Домашнее задание 6.

Дудников Кирилл.

**Отчет об обнаруженных проблемах с исправлениями.**

Здесь и в прикрепленном проекте ошибки исправляются потому, что нужно же как-то проверить все требования, а что делать, если самый первый же метод callLogin написан ошибочно, и не дает проверить все остальное.

Ссылка на этот же документ, но в Google Docs: <https://docs.google.com/document/d/1TFtRkDSnf6XrPrOjb0LJdWoPzGuGybpes2vbTHDlw0I/edit?usp=sharing>

**Формат:**

1. код до исправления;
2. данные, на которых наблюдается некорректное поведение;
3. полученное значение, ожидаемое значение;
4. код после исправлений.

**Примечания:**

* Все методы располагаются в классе root.integrat.AccountManager.
* Во входных данных:   
  am - это spy-объект для класса AccountManager,   
  server - это заглушка для интерфейса IServer.

**Внимание!** В ТЗ есть некоторое несоответствия.

1) Написано, что:

*“INCORRECT\_RESPONCE = 4 — возвращается, когда от сервера пришел*

*некорректный ответ*

*UNDEFINED\_ERROR = 5 — возвращается при возникновении непредусмотренной*

*ошибки”*

Однако ни в одном методе в ТЗ не упоминается возвращение INCORRECT\_RESPONCE. И в callLogin, и в withdraw и в др. методах используется только UNDEFINED\_ERROR. То есть INCORRECT\_RESPONCE - это упомянутая, но неиспользуемая в ТЗ информация. А в коде используется только INCORRECT\_RESPONCE, а UNDEFINED\_ERROR - нигде. Так что формально я считаю, что ТЗ правильно, и значит, в строках 29, 42, 66, 89, 107 - ошибка - должно возвращаться UNDEFINED\_ERROR, а не INCORRECT\_RESPONCE.

2) В “описании методов” класса AccountManager во всех методах написано, что есть возвращаемый код “SUCCESS”. Однако ниже, в AccountManagerResponse, кода SUCCESS нет, есть только SUCCEED. Я понимаю, что, в отличие от предыдущего несоответствия, это скорее всего опечатка, а не ошибка ТЗ. Но все равно.

Также хотел сказать, что в коде есть места, куда просто нельзя зайти методом черного ящика, то есть не изменяя значения переменных через рефлексию. Это, например, ветка case NOT\_LOGGED в switch в методе callLogout и другие подобные.

**№ 1.**

**Метод callLogin(String, String):**

| **public** AccountManagerResponse **callLogin**(String login, String password) {  Long session = activeAccounts.get(login);  **if**(session!=**null**)  **return** AccountManagerResponse.ACCOUNT\_MANAGER\_RESPONSE;  ServerResponse ret = server.login(login, makeSecure(password));  **switch**(ret.code) {  **case** ServerResponse.ALREADY\_LOGGED:  **return** AccountManagerResponse.ACCOUNT\_MANAGER\_RESPONSE;  **case** ServerResponse.NO\_USER\_INCORRECT\_PASSWORD:  **return** AccountManagerResponse.NO\_USER\_INCORRECT\_PASSWORD\_RESPONSE;  **case** ServerResponse.SUCCESS:{  Object answ = ret.response;  **if**(answ **instanceof** Long)  **return** **new** AccountManagerResponse(AccountManagerResponse.SUCCEED, answ);  **break**;   }  }  **return** **new** AccountManagerResponse(AccountManagerResponse.INCORRECT\_RESPONSE, ret);  } |
| --- |

Входные данные 1:

| doReturn("hash1").when(am).makeSecure("pass1");  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, 322L)).when(server).login("user1", "hash1");  am.init(server);  AccountManagerResponse res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.SUCCEED, res.code);  assertEquals(322L, res.response);  res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.ALREADY\_LOGGED, res.code); |
| --- |

