**ЛАБОРАТОРНАЯ РАБОТА №5**

по дисциплине «Модульное тестирование»

«**РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ (TDD)»**

Автор: студент гр. 10701118 Воробей И.А.

Проверил: доцент, к.т.н. Попова Ю.Б.

**Цель работы**: закрепление практических навыков по использованию технологии TestDrivenDevelopment.

**Изучаемые вопросы:**

разработка через тестирование;

циклы разработки через тестирование;

общая структура тестов;

Mock;

Stub;

Spy;

Fake;

NUnit, JUnit, DUnit, QUnit.

**Задание:**

1. Ознакомиться с основами технологии TDD.

2. Разработать необходимые интерфейсы, классы, модули без реализации (**при разработке необходимо придерживаться правил и принципов кодирования на языках *C#, Java, Delphi, JavaScript*).**

3. Разработать тесты для предполагаемых методов, продумав, как они будут работать (**10 непохожих тестов с использованием технологии Mock, предусмотрев положительные и отрицательные тесты, реагирующие на исключения).**

4. Запустить разработанные тесты. Ни один тест не должен пройти. Занести результаты пункта в отчет в виде копий экранов с не пройденными тестами и листингом кода.

5. Произвести разработку кода, не прибегая к изменению тестов. Запустить тесты. Занести результаты пункта в отчет в виде копий экранов с результатами тестов и листингом разработанного кода.

6. Произвести рефакторинг кода и тестов. В отчет занести изменения по коду и комментарии к ним.

7. Сделать выводы по работе.

Буду разрабатывать класс BidDaoImpl.

Класс имеет предка GenericDao (в листинге не привожу).

GenericDao содержит protected методы обращение к БД.

Эти методы принимают запросы БД в виде строк и параметры PreparedStatement.

Кроме того методы могут принимать объект Connection, когда необходима транзакция.

Класс перед началом разработки:

public class BidDaoImpl extends GenericDao<Bid> implements BidDao {  
 public BidDaoImpl() {  
 super(new BidBoulder());  
 }  
  
 @Override  
 public void placeTopUpBid(Long accountId, Long amount, BigDecimal amountInChosenCurrency,  
 CustomCurrency currency, String message) throws DaoException {  
 }  
  
 @Override  
 public void placeWithdrawBid(Long accountId, Long amount, BigDecimal amountInChosenCurrency,  
 CustomCurrency currency, String message) throws DaoException {  
 }  
  
 @Override  
 public List<Bid> findInProgressBids(int limit, int offset) throws DaoException {  
 return null;  
 }  
  
 @Override  
 public List<Bid> findByAccountId(Long accountId, int limit, int offset) throws DaoException {  
 return null;  
 }  
  
 @Override  
 public Long findAmountOfInProgressBids() throws DaoException {  
 return null;  
 }  
  
 @Override  
 public Long findAmountOfBidsByAccountId(Long accountId) throws DaoException {  
 return null;  
 }  
  
 @Override  
 public boolean isPresentInProgressBids(Long accountId) throws DaoException {  
 return false;  
 }  
  
 @Override  
 public boolean isTopUpBid(Long bidId) throws DaoException {  
 return false;  
 }  
  
 @Override  
 public void approveTopUpBid(Long topUpBidId, String adminComment) throws DaoException {  
 }  
  
 @Override  
 public void approveWithdrawBid(Long withdrawBidId, String adminComment) throws DaoException {  
 }  
  
 @Override  
 public void rejectTopUpBid(Long topUpBidId, String adminComment) throws DaoException {  
 }  
  
 @Override  
 public void rejectWithdrawBid(Long withdrawBidId, String adminComment) throws DaoException {  
 }  
}

Теперь определяю как должен вести себя этот класс и пишу тесты под это поведение.

В данном случае необходим именно Spy.

Spy сохраняет поведение оригинального объекта и в то же время может быть использыван в when() & verify().

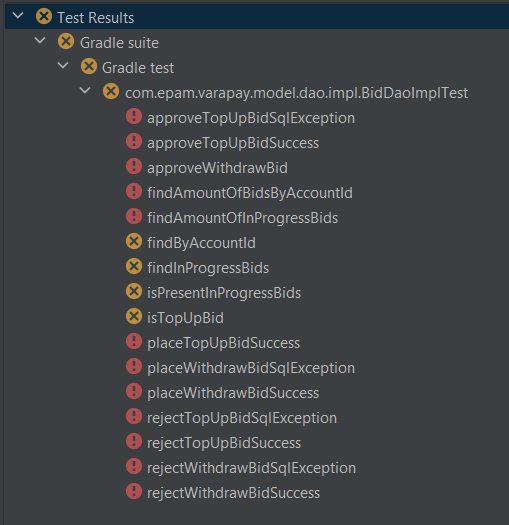
Тестируемый объект будет Spy. Это позволит проверить вызвались ли на этом объекте методы класса предка GenericDao из тестируемых методов.

