Factory Pattern

팩토리 메서드와 추상 팩토리

목차

돌아보는 Async cafe

커피 주세요!
라떼 주세요!
프라푸치노 주세요!

> 참고
> 이 내용은 Async / Sync 와 관련이 없습니다.

```
class ASyncCafe {

    orderDrink(type) {
        let drink;

        if(type === '아메리카노') {
            drink = new Americano();
        } else if (type === '카페라떼') {
            drink = new CafeLatte();
        } else if (type === '프라푸치노') {
            drink = new Frappuccino();
        }

        drink.prepare();
        drink.putCup();

        return drink;
    }
}
```

시간은 흘러 `아샷추 (아이스티에 샷추가)`가 **SNS**를 뜨겁게 달구고....

```
    if(type === '아메리카노') {
        drink = new Americano();
    } else if (type === '카페라떼') {
        drink = new CafeLatte();
    } else if (type === '프라푸치노') {
        drink = new Frappuccino();

    } else if (type === '아샷추') {
        drink = new IceTeaAddShot();        // NEW!! ✨
    }
```

위의 코드는 개발자 정신 건강에 해롭다.

새로운 타입이 추가되거나,
삭제할 일이 있으면 찾아가서 하나 하나 고쳐줘야함

객체를 직접 생성해서 사용하면 무엇이 문제일까

> new 는 `구체 클래스에 대한 인스턴스` 이다.

추상화가 아닌 특정 구현에 의존하기 때문에 결합도가 높아진다.
뿐만 아니라 생성자의 매개변수까지 알아야한다

그럼 객체 생성을 캡슐화 해보자

1 Simple Factory

```
createCoffee(type) {
    let drink;
    if(type === '아메리카노') {
        drink = new Americano();
    } else if (type === '카페라떼') {
        drink = new CafeLatte();
    } else if (type === '프라푸치노') {
        drink = new Frappuccino();
    } else if (type === '아샷추') {
        drink = new IceTeaAddShot();
    }

    return drink;
}

//이제 이 아이는 음료 추가와 무관하게 변경에 자유롭게 된다
orderCoffee(type) {
    drink = createCoffee(type);
    drink.prepare();
    drink.putCup();
}
```

그게 그거 같은데.. 😯
객체를 생성하기 위해 `new` 는 피할 수 없다

하지만 이렇게나마 생성과 사용을 분리하면,
객체 내에 drink를 사용하는 다른 메서드는
객체가 필요하면 `createCoffee()` 를 이용해 사용할 수 있다.

⚒️ 팩토리 메서드 편

Async Cafe는 개발자 사이에 큰 인기를 얻어 분점이 생겼다. 💵

하지만 Async Cafe의 ~~아메리카노는 산미가 가득했고~~,
~~카페라떼는 우유로 만들어 유당불내증인 사람들이 먹을 수 없었다~~.

돈 욕심이 많았던 사장은 더 많은 사람들에게 음료를 팔기위해 **산미가 없는 아메리카노**와
두유 카페라떼를 만들어 판매하는 **Sync Cafe**를 만들기로 했다.

```
class Cafe {
    createCoffee(type) {
    }

    orderCoffee(type) {
        drink = createCoffee(type);
        drink.prepare();
        drink.putCup();
    }
}
```

```
class ASyncCafe extends Cafe {
    createCoffee(type) {
        let drink;

        if(type === '아메리카노') {
            drink = new Americano();
        } else if (type === '카페라떼') {
            drink = new CafeLatte();
        } else if (type === '프라푸치노') {
            drink = new Frappuccino();
        }

        return drink;
    }
}
```

```
class SyncCafe extends Cafe {
    createCoffee(type) {
        let drink;

        if(type === '아메리카노') {
            drink = new 산미가적은Americano();
        } else if (type === '카페라떼') {
            drink = new 두유CafeLatte();
        } else if (type === '프라푸치노') {
            drink = new Frappuccino();
        }

        return drink;
    }
}
```

그래서 팩토리메서드는

createCoffee()를 추상화시키고 -> **생성 메서드**를 추상 메서드로 만들어
서브 클래스에서 이를 구현한다. -> 직접 구현하는 네가 너의 입맛대로 만들어