Ожидаемое значение: AccountManagerResponse.ALREADY\_LOGGED

Полученное значение: AccountManagerResponse.SUCCEED

Входные данные 2:

| doReturn("hash1").when(am).makeSecure("pass1");  doReturn(**new** ServerResponse(ServerResponse.UNDEFINED\_ERROR, **null**)).when(server).login("user1", "hash1");  am.init(server);  AccountManagerResponse res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.UNDEFINED\_ERROR, res.code); |
| --- |

Ожидаемое значение: AccountManagerResponse.UNDEFINED\_ERROR

Полученное значение: AccountManagerResponse.INCORRECT\_RESPONSE

Код после исправлений:

| **public** AccountManagerResponse **callLogin**(String login, String password) {  Long session = activeAccounts.get(login);  **if**(session!=**null**)  **return** AccountManagerResponse.ACCOUNT\_MANAGER\_RESPONSE;  ServerResponse ret = server.login(login, makeSecure(password));  **switch**(ret.code) {  **case** ServerResponse.ALREADY\_LOGGED:  **return** AccountManagerResponse.ACCOUNT\_MANAGER\_RESPONSE;  **case** ServerResponse.NO\_USER\_INCORRECT\_PASSWORD:  **return** AccountManagerResponse.NO\_USER\_INCORRECT\_PASSWORD\_RESPONSE;  **case** ServerResponse.SUCCESS:{  Object answ = ret.response;  **if**(answ **instanceof** Long) {  activeAccounts.put(login, (Long) answ);  **return** **new** AccountManagerResponse(AccountManagerResponse.SUCCEED, answ);  }  **break**;   }  }  **return** **new** AccountManagerResponse(AccountManagerResponse.UNDEFINED\_ERROR, ret);  } |
| --- |

**№ 2.**

**Метод callLogout(String, Long):**

| **public** AccountManagerResponse **callLogout**(String user, **long** session) {  Long rem = activeAccounts.remove(user);  **if**(rem == **null**)  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  ServerResponse resp = server.logout(session);  **switch**(resp.code){  **case** ServerResponse.NOT\_LOGGED:  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **case** ServerResponse.SUCCESS:  **return** AccountManagerResponse.SUCCEED\_RESPONSE;  }  **return** **new** AccountManagerResponse(AccountManagerResponse.INCORRECT\_RESPONSE, resp);    } |
| --- |

Входные данные 1:

| doReturn("hash1").when(am).makeSecure("pass1");  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, 321L)).when(server).login("user1", "hash1");  doReturn(**new** ServerResponse(ServerResponse.NOT\_LOGGED, **null**)).when(server).logout(666L);  am.init(server);  AccountManagerResponse res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.SUCCEED, res.code);  assertEquals(321L, res.response);  res = am.callLogout("user1", 666L);  assertEquals(AccountManagerResponse.INCORRECT\_SESSION, res.code); |
| --- |

Ожидаемое значение: AccountManagerResponse.INCORRECT\_SESSION

Полученное значение: AccountManagerResponse.NOT\_LOGGED

Входные данные 2:

Пояснение: Сначала я логиню пользователя user1 под сессией 321. После этого вызываю callLogout с пользователем user1 и некорректной сессий 666. После исправления выше мне вернется INCORRECT\_SESSION. После этого я попытаюсь выйти уже с корректной сессией 321 и получу ответ NOT\_LOGGED, хотя должно мне вернуться SUCCEED. Это происходит потому, что в коде callLogout сначала удаляется из словаря значения пользователя с сессией и только потом проверяется, а работает ли logout на сервере. И когда оказывается, что у пользователя вообще другая сессия, на сервере эта сессия все еще работает, а в менеджере уже удалена. Это в будущем приведет к тому, что я больше никогда не смогу пользоваться этим пользователем, т.к. в менеджере его нет, а сервер считает, что сессия уже открыта. Поэтому нужно сначала отправлять запрос к серверу, а потом, если все разлогинилось, уже удалять на менеджере. Ну или просто сначала проверять корректность сессии в словаре удаления.

