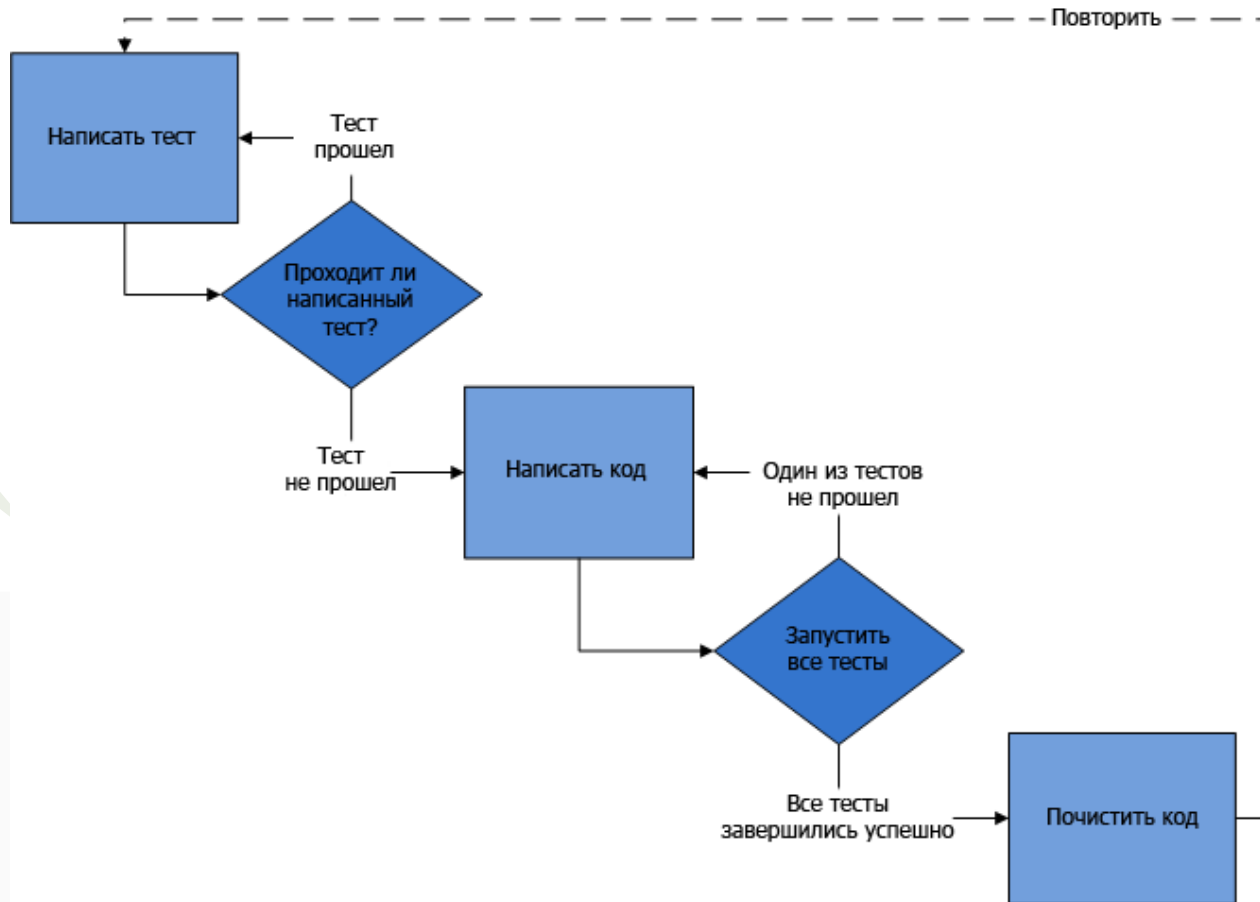


Глоток Moskito или ближе к КОДУ

Ткачук Алексей

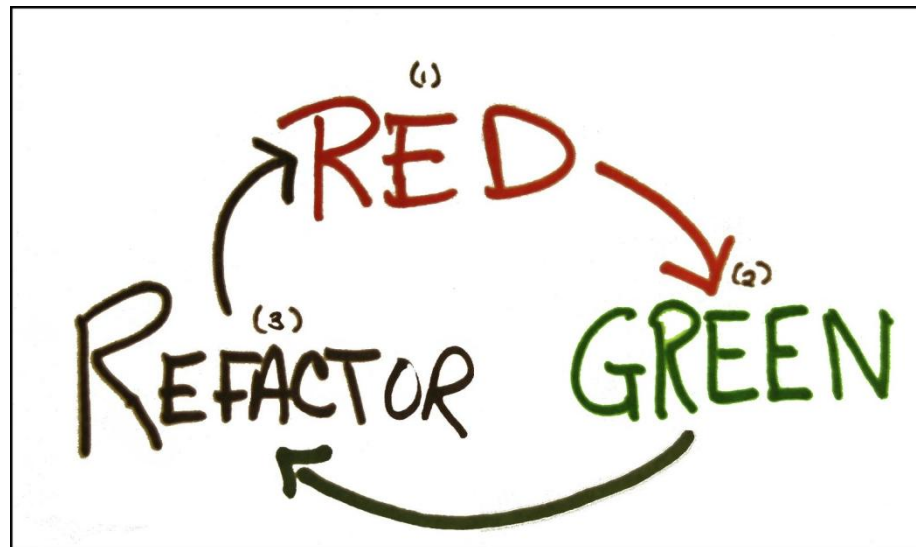


TDD или тот еще конвейер



Stylish

- «делай проще, дурачок» (keep it simple, stupid, KISS)
- «вам это не понадобится» (you ain't gonna need it, YAGNI)
- «подделай, пока не сделаешь» (fake it till you make it)



Погнали? Test #1

Когда пользователь вводит правильный идентификатор и пароль, состояние аккаунта меняется на «вошедший». Тестируем:

```
package present_mock;

import org.junit.Test;
import static org.mockito.Mockito.*;

public class LoginServiceTest {

    @Test
    public void itShouldSetAccountToLoggedInWhenPasswordMatches () {
        AccI account = mock(AccI.class);
        when(account.passMatch(anyString())).thenReturn(true);

        AccountRepoI accountRepository = mock(AccountRepoI.class);
        when(accountRepository.find(anyString())).thenReturn(account);

        LoginService service = new LoginService(accountRepository);

        service.login("brett", "password");

        verify(account, times(1)).setLogged(true);
    }
}
```

Начинаем с интерфейсов

```
package present_mock;
```

```
public class LoginService {
```

```
    public LoginService(AccountRepoI accountRepository) {
```

```
        public void login(String accId, String password) {
```

```
    }
```

```
package present_mock;
```

```
public interface AccountRepoI {
```

```
    AccI find(String accId);
```

```
}
```

```
package present_mock;
```

```
public interface AccI {
```

```
    void setLogged(boolean value);
```

```
    boolean passMatch(String candidate);
```

```
}
```

«Первая кровь»

No test passed, 1 test failed.(0,317 s)

```
present_mock.LoginServiceTest FAILED
  itShouldSetAccountToLoggedInWhenPasswordMatches FAILED: Wanted but not invoked: accI.setLogged(true); -> at present_mock.LoginServiceTest.itShouldSetAccountToLoggedInWhenPasswordMatches(LoginServiceTest.java:20) Actually, there were zero interactions with this mock.
