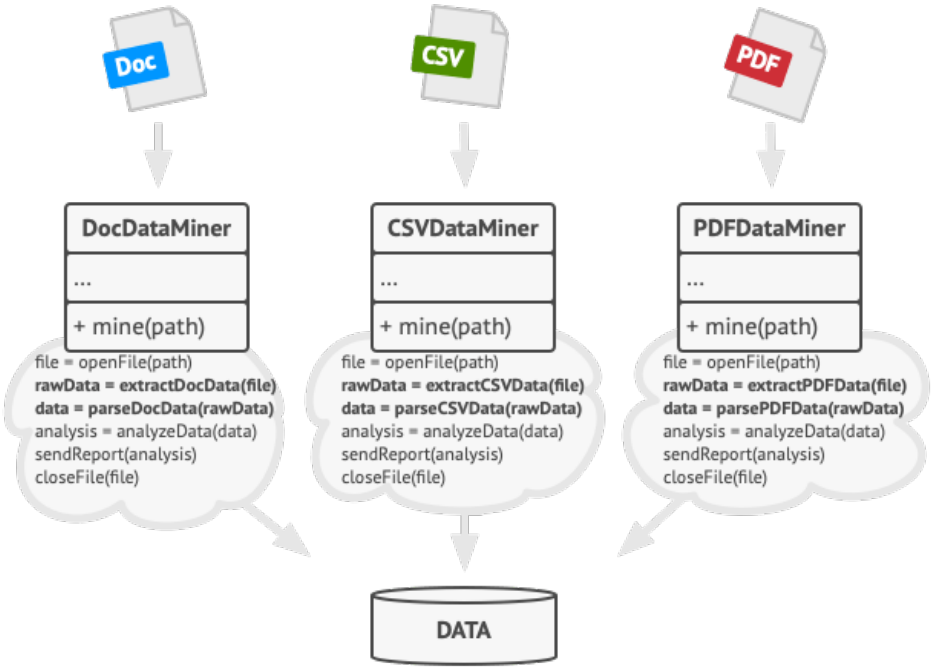
템플릿 메소드 패턴 (Template Method Pattern)



템플릿 메소드 패턴의 개념과 역할

템플릿 메소드 패턴은 객체 지향 프로그래밍에서 쓰이는 디자인 패턴 중 하나입니다. 이 패턴은 추상 클래스에 템플릿 메소드를 정의하고, 이 메소드 내에서 일련의 작업 단계를 정의합니다. 그리고 이 작업 단계 중 일부는 추상 메소드 형태로 선언합니다. 이 추상 메소드는 서브클래스에서 구현됩니다. 이렇게 함으로써, 서브클래스에서는 알고리즘의 일부분을 변경할 수 있게 되는데, 추상 클래스에서 정의된 나머지 작업 단계는 변경되지 않기 때문입니다.

즉, 템플릿 메소드 패턴은 공통적인 알고리즘의 일부분을 추상 클래스에서 정의하고, 그 일부분은 추상 메소드로 정의하여 서브클래스에서 구현할 수 있도록 합니다. 이렇게 함으로써, 공통적인 알고리즘을 갖는 클래스들을 쉽게 구현할 수 있게 됩니다. 이러한 패턴을 사용함으로써 코드의 재사용성과 유지보수성이 향상되는 효과가 있습니다.

템플릿 메소드 패턴의 구성 요소

추상 클래스(Abstract Class)

- 템플릿 메소드를 가지는 클래스
- 추상 메소드(Abstract Method) 또는 구현된 메소드가 포함될 수 있음
- 템플릿 메소드에서 사용되는 메소드 중 하나 이상이 추상 메소드로 선언되면, 서브 클래스에서 반드시 이를 구현해야 함
- 서브 클래스에서 공통적으로 사용되는 메소드는 구현된 메소드로 제공함

구체적인 서브 클래스(Concrete Subclass)

- 추상 클래스(Abstract Class)를 상속받아 구현한 클래스
- 추상 클래스(Abstract Class)에서 선언된 추상 메소드(Abstract Method)를 구현함

- 서브 클래스에서는 추상 클래스(Abstract Class)에서 정의한 템플릿 메소드(Template Method)를 그대로 상속받아 사용할 수 있음

템플릿 메소드 패턴의 동작 방식

1. 추상 클래스(Abstract Class) 생성: 알고리즘을 수행하는 메소드를 가진 추상 클래스를 생성합니다.
2. 추상 메소드(Abstract Method) 생성: 추상 클래스에서 수행할 알고리즘 중 일부를 추상 메소드로 선언합니다.
3. 구체적인 클래스(Concrete Class) 생성: 추상 클래스를 상속받아 구체적인 클래스를 생성합니다. 이 클래스는 추상 메소드를 구현해야합니다.
4. 템플릿 메소드(Template Method) 생성: 추상 클래스에서 알고리즘을 수행할 때, 추상 메소드를 호출하도록 하고, 이를 템플릿 메소드로 생성합니다.
5. 템플릿 메소드 호출: 구체적인 클래스에서 생성된 객체를 통해 템플릿 메소드를 호출합니다. 이 때, 구체적인 클래스에서 구현된 추상 메소드가 호출되어 알고리즘이 수행됩니다.

