10장. 응집성: 소프트웨어의 관련 요소들은 한곳에

> 서로 관련이 없는 것은 더 멀리 떨어뜨리고, 서로 관련된 것은 더 가깝게 배치하자.

응집성을 얘기할 때, 단순히 서로 관련 없는 모음을 파일에 던져넣는다고 해서 코드가 모듈화가 되는게 아님

- 다른 컴포넌트(모듈)로부터 정보를 은닉하는 시스템 컴포넌트를 의미

응집성을 개선하기 위한 리펙토링 사례 하나

▼ 코드 10.1 순진한 응집성을 보여주는 정말 나쁜 코드

```
public class ReallyBadCohesion
{
    public boolean loadProcessAndStore( ) throws IOException
    {
        String[] words;
        List<String> sorted;

        try (FileReader reader =
                    new FileReader("./resources/words.txt"))
        {
            char[] chars = new char[1024];
            reader.read(chars);
            words = new String(chars).split(" |\0");
        }
        sorted = Arrays.asList(words);
        sorted.sort(null);

        try (FileWriter writer =
                    new FileWriter("./resources/test/sorted.txt"))
        {
            for (String word : sorted)
            {
                writer.write(word);
                writer.write("\n");
            }
            return true;
        }
    }
}
```

- 위 코드는 관심사 분리 불량, 모듈성 불량, 높은 결합도. 거의 추상화 제로를 외치고 있음
- 응집성 측면에서는?
    - 순진한 관점 : 모든 내용을 함께 쉽게 볼 수 있다.

```
public class BadCohesion
{
    public boolean loadProcessAndStore() throws IOException
    {
        String[] words = readWords();
        List<String> sorted = sortWords(words);
        return storeWords(sorted);
    }

    private String[] readWords() throws IOException
    {
        try (FileReader reader =
                    new FileReader("./resources/words.txt"))
        {
            char[] chars = new char[1024];
            reader.read(chars);
            return new String(chars).split(" |\0");
        }
    }

    private List<String> sortWords(String[] words)
    {
```



```
        List<String> sorted = Arrays.asList(words);
        sorted.sort(null);
        return sorted;
    }

    private boolean storeWords(List<String> sorted) throws IOException
    {
        try (FileWriter writer =
                    new FileWriter("./resources/test/sorted.txt"))
        {
            for (String word : sorted)
            {
                writer.write(word);
                writer.write("\n");
            }
            return true;
        }
    }
}
```

- 위 코드 보다는 코드에서 밀접하게 관련된 부분이 더 명확하게 묘사되고 문자 그대로 서로 더 가깝게 연결되어 있어 응집성이 훨씬 더 높다.

> 코드의 주요 목표는 아이디어를 사람에게 전달하는 것이다.

결합도와 응집성 사이의 관계

- 결합도 (coupling)
  - 코드 A와 B가 주어졌을 때, A가 변경되었기 때문에 B가 동작을 변경해야 할 때 결합도가 생김
- 응집성 (cohesion)
  - A를 변경하면 B가 변경되어 둘 다 새로운 가치를 추가할 수 있을 때 응집성이 생김

> 결합도는 낮추고 응집성은 높이려고 하자

응집성 있는 소프트웨어를 만들려면

▼ 코드 10.3 더 열악한 응집성

```
class PoorCohesion:
    def __init__(self):
        self.a = 0
        self.b = 0

    def process_a(x):
        a = a + x

    def process_b(x):
        b = b * x
```

- 두 변수는 서로 다른 메서드에 밀접하지만 서로 관련이 없음에도 불구하고 클래스 수준에서 함께 저장된다.

▼ 코드 10.4 더 나은 응집성

```
class BetterCohesionA:
    def __init__(self):
        self.a = 0

    def process_a(x):
        a = a + x

class BetterCohesionB:
    def __init__(self):
        self.b = 0
    def process_b(x):
        b = b * x
```

- 위 코드는 코드10.3 보다 더 멋지고 응집성 있는 해법을 보여준다.
- 응집성이 더 높을 뿐만아니라 더 모듈화되어 있고 관심사를 더 잘 분리하고 있다는 점에 주목하자.

▼ 코드 10.5 세 가지 응집성 예시

```
def add_to_cart1(self, item):
    self.cart.add(item)

    conn = sqlite3.connect('my_db.sqlite')
    cur = conn.cursor()
    cur.execute('INSERT INTO cart (name, price)
    values (item.name, item.price)')
    conn.commit()
    conn.close()

    return self.calculate_cart_total();

def add_to_cart2(self, item):
    self.cart.add(item)
    self.store.store_item(item)
    return self.calculate_cart_total();

def add_to_cart3(self, item, listener):
    self.cart.add(item)
    listener.on_item_added(self, item)
```

- 뭐가 가장 읽기 좋아 보이는가?

...

