참고자료

- https://refactoring.guru/design-patterns/template-method

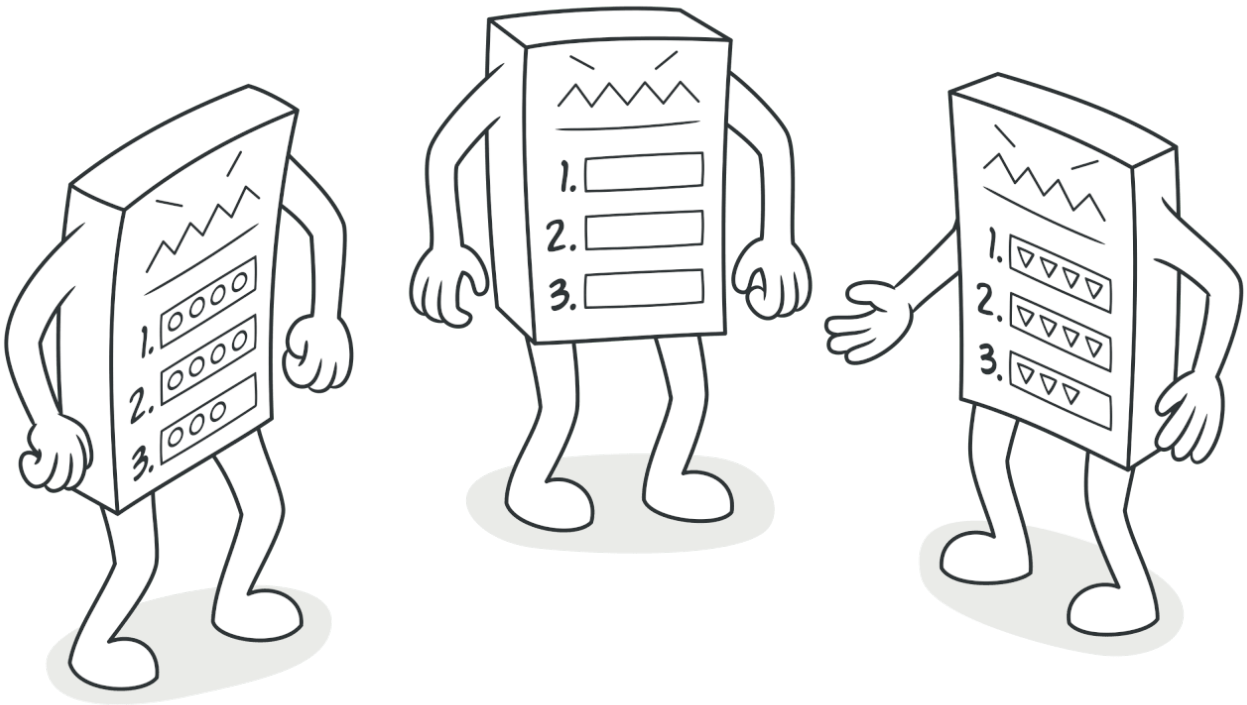
Intent

분류

- behavioral design pattern
  - behavioral class pattern으로, 클래스간 **행동(behavioral)을 분산**하기 위해 **상속**을 이용하는 패턴

정의

- 알고리즘의 뼈대(skeleton)를 super 클래스에 정의하고, subclass는 super클래스의 뼈대를 바꾸지 않은 채 특정한 알고리즘을 나타내는 method를 오버라이딩하는 패턴



