**White Box Unit Testing Report**

[**Create Transaction: Source Listing**](#_e2yzh9f2q4ir) **2**

[**Create Transaction: Test Cases, Inputs and Results**](#_qsrdn1b7h27r) **4**

[CT- Test Cases](#_o1cvbe6rilvm) 4

[CT- Test Inputs](#_ggawo795r7nu) 5

[CT- Test Results](#_khnkcxouqgl) 6

[**Refund Transaction: Source Listing**](#_7uk8zxcj65t) **7**

[**Refund Transaction: Test Cases, Inputs and Results**](#_5y7j2nixyo85) **9**

[RT-Test Cases](#_ncu222j8ghyh) 9

[RT- Test Inputs](#_5r1xdirj0b5o) 11

[RT- Test Results](#_7fspzjbusgor) 12

## 

## 

## **Create Transaction: Source Listing**

Create transaction excerpt from src/main/java/helpers/TransactionApplier.java in method applyTransactions():

public void applyTransactions() {

System.out.println("Applying transactions...");

for(DailyTransaction dt: transactions) {

switch(dt.getTransactionCode()) {

case create:

System.out.println(" ");

System.out.println("-------------Test-Case--------------------------");

System.out.println("Create transaction for user " + dt.getUsername1() + ":");

UserAccount ua = cuaf.findUserByUsername(dt.getUsername1());

if(ua == null) {

cuaf.addUser(dt.getUserAcc1());

System.out.println(" Successfully applied create transaction");

System.out.println("-------------Success----------------------------");

System.out.println(" ");

}

else {

System.out.println("ERROR: Username "+dt.getUsername1()+" already exists.");

System.out.println("-------------Success----------------------------");

System.out.println(" ");

break;

}

and for one of the test cases:

Excerpt from DailyTransactionsFile.java

private void readFile() {

BufferedReader br;

try {

..

catch(NumberFormatException e)

{

System.out.println(" ");

System.out.println("-------------Test-Case--------------------------");

System.out.println("ERROR: Maximum of 15 Characters for username”);

System.out.println("-------------Success----------------------------");

System.out.println(" ");

}

## 

## 

## **Create Transaction: Test Cases, Inputs and Results**

##### **CT- Test Cases**

The creation transaction will be using output partitioning testing. We have created a new copy of TransactionApplier.java called TransactionApplierCreateTest.java. There are four total test cases:

1. when the username input is less than 15 characters long.
2. when the username input is equal to 15 characters long.
3. when the username input is greater than 15 characters long.
4. when the username input is a username that already exists.

##### 

##### 

##### 

##### 

##### **CT- Test Inputs**

Each test has a different user added inside the .cuaf file depending on the test case and is coming from the daily transaction files.

|  |  |  |
| --- | --- | --- |
| **Test Output** | **Daily Transactions Input** | **Current Users Accounts File Input** |
| username < 15 chars | 01 danica AA 000000.00 | danica AA 000000.00 |
| username = 15 chars | 01 kaleeeeeeeeeeee AA 000000.00 | kaleeeeeeeeeeee AA 000000.00 |
| username > 15 chars | 01 UsernameGreatereeerAA 000000.00 | UsernameGreatereeerAA 000000.00 |
| username already exists | 01 fullStandard FS 000000.00 | fullStandard FS 000000.00 |

##### 

##### 

##### 

##### 

##### **CT- Test Results**

|  |  |  |
| --- | --- | --- |
| **Expected Outputs** | **Actual Outputs and Results** | **Failures** |
| admin AA 000999.99  adminUser AA 000999.99  fullStandard FS 000000.00  buyStandard BS 000000.00  sellStandard SS 000000.00  danica AA 000000.00  END | admin AA 000999.99  adminUser AA 000999.99  fullStandard FS 000000.00  buyStandard BS 000000.00  sellStandard SS 000000.00  danica AA 000000.00  END  Create transaction for user danica:  Successfully applied create transaction | None. |
| admin AA 000999.99  adminUser AA 000999.99  fullStandard FS 000000.00  buyStandard BS 000000.00  sellStandard SS 000000.00  kaleeeeeeeeeeee AA 000000.00  END | admin AA 000999.99  adminUser AA 000999.99  fullStandard FS 000000.00  buyStandard BS 000000.00  sellStandard SS 000000.00  kaleeeeeeeeeeee AA 000000.00  END  Create transaction for user kaleeeeeeeeeeee:  Successfully applied create transaction | None. |
| admin AA 000999.99  adminUser AA 000999.99  fullStandard FS 000000.00  buyStandard BS 000000.00  sellStandard SS 000000.00  END | admin AA 000999.99  adminUser AA 000999.99  fullStandard FS 000000.00  buyStandard BS 000000.00  sellStandard SS 000000.00  END  Create transaction for user fullStandard:  ERROR: Username fullStandard already exists. | None, as there was already a user named “fullstandard” from the current user accounts file. |
| admin AA 000999.99  adminUser AA 000999.99  fullStandard FS 000000.00  buyStandard BS 000000.00  sellStandard SS 000000.00  END | admin AA 000999.99  adminUser AA 000999.99  fullStandard FS 000000.00  buyStandard BS 000000.00  sellStandard SS 000000.00  END  ERROR: Maximum of 15 Characters for username. | None as the transaction didn’t go through because there were too many characters for the username. |

## **Refund Transaction: Source Listing**

The section of the code to test for refund is a portion within the method applyTransaction() in the file src/main/java/helpers/TransactionApplier.java.

public void applyTransactions() {

System.out.println("Starting transaction applying process...");

for(DailyTransaction dt: transactions) {

switch(dt.getTransactionCode()) {

...

case refund:

System.out.println("Refund transaction for seller " + dt.getUsername1() + " and buyer " + dt.getUsername2() + ":");

//Get refund amount from dtf file.

