# 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%** | **20.0** | **failed** | **0.0** |
| **WRE3** | **STUDENT (true)** | **LARGE\_DEPOSIT (300)** | **SUBSTANTIAL\_BALANCE (3000)** | **0.5%** | **150.0** | **failed** | **3.0** |
| **WRE4** | **NOT\_STUDENT (false)** | **NEGATIVE\_D﻿EPOSIT (-10)** | **AMPLE\_BALANCE (7000)** | **0.1%** | **-1.0** | **failed** | **-0.0** |
| **WRE5** | **STUDENT (true)** | **SMALL\_DEPOSIT (25)** | **ABUNDANT\_BALANCE (20000)** | **0.5%** | **12.5** | **failed** | **0.125** |
| **WRE6** | **NOT\_STUDENT (false)** | **MODERATE\_DEPOSIT(200)** | **NEGATIVE\_BALANCE (-10)** | **0.1%** | **20.0** | **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