JAVA MOCKING

FIZZ BUZZ WHIZZ

你是一名体育老师，在某次课距离下课还有五分钟时，你决定搞一个游戏。

首先你说出三个不同的特殊数, 要求必须是个位数, 比如3, 5, 7.

此时上课的100名学生拍成一队, 按顺序进行报数.

规则一

- 如果所报数字是第一个特殊数(3)的倍数, 那么不能说该数字, 而要说Fizz
- 如果所报数字是第二个特殊数(5)的倍数, 那么要说Buzz
- 如果所报数字是第三个特殊数(7)的倍数, 那么要说Whizz.

规则二

- 如果所报数字同时是两个特殊数的倍数情况下, 那么不能说该数字, 而是要说FizzBuzz, 以此类推.
- 例如要报15的同学数字同时是第一个特殊数(3)和第二个特殊数(5)的倍数, 所以要说FizzBuzz.
- 如果所报数字同时是三个特殊数的倍数, 那么要说FizzBuzzWhizz.

规则三

- 如果所报数字包含了第一个特殊数, 那么也不能说该数字, 同时忽略规则一和规则二, 而是要说相应的单词.
- 例如要报13的同学数字中包含了第一个特殊数(3), 所以应该说Fizz, 而不是13.
- 例如要报35的同学数字中包含了第一个特殊数(3), 所以应该说Fizz, 而不是BuzzWhizz.

实现

```
public class Student {
    public String say(int number) {
        return Integer.toString(number);
    }
}

@Test
public void shouldSayNumberDirectly() throws Exception {
    Student student = new Student();
    assertThat(student.say(1), is("1"));
    assertThat(student.say(2), is("2"));
    assertThat(student.say(4), is("4"));
}
```

规则一

```
public class Student {
    public String say(int number) {
        if (number % 3 == 0) {
            return "Fizz";
        } else if (number % 5 == 0) {
            return "Buzz";
        } else if (number % 7 == 0) {
            return "Whizz";
        }
        return Integer.toString(number);
    }
}
```

```
@Test
public void shouldSayCorrespondingWordIfNumberIsAMultipleOfSpecialNumber() {
    Student student = new Student();
    assertThat(student.say(3), is("Fizz"));
    assertThat(student.say(5), is("Buzz"));
    assertThat(student.say(7), is("Whizz"));
}
```

规则二

```
public class Student {
    public String say(int number) {
        StringBuilder sb = new StringBuilder();

        if (number % 3 == 0) sb.append("Fizz");
        if (number % 5 == 0) sb.append("Buzz");
        if (number % 7 == 0) sb.append("Whizz");

        return sb.length() != 0 ? sb.toString() : Integer.toString(number);
    }
}
```

```
@Test
public void shouldSayCorrespondingWordsIfNumberIsAMultipleOfMultipleNumbers() {
    Student student = new Student();
    assertThat(student.say(15), is("FizzBuzz"));
    assertThat(student.say(35), is("BuzzWhizz"));
    assertThat(student.say(105), is("FizzBuzzWhizz"));
}
```

规则三

```
public class Student {
    public String say(int number) {
        if (Integer.toString(number).contains("3")) return "Fizz";

        StringBuilder sb = new StringBuilder();
        if (number % 3 == 0) sb.append("Fizz");
        if (number % 5 == 0) sb.append("Buzz");
        if (number % 7 == 0) sb.append("Whizz");
        return sb.length() != 0 ? sb.toString() : Integer.toString(number);
    }
}
```

```
@Test
public void shouldSayFizzIfNumberContains3() {
    Student student = new Student();
    assertThat(student.say(13), is("Fizz"));
    assertThat(student.say(35), is("Fizz"));
}
@Test
public void shouldSayCorrespondingWordsIfNumberIsAMultipleOfMultipleNumbers() {
    Student student = new Student();
    assertThat(student.say(15), is("FizzBuzz"));
    assertThat(student.say(35), is("BuzzWhizz")); // failed!
    assertThat(student.say(105), is("FizzBuzzWhizz"));
}
```

OCP(开放封闭原则)


