Visitor Pattern

비지터 패턴은 어떤 구조로부터 그 안에 있는 연산들을 분리해내는 방법을 설계한 것이다. 보통의 옵저버들은 클로저의 멀티메서드, 프로토콜, 타입, 프록시와 유사해 보이면서도 그 특징들을 구체화하고 있다.

자바와 클로저

나탈리우스 S. 셀비즈(Natanius S. Selbys)는 사용자들이 자신의 메시지와 활동을 다른 포맷으로 내보내는(export) 기능 구현을 제안했다.

이브: 어떻게 하실 생각이세요?

페드로: 항목들(Message, Activity)의 계층도와 파일 포멧들(PDF, XML)의 계층도가 있는 거죠

```
abstract class Format { }
class PDF extends Format { }
class XML extends Format { }

public abstract class Item {
  void export(Format f) {
    throw new UnknownFormatException(f);
  }
  abstract void export(PDF pdf);
  abstract void export(XML xml);
}

class Message extends Item {
  @Override
  void export(PDF f) {
    PDFExporter.export(this);
  }

  @Override
  void export(XML xml) {
    XMLExporter.export(this);
  }
}

class Activity extends Item {
  @Override
  void export(PDF pdf) {
    PDFExporter.export(this);
  }

  @Override
  void export(XML xml) {
    XMLExporter.export(this);
```

```
  }
}
```

페드로: 이게 다에요

이브: 좋아요, 그런데 인수의 타입에 따른 디스패치는 어떻게 하죠?

페드로: 그게 뭐죠?

이브: 다음의 코드를 보시죠

```
Item i = new Activity();
Format f = new PDF();
i.export(f);
```

페드로: 이 코드에는 의심스러운 부분이 없어 보이는데요

이브: 실제로 이 코드를 실행해 보면 UnknownFormatException이 발생해요

페드로: 잠깐만요… 정말요?

이브: 자바에서는 단일 디스패치만이 가능해요. 다시 말해 i.export(f)를 호출하면, i의 실제 타입에 따라 디스패치 될 뿐 인수인 f는 전혀 고려의 대상이 되지 않아요.

페드로: 놀랍군요. 그러면 인수의 타입에 따른 디스패치는 안된다는 건가요?

이브: 그래서 방문자 패천이 생긴 것이죠. 먼저 i 타입에 기반해 디스패치한 후, 다시 f.someMethod(i)를 호출해 f의 타입에 기반해서 디스패치하는 방식으로요.

페드로: 그런 식의 코드는 어떻게 작성하죠?

이브: 모든 타입에 export 메소드를 일일이 방문자로 정의하면 됩니다.

```
public interface Visitor {
  void visit(Activity a);
  void visit(Message m);
}

public class PDFVisitor implements Visitor {
  @Override
  public void visit(Activity a) {
    PDFExporter.export(a);
  }

  @Override
  public void visit(Message m) {
    PDFExporter.export(m);
  }
}
```

이브: 각 아이템은 다른 방문자를 받아들일 수 있도록 인수 형식을 바꾸어 줍니다.

```
public abstract class Item {
  abstract void accept(Visitor v);
}

class Message extends Item {
  @Override
  void accept(Visitor v) {
    v.visit(this);
  }
}

class Activity extends Item {
  @Override
  void accept(Visitor v) {
    v.visit(this);
  }
}
```

이브: 그리고 다음과 같은 방식으로 호출합니다.

```
Item i = new Message();
Visitor v = new PDFVisitor();
i.accept(v);
```

이브: 모든 것이 제대로 작동합니다. 게다가 Activity와 Message의 코드를 변경하지 않고, 단순히 새로운 방문자를 정의해서 새로운 동작을 추가할 수 있어요.

페드로: 정말로 유용하네요. 하지만 구현은 쉽지 않네요. 클로저에서도 마찬가지인가요?

이브: 그렇지 않아요. 클로저에서는 멀티메소드를 통해 쉽게 구현할 수 있어요.

페드로: 멀티 뭐라고요?

이브: 코드를 보시죠. 먼저 디스패처(dispatcher) 함수를 정의합니다.

```
(defmulti export
  (fn [item format] [(:type item) format]))
```

이브: 이 함수는 item과 format을 인수로 받습니다. 예를 들면,

```
;; Message item
{:type :message :content "Say what again!"}

;; Activity item
{:type :activity :content "Quoting Ezekiel 25:17"}
```

```
;; Formats
:pdf, :xml
```

이브: 그리고 다음과 같이 다양한 조합의 인수를 받는 함수들을 정의합니다. 그러면 디스패처가 어떤 함수를 호출할지 결정합니다.

```
(defmethod export [:activity :pdf] [item format]
  (exporter/activity->pdf item))

(defmethod export [:activity :xml] [item format]
  (exporter/activity->xml item))

(defmethod export [:message :pdf] [item format]
  (exporter/message->pdf item))

(defmethod export [:message :xml] [item format]
  (exporter/message->xml item))
```

페드로: 모르는 포맷이 인수로 건네지면 어떻게 하죠?

이브: 다음처럼 디폴트 함수를 지정해 줄 수 있어요

```
(defmethod export :default [item format]
  (throw (IllegalArgumentException. "not supported")))
```

페드로: 좋습니다. 하지만 :pdf와 :xml 사이에는 아루먼 상하 관계(hierarchy)가 존재하지 않네요. 단순히 키워드일 뿐이잖아요?

이브: 맞아요. 단순한 문제여서 해법도 단순해요. 이와 같은 고급 기능이 필요하면 다음과 같이 임의로 계층 관계를 지정해 줄 수도 있고, class 함수를 디스패처로 사용할 수도 있어요

```
(derive ::pdf ::format)
(derive ::xml ::format)
```

페드로: 콜론이 두 개 있네요!

이브: 일단은 그냥 키워드와 같다고 생각하세요.

페드로: 알겠습니다.

이브: 그리고 ::pdf와 ::xml, ::format에 해당하는 함수들을 다음처럼 추가해줍니다.

```
(defmethod export [:activity ::pdf])
(defmethod export [:activity ::xml])
(defmethod export [:activity ::format])
```

이브: 만약 새로운 포맷(예를 들면, csv)처리가 필요하면, 다음과 같이 해 줍니다.

```
(derive ::csv ::format)
```

이브: ::csv를 처리하는 함수를 별도로 제공하지 않으면, ::format을 처리하는 하수가 이를 처리해 줘요.

페드로: 훌륭해 보이네요

이브: 물론지요, 게다가 훨씬 쉽지요.

페드로: 그렇다면, 언어가 기본적으로 다중 디스패처를 지원하면, 방문자 패턴은 필요 없다는 건가요?

이브: 맞아요

```
(def msg-item
  {:type :message :content "Say what again!"})

(def act-item
  {:type :activity :content "Quoting Ezekiel 25:17"})
(derive ::pdf ::format)
(derive ::xml ::format)
(derive ::csv ::format)


(defmethod export [:activity ::pdf] [item format]
  (str item format ::pdf))
(defmethod export [:activity ::xml])
(defmethod export [:activity ::format] [item  format]
  (str item format ::format))


(export act-item ::csv)
;=> "{:type :activity, :content \"Quoting Ezekiel 25:17\"}:clojure-design-pattern.visitor/csv:clojure-design-pattern.visitor/format"
```

관련 문서

- Clojure Design Patterns