Также для доступа к protected методам класса предка GenericDao я создал новый класс, который наследует тестируемый класс BidDaoImpl. Я переопределил только методы класса GenericDao, а тестируемые методы BidDaoImpl остались как были.

Кроме того этот внутренний класс предотвращает инициализацию ConnectionPool. Это происходит с помощью переопределения метода init(). Переопределенный метод ничего не делает, пожтому пул не инициализируется.

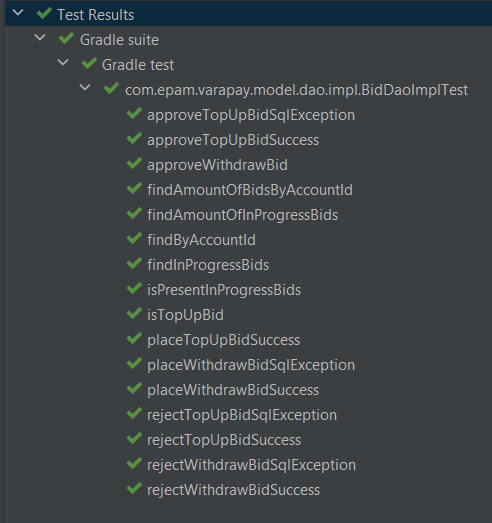
public class BidDaoImplTest {  
 @Mock  
 private ConnectionPool pool;  
 @Mock  
 private Connection connection;  
 @Spy  
 @InjectMocks  
 private ProtectedAccessToPublicBidDao bidDaoSpy;  
 private static List<Bid> *executeQueryExpectedReturn* = Collections.*emptyList*();  
  
 static class ProtectedAccessToPublicBidDao extends BidDaoImpl {  
 @Override  
 public void init() {  
 }  
  
 @Override  
 public void endTransaction(Connection connection) {  
 }  
  
 @Override  
 public void executeUpdate(String query, Connection connection, Object... parameters) {  
 }  
  
 @Override  
 public void executeUpdate(String query, Object... parameters) {  
 }  
  
 @Override  
 public List<Bid> executeQuery(String query, Object... parameters) {  
 return *executeQueryExpectedReturn*;  
 }  
  
 @Override  
 public Optional<Long> findLong(String query, String columnLabel, Object... params) {  
 return Optional.*of*(1L);  
 }  
  
 @Override  
 public Optional<Long> findLong(String query, Connection connection, String columnLabel, Object... params) {  
 return Optional.*of*(2L);  
 }  
  
 @Override  
 public boolean findBoolean(String query, String columnLabel, Object... params) {  
 return true;  
 }  
 }  
  
 @BeforeMethod  
 public void setup() {  
 *initMocks*(this);  
 *when*(pool.getConnection()).thenReturn(connection);  
 }  
  
 @Test  
 public void placeTopUpBidSuccess() throws DaoException {  
 bidDaoSpy.placeTopUpBid(1L, 2L, new BigDecimal(3), CustomCurrency.*BITCOIN*, "hi");  
  
 *verify*(bidDaoSpy).executeUpdate("insert into bids (account\_id, amount, " +  
 "amount\_in\_chosen\_currency, currency\_id, client\_message, is\_top\_up) " +  
 "VALUES (?, ?, ?, ?, ?, true)",  
 1L, 2L, new BigDecimal(3), CustomCurrency.*BITCOIN*.ordinal(), "hi");  
 }  
  
 @Test  
 public void placeWithdrawBidSuccess() throws DaoException {  
 bidDaoSpy.placeWithdrawBid(1L, 2L, new BigDecimal(3), CustomCurrency.*BITCOIN*, "hi");  
  
 *verify*(bidDaoSpy).endTransaction(connection);  
 }  
  
 @Test(expectedExceptions = DaoException.class)  
 public void placeWithdrawBidSqlException() throws DaoException {  
 *doThrow*(SQLException.class).when(bidDaoSpy).endTransaction(connection);  
  
 bidDaoSpy.placeWithdrawBid(1L, 2L, new BigDecimal(3), CustomCurrency.*BITCOIN*, "hi");  
 }  
  
 @Test  
 public void findInProgressBids() throws DaoException {  
 List<Bid> actual = bidDaoSpy.findInProgressBids(1, 2);  
  