Problem

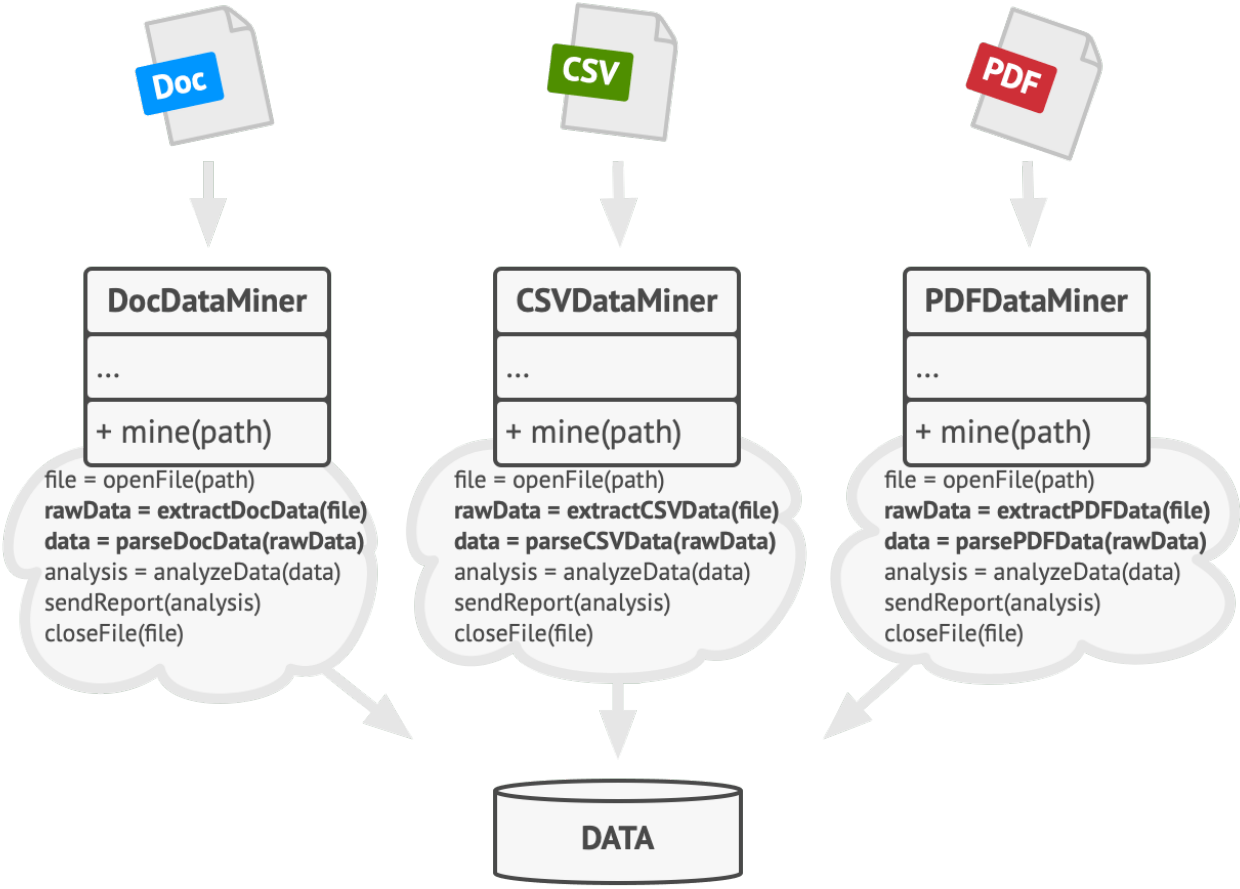
상황

- 데이터 마이닝 앱을 생각해봅시다.
- 이 앱은 기업 문서를 분석합니다.
- 유저가 데이터를 넣어줍니다.
- 그러면 통일된 형식으로 파싱하고 이를 분석하여 의미있는 데이터를 추출합니다.

문제

input type 추가 지원

처음에는 Doc타입 문서만 데이터 마이닝을 하였지만 시간이 흘러 csv파일, pdf파일도 지원하게 되었습니다.



문제 인지

1. 세가지 클래스는 유사한 코드를 가지고 있습니다.
    - 파일을 읽고 -> 데이터를 추출하고 -> 데이터를 파싱하고 -> 데이터를 분석하고 -> 분석결과를 리포팅하고 -> 파일을 닫는 **동일한 절차**를 지닙니다.
    - doc, csv, pdf 세 클래스는 **중복된 코드**를 지닙니다.

      ex:) openFile, analyzeData, sendReport, closeFile
2. 분기 로직이 많습니다.
    - client코드에서 분기처리로 각 클래스의 메서드를 호출하기 때문입니다.

어떻게 하면 동일한 구조를 지닌 클래스들의 중복을 피할 수 있을까요?