- 첫번째 함수는 응집성 있는 코드가 아님
- 두번째 함수는 조금 더 나은 예임
  - 저장이라는 명령어가 논쟁의 여지가 있겠지만 아무튼 1번보단 더 낫다.
- 마지막 함수가 가장 응집성이 있다고 저자는 주장하고 있다.
- 유용한 작업을 수행하려면 장바구니에 품목을 추가하고 잠재적으로 이해관계가 있는 다른 당사자에게 품목이 추가되었음을 알릴 필요가 있다.

응집성이 부족할 때 치러야 할 대가

- 응집성이 떨어지면 코드와 시스템의 유연성이 떨어지고 테스트하거나 작업 시 더 어려워진다는 문제가 생김
- 위의 예에서 add_to_cart3 함수가 항상 최고일 거 같지만 응집성을 위한 최적의 지점이 존재한다.
- add_to_cart3 함수는 설계적으로는 더 유연하지만 여전히 명확성이 부족하다.
  - 너무 느슨해도 명확성을 확보하는 데 비용이 들지만 장점도 있다는 것이다.

---

1. 응집성의 정의 및 설계 원칙

- 컴퓨터 과학에서 **응집성은 "모듈 내부의 요소들이 함께 속해 있는 정도"**로 정의됩니다.
- 좋은 소프트웨어 설계는 **"서로 관련이 없는 것은 더 멀리 떨어뜨리고, 서로 관련된 것은 더 가깝게 배치하자"**는 원칙을 따르며, 응집성은 관련 개념들을 코드 내에 가깝게 배치해야 할 필요성 때문에 중요합니다.
- 응집성은 단순히 모듈성에 대한 순진한 주장을 넘어, 시스템의 복잡성을 관리하기 위해 필요한 핵심 기술 중 하나입니다.

2. 응집성 개선을 위한 실천 방안

- **가독성과 간결함:** 코드는 의사소통 도구이므로, 타이핑된 문자 수로 효율성을 평가하기보다는 **가능한 한 명확하고 간결하게 아이디어를 표현**하는 것이 중요합니다. 응집성을 개선하면 코드를 훨씬 더 읽기 쉽고, 결과적으로 수정하기가 훨씬 더 쉬워집니다.
- **도메인 주도 설계(DDD):** DDD의 '경계 컨텍스트(bounded context)'와 '유비쿼터스 언어' 같은 개념을 사용하면 설계에 지침을 제공하고 코드의 모듈성, 응집성, 관심사 분리를 개선하는 데 도움이 되며, 자연스레 느슨하게 결합된 코드 조직을 유도합니다.
- **TDD(테스트 주도 개발):** TDD를 통해 테스트 가능한 설계를 달성하려고 노력하면, 시스템의 응집성을 향상하도록 설계에 압력을 가할 수 있습니다. TDD의 규율은 최적의 응집성 지점에 도달하도록 장려하며, 프로그래머의 기술과 경험을 증폭시키는 **"재능 증폭기"** 역할을 합니다.
- **본질적 복잡성과 우발적 복잡성 분리:** 응집성을 개선하는 효과적인 방법은 시스템의 **본질적인 복잡성**(문제 해결 자체에 내재된 복잡성)을 **우발적인 복잡성**(컴퓨터 실행 때문에 발생하는 데이터 영속성, 화면 표출 등 부수적인 문제)과 명확하게 분리하는 것입니다.

3. 낮은 응집성의 대가 및 조직적 중요성

- 응집성이 떨어지면 코드와 시스템의 유연성이 떨어지고 테스트하거나 작업하기가 더 어려워집니다. 특히 코드가 다양한 책임을 혼동하는 경우, 가독성이 부족해 무엇을 하는지 파악하기 어려워집니다.
- 실제로 **고성능 시스템**은 복잡한 코드가 아닌 단순하고 잘 설계된 코드가 필요하며, 코드가 복잡할수록 컴파일러의 최적화 기능이 떨어집니다.
- 응집성은 조직 구조에도 영향을 미칩니다. 높은 성과를 내는 팀의 주요 특징 중 하나는 **팀 외부의 누구에게도 허락을 구할 필요 없이** 스스로 결정을 내릴 수 있는 능력인데, 이는 팀의 정보와 기술에 응집성이 있다는 의미입니다.

4. 결합도와의 관계

- 응집성은 변경의 범위 또는 비용을 측정하는 핵심 척도이며, 응집성이 높은 시스템 영역에서는 더 긴밀하게 결합될 가능성이 높습니다.
- **결합도는 어떤 의미에서 응집성의 비용**이라고 할 수 있으며, 응집성과 관심사 분리는 결합도를 줄이는 데 도움이 되는 핵심 기술입니다.

응집성이란 관련된 개념들을 코드 내에 함께 배치하는 것으로, 복잡성을 관리하는 데 있어 가장 중요한 아이디어 중 하나이며, 다른 복잡성 관리 원칙들(모듈성, 관심사 분리)과 밀접하게 연관되어 있습니다. 응집성은 소프트웨어 개발을 위해 우리가 만들고자 하는 시스템의 핵심적인 구조를 잡는 데 필수적인 기초 작업과 같습니다.