
# Small Business Discount Network

## Practical work on VDM++

Formal Methods

Master in Informatics and Computing Engineering

Maria João Mira Paulo - up201403820@fe.up.pt
Pedro Duarte da Costa - up201403291@fe.up.pt

December 29, 2017

# Contents


**1 Informal system description and list of requirements 3**
- 1.1 Informal system description . . . 3
- 1.2 List of requirements . . . 4

**2 Visual UML model 5**
- 2.1 Use case model . . . 5
- 2.2 Class model . . . 9

**3 Formal VDM++ model 11**
- 3.1 Customer Class . . . 11
- 3.2 DiscountCard Class . . . 13
- 3.3 DiscountSystem Class . . . 15
- 3.4 Merchant Class . . . 18
- 3.5 Product Class . . . 20
- 3.6 User Class . . . 22

**4 Model validation 24**
- 4.1 DiscountSystemTest Class . . . 24

**5 Model verification 31**
- 5.1 Example of domain verification . . . 31
- 5.2 Example of invariant verification . . . 31

**6 Conclusions 32**

**7 References 32**

# 1 Informal system description and list of requirements

## 1.1 Informal system description

The Small Business Discount Network project is a platform responsible for all the information and transactions managed by a small business discount network.

Small merchants can join the system. Customers can join the system by receiving a discount card (physical or virtual) that they use in purchases from affiliated merchants.

In each purchase, there is a charge that is charged to the merchant (for example, 5% of the value of the sale), a part of which is returned to the customer (for example, 3%) and the remainder is a service charge to pay the maintenance of the system (e.g., 2%). The customer may use his balance on subsequent purchases from any affiliated merchant. It is also possible to transfer part of balance to other adherent customer.

Customers who get new customers or merchants who raise new merchants also receive bonuses. Merchants can offer extra discounts on selected products, increasing the percentage that reverts to the customer.

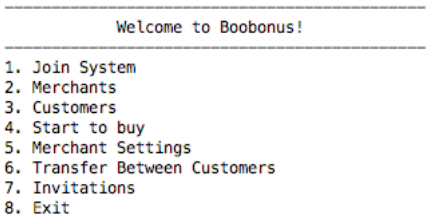


Figure 1: Main Menu

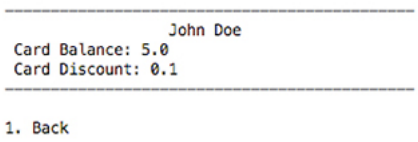


Figure 2: Customer Profile

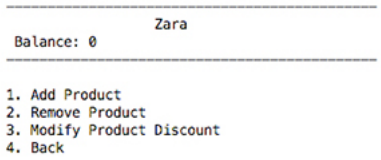


Figure 3: Merchant Settings

## 1.2 List of requirements

| Id | Priority | Description |
|---|---|---|
| R1 | Mandatory | Small Merchants can join the system. |
| R2 | Mandatory | Consumers / customers can join the system by receiving a discount card. |
| R3 | Mandatory | Consumer purchase something from the affiliated merchant. |
| R4 | Mandatory | Each purchase charges a fee, from the value of sale. This includes the discount, that is returned to the customer and a default fee which is used for operating costs. |
| R5 | Mandatory | The customer may use his balance on subsequent purchases from any affiliated merchant. |
| R6 | Mandatory | The consumer can transfer part of his balance to other adherent customer. |
| R7 | Mandatory | A Customer who get new customers receives a bonus. |
| R8 | Mandatory | A merchant who raise new merchants receives a bonus. |
| R9 | Mandatory | Merchants can offer extra discounts on selected products (increasing the % that reverts to the customer). |
| R10 | Optional | A merchant can add new products |
| R11 | Optional | A merchant can remove a product |

