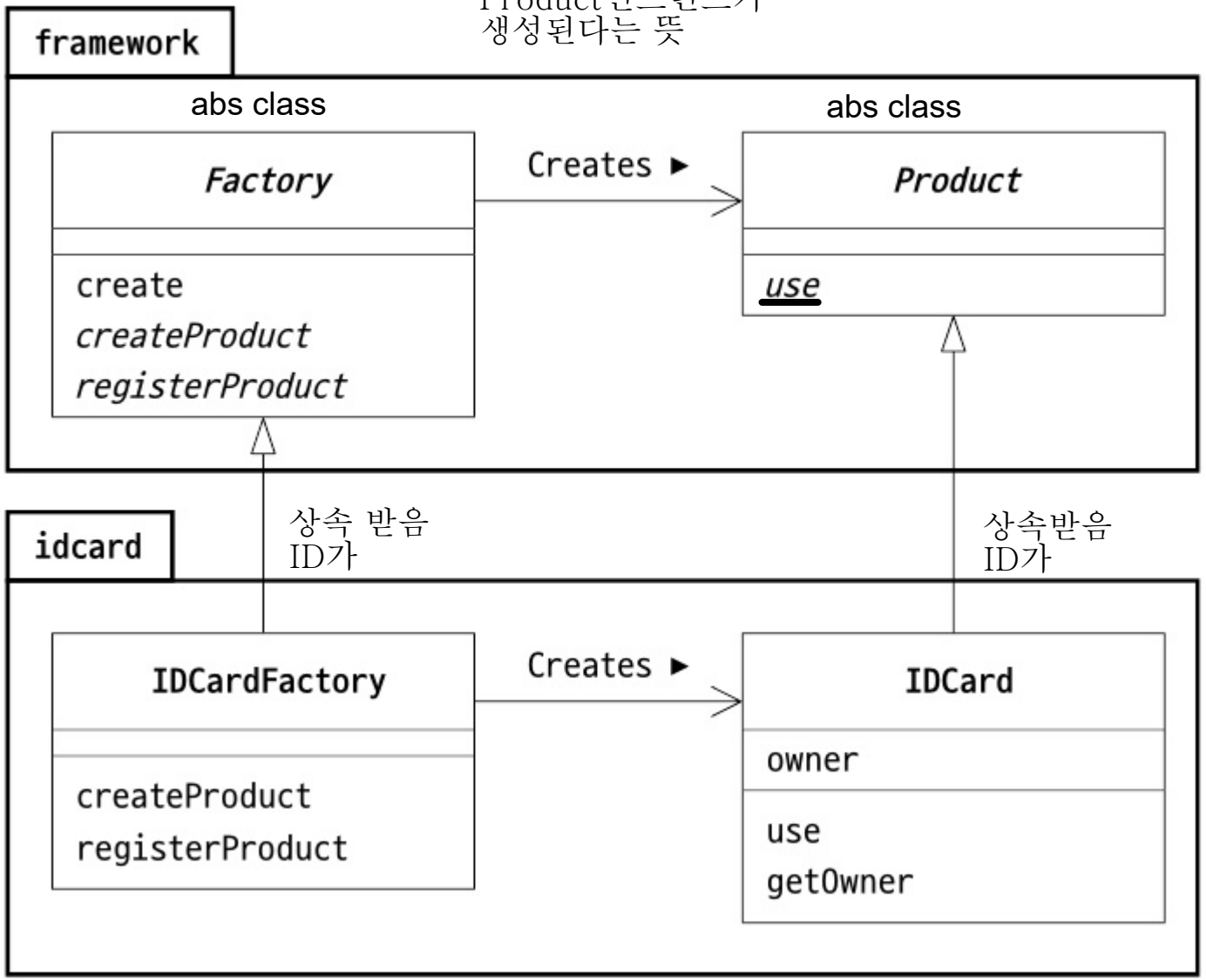
- 신분증 카드(ID 카드)를 만드는 공장
- 프레임워크 역할
  - Product 클래스와 Factory 클래스
- 구체적인 내용 역할
  - IDCard 클래스와 IDCardFactory 클래스