    Wanted but not invoked:accI.setLogged(true);-> at present_mock.LoginServiceTest.itShouldSetAccountToLoggedInWhenPasswordMatches(LoginServiceTest.java:20)Actually, there were zero interactions with this mock.
    junit.framework.AssertionFailedError
    at present_mock.LoginServiceTest.itShouldSetAccountToLoggedInWhenPasswordMatches(LoginServiceTest.java:20)
```

org.mockito.exceptions.verificaton.WantedBut
NotInvoked:
Wanted but not invoked:
AccI.setLogged(true);
 at present_mock.LoginServiceTest.
itShouldSetAccountToLoggedInWhenPassword
Matches(LoginServiceTest.java)
Actually there were zero interactions with this
mock.

Refactor

Мы отправляем сообщение конкретному объекту AccI. LoginService получает аккаунт через его AccountRepoI (см. конструктор). Отсюда следует, что нам просто нужно запомнить конкретный объект AccRepoI и использовать его:

```
package present_mock;

public class LoginService {

    private final AccountRepoI repo;

    public LoginService(AccountRepoI accRepo) {
        this.repo = accRepo;
    }

    public void login(String accId, String password) {
        AccI account = repo.find(accId);
        account.setLogged(true);
    }
}
```

The test passed.(0,25 s)

present_mock.LoginServiceTest passed

Test #2

После 3 неудачных логинов «обнуляем» аккаунт пользователя (ограничения на количество попыток):

```
@Test
public void itShouldSetAccountToRevokedAfterThreeFailedLoginAttempts () {
    AccI account = mock(AccI.class);
    when(account.passMatch(anyString())) .thenReturn(false);

    AccountRepoI accountRepository = mock(AccountRepoI.class);
    when(accountRepository.find(anyString())) .thenReturn(account);

    LoginService service = new LoginService(accountRepository);

    for (int i = 0; i < 3; ++i) {
        service.login("brett", "password");
    }

    verify(account, times(1)).setRevoked(true);
}
```

Refactor

50,00 %

1 test passed, 1 test failed.(0,377 s)

present_mock.LoginServiceTest **FAILED**

itShouldSetAccountToRevokedAfterThreeFailedLoginAttempts **FAILED: Wanted but not invoked: accI.setRevoked(true); -> at present_mock.LoginServiceTest.itShouldSetAccountToRevokedAfterThreeFailedLoginAttempts(LoginServiceTest.java:37)**