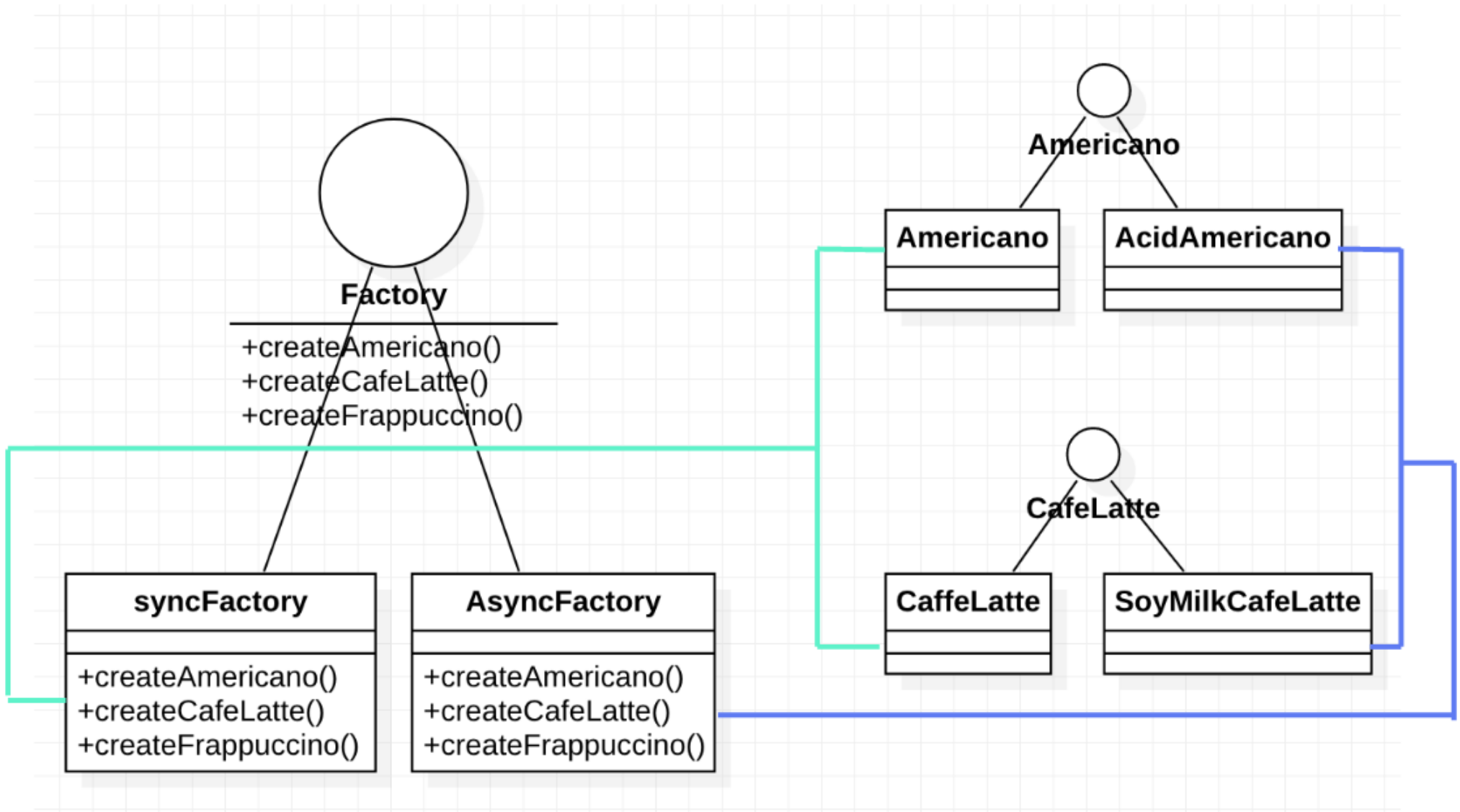
상속

구체적으로 어떤 클래스의 객체를 생성할지 미리 알지 못할 경우에 유용하다.
그래서 어떤 객체를 생성할 것인지에 상관없이 개발이 가능

🏭 추상 팩토리 편

시간이 흘러 5차 산업혁명이 도래하고 무선 샤워기가 나올만큼 과학이 발전하는데...
이에 바리스타를 다 해고하고 버튼을 누르면 알아서 음료를 제조하는
`DrinkFactory`를 해 사장은 인건비를 절약했다.

그런데 Async Cafe와 Sync Cafe는 서로 음료가 약간씩 다르다.


Factory
+createAmericano()
+createCafeLatte()
+createFrappuccino()
syncFactory
+createAmericano()
+createCafeLatte()
+createFrappuccino()
AsyncFactory
+createAmericano()
+createCafeLatte()
+createFrappuccino()
Americano
Americano
AcidAmericano
CafeLatte
CaffeLatte
SoyMilkCafeLatte

DrinkFactory

```
class DrinkFactory {

    createAmericano() {

    }
    createCafeLatte() {

    }
    createFrappuccino() {

    }
}
```

```
class ASyncDrinkFactory extends DrinkFactory {

    createAmericano() {
        return new Americano();
    }
    createCafeLatte() {
        return new CaffeLatte();
    }
    //...
}
```

```
class SyncDrinkFactory extends DrinkFactory {

    createAmericano() {
        return new 산미가적은Americano();
    }
    createCafeLatte() {
        return new 두유CafeLatte();
    }
    //...
}
```

```
class Cafe {
    DrinkFactory;
    construnctor(DrinkFactory) {
        this.DrinkFactory = DrinkFactory;
    }

    createCoffee(type) {
        let drink;

        if(type === '아메리카노') {
            drink = DrinkFactory.createAmericano();
        } else if (type === '카페라떼') {
            drink = DrinkFactory.createCafeLatte();
        } else if (type === '프라푸치노') {
            drink = DrinkFactory.createFrappuccino();
        }

        return drink;
    }

    orderCoffee(type) {
        drink = createCoffee(type);
        drink.prepare();
        drink.putCup();
    }
}
```

그래서 추상 팩토리는

공통 테마를 가지는 친구들을 생성하는 인터페이스를 제공한다
친구들이 어떻게 만들어지는지는 서브클래스에서 정의된다.

인터페이스를 구현할 때 구체적인 객체를 구성(composition)한다.

합성

감사합니다 :)