OPEN CLOSED PRINCIPLE
Open Chest Surgery Is Not Needed When Putting On A Coat

重构

```
public class Student {
    public String say(int number) {
        // 规则三
        if (Integer.toString(number).contains("3")) return "Fizz";
        // 规则二 & 一
        StringBuilder sb = new StringBuilder();
        if (number % 3 == 0) sb.append("Fizz");
        if (number % 5 == 0) sb.append("Buzz");
        if (number % 7 == 0) sb.append("Whizz");
        // 默认规则
        return sb.length() != 0 ? sb.toString() : Integer.toString(number);
    }
}
```

- Student按照优先级匹配规则
- 每个规则处理不同类型的数字

SRP(单一职责原则)


SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should

STUDENT按照优先级匹配规则

```
public class Student {
    private List<Rule> rules;
    public Student(Rule... rules) {
        this.rules = ImmutableList.copyOf(rules);
    }
    public String say(int number) {
        for (Rule rule : rules) {
            Optional<String> result = rule.apply(number);
            if (result.isPresent()) {
                return result.get();
            }
        }

        throw new IllegalStateException();
    }
}
```

```
interface Rule {
    Optional<String> apply(int number);
}
```

默认规则

```
class DefaultRule implements Rule {
    public Optional<String> apply(int number) {
        return Optional.of(Integer.toString(number));
    }
}
```

```
@Test
public void shouldSayNumberDirectly() throws Exception {
    Student student = new Student(new DefaultRule());
    assertThat(student.say(1), is("1"));
    assertThat(student.say(2), is("2"));
    assertThat(student.say(4), is("4"));
}
```

规则一 & 规则二

```
public class MultipleNumberRule implements Rule {

    public Optional<String> apply(int number) {
        StringBuilder sb = new StringBuilder();
        if (number % 3 == 0) sb.append("Fizz");
        if (number % 5 == 0) sb.append("Buzz");
        if (number % 7 == 0) sb.append("Whizz");
        return Optional.fromNullable(sb.length() == 0 ? null : sb.toString());
    }
}
```

```
@Test
public void shouldSayCorrespondingWordsIfNumberIsAMultipleOfMultipleNumbers() {
    Student student = new Student(new MultipleNumberRule(), new DefaultRule());
    assertThat(student.say(15), is("FizzBuzz"));
    assertThat(student.say(35), is("BuzzWhizz"));
    assertThat(student.say(105), is("FizzBuzzWhizz"));
}
```

规则三

```
public class SpecialNumberRule implements Rule {
    public Optional<String> apply(int number) {
        if (Integer.toString(number).contains("3")) {
            return Optional.of("Fizz");
        } else {
            return Optional.absent();
        }
    }
}
```

```
@Test
public void shouldSayFizzIfNumberContains3() {
    Student student = new Student(new SpecialNumberRule(),
        new MultipleNumberRule(), new DefaultRule());
    assertThat(student.say(13), is("Fizz"));
    assertThat(student.say(15), is("FizzBuzz"));
    assertThat(student.say(35), is("Fizz"));
}
```

INVERSION OF CONTROL(控制反转)

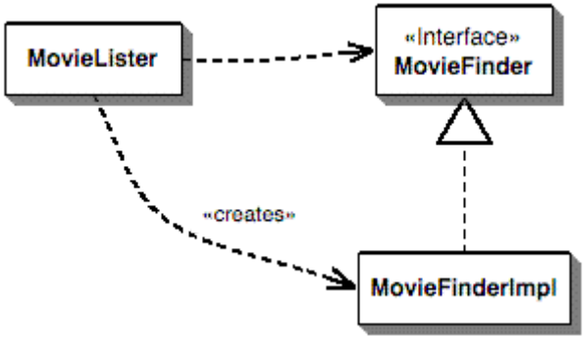

Figure 1 shows the dependencies for this situation. The MovieLister class is dependent on both the MovieFinder interface and upon the implementation.

