

QUINTERAC- Assignment 5

Backend Whitebox Testing Report

MEMBER	ROLES	TIME
LUVIT CHUMBER – 10190467	Withdraw Testing Lead Withdraw Report	7 hours
JESSICA WONG – 10181646	Withdraw Code Lead Withdraw Report	7 hours
SAMIR MEKHDI – 10191295	Createacct Testing Lead Creareacct Report	5 hours
TAYYAB AHMAD – 10197212	Createacct Code Lead Createacct Report	5 hours

Withdraw – Decision Coverage Testing

Source Listing

```
public class
backend {

    public static void main(String[] args) throws IOException {
        ...
        masterAccts = updateMasterAccts(masterAccts,mergedTsffFileNames);
        ...
    }

    ...

    private static ArrayList<Account> updateMasterAccts(ArrayList<Account>
masterAccts, String[] mergedTsffFileNames) throws FileNotFoundException {

        for(int i = 0; i < mergedTsffFileNames.length; i++) {
            File tsf = new File(mergedTsffFileNames[i]);
            Scanner sc = new Scanner(tsf);
            while
            (sc.hasNextLine()) {
                String inRaw = sc.nextLine();
                1         if (inRaw.equals("EOS") || !sc.hasNextLine()) {
                        continue; // if eos or no more lines, leave
                        current iteration.
                }
                ...

                acctFrom = validateAcctNumandReturn(acctFrom);
                int amount = validateAmountandReturn(amountStr);

                try
                {
                    5         switch(type) {
                        ...
                        5         case "WDR":
                                masterAccts =
                                transaction(masterAccts,type,acctFrom,nam
                                e,amount);
                                break;
                                ...
                                5         default:
                                    throw new
                                    IllegalArgumentException("Illegal
                                    Transaction Code");
                                } //end switch
                } catch (IllegalArgumentException e) {
```

```

        //e.printStackTrace();
        System.err.println(e.getMessage());
        sc.close();
        throw
        e;
    }
    ...
}
private static ArrayList<Account> transaction(ArrayList<Account>
masterAccts,String type,String acct,String name,int amount) throws
IllegalArgumentException {
    //          check if account exists else throw fatal error
6    if (!type.equals("NEW") && !DoesAcctNumExist(masterAccts,acct)) {
        throw new IllegalArgumentException("Account does not exist");
    }
    ...
8    switch(type) {
        ...
8        case "WDR":
9            if(amount != -
                1) {
                idx = masterAccts.indexOf(temp);
                temp = masterAccts.get(idx);
                currentAmount = temp.getAmount();
10            if (amount<currentAmount) {
                temp.setAmount(currentAmount-amount);
                masterAccts.set(idx, temp);
11            }else
                {
                //          error
                throw new
                IllegalArgumentException("Insufficent
                funds to transfer");
                }
9            }else {
                throw new IllegalArgumentException("Cannont
                Withdraw from Account, Invalid Amount Value");
            }
            break;
        }
    return masterAccts;
}
private static int validateAmountandReturn(String in) {
    int num = -1;
    try {
        double test = Double.parseDouble(in);

```

3

```

    if (test % 1
        > 0)
        test *= 100; //left shift amount 2 if amount entered as
        XX.x
    num=(int)test
    ;

```

4

```

    if (in.length() < 3 || num > 99999999 || num < 0)
        num = -1;
} catch (NumberFormatException
e){
    e.printStackTrace();
}

```

```

return num;

```

```

}

```

...

```

private static String validateAcctNumandReturn(String in) {

```

```

    String acct =
    "";

```

```

    in = in.trim();

```

2

```

    if (in.charAt(0)!='0' && in.length()==7) { //error if it starts with 0, or
    its length is longer than 7 characters

```

```

        try
        {
            Integer.parseInt(in);
            acct=in
            ;
        }catch (NumberFormatException e){
            e.printStackTrace();
        }

```

2

```

    }else {
        acct="NotValid";

```

```

    }
    return acct;

```

```

}

```

```

private static boolean DoesAcctNumExist(ArrayList<Account> accts, String in) {

```

```

    in = in.trim();
    Account cache = new Account(in,-1,null);

```

7

```

    if
    (accts.contains(cache))
        return true;