Solution

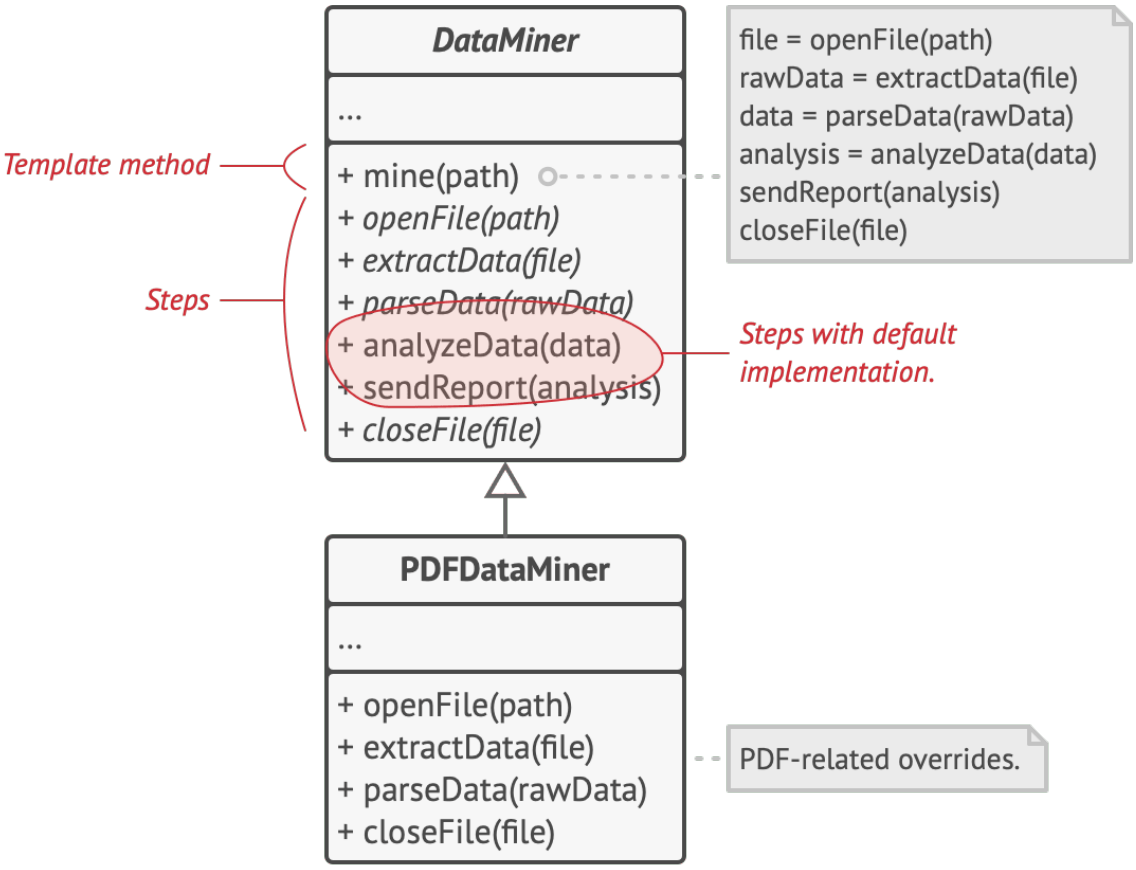
핵심

- 공통 로직은 super 클래스에서 처리!
- 상이한 로직은 sub 클래스에서 처리!
- 일련의 과정은 묶어서 처리!

방법

1. DataMiner라는 super 클래스를 만듭니다.
    - 공통된 스텝은 이 클래스에서 구현을 합니다.
    - 각 타입의 문서가 처리해야할 로직은 abstract로 둡니다.
    - 일련의 메서드를 실행할 수 있도록 mine이라는 template method를 둡니다.
2. 각 문서 포맷에 대한 클래스들은 DataMiner를 상속받습니다.
    - 각 문서 포맷의 클래스는 자신이 작성할 메서드를 오버라이딩하여 구현하도록 합니다.