Wanted but not invoked:accI.setRevoked(true);-> at present_mock.LoginServiceTest.itShouldSetAccountToRevokedAfterThreeFailedLoginAttempts(LoginServiceTest.java:37)

junit.framework.AssertionFailedError

at present_mock.LoginServiceTest.itShouldSetAccountToRevokedAfterThreeFailedLoginAttempts(LoginServiceTest.java:37)

Ошибка похожа на то, что уже было раньше, соответственно, действуем аналогично, но еще добавляем метод `setRevoked`:

```
private final AccountRepoI repo;

public void login(String accountId, String password) {
    AccI account = repo.find(accountId);
    account.setLogged(true);
    if (!account.passMatch(password))
        ++failedAttempts;
    if (failedAttempts == 3)
        account.setRevoked(true);
}
```

Both tests passed.(0,267 s)

present_mock.LoginServiceTest **passed**

Стремимся к прекрасному

```
public class LoginServiceTest {
    private AccI account;
    private AccountRepoI accountRepository;
    private LoginService service;

    @Before
    public void init() {
        account = mock(AccI.class);
        accountRepository = mock(AccountRepoI.class);
        when(accountRepository.find(anyString())).thenReturn(account);
        service = new LoginService(accountRepository);
    }

    private void willPasswordMatch(boolean value) {
        when(account.passMatch(anyString())).thenReturn(value);
    }

    @Test
    public void itShouldSetAccountToLoggedInWhenPasswordMatches() {
        willPasswordMatch(true);
        service.login("brett", "password");
        verify(account, times(1)).setLogged(true);
    }

    @Test
    public void itShouldSetAccountToRevokedAfterThreeFailedLoginAttempts() {
        willPasswordMatch(false);

        for (int i = 0; i < 3; ++i)
            service.login("brett", "password");

        verify(account, times(1)).setRevoked(true);
    }
}
```

Test #3

Тест заключается в том, что пользователь, который ввел неверный пароль, не должен «залогиниться».

```
@Test
public void itShouldNotSetAccountLoggedInIfPasswordDoesNotMatch() {
    willPasswordMatch(false);
    service.login("brett", "password");
    verify(account, never()).setLoggedIn(true);
}
```

Test #3

С помощью первых двух тестов мы добились
значительного прогресса. Вызван «нежелательный» метод 😊
Меняем метод login.

2 tests passed, 1 test failed.(0,362 s)

present_mock.LoginServiceTest FAILED
itShouldNotSetAccountLoggedInIfPasswordDoesNotMatch FAILED: accI.setLogged(true); Never wanted here:

```
public void login(String accountId, String password) {  
    AccI account = repo.find(accountId);  
  
    if (account.passMatch(password)) {  
        account.setLogged(true);  
    } else {  
        ++failedAttempts;  
    }  
  
    if (failedAttempts == 3) {  
        account.setRevoked(true);  
    }  
}
```

All 3 tests passed.(0,265 s)

present_mock.LoginServiceTest passed

Test #4

Создаем аккаунт, который не сможет пройти проверку при любом вводе пароля с целью залогиниться. Пытаемся логиниться с двух аккаунтов (2 раза с одного и 1 раз с другого). Все попытки не проходят, но ни один из аккаунтов не должен быть «обнулен». Проверяем, что аккаунт не был «обнулен».

Ошибка один в один той, которая была в предыдущем тесте.

```
@Test
public void itShouldNotRevokeSecondAccountAfterTwoFailedAttemptsFirstAccount() {
    willPasswordMatch(false);

    AccI secondAccount = mock(AccI.class);
    when(secondAccount.passMatch(anyString())).thenReturn(false);
    when(accountRepository.find("schuchert")).thenReturn(secondAccount);

    service.login("brett", "password");
    service.login("brett", "password");
    service.login("schuchert", "password");

    verify(secondAccount, never()).setRevoked(true);
}
```

3 tests passed, 1 test failed.(0,368 s)

present_mock.LoginServiceTest FAILED

itShouldNotRevokeSecondAccountAfterTwoFailedAttemptsFirstAccount FAILED: accI.setRevoked(true); Never wanted here:

Refactoring

```
public void login(String accountId, String password) {
    AccI account = repo.find(accountId);

    if (account.passMatch(password)) {
        account.setLogged(true);
    } else {
        if (previAccId.equals(accountId))
            ++failedAttempts;
        else {
            failedAttempts = 1;
            previAccId = accountId;
        }
    }

    if (failedAttempts == 3)
        account.setRevoked(true);
}
```



All 4 tests passed.(0,267 s)