DEPENDENCY INJECTION(依赖注入)

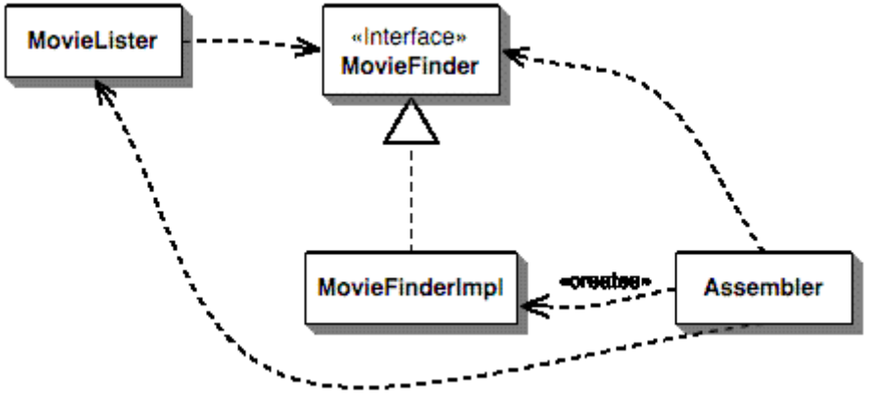

Figure 2: The dependencies for a Dependency Injector

IOC

The main control of the program was inverted, moved away from you to the framework.

问题

- 规则改变, 当数字中包含3, 则返回Buzz

```
@Test
public void shouldSayFizzIfNumberContains3() {
  Student student = new Student(new SpecialNumberRule(),
      new MultipleNumberRule(), new DefaultRule());
  assertThat(student.say(13), is("Fizz"));    // failed!
  assertThat(student.say(15), is("FizzBuzz"));
  assertThat(student.say(35), is("Fizz"));    // failed!
}
```

再次理解STUDENT逻辑

```
public class Student {
    private List<Rule> rules;
    public Student(Rule... rules) {
        this.rules = ImmutableList.copyOf(rules);
    }
    public String say(int number) {
        for (Rule rule : rules) {
            Optional<String> result = rule.apply(number);
            if (result.isPresent()) {
                return result.get();
            }
        }

        throw new IllegalStateException();
    }
}
```

SUT

- System Under Test

SPECIALNUMBERRULE的测试

- Case 1
  - Given一个数字
  - When数字中包含3
  - Then返回Buzz
- Case 2
  - Given一个数字
  - When数字中没有3
  - Then不返回内容

```
public class SpecialNumberRuleTest {
    @Test
    public void shoudReturnFizzIfNumberContains3() throws Exception {
        SpecialNumberRule rule = new SpecialNumberRule();
        assertThat(rule.apply(3).get(), is("Buzz"));
        assertThat(rule.apply(13).get(), is("Buzz"));
    }

    @Test
    public void shoudReturnNothingIfNumberDoesNotContain3() throws Exception {
        SpecialNumberRule rule = new SpecialNumberRule();
        assertThat(rule.apply(5).isPresent(), is(false));
        assertThat(rule.apply(15).isPresent(), is(false));
    }
}
```

STUDENT的测试

- Case 1
  - Given两个规则
  - When第一个规则返回了Fizz
  - Then返回Fizz
- Case 2
  - Given两个规则
  - When第一个规则没有返回值
  - And第二个规则返回了Buzz
  - Then返回Buzz

通过假实现完成STUDENT的测试

```
public class StudentTest {
    @Test
    public void shouldSayWordAccrodingToTheRules() {
        Student student = new Student(new Rule() {
            public Optional<String> apply(int number) {
                return Optional.fromNullable(number == 3 ? "Fizz" : null);
            }
        }, new Rule() {
            public Optional<String> apply(int number) {
                return Optional.of("Buzz");
            }
        });

        assertThat(student.say(3), is("Fizz"));
        assertThat(student.say(5), is("Buzz"));
    }
```

TEST DOUBLES

- Dummy

  Objects are passed around but never actually used. Usually they are just used to fill parameter lists.

- Fake

  Objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).

- Stubs

  Provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'.

