# Deposit Weak Robust Equivalence Class Analysis:

I divided the domain of the isStudent, amount, and balance variables into equivalence classes based on the conditions in which they are used. Since I am using a robust version of equivalence class analysis, I added two extra classes for negative values of the amount and balance variables. For testing, I assume that if a test case includes a negative value for either amount or balance, the expected result follows the normal specification logic (no error should be triggered).

Equivalence Classes:

**STUDENT** = { client | client = student }

**NOT\_STUDENT** = STUDENT'

**SMALL\_DEPOSIT** = { amount | 0 <= amount <= 50 }

**MODERATE\_DEPOSIT** = { amount | 50 < amount <= 250 }

**LARGE\_DEPOSIT** = { amount | amount > 250 }

**HUMBLE\_BALANCE** = { balance | 0 <= balance <= 500 }

**MODEST\_BALANCE** = { balance | 500 < balance <= 2500 }

**SUBSTANTIAL\_BALANCE** = { balance | 2500 < balance <= 5000 }

**AMPLE\_BALANCE** = { balance | 5000 < balance <= 10000 }

**ABUNDANT\_BALANCE** = { balance | balance > 10000 }

**NEGATIVE\_DEPOSIT** = { amount | amount < 0 }

**NEGATIVE\_BALANCE** = { balance | balance < 0 }

Chosen Representative Values:

**STUDENT** = true

**NOT\_STUDENT** = false

**SMALL\_DEPOSIT** = 25

**MODERATE\_DEPOSIT** = 200

**LARGE\_DEPOSIT** = 300

**HUMBLE\_BALANCE** = 250

**MODEST\_BALANCE** = 1000

**SUBSTANTIAL\_BALANCE** = 3000

**AMPLE\_BALANCE** = 7000

**ABUNDANT\_BALANCE** = 20000

**NEGATIVE\_DEPOSIT** = -10

**NEGATIVE\_BALANCE** = -10

**Deposit Weak Robust Equivalence Class Tests**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **case id** | **client** | **amount** | **Balance** | **expected fee** | **expected output** | **results** | **value** |
| **WRE1** | **STUDENT (true)** | **SMALL\_DEPOSIT (25)** | **HUMBLE\_BALANCE (250)** | **0.0%** | **0.0** | **failed** | **0.05** |
| **WRE2** | **NOT\_STUDENT (false)** | **MODERATE\_DEPOSIT (200)** | **MODEST\_BALANCE (1000)** | **0.1%** | **0.2** | **failed** | **0.0** |
| **WRE3** | **STUDENT (true)** | **LARGE\_DEPOSIT (300)** | **SUBSTANTIAL\_BALANCE (3000)** | **0.5%** | **1.5** | **failed** | **3.0** |
| **WRE4** | **NOT\_STUDENT (false)** | **NEGATIVE\_D﻿EPOSIT (-10)** | **AMPLE\_BALANCE (7000)** | **0.1%** | **-0.01** | **failed** | **-0.0** |
| **WRE5** | **STUDENT (true)** | **SMALL\_DEPOSIT (25)** | **ABUNDANT\_BALANCE (20000)** | **0.5%** | **0.125** | **pass** | **0.125** |
| **WRE6** | **NOT\_STUDENT (false)** | **MODERATE\_DEPOSIT(200)** | **NEGATIVE\_BALANCE (-10)** | **0.1%** | **0.2** | **failed** | **0.0** |

# Transfer Decision Table Analysis

Process and Assumptions to generate test cases.

We have our conditions as whether the customer is a student, amount sent broken into various categories, source balance (account money is coming from) broken out, sink balance (account money is going to) broken out into categories. We have our actions as the Fee percentage.

With decision tables we have at least as many test cases as rules in our code. We observe 16 main control paths in the Transfer logic. We have at least that many test cases. The test cases increase because of our many conditions. The different possibilities can then be enumerated to give all possible combinations, which causes an increase in test cases.

Test Cases



Test Results

# Withdrawal Robust Worst Case Boundary Value Analysis:

To start we look at the values involved in a withdrawal which are Balance, isStudent and isWeekend. According to requirements Balance will have boundaries at $1000 and $5000. We are assuming that the user has a balance great enough to handle the amount being withdrawn. Another assumption, the user must have more than $5000 for a 0.0% fee.

“The balance is $1,000, or more, but less than $5,000, the fee is 0.1% of the  
amount withdrawn.  
If the balance is more than $5,000, then there is no fee” (Assignment 1). No rule is applied to exactly $5000 so this assumption is made.

For Robust worst case testing we observe values in and out the boundaries. Using the defined variables, we can create the following table.

Test Cases

A screenshot of a table

Description automatically generated

Test ResultsA screenshot of a data

Description automatically generated

# ATM Session:

**Robust Worst Case Boundary Value Analysis for an Invalid Amount:**

The invalid amount test requires validating the input for the withdrawal transaction to ensure it is a multiple of 20 and 50. The value involved for the analysis is the amount, which defines the amount chosen for the transaction. Since the account has a daily limit of $1000, $1000 is our upper bound, and the lower bound is 0. The test cases operate under the assumption that the transaction is the sole transaction of the day, therefore the full $1000 daily limit is still intact. Additionally, the tests operate under the assumption that the account has sufficient funds to perform the transaction.

Therefore, under these assumptions, the test cases operate on a singular value, the amount chosen for withdrawal. The test cases aim to check if the daily transaction limit is surpassed in the transaction, and the amount is a product of 20 and 50.

For Robust Worst Case Boundary Value Analysis, we observe values slightly above and below the border, on the border, as well as a nominal value, leading to the following test cases for the amount value.

Test cases:  
A white sheet with black text

Description automatically generated

Results from test cases:

A white background with black text

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Robust Worst Case Boundary Value Analysis for an Invalid PIN:**

The test cases for the PIN format must validate the inputted PIN against the definition of a valid PIN format, which is a PIN of length 5. To test the PIN validation, I used Robust Worst Case Boundary Value Analysis, with a single variable being the length of the inputted PIN. The only valid value for the PIN length variable is 5.

Robust Worst Case Boundary Value Analysis can be achieved by having a test case one below and one above the accepted range which consists only of a PIN length of 5. Therefore, PIN lengths 4 and 6 will be used as the values slightly outside of the accepted range for robust testing. The test case operates under the assumption that the user can only enter digits in the PIN.

Test cases:

A white background with black text

Description automatically generated

Test case results:

A grid of words on a white background

Description automatically generated

A screenshot of a computer code

Description automatically generated