📦 ✓ present_mock.LoginServiceTest passed

Test #5

Пользователю нету смысла логиниться, если он уже это сделал.

```
@Test(expected = AccLoginLimitException.class)
public void itShouldNowAllowConcurrentLogins () {
    willPasswordMatch(true);
    when(account.isLogged()).thenReturn(true);
    service.login("brett", "password");
}
```

4 tests passed, 1 test failed.(0,277 s)

[-] [!] present_mock.LoginServiceTest FAILED
[+] [!] itShouldNowAllowConcurrentLogins FAILED: Expected exception: present_mock.AccLoginLimitException

Refactoring

- Добавляем новый класс:

```
public class AccLoginLimitException extends RuntimeException{  
    private static long cons = 1L;  
}
```

- Меняем функцию login:

```
AccI account = repo.find(accountId);  
  
if (account.passMatch(password)) {  
    if (account.isLogged()) {  
        throw new AccLoginLimitException();  
    }  
    account.setLogged(true);  
}
```

- Success!!!



All 5 tests passed. (0,28 s)

present_mock.LoginServiceTest passed

Test #6

Предположим, что при поиске в репозитории аккаунт НЕВОЗМОЖНО найти.

```
@Test(expected = AccountDExistException.class)
public void ItShouldThrowExceptionIfAccountNotFound() {
    when(accountRepository.find("schuchert")).thenReturn(null);
    service.login("schuchert", "password");
}
```

5 tests passed, 1 test caused an error.(0,289 s)

present_mock.LoginServiceTest FAILED

ItShouldThrowExceptionIfAccountNotFound caused an ERROR: Unexpected exception,

Refactoring

Реагируем на null в функции login:

```
if (account == null) {  
    throw new AccountDExistException();  
}
```



All 6 tests passed.(0,285 s)

present_mock.LoginServiceTest passed

Test #7

«Обнуленный» аккаунт не должен иметь
ВОЗМОЖНОСТИ ЗАЛОГИНИТЬСЯ:

```
@Test(expected = AccRevokedException.class)
public void ItShouldNotBePossibleToLogIntoRevokedAccount() {
    willPasswordMatch(true);
    when(account.isRevoked()).thenReturn(true);
    service.login("brett", "password");
}
```

Все абсолютно аналогично, по этому просто
добавляем код:

```
if (account.isRevoked()) {
    throw new AccRevokedException();
}
```

All 7 tests passed. (0,38 s)

  present_mock.LoginServiceTest passed

Предварительные итоги

Все хорошо, но остается две проблемы:

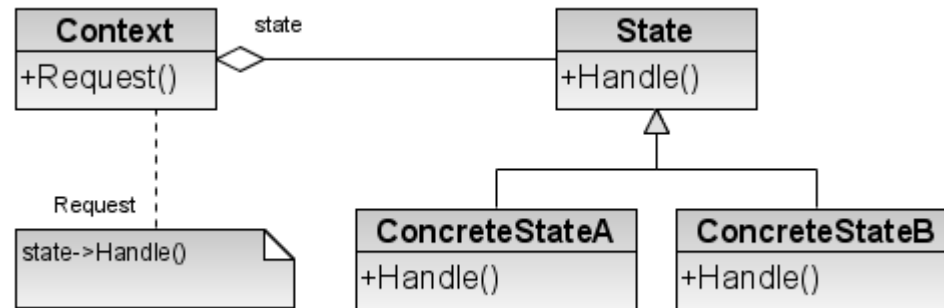
1. LoginService проверяет разнообразные действия наших аккаунтов, так что последние должны больше «отвечать»
2. Ответы LoginService зависят от результата предыдущих действий.