```

7

```

    return false;
}

```

```

}

```

Test Cases

Decision		Input		Test		Output
No.	Type/Result	TransCode	Amount	AcctNum	Number	Printed Line
1	If: True	EOS	-	-	T2	System.out.println("WDRStatementReached.TestingLine01");
1	If: False	WDR	100	1234567	T1	
2	If: True	WDR	100	1234567	T1	System.out.println("WDRStatementReached.TestingLine02");
2	If: False	WDR	100	111111	T3	System.out.println("WDRStatementReached.TestingLine03");
3	If: True	WDR	1.00	1234567	T4	System.out.println("WDRStatementReached.TestingLine04");
3	If: False	WDR	100	1234567	T1	
4	If: True	WDR	-100	1234567	T5	System.out.println("WDRStatementReached.TestingLine05");
4	If: False	WDR	100	1234567	T1	
5	Switch: WDR	WDR	100	1234567	T1	System.out.println("WDRStatementReached.TestingLine06");
5	Switch: Not WDR	ABC	100	1234567	T6	System.out.println("WDRStatementReached.TestingLine07");
6	If: True	WDR	100	2345678	T7	System.out.println("WDRStatementReached.TestingLine08");
6	If: False	WDR	100	1234567	T1	
7	If: True	WDR	100	1234567	T1	System.out.println("WDRStatementReached.TestingLine09");
7	If: False	WDR	100	2345678	T7	System.out.println("WDRStatementReached.TestingLine10");
8	Switch: WDR	WDR	100	1234567	T1	System.out.println("WDRStatementReached.TestingLine11");
8	Switch: Not WDR	ABC	100	1234567	T6	*Impossible, throws error before reaching
9	If: True	WDR	100	1234567	T1	System.out.println("WDRStatementReached.TestingLine12");
9	If: False	WDR	-100	1234567	T5	System.out.println("WDRStatementReached.TestingLine13");
10	If: True	WDR	100	1234567	T1	System.out.println("WDRStatementReached.TestingLine14");
10	If: False	WDR	99999999	1234567	T8	System.out.println("WDRStatementReached.TestingLine15");

Test Inputs

Case	Type	Acct To*	Amount	Acct From	Name*
T1	WDR	0000000	100	1234567	***
T2	EOS				
T3	WDR	0000000	100	111111	***
T4	WDR	0000000	1.00	1234567	***
T5	WDR	0000000	-100	1234567	***
T6	ABC	0000000	100	1234567	***
T7	WDR	0000000	100	2345678	***
T8	WDR	0000000	99999999	1234567	***

* toString method will always output 0000000 and *** for fields not needed for withdraw

Test Method and Template

Used the below template to create all the tests, updated the transaction and output per test.

```
//template
@Test
public void testWDRTX() throws Exception {
    String a[] = new String[]{"7654321 123 Jane Doe",
        "1234567 11607 John Doe",
        "1000002 74076 Jane Hancock",
        "1000001 74070 John Hancock"
        /*master accts contents*/};

    String b[] = new String[]{"WDR 0000000 100 1234567 ***",
        "EOS",
        "EOS" /*mergedTSF contents*/};

    String c[] = new String[] { /*expectedOutput*/
        /*always output from reading master accts*/
```

```

"WDRStatementReached.TestingLine02",
"WDRStatementReached.TestingLine02",
"WDRStatementReached.TestingLine02",
"WDRStatementReached.TestingLine02",
"WDRStatementReached.TestingLine03",
/*test specific output*/
"WDRStatementReached.TestingLine01",
"WDRStatementReached.TestingLine02",
"WDRStatementReached.TestingLine03",
"WDRStatementReached.TestingLine04",
"WDRStatementReached.TestingLine05",
"WDRStatementReached.TestingLine06",
"WDRStatementReached.TestingLine07",
"WDRStatementReached.TestingLine08",
"WDRStatementReached.TestingLine09",
"WDRStatementReached.TestingLine10",
"WDRStatementReached.TestingLine11",
"WDRStatementReached.TestingLine12",
"WDRStatementReached.TestingLine13",
"WDRStatementReached.TestingLine14",
"WDRStatementReached.TestingLine15",
/*clean up output*/
"WDRStatementReached.TestingLine01",
"WDRStatementReached.TestingLine01"
};

```

```
String d[] = new String[] {""};
```










```

    runAndTest(Arrays.asList(a), //
               Arrays.asList(b), //
               Arrays.asList(c), //
               Arrays.asList(d), false);
}