- Mocks

  Objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

通过MOCK完成STUDENT的测试

```
import static org.mockito.Mockito.*;
public class StudentTest {
    @Test
    public void shouldSayWordAccrodingToTheRules() {
        Rule rule1 = mock(Rule.class);
        when(rule1.apply(anyInt())).thenReturn(Optional.<String>absent());
        when(rule1.apply(3)).thenReturn(Optional.of("Fizz"));

        Rule rule2 = mock(Rule.class);
        when(rule2.apply(anyInt())).thenReturn(Optional.of("Buzz"));

        Student student = new Student(rule1, rule2);
        assertThat(student.say(3), is("Fizz"));
        assertThat(student.say(5), is("Buzz"));
    }
}
```

MOCKITO

```
import static org.mockito.Mockito.*;

List<String> mockedList = mock(List.class);

when(mockedList.get(0)).thenReturn("first");
when(mockedList.get(1)).thenThrow(new RuntimeException());

assertThat(mockedList.get(0), is("first"));
```

MOCKITO STUB

```
Car boringStubbedCar = when(mock(Car.class).shiftGear())
        .thenThrow(EngineNotStarted.class)
        .getMock();

doThrow(new RuntimeException()).when(mockedList).clear();
```

WEATHER NOTIFIER

- 小明是一名程序员, 为了体现对女朋友的关心, 小明决定写一个程序自动在天气变化时向女朋友手机上发送一条提醒信息
- 例如当天如果下雨, 则自动向她的手机上发送一条"今天下雨,记得带伞哦"

- 小明经过一番努力, 发现某网站提供如下两个API, 能够帮助他实现梦想:
  - GET http://api.com/weather-api?location={CityName}
    获取指定城市的天气情况, 会返回sunny, rain等状态
  - POST http://api.com/sms-gateway?number={PhoneNumber}&content={Message}
    向指定手机号发送短信

```
public interface HttpClient
{
    String get(String url, String params);
    String post(String url, String params);
    // other irrelevant methods
}
```

```
public class WeatherNotifier {
    private HttpClient client;
    public WeatherNotifier(HttpClient client) {
        this.client = client;
    }
    void check() {
        if (getWeather().equals("rain")) sendNotification();
    }
    private void sendNotification() {
        client.post("http://api.com/sms-gateway", String.format(
                "number=%s&content=%s", "13012345678", "今天下雨,记得带伞哦"));
    }
    private String getWeather() {
        return client.get("http://api.com/weather-api",
                "location=xian");
    }
```

测试?

- 获取天气的API需要网络连接, 并且最近持续是晴天
- 发送短信的API同样需要网络连接, 并且还会收费

WHY MOCK

- 被测对象的外部依赖难以构造
- 由于外部依赖的性能瓶颈导致测试运行缓慢(例如数据库)
- 由于外部依赖的稳定性较差导致测试不稳定(例如网络, 时间)
- 需要构造一些难以触发的异常场景, 用以验证被测对象的特定分支
- 外部依赖尚未实现, 或是其逻辑即将改变

MOCKITO MOCK

```
@Test
public void shouldVerifyMockedMethods() throws Exception {
    List mockedList = mock(List.class);

    mockedList.add("one");
    mockedList.clear();

    verify(mockedList).add("one");
    verify(mockedList).clear();
}
```

```java
@Test
public void shouldVerifyMockedMethods() throws Exception {
    List mockedList = mock(List.class);

    mockedList.add("one");
    mockedList.clear();

    verify(mockedList).add("two");
    verify(mockedList).clear();
}
```

Test Result:

```
Argument(s) are different! Wanted:
list.add("two");
-> at MockTest.shouldVerifyMockedMethods(MockTest.java:28)
Actual invocation has different arguments:
list.add("one");
-> at MockTest.shouldVerifyMockedMethods(MockTest.java:25)

Comparison Failure:
Expected :list.add("two");
Actual   :list.add("one");
```

ARGUMENT MATCHERS

```
when(mockedList.get(anyInt())).thenReturn("element");

assertThat(mockedList.get(999), is("element"));

verify(mockedList).get(anyInt());
```

VERIFYING NUMBER OF INVOCATIONS

```
mockedList.add("twice");
mockedList.add("twice");

mockedList.add("three times");
mockedList.add("three times");
mockedList.add("three times");

verify(mockedList, times(2)).add("twice");

verify(mockedList, never()).add("never happened");
verify(mockedList, atLeastOnce()).add("three times");
verify(mockedList, atLeast(2)).add("five times");
verify(mockedList, atMost(5)).add("three times");
```

练习

帮助小明完成测试

参考资料

- SOLID
- Mocks Aren't Stubs
- Mockito