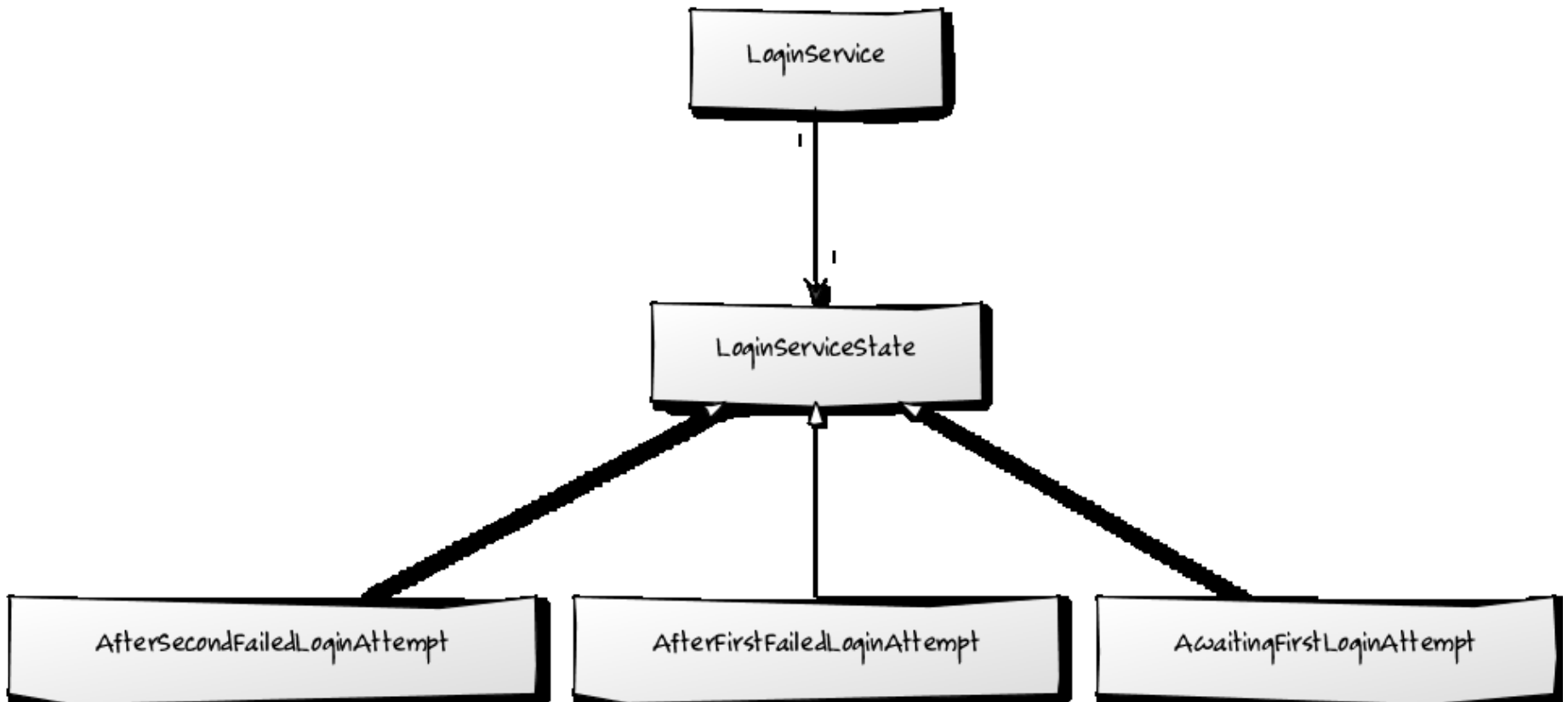
Состояние (шаблон проектирования)

Паттерн состоит из 3 блоков:

- **Widget** — класс, объекты которого должны менять свое поведение в зависимости от состояния.
- **IState** — интерфейс, который должно реализовать каждое из конкретных состояний.
- **StateA ... StateZ** — классы конкретных состояний.



Применяем шаблон



Refactoring

Создаем иерархию:

- Создаем абстрактный класс LoginServiceBase
- Создаем классы, которые наследуются (наши состояния):
 - WaitFirstLogin
 - FirstLoginFailed
 - SecondLoginFailed

» *Необходимо помнить, что рефакторинг – это улучшение структуры кода без влияния на его поведение. В нашем случае поведение определено тестами. По этому разобьем все на мелкие этапы, чтобы наш код был компилируемым и проходил все тесты.*

Общие шаги

Чтобы все компилировалось и тесты проходились следуем простым советам:

- Создаем место, куда будем добавлять новую функциональность
- Копируем (не вырезаем) функциональность из одного места в другое
- Компилируем
- Делигируем из начального места в новое

» Делегирование означает ситуацию, когда объект для предоставления определённого набора функциональности полагается на другой объект

Вставляем старый код

```
package present_mock;

public class LoginServiceBase {
    public void login(LoginService service, AccI account, String password) {
        if (account == null) {
            throw new AccountDExistException();
        }
        if (account.passMatch(password)) {
            if (account.isLogged()) {
                throw new AccLogInLimitException();
            }
            if (account.isRevoked()) {
                throw new AccRevokedException();
            }
            account.setLogged(true);
        } else {
            if (previAccId.equals(accountId)) {
                ++failedAttempts;
            } else {
                failedAttempts = 1;
                previAccId = accountId;
            }
        }

        if (failedAttempts == 3) {
            account.setRevoked(true);
        }
    }
}
```



Проблемы

1. prevAccId не существует: добавляем параметр функции
2. Не существует идентификатора аккаунта: исправляем геттером (а так же корректируем тесты)
3. Количество неуспешных попыток отсутствует: добавляем параметр

Прохождение тестов

После добавления метода getId в LoginService.login два теста «ложаться»:

```
else {  
    if (previAccId.equals(account.getId())) {  
        ++failedAttempts;  
    } else {  
        failedAttempts = 1;  
        previAccId = account.getId();  
    }  
}
```

5 tests passed, 2 tests caused an error. (0,403 s)