```

Test Report

```

 backendTest [Runner: JUnit 5] (0.504 s)
   testWDRT1() (0.435 s)
   testWDRT2() (0.011 s)
   testWDRT3() (0.009 s)
   testWDRT4() (0.013 s)
   testWDRT5() (0.009 s)
   testWDRT6() (0.007 s)
   testWDRT7() (0.008 s)
   testWDRT8() (0.012 s)

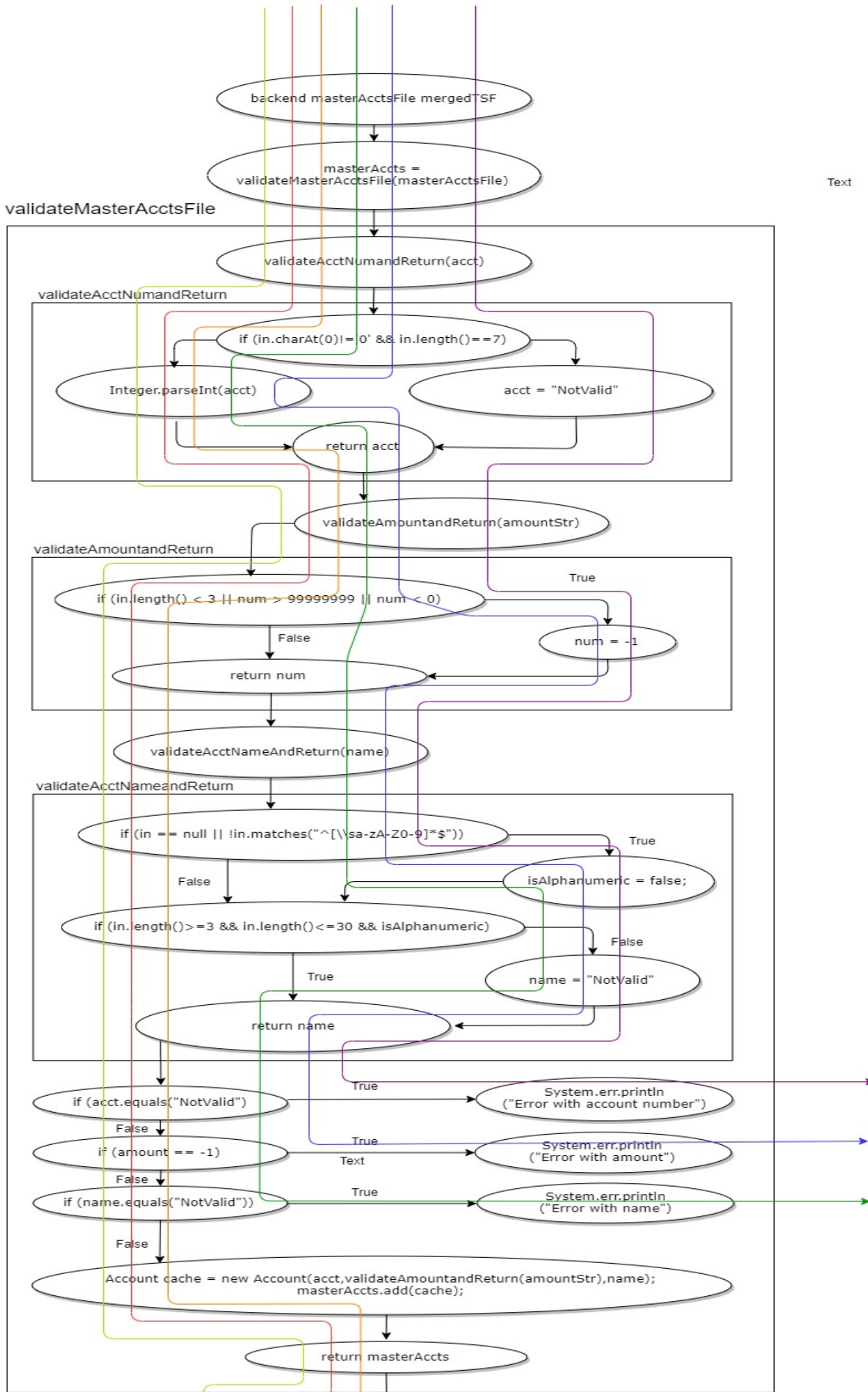
```