 *assertSame*(actual, *executeQueryExpectedReturn*);  
 }  
  
 @Test  
 public void findByAccountId() throws DaoException {  
 List<Bid> actual = bidDaoSpy.findByAccountId(1L, 2, 2);  
  
 *assertSame*(actual, *executeQueryExpectedReturn*);  
 }  
  
 @Test  
 public void findAmountOfInProgressBids() throws DaoException {  
 long actual = bidDaoSpy.findAmountOfInProgressBids();  
 long expected = 1;  
  
 *assertEquals*(actual, expected);  
 }  
  
 @Test  
 public void findAmountOfBidsByAccountId() throws DaoException {  
 long actual = bidDaoSpy.findAmountOfBidsByAccountId(1L);  
 long expected = 1;  
  
 *assertEquals*(actual, expected);  
 }  
  
 @Test  
 public void isPresentInProgressBids() throws DaoException {  
 boolean actual = bidDaoSpy.isPresentInProgressBids(1L);  
  
 *assertTrue*(actual);  
 }  
  
 @Test  
 public void isTopUpBid() throws DaoException {  
 boolean actual = bidDaoSpy.isTopUpBid(1L);  
  
 *assertTrue*(actual);  
 }  
  
 @Test  
 public void approveTopUpBidSuccess() throws DaoException {  
 bidDaoSpy.approveTopUpBid(1L, "comment");  
  
 *verify*(bidDaoSpy).endTransaction(connection);  
 }  
  
 @Test(expectedExceptions = DaoException.class)  
 public void approveTopUpBidSqlException() throws DaoException {  
 *doThrow*(SQLException.class).when(bidDaoSpy).endTransaction(connection);  
  
 bidDaoSpy.approveTopUpBid(1L, "comment");  
 }  
  
 @Test  
 public void approveWithdrawBid() throws DaoException {  
 bidDaoSpy.approveWithdrawBid(1L, "comment");  
  
 *verify*(bidDaoSpy).executeUpdate("update bids set bid\_state\_id = 2, " +  
 "admin\_comment = ? where bid\_id = ?", "comment", 1L);  
 }  
  
 @Test  
 public void rejectTopUpBidSuccess() throws DaoException {  
 bidDaoSpy.rejectTopUpBid(1L, "comment");  
  
 *verify*(bidDaoSpy).endTransaction(connection);  
 }  
  
 @Test(expectedExceptions = DaoException.class)  
 public void rejectTopUpBidSqlException() throws DaoException {  
 *doThrow*(SQLException.class).when(bidDaoSpy).endTransaction(connection);  
  
 bidDaoSpy.rejectTopUpBid(1L, "comment");  
 }  
  
 @Test  
 public void rejectWithdrawBidSuccess() throws DaoException {  
 bidDaoSpy.rejectWithdrawBid(1L, "comment");  
  
 *verify*(bidDaoSpy).endTransaction(connection);  
 }  
  
 @Test(expectedExceptions = DaoException.class)  
 public void rejectWithdrawBidSqlException() throws DaoException {  
 *doThrow*(SQLException.class).when(bidDaoSpy).endTransaction(connection);  
  
 bidDaoSpy.rejectWithdrawBid(1L, "comment");  
 }  
}

Запускаю тесты:



После множества циклов (написал метод, запустил тест, отрефакторил, снова запустил тест) получился класс, методы которого удовлетворяют тестам. Тесты не были изменены.