| doReturn("hash1").when(am).makeSecure("pass1");  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, 321L)).when(server).login("user1", "hash1");  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, **null**)).when(server).logout(321L);  doReturn(**new** ServerResponse(ServerResponse.NOT\_LOGGED, **null**)).when(server).logout(666L);  am.init(server);  AccountManagerResponse res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.SUCCEED, res.code);  assertEquals(321L, res.response);  res = am.callLogout("user1", 666L);  assertEquals(AccountManagerResponse.INCORRECT\_SESSION, res.code);  assertNull(res.response);  res = am.callLogout("user1", 321L);  assertEquals(AccountManagerResponse.SUCCEED, res.code); |
| --- |

Ожидаемое значение: AccountManagerResponse.SUCCEED

Полученное значение: AccountManagerResponse.NOT\_LOGGED

Входные данные 3:

| doReturn("hash1").when(am).makeSecure("pass1");  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, 321L)).when(server).login("user1", "hash1");  doReturn(**new** ServerResponse(ServerResponse.UNDEFINED\_ERROR, **null**)).when(server).logout(321L);  am.init(server);  AccountManagerResponse res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.SUCCEED, res.code);  assertEquals(321L, res.response);  res = am.callLogout("user1", 321L);  assertEquals(AccountManagerResponse.UNDEFINED\_ERROR, res.code); |
| --- |

Ожидаемое значение: AccountManagerResponse.UNDEFINED\_ERROR

Полученное значение: AccountManagerResponse.INCORRECT\_RESPONSE

Код после исправлений:

| **public** AccountManagerResponse **callLogout**(String user, **long** session) {  Long correctSession = activeAccounts.get(user);  **if**(correctSession == **null**)  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **if**(correctSession != session)  **return** AccountManagerResponse.INCORRECT\_SESSION\_RESPONSE;  ServerResponse resp = server.logout(session);  **switch**(resp.code){  **case** ServerResponse.SUCCESS:  **return** AccountManagerResponse.SUCCEED\_RESPONSE;  }  **return** **new** AccountManagerResponse(AccountManagerResponse.UNDEFINED\_ERROR, resp);  } |
| --- |

**№ 3.**

**Метод getBalance(String, long):**

| **public** AccountManagerResponse **getBalance**(String login, **long** session){  Long stored = activeAccounts.get(login);  **if**(stored == **null**)  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **if**(stored!=session)  **return** AccountManagerResponse.INCORRECT\_SESSION\_RESPONSE;  ServerResponse resp = server.getBalance(session);  **switch**(resp.code){  **case** ServerResponse.NOT\_LOGGED:  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **case** ServerResponse.SUCCESS:  Object r = resp.response;  **if**(r!=**null** && r **instanceof** Double)  **return** **new** AccountManagerResponse(AccountManagerResponse.SUCCEED, (Double)r);  **break**;  }  **return** **new** AccountManagerResponse(AccountManagerResponse.INCORRECT\_RESPONSE, resp);  }  **protected** **abstract** String **makeSecure**(String password); } |
| --- |

Входные данные:

| doReturn("hash1").when(am).makeSecure("pass1");  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, 111L)).when(server).login("user1", "hash1");  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, **null**)).when(server).logout(111L);  doReturn(**new** ServerResponse(ServerResponse.UNDEFINED\_ERROR, **null**)).when(server).getBalance(111L);  am.init(server);  AccountManagerResponse res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.SUCCEED, res.code);  assertTrue(res.response **instanceof** Long);  assertEquals(111L, res.response);   res = am.getBalance("user1", 111L);  assertEquals(AccountManagerResponse.UNDEFINED\_ERROR, res.code); |
| --- |

Ожидаемое значение:

Полученное значение:

Код после исправлений:

| **public** AccountManagerResponse **getBalance**(String login, **long** session){  Long stored = activeAccounts.get(login);  **if**(stored == **null**)  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **if**(stored!=session)  **return** AccountManagerResponse.INCORRECT\_SESSION\_RESPONSE;  ServerResponse resp = server.getBalance(session);  **switch**(resp.code){  **case** ServerResponse.NOT\_LOGGED:  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **case** ServerResponse.SUCCESS:  Object r = resp.response;  **if**(r!=**null** && r **instanceof** Double)  **return** **new** AccountManagerResponse(AccountManagerResponse.SUCCEED, (Double)r);  **break**;  }  **return** **new** AccountManagerResponse(AccountManagerResponse.UNDEFINED\_ERROR, resp);  } |
| --- |

**№ 4.**

**Метод deposit(String, long, double):**

| **public** AccountManagerResponse **deposit**(String login, **long** session, **double** amount){  Long stored = activeAccounts.get(login);  **if**(stored == **null**)  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **if**(stored!=session)  **return** AccountManagerResponse.INCORRECT\_SESSION\_RESPONSE;  ServerResponse resp = server.deposit(session, amount);  **switch**(resp.code){  **case** ServerResponse.NOT\_LOGGED:  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **case** ServerResponse.NO\_MONEY:  Object r = resp.response;  **if**(r!=**null** && r **instanceof** Double)  **return** **new** AccountManagerResponse(AccountManagerResponse.NO\_MONEY, (Double)r);  **break**;  **case** ServerResponse.SUCCESS:  r = resp.response;  **if**(r!=**null** && r **instanceof** Double)  **return** **new** AccountManagerResponse(AccountManagerResponse.SUCCEED, (Double)r);  **break**;  }  **return** **new** AccountManagerResponse(AccountManagerResponse.INCORRECT\_RESPONSE, resp);  } |
| --- |

Входные данные 1:

| doReturn("hash1").when(am).makeSecure("pass1");  when(server.login("user1", "hash1")).thenReturn(**new** ServerResponse(ServerResponse.SUCCESS, 111L));  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, **null**)).when(server).logout(111L);  when(server.deposit(111L, 28999.95))  .thenReturn(**new** ServerResponse(ServerResponse.NO\_MONEY, 10.1));  am.init(server);  AccountManagerResponse res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.SUCCEED, res.code);  assertTrue(res.response **instanceof** Long);  assertEquals(111L, res.response);   res = am.deposit("user1", 111L, 28999.95);  assertEquals(AccountManagerResponse.UNDEFINED\_ERROR, res.code); |
| --- |

Ожидаемое значение: AccountManagerResponse.UNDEFINED\_ERROR

Полученное значение: AccountManagerResponse.NO\_MONEY

Входные данные 2:

| doReturn("hash1").when(am).makeSecure("pass1");  when(server.login("user1", "hash1")).thenReturn(**new** ServerResponse(ServerResponse.SUCCESS, 111L));  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, **null**)).when(server).logout(111L);  when(server.deposit(111L, 28999.95)).thenReturn(**new** ServerResponse(ServerResponse.UNDEFINED\_ERROR, **null**));  am.init(server);  AccountManagerResponse res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.SUCCEED, res.code);  assertTrue(res.response **instanceof** Long);  assertEquals(111L, res.response);   res = am.deposit("user1", 111L, 28999.95);  assertEquals(AccountManagerResponse.UNDEFINED\_ERROR, res.code); |
| --- |