Double transferredAmount = dt.getrefundCredit();

if (transferredAmount < 0) {

System.out.println("Transferred amount has to be more than 0.");

}

else {

UserAccount seller = cuaf.findUserByUsername(dt.getUsername1());

UserAccount buyer = cuaf.findUserByUsername(dt.getUsername2());

if (seller == null) {

System.out.println("ERROR: Seller username '" + dt.getUsername1() + "' does not exist.");

}

else {

if (buyer == null) {

System.out.println("ERROR: Buyer username '" + dt.getUsername2() + "' does not exist.");

}

else {

//Buyer and seller update credit.

double sellerNewCredit = seller.getAvailableCredit() - transferredAmount;

seller.setAvailableCredit(sellerNewCredit);

cuaf.updateUserCredit(seller);

double buyerNewCredit = buyer.getAvailableCredit() + transferredAmount;

seller.setAvailableCredit(sellerNewCredit);

buyer.setAvailableCredit(buyerNewCredit);

System.out.println(" Successfully applied refund transaction");

}

}

}

break;

## 

## 

## 

## **Refund Transaction: Test Cases, Inputs and Results**

##### **RT-Test Cases**

The refund transaction uses decision coverage testing. A copy of TransactionApplier.java was made and System.out.println() was used for instrumentation injection to see which branch of the decision was executed. The new file is src/test/java/helpers.test/TransactionApplierRefund.java. There are 3 decisions as highlighted below. The first decision is to check the transferred amount in the transaction if it is equal or greater than 0, the second decision checks if seller user is null, and the third decision checks if the buyer user is null.

System.out.println("Refund transaction for seller " + dt.getUsername1() + " and buyer " + dt.getUsername2() + ":");

//Get refund amount from dtf file.

Double transferredAmount = dt.getrefundCredit();

if(transferredAmount <= 0)

{

System.out.println("Decision 1: true, transferred amount <= 0");

System.out.println("Transferred amount has to be more than 0.");

}

else {

System.out.println("Decision 1: false, transferred amount > 0");

UserAccount seller = cuaf.findUserByUsername(dt.getUsername1());

UserAccount buyer = cuaf.findUserByUsername(dt.getUsername2());

if (seller == null) {

System.out.println("Decision 2: true, seller == null");

System.out.println("ERROR: Seller username '" + dt.getUsername1() + "' does not exist.");

}

else {

System.out.println("Decision 2: false, seller != null");

if (buyer == null) {

System.out.println("Decision 3: true, buyer == null");

System.out.println("ERROR: Buyer username '" + dt.getUsername2() + "' does not exist.");

}

else {

System.out.println("Decision 3: false, buyer != null");

//Buyer and seller update credit.

double sellerNewCredit = seller.getAvailableCredit() - transferredAmount;

seller.setAvailableCredit(sellerNewCredit);

cuaf.updateUserCredit(seller);

double buyerNewCredit = buyer.getAvailableCredit() + transferredAmount;

seller.setAvailableCredit(sellerNewCredit);

buyer.setAvailableCredit(buyerNewCredit);

System.out.println(" Successfully applied refund transaction");

}

}

}

##### 

##### 

##### 

##### 

##### 

##### **RT- Test Inputs**

Each test will use the same users file with 3 users: admin, buyStandard, sellStandard. The inputs related to the transaction that affects the decisions are buyer username, seller username, refund amount.

|  |  |  |
| --- | --- | --- |
| **Decision** | **Daily Transactions Input** | **Current Users Accounts File Input** |
| 1: true | 05 buyStandard sellStandard -00001.00 | admin AA 000000.00  buyStandard BS 000000.00  sellStandard SS 000000.00  END |
| 1: false | 05 buyStandard sellStandard 000001.00 |
| 2: true | 05 user1 sellStandard 000001.00 |
| 2: false | 05 buyStandard sellStandard 000001.00 |
| 3: true | 05 buyStandard user2 000001.00 |
| 3: false | 05 buyStandard sellStandard 000001.00 |

##### 

##### 

##### 

##### 

##### 

##### **RT- Test Results**

|  |  |
| --- | --- |
| Decision | Expected output |
| 1: true | Decision 1: true, transferred amount <= 0 |
| 1: false | Decision 1: false, transferred amount > 0 |
| 2: true | Decision 2: true, seller == null |
| 2: false | Decision 2: false, seller != null |
| 3: true | Decision 3: true, buyer == null |
| 3: false | Decision 3: false, buyer != null |

|  |  |  |
| --- | --- | --- |
| Decision | Actual output result | Failures |
| 1: true | Decision 1: true, transferred amount <= 0 | None, but fixed the source code to include credit amount of 0. It originally looked at amount < 0 and would let refund credit of 0 processes through even though it would not affect a user’s credit in the end |
| 1: false | Decision 1: false, transferred amount > 0 | None |
| 2: true | Decision 2: true, seller == null | None |
| 2: false | Decision 2: false, seller != null | Originally got a null pointer exception because the else statement was not properly added in, but is fixed now. |
| 3: true | Decision 3: true, buyer == null | None |
| 3: false | Decision 3: false, buyer != null | None |