패키지	이름	설명
framework	Product	추상 메소드 use만 정의한 추상 클래스
framework	Factory	메소드 create를 구현한 추상 클래스
idcard	IDCard	메소드 use를 구현한 클래스
idcard	IDCardFactory	메소드 createProduct, registerProduct를 구현한 클래스
이름 없음	Main	동작 테스트용 클래스

Factory Method 패턴

예제 프로그램의 클래스 다이어그램

Factory인스턴스의
create메서드를 통해서
Product인스턴스가
생성된다는 뜻



IDCardFactory인스턴스의
createProduct메서드를 통해
IDCard라는 제품이
생성된다는 뜻 이때
추상 메서드들을 자세히
정의하므로써
생성시 필요한 조치를
자식에서 정함

framework/Product.java

```
public abstract class Product {
    public abstract void use();
}
```

Factory Method 패턴

framework/Factory.java

```
public abstract class Factory {
    public final Product create(String owner) {
        Product p = createProduct(owner);
        registerProduct(p);
        return p;
    }

    protected abstract Product createProduct(String owner);
    protected abstract void registerProduct(Product product);
}
```

new IDCard( ) 한다고 바로 되는게 아니라

어떻게 해당 제품을 만들때 어떤 조치를 취해야하는지를
부모에서 정하지 않고

자식이 정하게끔

추상메서드로 정의함

idcard/IDCard.java

```
public class IDCard extends Product {
    private String owner;

    public IDCard(String owner) {
        System.out.println(owner + "의 카드를 만듭니다.");
        this.owner = owner;
    }

    @Override
    public void use() {
        System.out.println(this + "을 사용합니다.");
    }

    @Override
    public String toString() {
        return "[IDCard:" + owner + "]";
    }
}
```

idcard/IDCardFactory.java

```
public class IDCardFactory extends Factory {
    @Override
    protected Product createProduct(String owner) {
        return new IDCard(owner);
    }

    @Override
    protected void registerProduct(Product product) {
        System.out.println(product + "을 등록했습니다.");
    }
}
```

Factory Method 패턴

Main.java

```
public class Main {
    public static void main(String[] args) {
        Factory factory = new IDCardFactory();
        Product card1 = factory.create("Youngjin Kim");
        Product card2 = factory.create("Heunmin Son");
        Product card3 = factory.create("Kane");
        System.out.println();

        card1.use();
        card2.use();
        card3.use();
    }
}
```

팩토리 하나 생성하고

팩토릴 통해 실제 제품 3개 만들고

실제 제품 사용

정의 할 때 데이터 베이스 관련 활동을 하게끔 정의 할 수 있음.

사용하는 사람은 모르죠.

JPA같은 DB연동 방법을 활용할 때 많이 쓰인다.

해당 방법은 표준으로 지정되어있다.

인터페이스 추상클래스로 이미 정의되어있다.