- [-] [!] present_mock.LoginServiceTest FAILED
 - [+] [!] itShouldSetAccountToRevokedAfterThreeFailedLoginAttempts caused an ERROR: java.lang.NullPointerException
 - [+] [!] itShouldNotRevokeSecondAccountAfterTwoFailedAttemptsFirstAccount caused an ERROR: java.lang.NullPointerException

Проходим все тесты

Добавляем строчку в функцию `init` в тестах:

```
account = mock(AccI.class);  
when(account.getId()).thenReturn("brett");
```

All 7 tests passed. (0,372 s)

...  present_mock.LoginServiceTest **passed**



Разделяй и властвуй

```
public void login(String accountId, String password) {
    AccI account = repo.find(accountId);

    if (account == null)
        throw new AccountDNotExistException();

    verifyLoginAttempt(account, password);
}

public void verifyLoginAttempt(AccI account, String password) {
    if (account.passMatch(password)) {
        if (account.isLogged()) {
            throw new AccLogInLimitException();
        }
        if (account.isRevoked()) {
            throw new AccRevokedException();
        }
        account.setLogged(true);
    } else {
        if (previAccId.equals(account.getId())) {
            ++failedAttempts;
        } else {
            failedAttempts = 1;
            previAccId = account.getId();
        }
    }

    if (failedAttempts == 3) {
        account.setRevoked(true);
    }
}

All 7 tests passed.(0,375 s)
present_mock.LoginServiceTest passed
```

Копирование кода

Копируем (с переименованием) `verifyLoginAttempt` в `LoginServiceState.login` и добавляем необходимые поля `prevAccId` и `failedAttempts`

```
package present_mock;

public abstract class LoginServiceBase {
    private String prevAccId = "";
    private int failedAttempts;

    public void login(AccI account, String password) {
        if (account.passMatch(password)) {
            if (account.isLogged())
                throw new AccLogInLimitException();
            if (account.isRevoked())
                throw new AccRevokedException();
            account.setLogged(true);
        } else {
            if (prevAccId.equals(account.getId()))
                ++failedAttempts;
            else {
                failedAttempts = 1;
                prevAccId = account.getId();
            }
        }

        if (failedAttempts == 3)
            account.setRevoked(true);
    }
}
```

All 7 tests passed. (0,368 s)

present_mock.LoginServiceTest passed

Делегирование

Добавляем отношение агрегации:

```
public class LoginService {  
    private LoginServiceBase state = new WaitFirstLogIn();  
}
```

И дописываем нашу многострадальную функцию:

```
public void login(String accountId, String password) {  
    AccI account = repo.find(accountId);  
  
    if (account == null) {  
        throw new AccountDExistException();  
    }  
  
    state.login(account, password);  
}
```

All 7 tests passed.(0,401 s)

present_mock.LoginServiceTest passed

Удаление кода

Удаляем метод `verifyLoginAttempt` и поля `failedAttempts`, `prevAcclId`:

```
package present_mock;

public class LoginService {
    private final AccountRepoI repo;
    private LoginServiceBase state = new WaitFirstLogIn();



    public LoginService(AccountRepoI repo) {
        this.repo = repo;
    }

    public void login(String accountId, String password) {
        AccI account = repo.find(accountId);

        if (account == null)
            throw new AccountDNotExistException();

        state.login(account, password);
    }
}
```

All 7 tests passed.(0,382 s)

  present_mock.LoginServiceTest passed

Готовимся к полиморфизму

LoginServiceState.login должен быть в каждом подклассе состояний (вспоминаем шаблон), после чего мы будем удалять код, который не относится к конкретному состоянию.

Копируем метод login в каждый подкласс, а сам метод делаем абстрактным в базовом классе.

Изменение состояний

```
package present_mock;

public abstract class LoginServiceBase {
    protected String prevAccId = "";
    protected int failedAttempts;


    public abstract void login(AccI account, String password);
}

package present_mock;

public class WaitFirstLogIn extends LoginServiceBase {
    @Override
    public void login(AccI account, String password) {
        if (account.passMatch(password)) {
            if (account.isLogged())
                throw new AccLogInLimitException();
            if (account.isRevoked())
                throw new AccRevokedException();
            account.setLogged(true);
        } else {
            if (prevAccId.equals(account.getId()))
                ++failedAttempts;
            else {
                failedAttempts = 1;
                prevAccId = account.getId();
            }
        }

        if (failedAttempts == 3)
            account.setRevoked(true);
    }
}
```

All 7 tests passed.(0,287 s)

  present_mock.LoginServiceTest passed

Дальнейшие планы

Сейчас целью является:

1. Удалить лишний код в каждом состоянии
2. «Переходить» из одного состояния в другое

Чтобы реализовать последнее есть два возможных варианта:

1. Возвращать следующее состояние
2. **Устанавливать в LoginService подходящие объекты состояний**

Задаем состояния (ожидание первого входа)

```
public class WaitFirstLogIn extends LoginServiceBase {

    @Override
    public void login(AccI account, String password, LoginService service) {
        if (account.passMatch(password)) {
            if (account.isLogged()) {
                throw new AccLogInLimitException();
            }
            if (account.isRevoked()) {
                throw new AccRevokedException();
            }
            account.setLogged(true);
        } else {
            service.setState(new FirstLogInFailed(account.getId()));
        }
    }

    public void setState(LoginServiceBase state) {
        this.state = state;
    }
}
```

Задаем состояния (после первой попытки входа)

```
public class FirstLogInFailed extends LoginServiceBase {

    @Override
    public void login(AccI account, String password, LoginService service) {
        if (account.passMatch(password)) {
            if (account.isLogged()) {
                throw new AccLogInLimitException();
            }
            if (account.isRevoked()) {
                throw new AccRevokedException();
            }
            account.setLogged(true);
        } else {
            if (prevAccId.equals(account.getId())) {
                service.setState(new SecondLogInFailed(account.getId()));
            } else {
                prevAccId = account.getId();
            }
        }
    }

    public FirstLogInFailed(String previousAccountId) {
        this.prevAccId = previousAccountId;
        failedAttempts = 1;
    }
}
```

Задаем состояния (после второй попытки входа)

```
public class SecondLogInFailed extends LoginServiceBase {  
  
    @Override  
    public void login(AccI account, String password, LoginService service) {  
        if (account.passMatch(password)) {  
            if (account.isLogged()) {  
                throw new AccLogInLimitException();  
            }  
            if (account.isRevoked()) {  
                throw new AccRevokedException();  
            }  
            account.setLogged(true);  
        } else {  
            if (prevAccId.equals(account.getId())) {  
                account.setRevoked(true);  
                service.setState(new WaitFirstLogIn());  
            } else {  
                service.setState(new FirstLogInFailed(account.getId()));  
            }  
        }  
    }  
  
    public SecondLogInFailed(String prevAccId) {  
        this.prevAccId = prevAccId;  
        failedAttempts = 2;  
    }  
}
```

All 7 tests passed.(0,285 s)

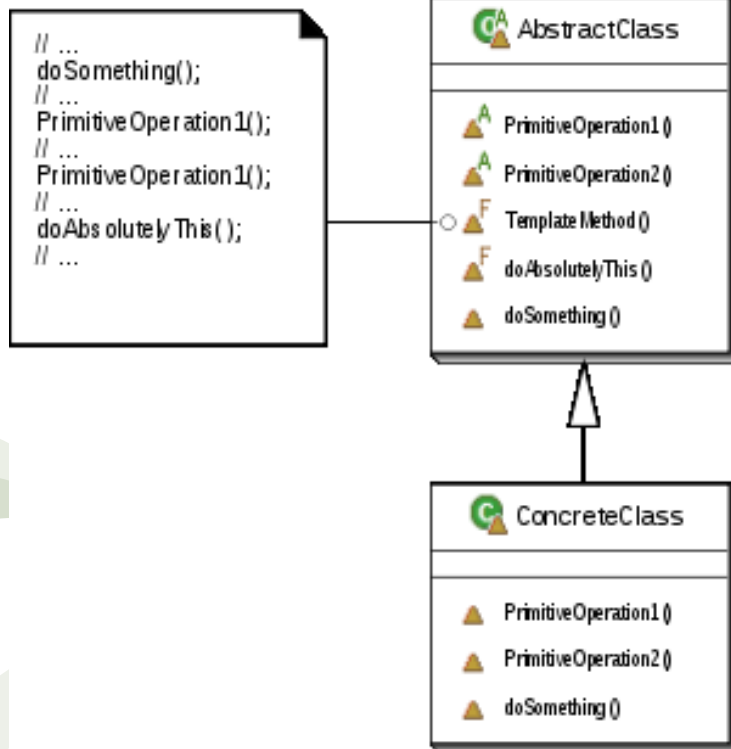
present_mock.LoginServiceTest passed

Применяем шаблонный метод (шаблон проектирования)