public class BidDaoImpl extends GenericDao<Bid> implements BidDao {  
 private static final String *FIND\_BID\_ACCOUNT\_BY\_BID\_ID* = "select account\_id from bids where bid\_id = ?";  
 private static final String *FIND\_BID\_AMOUNT\_BY\_ID* = "select amount from bids where bid\_id = ?";  
 private static final String *FIND\_IN\_PROGRESS\_BIDS* =  
 "select bid\_id, state, bids.amount, amount\_in\_chosen\_currency, currency\_name, is\_top\_up, client\_message, " +  
 "admin\_comment, placing\_date\_time, accounts.account\_id, accounts.account\_id\n" +  
 " balance, is\_active, users.user\_id, users.login, password, salt,users.email,\n" +  
 " users.firstname, users.lastname, users.birth, roles.role\_name from bids\n" +  
 " join bid\_states on bids.bid\_state\_id = bid\_states.bid\_state\_id\n" +  
 " join accounts on bids.account\_id = accounts.account\_id\n" +  
 " and bids.bid\_state\_id = 1\n" +  
 " join users on accounts.user\_id = users.user\_id\n " +  
 " join currencies on bids.currency\_id = currencies.currency\_id " +  
 " join roles on users.role\_id = roles.role\_id order by placing\_date\_time desc limit ? offset ?";  
 private static final String *FIND\_BY\_ACCOUNT\_ID* =  
 "select bid\_id, state, bids.amount, amount\_in\_chosen\_currency, currency\_name, is\_top\_up, client\_message, " +  
 "admin\_comment, placing\_date\_time, accounts.account\_id, accounts.account\_id\n" +  
 " balance, is\_active, users.user\_id, users.login, password, salt,users.email,\n" +  
 " users.firstname, users.lastname, users.birth, roles.role\_name from bids\n" +  
 " join bid\_states on bids.bid\_state\_id = bid\_states.bid\_state\_id\n" +  
 " join accounts on bids.account\_id = ? and bids.account\_id = accounts.account\_id\n" +  
 " join users on accounts.user\_id = users.user\_id\n" +  
 " join currencies on bids.currency\_id = currencies.currency\_id " +  
 " join roles on users.role\_id = roles.role\_id order by placing\_date\_time desc limit ? offset ?";  
 private static final String *PLACE\_TOP\_UP\_BID* = "insert into bids (account\_id, amount, amount\_in\_chosen\_currency," +  
 " currency\_id, client\_message, is\_top\_up) VALUES (?, ?, ?, ?, ?, true)";  
 private static final String *PLACE\_WITHDRAW\_BID* = "insert into bids (account\_id, amount, " +  
 "amount\_in\_chosen\_currency, currency\_id, client\_message, is\_top\_up) VALUES (?, ?, ?, ?, ?, false)";  
 private static final String *ADD\_ACCOUNT\_BALANCE* = "update accounts set balance = balance + ? where account\_id = ?";  
 private static final String *UPDATE\_ADMIN\_COMMENT* = "update bids set admin\_comment = ? where bid\_id = ?";  
 private static final String *SET\_STATE\_REJECTED* = "update bids set bid\_state\_id = 3 where bid\_id = ?";  
 private static final String *SET\_ADMIN\_COMMENT\_AND\_STATE\_APPROVED* = "update bids set bid\_state\_id = 2, " +  
 "admin\_comment = ? where bid\_id = ?";  
 private static final String *RECOVER\_ACCOUNT\_BALANCE\_CONSIDERING\_WITHDRAW\_BID\_ID* =  
 "update accounts set balance = balance + (select amount from bids where bid\_id = ?) \n" +  
 "where account\_id = (select account\_id from bids where bid\_id = ?)";  
 private static final String *FIND\_NUMBER\_OF\_RECORDS* = "select count(\*) from bids where bid\_state\_id = 1";  
 private static final String *FIND\_NUMBER\_OF\_RECORDS\_BY\_ACCOUNT\_ID* =  
 "select count(\*) from bids where account\_id = ?";  
 private static final String *IS\_PRESENT\_IN\_PROGRESS\_BID\_BY\_ACCOUNT\_ID* = "select exists(select 1 from bids " +  
 "where bid\_state\_id = 1 and account\_id = ?)";  
 private static final String *IS\_TOP\_UP\_BID* = "select is\_top\_up from bids where bid\_id = ?";  
  
 public BidDaoImpl() {  
 super(new BidBoulder());  
 }  
  
 @Override  
 public void placeTopUpBid(Long accountId, Long amount, BigDecimal amountInChosenCurrency,  
 CustomCurrency currency, String message) throws DaoException {  
 executeUpdate(*PLACE\_TOP\_UP\_BID*, accountId, amount, amountInChosenCurrency, currency.ordinal(), message);  
 }  
  
 @Override  
 public void placeWithdrawBid(Long accountId, Long amount, BigDecimal amountInChosenCurrency,  
 CustomCurrency currency, String message) throws DaoException {  
 Connection connection = pool.getConnection();  
 try {  
 startTransaction(connection);  
 executeUpdate(*PLACE\_WITHDRAW\_BID*, accountId, amount, amountInChosenCurrency, currency.ordinal(), message);  
 executeUpdate(*ADD\_ACCOUNT\_BALANCE*, connection, -amount, accountId);  
 endTransaction(connection);  
 } catch (SQLException e) {  
 cancelTransaction(connection);  
 throw new DaoException("can not get access to db", e);  
 } finally {  
 pool.releaseConnection(connection);  
 }  
 }  
  
 @Override  
 public List<Bid> findInProgressBids(int limit, int offset) throws DaoException {  
 return executeQuery(*FIND\_IN\_PROGRESS\_BIDS*, limit, offset);  
 }  
  