Ожидаемое значение: AccountManagerResponse.UNDEFINED\_ERROR

Полученное значение: AccountManagerResponse.INCORRECT\_RESPONSE

Код после исправлений:

| **public** AccountManagerResponse **deposit**(String login, **long** session, **double** amount){  Long stored = activeAccounts.get(login);  **if**(stored == **null**)  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **if**(stored!=session)  **return** AccountManagerResponse.INCORRECT\_SESSION\_RESPONSE;  ServerResponse resp = server.deposit(session, amount);  **switch**(resp.code){  **case** ServerResponse.NOT\_LOGGED:  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **case** ServerResponse.SUCCESS:  Object r = resp.response;  **if**(r!=**null** && r **instanceof** Double)  **return** **new** AccountManagerResponse(AccountManagerResponse.SUCCEED, (Double)r);  **break**;  }  **return** **new** AccountManagerResponse(AccountManagerResponse.UNDEFINED\_ERROR, resp);  } |
| --- |

**№ 5.**

**Метод withdraw(String, long, double):**

| **public** AccountManagerResponse **withdraw**(String login, **long** session, **double** amount){  Long stored = activeAccounts.get(login);  **if**(stored == **null**)  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **if**(stored!=session)  **return** AccountManagerResponse.INCORRECT\_SESSION\_RESPONSE;  ServerResponse resp = server.withdraw(session, amount);  **switch**(resp.code){  **case** ServerResponse.NOT\_LOGGED:  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **case** ServerResponse.NO\_MONEY:  Object r = resp.response;  **if**(r!=**null** && r **instanceof** Double)  **return** **new** AccountManagerResponse(AccountManagerResponse.NO\_MONEY, (Double)r);  **break**;  **case** ServerResponse.SUCCESS:  r = resp.response;  **if**(r!=**null** && r **instanceof** Double)  **return** **new** AccountManagerResponse(AccountManagerResponse.SUCCEED, (Double)r);  **break**;  }  **return** **new** AccountManagerResponse(AccountManagerResponse.INCORRECT\_RESPONSE, resp);  } |
| --- |

Входные данные:

| doReturn("hash1").when(am).makeSecure("pass1");  when(server.login("user1", "hash1")).thenReturn(**new** ServerResponse(ServerResponse.SUCCESS, 111L));  doReturn(**new** ServerResponse(ServerResponse.SUCCESS, **null**)).when(server).logout(111L);  when(server.withdraw(111L, 999999.23232)).thenReturn(**new** ServerResponse(ServerResponse.UNDEFINED\_ERROR, **null**));  am.init(server);  AccountManagerResponse res = am.callLogin("user1", "pass1");  assertEquals(AccountManagerResponse.SUCCEED, res.code);  assertTrue(res.response **instanceof** Long);  assertEquals(111L, res.response);   res = am.withdraw("user1", 111L, 999999.23232);  assertEquals(AccountManagerResponse.UNDEFINED\_ERROR, res.code); |
| --- |

Ожидаемое значение: AccountManagerResponse.UNDEFINED\_ERROR

Полученное значение: AccountManagerResponse.INCORRECT\_RESPONSE

Код после исправлений:

| **public** AccountManagerResponse **withdraw**(String login, **long** session, **double** amount){  Long stored = activeAccounts.get(login);  **if**(stored == **null**)  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **if**(stored!=session)  **return** AccountManagerResponse.INCORRECT\_SESSION\_RESPONSE;  ServerResponse resp = server.withdraw(session, amount);  **switch**(resp.code){  **case** ServerResponse.NOT\_LOGGED:  **return** AccountManagerResponse.NOT\_LOGGED\_RESPONSE;  **case** ServerResponse.NO\_MONEY:  Object r = resp.response;  **if**(r!=**null** && r **instanceof** Double)  **return** **new** AccountManagerResponse(AccountManagerResponse.NO\_MONEY, (Double)r);  **break**;  **case** ServerResponse.SUCCESS:  r = resp.response;  **if**(r!=**null** && r **instanceof** Double)  **return** **new** AccountManagerResponse(AccountManagerResponse.SUCCEED, (Double)r);  **break**;  }  **return** **new** AccountManagerResponse(AccountManagerResponse.UNDEFINED\_ERROR, resp);  } |
| --- |