**팩토리 패턴**

**1. 모든 팩토리 패턴에서는 객체 생성을 캡슐화 한다.**

**2. 팩토리 메서드 패턴과 추상 팩토리 패턴이 있다.**

**3 .팩토리 메서드 패턴 : 객체를 생성하기 위한 인터페이스를 정의하는데, 어떤 클래스의 인스턴스를 만들지는 서브클래스에서 결정한다.**

**추상 팩토리 패턴 : 인터페이스를 이용하여 서로 연관된, 또는 의존하는 객체를 구상 클래스를 지정하지 않고도 생성할 수 있다. 추상 팩토리 패턴에는 팩토리 메서드 패턴이 포함될 수 있다.**

**디자인원칙 중 ‘추상화된 것에 의존하도록 만들어라. 구상클래스에 의존하지 않도록 만든다.’에 기인한 패턴.**

**팩토리 패턴의 핵심은 클래스의 인스턴스를 만드는 것을 서브클래스에서 결정하도록 한다는 것입니다. 즉, new 키워드를 사용하는 부분을 서브클래스에 위임함으로 써 객체 생성을 캡술화하고 구상클래스에 대한 의존성이 줄어든다는 이점을 얻을 수 있습니다.**

**특히 구상클래스에 대한 의존성이 줄어드는 것은 의존성 뒤집기 원칙(Dependency Inversion Principle : DI)에 기인하는데, DI는 자바진영에서 널리 쓰이고 있는 Spring 프레임워크의 핵심 개념 중 하나입니다. 싱글톤패턴과 더불어 가장 유명하고 널리 쓰이는 디자인패턴 중 하나라고 할 수 있겠네요.**

**Abstract Factory Pattern 개요**

**•추상 팩토리 패턴에서는 서로 호환성 있는 개체들을 생성하는 부분만 담당하는 개체를 제공하는 것입니다. 이처럼 호환성 있는 개체를 생성하는 개체를 팩토리 개체라 부릅니다. 하나의 팩토리 개체를 통해 생성되는 개체들은 상호 호환성을 보장받기 때문에 사용자는 호환성에 대해 고민을 할 필요가 없게 됩니다.**

**•**

**•팩토리 형식에서는 호환성 있는 개체를 생성하는 메서드를 표준화된 인터페이스에 약속합니다. 이를 위해서 팩토리 개체를 통해 생성되는 개체들도 일반화합니다. 이로써 사용자는 자신이 사용해야 할 목적과 환경에 맞는 팩토리 개체가 무엇인지만 판단하면 됩니다.**

**Abstract factory와 factory method의 차이**

**•Factory 클래스에서 객체에 대한 생성을 지원하는 범위**

**팩토리 메소드 패턴**

**한 팩토리당 한 종류 ( create 메서드가 Factory 클래스에 1개)**

**한 개의 메서드로 여러 개의 객체를 만듦**

**\*추상 팩토리 패턴**

**한 팩토리에서 서로 연관된 여러 종류 모두 지원( create() 메서드가 팩토리 클래스에 여러 개) 구상 클래스에 의존하지 않고 여러 개의 관련된 객체를 하나의 팩토리로 묶음**

**\*포커스**

**팩토리 메서드 패턴메서드(Factory Method) 레벨에서 포커스를 맞춤 라이언트의 ConcreteProduct 인스턴스 생성 및 구성에 대한 책임을 덜어줌**

**추상 팩토리 패턴**

**클래스(Abstract Factory) 레벨에서 포커스를 맞춤. 각 Product들이 다른 클래스와 함께 사용될 때의 제약사항을 강제할 수 있다. 단, 새로운 ConcreteFactory를 추가할 때 많은 작업이 필요하다.**

**\*메서드와 오브젝트**

**팩토리 메서드는 단일 Method이다. 추상 팩토리는 Object이다. (팩토리 오브젝트 생성)**

**\*Inhritance(상속), Composition(구성)**

**팩토리 메서드 패턴 상속을 사용하여 객체의 인스턴스 생성에 대해서는 서브클래스에 의존 ! 지역 레퍼런스 없이 바로 하위 클래스의 함수를 호출하여 객체를 만듦**

**\*추상 팩토리 패턴**

**지역 레퍼런스를 두어 , 외부로부터 Factory 객체를 DI 받아서 위임한다고 한다.**