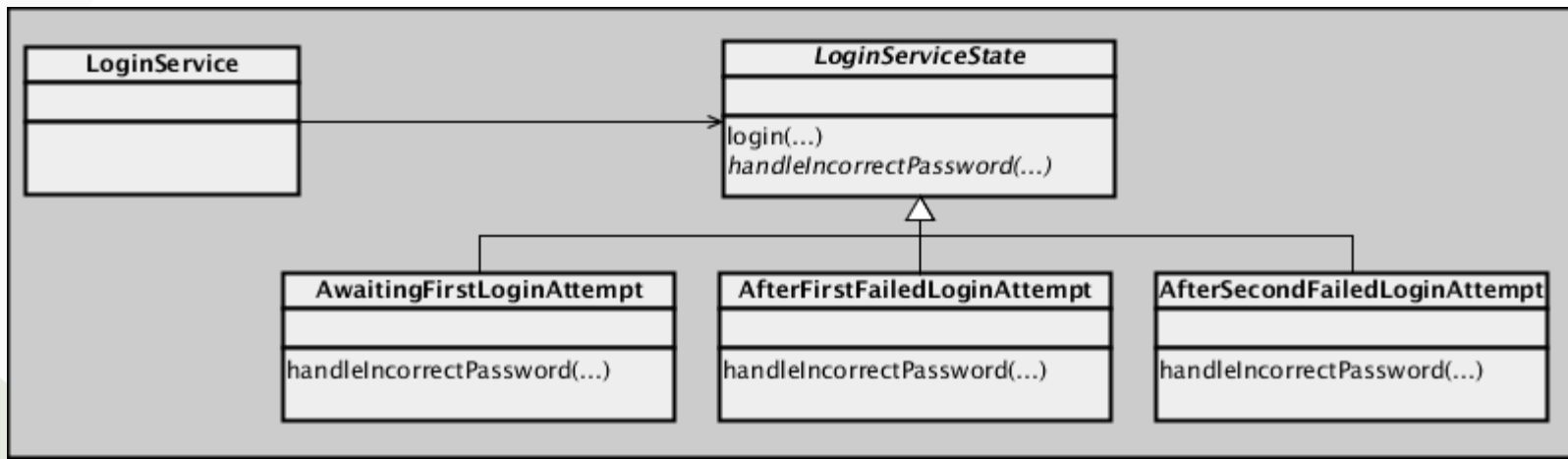
Состоит из:

1. **Abstract class** (абстрактный класс) - определяет абстрактные операции, замещаемые в наследниках для реализации шагов алгоритма
2. **Concrete class** (конкретный класс) - реализует замещаемые операции необходимым для данной реализации способом.

Общая схема шаблона



Применение



Мы почти у цели

```
package present_mock;

public abstract class LoginServiceBase {

    public final void login(LoginService service, AccI account,
        String password) {
        if (account.passMatch(password)) {
            if (account.isLogged()) {
                throw new AccLogInLimitException();
            }
            if (account.isRevoked()) {
                throw new AccRevokedException();
            }
            account.setLogged(true);
        } else {
            handleIncorrectPassword(service, account, password);
        }
    }

    public abstract void handleIncorrectPassword(LoginService service,
        AccI account, String password);
}
```

```
package present_mock;

public class WaitFirstLogIn extends LoginServiceBase {

    @Override
    public void handleIncorrectPassword(LoginService context, AccI account,
        String password) {
        context.setState(new FirstLogInFailed(account.getId()));
    }
}
```

```

package present_mock;

public class FirstLogInFailed extends LoginServiceBase {

    private String prevAccId;

    public FirstLogInFailed(String previousAccountId) {
        this.prevAccId = previousAccountId;
    }

    @Override
    public void handleIncorrectPassword(LoginService context, AccI account,
        String password) {
        if (prevAccId.equals(account.getId())) {
            context.setState(new SecondLogInFailed(account.getId()));
        } else {
            prevAccId = account.getId();
        }
    }
}

package present_mock;

public class SecondLogInFailed extends LoginServiceBase {

    private String prevAccId;

    public SecondLogInFailed(String previousAccountId) {
        this.prevAccId = previousAccountId;
    }

    @Override
    public void handleIncorrectPassword(LoginService context, AccI account,
        String password) {
        if (prevAccId.equals(account.getId())) {
            account.setRevoked(true);
            context.setState(new WaitFirstLogIn());
        } else {
            context.setState(new FirstLogInFailed(account.getId()));
        }
    }
}

```

All 7 tests passed.(0,277 s)





present_mock.LoginServiceTest passed

Последний тест

После успешного входа нужно переводить аккаунт в «нормальное» состояния

```
@Test
public void itShouldResetBackToInitialStateAfterSuccessfulLogin() {
    willPasswordMatch(false);
    service.login("brett", "password");
    service.login("brett", "password");
    willPasswordMatch(true);
    service.login("brett", "password");
    willPasswordMatch(false);
    service.login("brett", "password");
    verify(account, never()).setRevoked(true);
}
```

7 tests passed, 1 test failed.(0,444 s)

  present_mock.LoginServiceTest FAILED
  itShouldResetBackToInitialStateAfterSuccessfulLogin

Последние изменения кода

- В LoginServiceBase:

```
account.setLogged(true);  
service.setState(new WaitFirstLogIn());
```

All 8 tests passed.(0,444 s)

🔍...✅ present_mock.LoginServiceTest passed



Спасибо за внимание
Есть вопросы?