УДК 621.311

БИБЛИОТЕКА MOCKITO – ЕЕ ИСПОЛЬЗОВАНИЕ, НЕДОСТАТКИ И ДОСТОИНСТВА

Игнатюк В. С.
Научный руководитель – Попова Ю. Б.

Разрабатывая любой программный продукт появляется необходимость проверить корректность его работы. Для этих целей пишутся тесты. Тесты дают уверенность, что программа работает как задумано.

По степени автоматизации выделяют:

- ручное – тестирование, которое проводится без инструментов автоматизации;
- автоматизированное – тестирование на всех уровнях выполняется с использованием средств автоматизации;
- полуавтоматизированное– предполагается, что для определенных целей применяется автоматизация (автоматизации развертки окружения, автоматизация функционального тестирования, автоматизация генерации данных и т.д.)

Успешное выполнение тестов покажет разработчику, что его изменения не сломали ничего. Провалившийся тест позволит обнаружить, что в коде сделаны изменения, которые меняют или ломают его поведение. Исследование ошибки, которую выдает провалившийся тест, и сравнение ожидаемого результата с полученным даст возможность понять, где возникла ошибка, будь она в коде или в требованиях.

Также существует философия разработки через тестирование. Ее основа – это написание кода, который нужен для прохождения тестов. Когда все тесты проходят, код нужно отрефакторить и почистить, а затем приступить к следующему участку.

Для разработки через тестирование используются юнит-тесты. Основное отличие юнит-тестов от системных и интеграционных в том, что они проверяют взаимодействие между модулями, и если тест провалится, ошибку будет легко обнаружить и исправить.

Но основная проблема состоит в том, как оградить части кода друг от друга, поскольку при тестировании юнитов все равно существует связь между ними. Поэтому найти место, где провалится тест будет сложнее.

Таким образом, чтобы проверить корректность работы каждого юнита, необходимо как-то оградить его от ошибок в других юнитах. Для этого пишутся заглушки. Заглушка – это фрагмент кода, который подставляется вместо другого компонента во время тестирования. В объектно-ориентированном тестировании используются mock-объекты – это конкретная фиктивная реализация интерфейса, предназначенная исключительно для тестирования взаимодействия. Во время unit-тестирования mock-объекты могут симулировать поведение бизнес-

объектов и бизнес-логику. Конечно, удобнее создавать и использовать разработанные заранее mock- библиотеки, чем придумывать все с нуля. Например, java-source.net приводит список из 7 таких библиотек: EasyMock, Mocquer, MockLib, Mockrunner, jMock, MockEJB, MockCreator.

Одним из таких тестовых framework-ов является библиотека Mockito для Java, произведенная под лицензией the MIT License. Исследования, проведенные в 2013 году на 10000 проектов GitHub установили, что Mockito является 9-ой наиболее популярной библиотекой Java.

По сравнению с другими подобными инструментами Mockito обладает следующими преимуществами:

1. Создание заглушек для классов и интерфейсов.
2. Возможность проверки порядка, количества и отсутствия вызовов.
3. Концепция “частичной заглушки”.
4. Возможность проверить, с какими параметрами вызывались методы класса.
5. Меньше кода в тестах, и они выглядят читабельнее.

Но у данного framework-а есть и свои недостатки. Mockito не умеет работать с:

1) private/final/static методами;
2) конструкторами;
3) final классами.

Если без этого при написании тестов совсем никак не обойтись, то можно использовать библиотеку PowerMock, которая является надстройкой над Mockito.

Рассмотрим основные функции mockito:

1. Для создания mock объекта необходимо импортировать библиотеку в проект и затем создать объект класс при помощи mock.
2. Чтобы при вызове метода mock-объекта появился нужный результат, используется метод when() совместно с методами thenAnswer(), thenReturn(), thenThrow(). Если возвращаемый объект не задавать, то вернется null, 0, false.
3. Для того чтобы вернуть примитив, нужно в thenReturn() указать true;
4. Для того чтобы вернуть экземпляр класса, нужно в thenReturn() указать new B();
5. Для того чтобы вернуть исключение, нужно в thenReturn() указать newException();

Для примера рассмотрим созданный тест.

```
@Test
publicvoidtestExecuteABI() {
        HttpServletRequest request = mock(HttpServletRequest.class) ;
        HttpServletResponse response = mock(HttpServletResponse.class) ;
        when(request.getParameter("password")).thenReturn("");
```

```
        when(request.getParameter("email")).thenReturn("");
        SignInsignIn = newSignIn ();
        String result = signIn.execute(request, response);
        String page = properties.getString("LOGIN_PAGE_PATH");
        assertEquals(result,page) ;
}
```

Сначала создаем мок-объекты. Затем в when прописываем, что при получении параметров password и email вернутся пустые строки, что указано после thenReturn(""); Далее создаем объект тестируемого класса и вызываем нужный метод. В конце теста сравниваем полученный результат с ожидаемым. В данном случае ожидаемый результат - это возврат нужной страницы. Таким образом было создано несколько тестовых проходов для выбранного метода и запущен тест в eclipse. В результате тесты прошли.

Таким образом, данная библиотека является хорошим помощником в написании тестов.

Литература

1. Javasamples&frameworks : Использование Mockito в Unit тестах[Электронный ресурс] Режим доступа: http://nixx78.blogspot.com.by/p/blog-page.html - дата доступа: 17.04.2016
2. Wikipedia: Mokito [Электронный ресурс] Режим доступа: https://en.wikipedia.org/wiki/Mockito- дата доступа: 17.04.2016