Factory 패턴

추상팩토리, 팩토리메서드

팩토리를 써야 하는 상황이 언제?

> 👉 생성 패턴의 한 종류, 클래스의 객체 생성 처리를 위임하는 패턴이다.

▼ 생성 패턴이란?

객체지향의 경우 수많은 객체를 필요로하게 되는데, 이때 직접 객체를 생성하고 관계를 설정하면 매우 강력한 결합 관계(= 종속 관계)를 가지게 된다. 이는 객체의 확장과 유지보수를 어렵하게 하는 원인이 되므로, 이 종속 관계를 느슨한 결합으로 변경하는 기법이 생성 패턴이다.

Factory 패턴

강력한 결합 관계	팩토리 패턴
new로 상속받은 클래스를 직접 생성	create 함수에 매개변수를 전달하여 해당하는 조건에 따라 클래스 생성 ⇒ 조건 로직 필수, 외부 매개변수값 필수 (문자열보다는 상수로 정의할 것) <객체 생성의 위임>
유연한 코드 확장 방해, 변경과 수정이 힘듬 → 유지보수에 안좋다!	객체의 생성처리를 동적으로 위임하여 향후 클래스 추가, 변경에 코드를 쉽게 수정 가능

참고 자료 및 예시 : https://yeah.tistory.com/2?category=949516

⇒ 그러나, 이 방법도 여전히 객체 생성에 있어 강력한 결합을 유지하고 있다. (new 키워드)

ex) 결합된 객체의 생성 조건이 수정되거나 확장 시 factory 내부가 수정되야함.

팩토리 메서드 패턴

> 👉 기존 팩토리 기법 + 추상화 기법을 사용하여 패턴을 확장

방식

👉 기존 factory 클래스의 create 메소드를 추상 메소드로 바꾼다!
이때, 1개의 하위 클래스 안에서 매개변수를 통해 조건에 따라 객체 생성을 한다.

<as-is : factory 패턴만 적용한 경우>

```
/* 객체 생성의 위임을 위한 Factory class */
abstract class Factory {
    /* 기존 팩토리 패턴 */
    public User create(String name, String job) {
        switch (job) {
            case "마피아":
                return new Mafia(name);
            case "경찰":/
              return new Police(name);
            default:    //시민
                return new Citizen(name);
        }
    }
}
```

<to-be : factory 메서드 패턴>

```
/* 객체 생성의 위임을 위한 Factory class */
abstract class Factory {
    /* 추상화를 결합한 팩토리 메서드 패턴 */
    public final User create(String name, String job) {
        //하위 클래스로 위임
        return this.createUser(name, job);
    }

    abstract public User createUser(String name, String job);
}
```

추상 클래스로 적용하여 객체의 생산과 사용을 분리한다. (강력 → 느슨한 결합)

따라서, Factory 클래스를 상속받은 하위클래스를 만들어 실재적인 추상 메서드 구현부 작성필요

- createUser() : 하위클래스에서 객체를 생산하도록 위임
- create() : 하위클래스에 위임한 함수를 실행하도록 함.

<Factory 메서드 구현부 클래스>

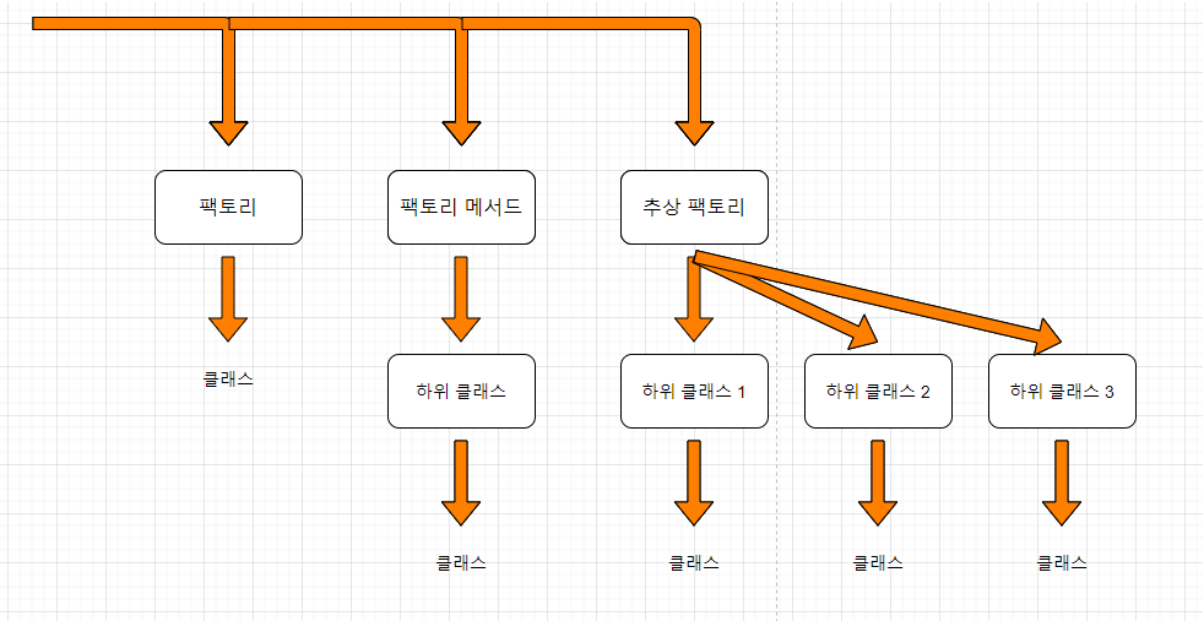
```
/* Factory(추상클래스)를 상속받아 실제 구현부를 담당하는 FactoryUser class */
public class FactoryUser extends Factory {
    @Override
    public User createUser(String name, String job) {
        switch (job) {
            case "마피아":
                return new Mafia(name);
            case "경찰":
                return new Police(name);
            default:    //시민
                return new Citizen(name);
        }
    }
}
```

as-is의 Factory 클래스의 모양을 띄고 있으나, 이제는 생성 조건 수정/추가 시 FactoryUser 코드만 수정하고 Factory 클래스 및 main도 수정이 필요 없음. —— ?????

⇒ 그러나, 이 방식 또한 조건문에 따라 객체 생성을 하기 때문에 다형성을 배제한 방법이 된다.

추상 팩토리 패턴

👉 팩토리 메서드 + 다형성 : 각 객체마다 하위 클래스를 생성하여 원하는 하위 클래스를 결합하도록 하는 방식 (여러개의 하위 팩토리 클래스)



추상 Factory 클래스를 상속받도록 하여 각 특징에 맞는 구현부(하위 클래스)를 생성하도록 한다.

예제 참고 링크 : https://yeah.tistory.com/13?category=949516

https://readystory.tistory.com/119?category=822867

팩토리 메서드 패턴	추상 팩토리 패턴
1개의 하위클래스 내 매개변수를 통해 선택적으로 객체 생성 처리	동일한 처리로직의 하위클래스 결합을 통해 선택적으로 객체 생성 가능
새로운 객체 추가시 조건 추가, 하위 클래스 덩치가 커지기 때문에 유지보수에 좋지 않음. →	새로운 객체 추가시 하위 클래스도 추가 확장시 모든 하위클래스의 수정이 필요할 수도 있음.

https://github.com/jun108059/til/blob/master/docs/SoftwareEngineering/03.Factory-Method-Pattern.md