템플릿 메소드 패턴과 전략 패턴의 차이점

템플릿 메소드 패턴과 전략 패턴은 모두 객체지향 디자인 패턴 중 하나로, 각각의 패턴은 서로 다른 목적을 가지고 있다는 점에서 차이가 있습니다.

템플릿 메소드 패턴은 상위 클래스에서 알고리즘의 구조를 정의하고, 하위 클래스에서 알고리즘의 구체적인 내용을 구현하는 패턴입니다. 즉, 공통적인 로직을 상위 클래스에서 처리하고, 상위 클래스에서 추상 메소드를 정의하고 하위 클래스에서 이를 구현함으로써 다양한 알고리즘을 지원합니다.

반면에 전략 패턴은 동일한 작업을 수행하는 여러 알고리즘을 정의하고, 이를 하나의 추상적인 인터페이스를 통해 접근할 수 있도록 합니다. 즉, 하나의 문제를 해결하는 다양한 전략을 정의하고, 이를 선택할 수 있는 유연성을 제공합니다.

따라서, 템플릿 메소드 패턴은 알고리즘의 구조를 일관성 있게 유지하면서 구현을 다양화하는 데에 사용되고, 전략 패턴은 다양한 전략 중 하나를 선택할 수 있도록 하는 데에 사용됩니다.

템플릿 메소드 패턴의 장단점

장점

- 중복된 코드를 제거하고 일반화된 추상 클래스에 공통 기능을 구현하므로 코드의 재사용성이 높아집니다.
- 알고리즘의 구조를 변경하지 않고도 일부 단계의 구현만 변경할 수 있으므로 유연성이 높아집니다.
- 알고리즘의 골격을 상위 클래스에 두므로 하위 클래스에서 구현해야 할 메소드를 강제할 수 있어 코드의 일관성과 안정성이 높아집니다.

단점

- 구현해야 할 메소드의 수가 많을수록 상위 클래스의 구현이 복잡해질 수 있습니다.
- 일반화된 추상 클래스가 추가되므로 전체적인 코드의 복잡도가 증가할 수 있습니다.

템플릿 메소드 패턴의 구현 방법과 주의사항

구현 방법

템플릿 메소드 패턴은 다음과 같은 구현 방법을 따릅니다.

1. 추상 클래스(Abstract Class)를 정의합니다.
    - 추상 클래스는 여러 개의 메소드를 가지고 있습니다.
    - 추상 클래스는 추상 메소드(Abstract Method)를 포함하고 있습니다.
    - 추상 메소드는 하위 클래스에서 구현해야 합니다.
2. 추상 클래스를 상속받는 구체적인 하위 클래스(Concrete Class)를 정의합니다.
    - 추상 클래스에서 정의한 추상 메소드를 구현합니다.
    - 추상 클래스에서 정의한 일련의 메소드를 오버라이딩할 수 있습니다.
    - 오버라이딩된 메소드는 하위 클래스에서 재정의됩니다.
3. 추상 클래스를 인스턴스화합니다.
    - 하위 클래스에서 오버라이딩된 메소드가 호출됩니다.
    - 하위 클래스에서 오버라이딩되지 않은 메소드는 추상 클래스에서 정의된 메소드를 호출합니다.

주의사항

템플릿 메소드 패턴을 구현할 때 주의해야 할 사항들은 다음과 같습니다.

1. 추상 클래스는 하위 클래스에서 재정의될 수 있는 일련의 메소드를 포함하고 있어야 합니다.
2. 추상 클래스에서는 하위 클래스에서 오버라이딩될 수 있는 메소드와 그렇지 않은 메소드를 모두 포함하고 있어야 합니다.
3. 추상 클래스에서는 하위 클래스에서 오버라이딩할 수 없는 final 메소드도 포함될 수 있습니다.
4. 추상 클래스에서는 하위 클래스에서 재정의할 수 있는 protected 메소드를 사용할 수 있습니다.

템플릿 메소드 패턴을 구현할 때는 상속을 사용해야 합니다. 따라서 하위 클래스에서 추상 메소드를 재정의하도록 유도할 수 있습니다. 하지만 상속은 유연하지 않고 복잡한 계층 구조를 만들 수 있습니다. 따라서 템플릿 메소드 패턴을 구현할 때는 상속 대신 더 유연한 디자인 패턴을 사용하는 것이 좋을 수도 있습니다.