DateMessageProvider에 대한 정리 및 느낀 점

잡 생각 때문에 잠이 오지 않아서 뒤척이다가 이 곳에서 코드를 가지고 더 많은 이야기를 했으면 좋겠다는 생각이 들었다. 프로그래머라면 코드를 통해 이야기할 때가 가장 즐겁기 때문이다. 그런 고민 후에 시작한 첫번째 글이 테스트를 모두 성공하려면 어떻게 해야 될까요? 글이다. 이 글에서 다룬 코드는 다음과 같다.

```
package net.slipp;

import java.util.Calendar;

public class DateMessageProvider {

  public String getDateMessage() {
    Calendar now = Calendar.getInstance();
    int hour = now.get(Calendar.HOUR_OF_DAY);

    if (hour < 12) {
      return "";
    }

    return "";
  }
}
```

위 소스 코드에 대한 단위 테스트를 만들어 자동화하려면 어떻게 해야될까에 대한 내용이다. 위 소스 코드에 대한 단위 테스트를 자동화하기 위한 해결 방법으로 세 가지 방법을 제시했다.

- DateMessageProvider 코드에 대한 첫번째 해결 방법 및 이슈 제기 : Calendar 인스턴스를 메소드 인자로 전달해 단위테스트를 실행. 즉, Dependency Injection을 활용해 문제 해결
- DateMessageProvider 코드에 대한 두번째 해결 방법 : Calendar 인스턴스 생성 부분을 별도의 method로 추출하고 이 method를 override해서 문제 해결
- DateMessageProvider 코드에 대한 세번째 해결 방법 : 메소드 인자로 시간을 전달하는 method를 추출해 Calendar 인스턴스와는 별개로 단위 테스트 진행

처음 위 예제 코드를 제시할 때는 이 세가지 방법 밖에 생각나지 않았다. 문제를 내고 몇 일 동안 하나씩 내용을 정리하다가 세번째 해결 방법에서 뭔가 찜찜했다. 이 찜찜함은 과거 소스 코드를 리팩토링하고 단위 테스트를 만들 때도 항상 발생하는 부분이었다. 찜찜한 부분은 DateMessageProvider 코드에 대한 세번째 해결 방법 글에서도 잠시 언급했지만 extract method 패턴으로 추출한 메소드를 단위 테스트를 위해 private이 아닌 default로 구현해야 한다는 것이다. 물론 일부 단위 테스트 프레임워크를 활용해 private 상태로 테스트할 수 있지만 그래도 뭔가 찜찜했다. 테스트 코드도 괜히 배보다 배꼽이 커질 수 있게다는 생각도 들었다.

```
package net.slipp;

import java.util.Calendar;

public class DateMessageProvider {
  public String getDateMessage() {
    Calendar now = Calendar.getInstance();
    int hour = now.get(Calendar.HOUR_OF_DAY);
    return getHourMessage(hour);
  }

  String getHourMessage(int hour) {
    if (hour < 12) {
      return "";
    }

    return "";
  }
}
```

몇 일전 잠자리에서 위 소스 코드 가지고 다른 방법이 없을까 고민해 봤다. 다양한 방식으로 고민해 본 결과 분명 소스 코드에 Bad Smell이 느껴진다. 뭔가 찜찜하고, 꾸리꾸리하다. 즉, DateMessageProvider가 너무 여러 가지 책임을 가지고 있다는 느낌이 든다. Calendar 인스턴스를 생성하고, 시간으로 구하고, 시간에 따른 메시지 생성까지 담당하고 있다. 그럼 클래스의 단일 책임 원칙에 따라 분리하는 것이 맞을까? 그런 생각으로 머릿속에서 한번 분리해봤다. 그랬더니 다음과 같은 소스 코드가 만들어 졌다.

```
package net.slipp;

import java.util.Calendar;

public class DateMessageProvider {
  public String getDateMessage() {
    Calendar now = Calendar.getInstance();
    int hour = now.get(Calendar.HOUR_OF_DAY);

    HourMessageProvider messageProvider = new HourMessageProvider();
    return messageProvider.getMessage(hour);
  }
}
```

```
package net.slipp;
public class HourMessageProvider {
  public String getMessage(int hour) {
    if (hour < 12) {
        return "";
    }
    return "";
  }
}
```