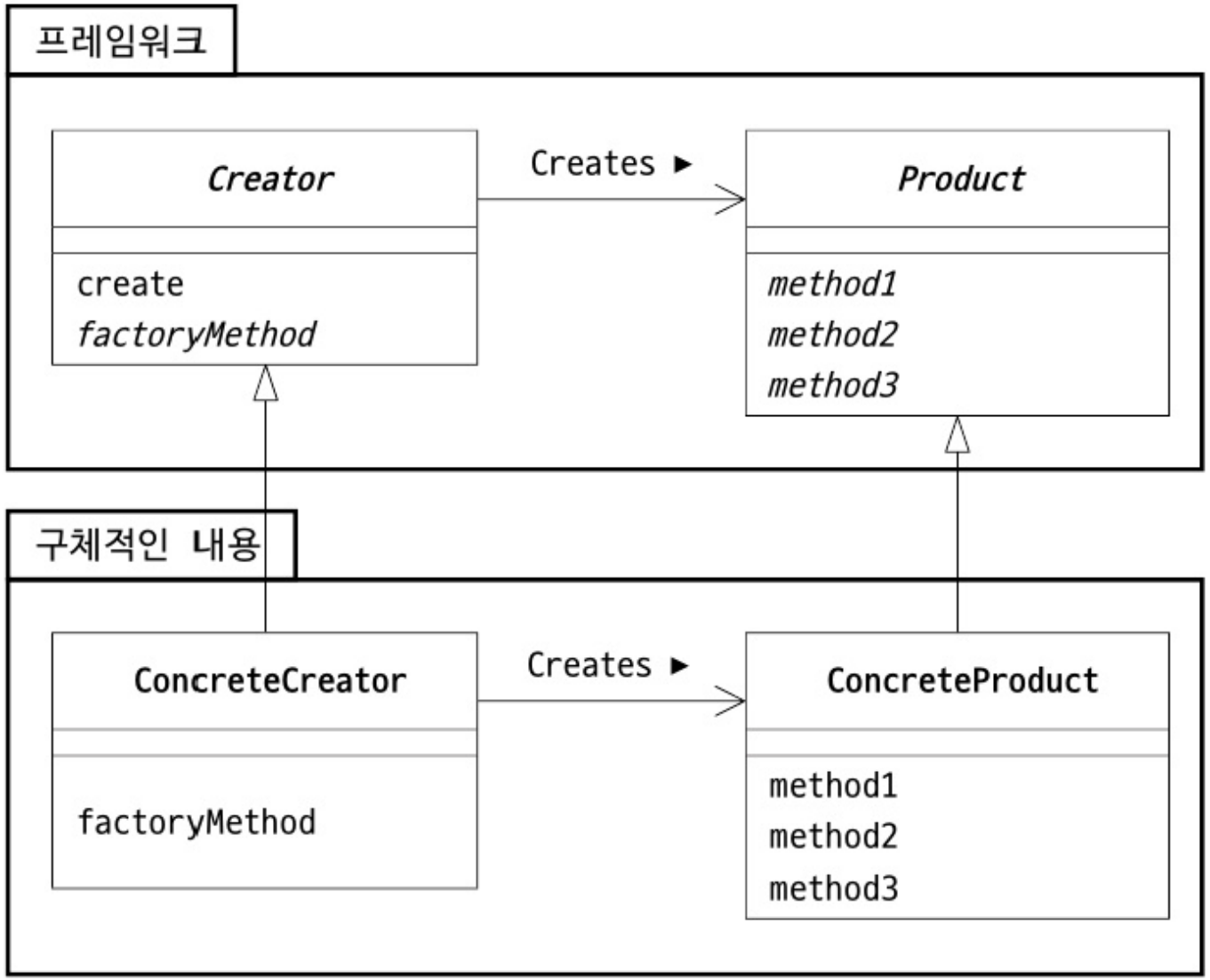
앞에서 봤던 framwork추상크래스들이java, spring등에서 제공하고 product는 회사마다 있다.
우리는 잘 활용하면 된다

```
Youngjin Kim의 카드를 만듭니다.
[IDCard:Youngjin Kim]을 등록했습니다.
Heunmin Son의 카드를 만듭니다.
[IDCard:Heunmin Son]을 등록했습니다.
Kane의 카드를 만듭니다.
[IDCard:Kane]을 등록했습니다.

[IDCard:Youngjin Kim]을 사용합니다.
[IDCard:Heunmin Son]을 사용합니다.
[IDCard:Kane]을 사용합니다.
```

Factory Method 패턴

Factory Method 패턴의 클래스 다이어그램



Factory Method 패턴

Factory Method 패턴의 사용

- 같은 프레임워크를 사용하여 전혀 다른 제품과 공장을 만드는 경우

→ framework 패키지의 내용은 수정하지 않아도 전혀 다른 제품과 공장을 만들 수 있음

사용측은 framework의 내용으로 처리하므로 사용 측의 코드 변경 없음

→ OCP