Chain Of Responsibility Pattern

뉴욕의 마케팅 조직인 “A Profit NY”는 그들의 공개 채팅 시스템에서 비속어 필터링을 요청했다.

페드로: 젠장, 그들은 '젠장 ’ 이라는 말을 싫어하나?

이브: 수익을 내야 하는 조직이니, 공개 채팅에서 누군가 비속어를 사용하면 수익을 잃겠죠.

페드로: 비속어 리스트는 누가 만들죠?

이브: 조지 칼린이요.

리스트를 보다가 웃으며

페드로: 오케이, 비속어들을 별표로 바꾸는 필터를 추가해 봅시다.

이브: 그 솔루션은 확장성이 있어야 해요, 다른 필터도 적용될 수도 있어야 하거든요.

페드로: 책임 연쇄 패턴은 여기에 딱 맞는 패턴인 것 같네요. 일단 먼저 추상 필터를 만들어 보죠.

```
public abstract class Filter {
  protected Filter nextFilter;

  abstract void process(String message);

  public void setNextFilter(Filter nextFilter) {
    this.nextFilter = nextFilter;
  }
}
```

페드로: 그리고 나서 실제로 적용할 필터들 구현해 봅시다.

```
class LogFilter extends Filter {
  @Override
  void process(String message) {
    Logger.info(message);
    if (nextFilter != null) nextFilter.process(message);
  }
}

class ProfanityFilter extends Filter {
  @Override
  void process(String message) {
    String newMessage = message.replaceAll("fuck", "f*ck");
    if (nextFilter != null) nextFilter.process(newMessage);
  }
}

class RejectFilter extends Filter {
```

```
  @Override
  void process(String message) {
    System.out.println("RejectFilter");
    if (message.startsWith("[A PROFIT NY]")) {
      if (nextFilter != null) nextFilter.process(message);
    } else {
      // reject message - do not propagate processing
    }
  }
}

class StatisticsFilter extends Filter {
  @Override
  void process(String message) {
    Statistics.addUsedChars(message.length());
    if (nextFilter != null) nextFilter.process(message);
  }
}
```

페드로: 마지막으로 메시지가 처리되는 순서를 정의하는 필터의 연쇄를 만듭시다.

```
Filter rejectFilter = new RejectFilter();
Filter logFilter = new LogFilter();
Filter profanityFilter = new ProfanityFilter();
Filter statsFilter = new StatisticsFilter();

rejectFilter.setNextFilter(logFilter);
logFilter.setNextFilter(profanityFilter);
profanityFilter.setNextFilter(statsFilter);

String message = "[A PROFIT NY] What the fuck?";
rejectFilter.process(message);
```

이브: 오케이, 이제 클로저로 해보죠. 각 필터는 함수로 정의합니다.

```
;; define filters
(defn log-filter [message]
  (logger/log message)
  message)

(defn stats-filter [message]
  (stats/add-used-chars (count message))
  message)

(defn profanity-filter [message]
  (clojure.string/replace message "fuck" "f*ck"))

(defn reject-filter [message]
```

```
  (if (.startsWith message "[A Profit NY]")
    message))
```

이브: 그리고 some→ 매크로를 사용해서 필터들을 연결합니다.

```
(defn chain [message]
  (some-> message
          reject-filter
          log-filter
          stats-filter
          profanity-filter))
```

이브: 얼마나 쉬운지 아시겠죠? 너무 자연스러워서, 매번 `if (nextFilter != null) nextFilter.process()` 호출할 필요가 없어요. some→에서 정의한 다음 필터는 setNext를 호출할 필요 없이, 자연스럽게 연결돼요.

페드로: 확실히 조립성(composability)이 더 좋네요. 하지만 왜 →가 아닌 some→을 썼죠?

이브: reject-filter 때문이죠. 더 이상 진행할 필요가 없을 수 있는데, 그래서 필터가 nil을 반환하면 some→은 곧바로 nil을 반환하죠.

페드로: 좀 더 설명해 주실 수 있어요?

이브: 사용 예를 보시죠.

```
(chain "fuck") => nil
(chain "[A Profit NY] fuck") => "f*ck"
```

페드로: 이해됐어요.

이브: 책임 연쇄 패턴은 단지 함수 합성일 뿐인 거죠.

관련 문서

Plugin Backlinks: 아무 것도 없습니다.