 @Override  
 public List<Bid> findByAccountId(Long accountId, int limit, int offset) throws DaoException {  
 return executeQuery(*FIND\_BY\_ACCOUNT\_ID*, accountId, limit, offset);  
 }  
  
 @Override  
 public Long findAmountOfInProgressBids() throws DaoException {  
 return findLong(*FIND\_NUMBER\_OF\_RECORDS*, ColumnLabel.*COUNT*).orElseThrow(DaoException::new);  
 }  
  
 @Override  
 public Long findAmountOfBidsByAccountId(Long accountId) throws DaoException {  
 return findLong(*FIND\_NUMBER\_OF\_RECORDS\_BY\_ACCOUNT\_ID*, ColumnLabel.*COUNT*, accountId)  
 .orElseThrow(DaoException::new);  
 }  
  
 @Override  
 public boolean isPresentInProgressBids(Long accountId) throws DaoException {  
 return findBoolean(*IS\_PRESENT\_IN\_PROGRESS\_BID\_BY\_ACCOUNT\_ID*, ColumnLabel.*EXISTS*, accountId);  
 }  
  
 @Override  
 public boolean isTopUpBid(Long bidId) throws DaoException {  
 return findBoolean(*IS\_TOP\_UP\_BID*, ColumnLabel.*IS\_TOP\_UP*, bidId);  
 }  
  
 @Override  
 public void approveTopUpBid(Long topUpBidId, String adminComment) throws DaoException {  
 Connection connection = pool.getConnection();  
 try {  
 startTransaction(connection);  
 Long bidAmount = findLong(*FIND\_BID\_AMOUNT\_BY\_ID*, connection,  
 ColumnLabel.*BID\_AMOUNT*, topUpBidId).orElseThrow(DaoException::new);  
 Long bidAccountId = findLong(*FIND\_BID\_ACCOUNT\_BY\_BID\_ID*, connection,  
 ColumnLabel.*ACCOUNT\_ID*, topUpBidId).orElseThrow(DaoException::new);  
 executeUpdate(*ADD\_ACCOUNT\_BALANCE*, connection, bidAmount, bidAccountId);  
 executeUpdate(*SET\_ADMIN\_COMMENT\_AND\_STATE\_APPROVED*, connection, adminComment, topUpBidId);  
 endTransaction(connection);  
 } catch (SQLException | DaoException e) {  
 cancelTransaction(connection);  
 throw e instanceof DaoException ? (DaoException) e : new DaoException(e);  
 } finally {  
 pool.releaseConnection(connection);  
 }  
 }  
  
 @Override  
 public void approveWithdrawBid(Long withdrawBidId, String adminComment) throws DaoException {  
 executeUpdate(*SET\_ADMIN\_COMMENT\_AND\_STATE\_APPROVED*,adminComment, withdrawBidId);  
 }  
  
 @Override  
 public void rejectTopUpBid(Long topUpBidId, String adminComment) throws DaoException {  
 Connection connection = pool.getConnection();  
 try {  
 startTransaction(connection);  
 executeUpdate(*UPDATE\_ADMIN\_COMMENT*, connection, adminComment, topUpBidId);  
 executeUpdate(*SET\_STATE\_REJECTED*, connection, topUpBidId);  
 endTransaction(connection);  
 } catch (SQLException e) {  
 cancelTransaction(connection);  
 throw new DaoException("can not get access to db", e);  
 } finally {  
 pool.releaseConnection(connection);  
 }  
 }  
  
 @Override  
 public void rejectWithdrawBid(Long withdrawBidId, String adminComment) throws DaoException {  
 Connection connection = pool.getConnection();  
 try {  
 startTransaction(connection);  
 executeUpdate(*UPDATE\_ADMIN\_COMMENT*, connection, adminComment, withdrawBidId);  
 executeUpdate(*RECOVER\_ACCOUNT\_BALANCE\_CONSIDERING\_WITHDRAW\_BID\_ID*, connection, withdrawBidId, withdrawBidId);  
 executeUpdate(*SET\_STATE\_REJECTED*, connection, withdrawBidId);  
 endTransaction(connection);  
 } catch (SQLException e) {  
 cancelTransaction(connection);  
 throw new DaoException("can not get access to db", e);  
 } finally {  
 pool.releaseConnection(connection);  
 }  
 }  
}



Выводы: изучил TDD. Разработал класс по TDD. 100% строк тестируемого класса покрыты тестами. Использывал Spy. Для расширения видимости методов создал доп класс.