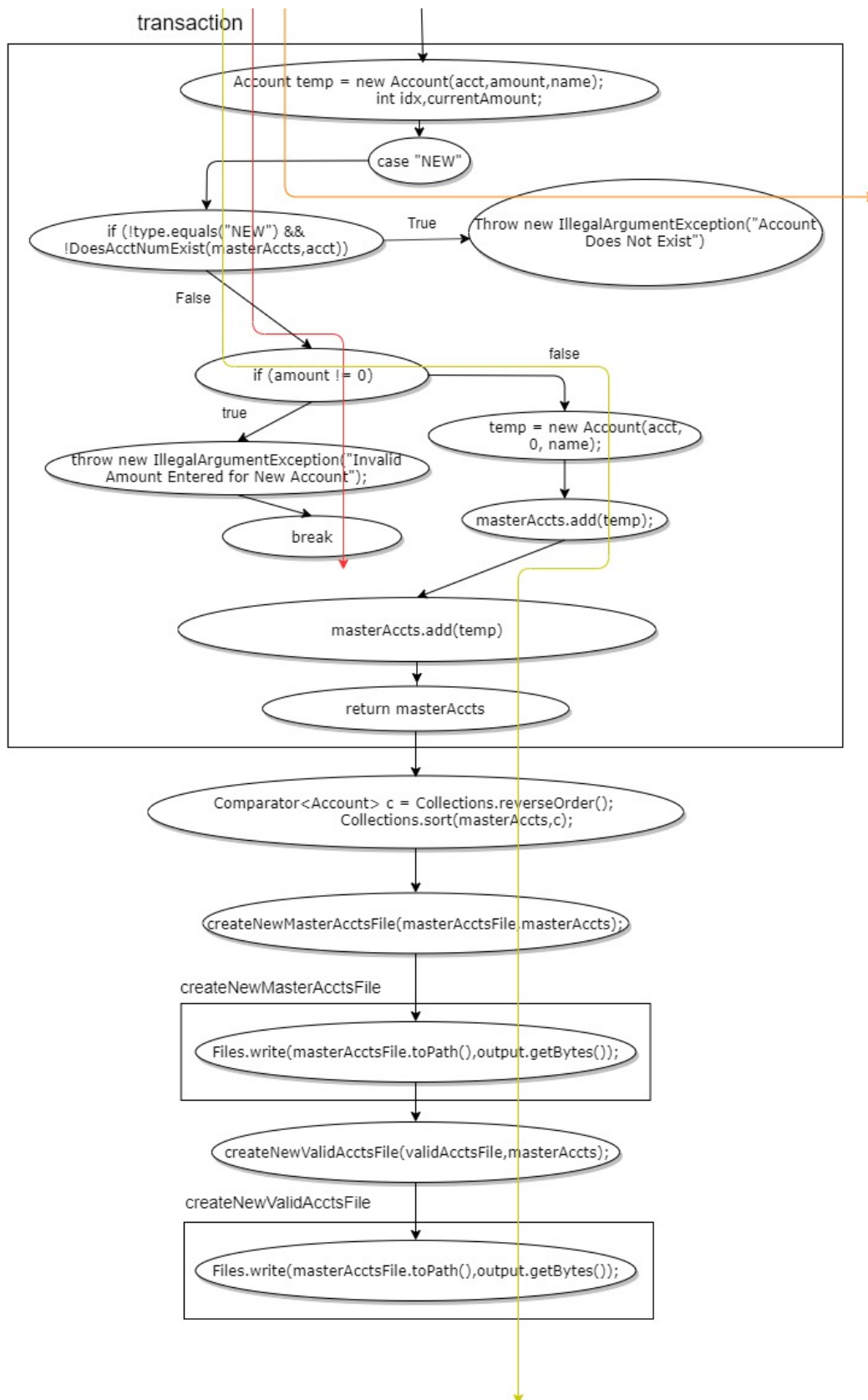
Createacct – Path Testing

Source Listing/Flow Diagram

validateMasterAcctsFile

Text





Test Inputs

System - path testing: one test case for each individual path (below)

Path	Type	Acct To	Amount	Acct From	Name
P1 (yellow)	NEW	1000002	000	0000000	John Doe
P2 (red)	NEW	1000001	111	0000000	John Doe
P3 (orange)	NEW	1000002 (<i>and account 1000002 already exists</i>)	000	0000000	John Doe
P4 (green)	NEW	1000001	000	0000000	???123
P5 (blue)	NEW	1000002	00000000 00000000 00	0000000	John Doe
P6 (purple)	NEW	10000001	000	0000000	John Doe

Test Report

Completion criterion – path testing: each test case for the paths are successful and without error

```
▼ backendTest [Runner: JUnit 5] (0.875 s)
  testNEWPath1() (0.592 s)
  testNEWPath2() (0.040 s)
  testNEWPath3() (0.009 s)
  testNEWPath4() (0.015 s)
  testNEWPath5() (0.031 s)
  testNEWPath6() (0.000 s)
```

Failure analysis: Testing CreateAcct path 1 uncovered a typo in an error message that has now been resolved.

This was the only failure uncovered by the path testing of CreateAcct.