위와 같이 클래스를 두 개로 분리하고 단위 테스트는 HourMessageProvider에 대해서만 다음과 같이 테스트하면 되지 않을까라는 생각을 했다.

```
package net.slipp;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;
import org.junit.Test;

public class HourMessageProviderTest {
  @Test
  public void () throws Exception {
    HourMessageProvider provider = new HourMessageProvider();
    assertThat( provider.getMessage(11), is(""));
  }

  @Test
  public void () throws Exception {
    HourMessageProvider provider = new HourMessageProvider();
    assertThat( provider.getMessage(16), is(""));
  }
}
```

위 소스 코드를 보는 순간 많은 개발자들이 불만의 목소리를 내지 않을까 생각한다. 아니 너무 한거 아니야. 저 인간이 학교로 가더니 이제 제정신이 아닌 상태가 되었구나라고 성토할 것 같은 생각이 들었다. 아마 보지 않아도 뻔하다. 그래서 다시 고민을 하기 시작했다. 분명 뭔가 찜찜하고, 꾸리꾸리한 냄새가 나는데 좋은 방법이 없을까? 그렇게 고민하던 끝에 다음과 같이 리팩토링한다면 클래스의 역할도 명확하고 개발자들도 수긍하지 않을까라는 생각을 했다.

```
package net.slipp;
import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;
import java.util.Calendar;
import org.junit.Test;
public class MyCalendarTest {
  @Test
  public void getHour() throws Exception {
    Calendar now = Calendar.getInstance();
    now.set(Calendar.HOUR_OF_DAY, 11);
    MyCalendar calendar = new MyCalendar(now);
    assertThat(calendar.getHour(), is(new Hour(11)));
  }
}
```

```
package net.slipp;
import java.util.Calendar;
public class MyCalendar {
  private Calendar calendar;
  public MyCalendar(Calendar calendar) {
    this.calendar = calendar;
  }
  public Hour getHour() {
    return new Hour(calendar.get(Calendar.HOUR_OF_DAY));
  }
}
```

먼저 Calendar의 시간을 구하는 책임을 위와 같이 MyCalendar를 만들어 구현한다. 그리고 시간을 int 형으로 반환하는 것이 아니라 Hour라는 클래스로 다음과 같이 구현한다.

```
package net.slipp;
import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;
import org.junit.Test;
public class HourTest {
  @Test
  public void () throws Exception {
    Hour hour = new Hour(11);
    assertThat(hour.getMessage(), is(""));
  }
  @Test
  public void () throws Exception {
    Hour hour = new Hour(16);
    assertThat(hour.getMessage(), is(""));
  }
}
```

```
package net.slipp;
public class Hour {
  private int hour;
  public Hour(int hour) {
    this.hour = hour;
  }
  public String getMessage() {
    if (hour < 12) {
      return "";
    }
    return "";
  }
  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    Hour other = (Hour) obj;
    if (hour != other.hour)
      return false;
    return true;
  }
}
```

위 소스 코드와 같이 Hour 클래스를 도출하고 각 시간에 따른 메시지 구하는 부분을 Hour 클래스에서 구현하도록 했다. 이와 같이 구현하니 클래스명도 확실하고 역할도 명확하게 구분되는 듯하다.

물론 지금까지 구현한 코드가 완전하지는 않지만 앞에서 이슈 제기한 private 메소드에 대한 이슈를 일정 부분 해결할 수 있어 만족스럽다. 앞으로도 이 같은 부분이 발생하면 각 역할에 맞는 클래스를 찾도록 노력해 봐야겠다.