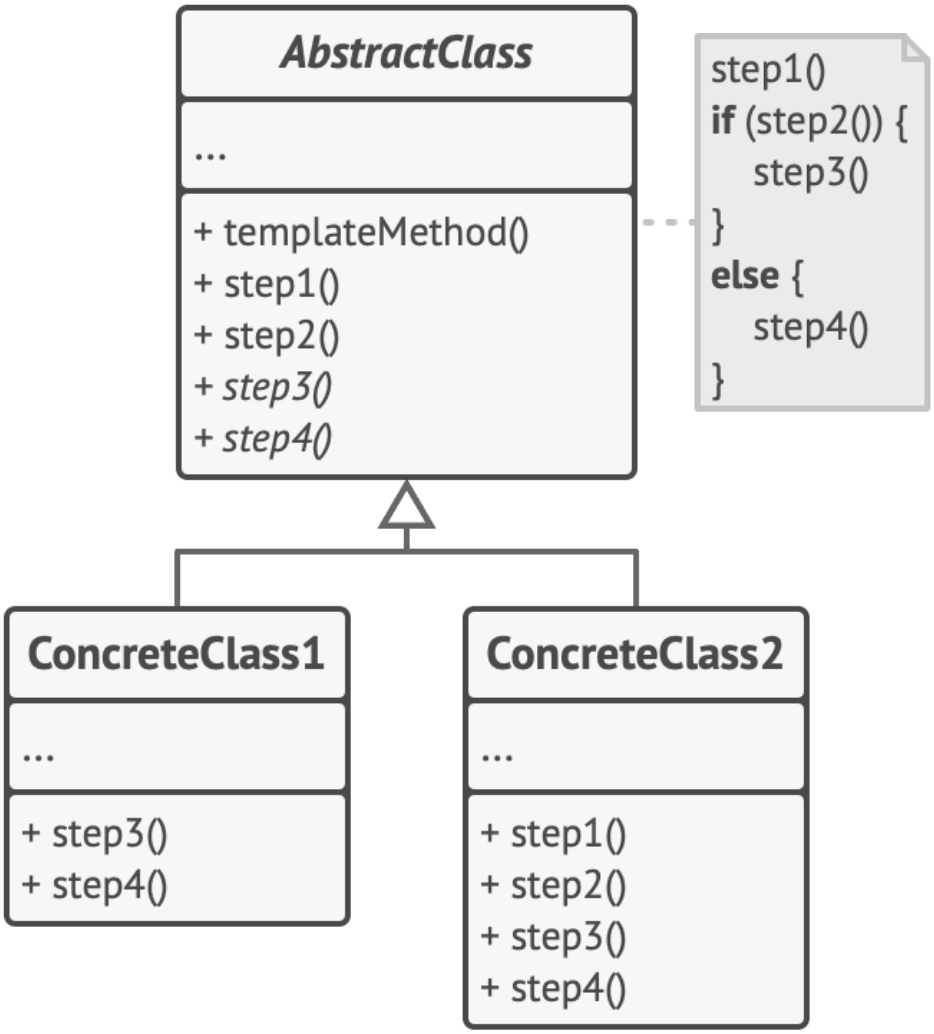
      ex:) PDF파일을 데이터 마이닝한다면, abstract로 선언된 parseData 메서드를 pdf를 파싱하는 로직으로 구현해야겠지요.



이와 같이 구현하면 공통 로직은 DataMiner에 둘 수 있고, 각 문서에 맞는 로직은 오버라이딩을 통해 구현할 수 있습니다.

그리고 일련의 실행과정은 template method를 통해 간단히 실행할 수 있게 됩니다.

구조



1. Abstract Class

- 알고리즘에서 각 단계를 메서드로 선언합니다.
- 이러한 메서드를 특정 순서대로 실행해주는 template method를 선언합니다.

- 추가로 후킹 메서드라는 것도 선언할 수 있습니다.
  - 후킹 메서드는 비어있는 코드로 되어 있습니다. (empty body)
  - 추가적인 로직이 필요할 때 이 후킹 메서드를 구현함으로써 로직을 추가합니다.
  - 따라서 보통 중요한 스텝 사이에 위치합니다.

2. Concrete Class

- 구현해야할 스텝을 오버라이딩하여 자신만의 로직은 구현합니다.

예제

- https://jdm.kr/blog/116