Table 1: List of requirements

# 2 Visual UML model

## 2.1 Use case model

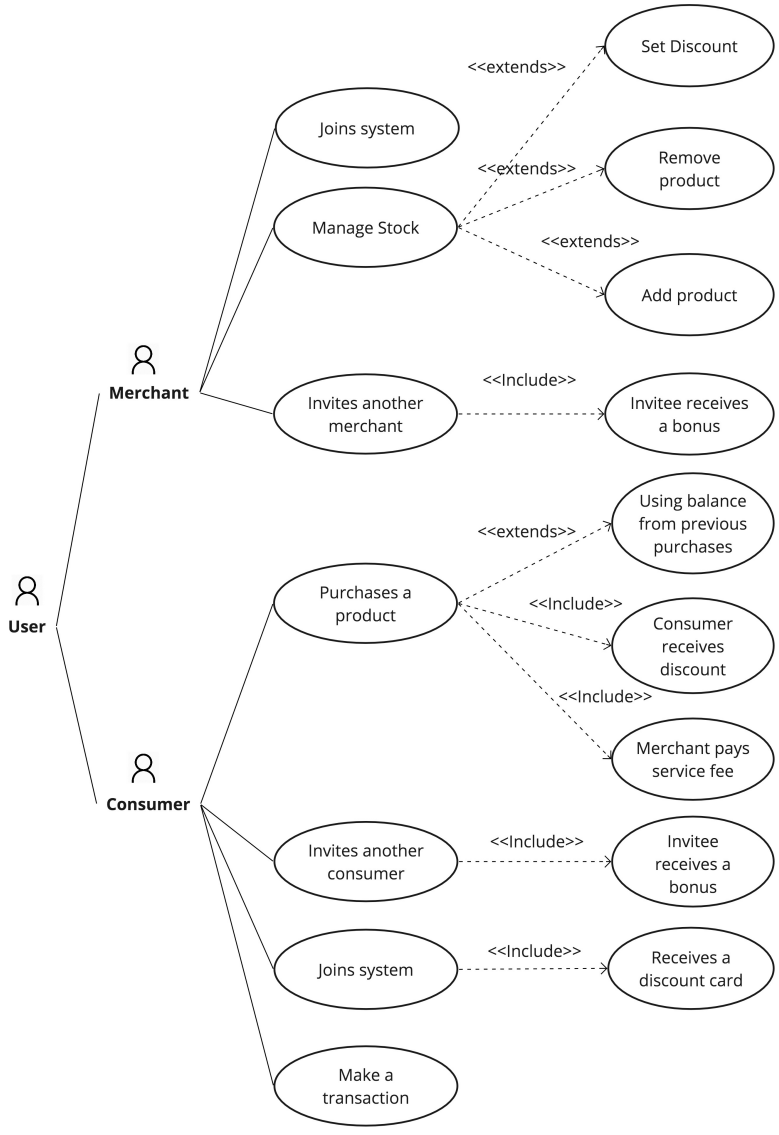


Figure 4: Use Case Model

| **Scenario** | **Buy a product** |
|---|---|
| **Description** | A consumer can buy a product and choose to deduct the price from his balance |
| **Pre-conditions** | The merchant has the product in stock (initial state) |
| **Post-conditions** | 1. The user deducts the cost from his balance, if he chooses (output)<br>2. The user balance increases accordingly to the product discount (output)<br>3. The merchant stock is updated (output) |
| **Steps** | 1. The consumer picks the product<br>2. The consumer pays the total cost of the product, may use its balance to pay<br>3. The merchant pays the discount back to the consumer<br>4. The consumer receives the discount on his account balance<br>5. The merchant pays a fee for the service |
| **Exceptions** | The merchant doesn't have the product |

Table 2: Buy a product scenario

| **Scenario** | **Invite a customer** |
|---|---|
| **Description** | A customer may invite a customer to be part of the system and, then, receives a bonus. |
| **Pre-conditions** | 1. Customer must be part of the system (initial state)<br>2. New customer must be new in the system (initial state) |
| **Post-conditions** | 1. New customer became part of the system (final system state)<br>2. The customer discount card balance increases the amount corresponding to the bonus (output) |
| **Steps** | 1. The customer invites a customer<br>2. The new customer enters the systems<br>3. The customer receives a bonus |
| **Exceptions** | The new customer is already in the system |

Table 3: Invite a customer scenario

| **Scenario** | **Invite a merchant** |
|---|---|
| **Description** | A merchant may invite a merchant to be part of the system and, then, receives a bonus. |
| **Pre-conditions** | 1. Merchant must me part of the system (initial state)<br>2. New merchant must be new on the system (initial state) |
| **Post-conditions** | 1. New merchant became part of the system (final system state)<br>2. The merchant balance increases the amount corresponding to the bonus (output) |
| **Steps** | 1. The merchant invites a merchant<br>2. The new merchant enters the systems<br>3. The merchant receives a bonus |
| **Exceptions** | The new merchant is already in the system |

Table 4: Invite a merchant scenario

| **Scenario** | **Add a product** |
|---|---|
| **Description** | A merchant can add new products |
| **Pre-conditions** | 1. The merchant must be part of the system (initial state)<br>2. The product should not exist on merchant's products list (initial state) |
| **Post-conditions** | The product should exist on merchant's products list (output) |
| **Steps** | The merchant adds a new product to the system |
| **Exceptions** | The product already exists on merchant's products list |

Table 5: Add a product scenario

| **Scenario** | **Removes a product** |
|---|---|
| **Description** | A merchant can remove a product from products list |
| **Pre-conditions** | 1. The merchant must be part of the system (initial state)<br>2. The product should exist on merchant's products list (initial state) |
| **Post-conditions** | The product should not exist on merchant's products list |
| **Steps** | The merchant removes a product from products list |
| **Exceptions** | The product doesn't exist on merchant's products list |

Table 6: Removes a product scenario

| Scenario | Consumer joins the system |
|---|---|
| **Description** | Consumers can join the system by receiving a discount card |
| **Pre-conditions** | The user is not in the system (initial system state) |
| **Post-conditions** | The user is now in the system (final system state) |
| **Steps** | 1. The consumer registers in the system<br>2. The consumer receives a discount card |
| **Exceptions** | The consumer is already in the system |

Table 7: Consumer joins the system scenario

| Scenario | Merchant joins the system |
|---|---|
| **Description** | Merchants can join the system |
| **Pre-conditions** | The Merchant is not in the system (initial system state) |
| **Post-conditions** | The Merchant is now in the system (final system state) |
| **Steps** | The Merchant registers in the system |
| **Exceptions** | The Merchant is already in the system |

Table 8: Merchant joins the system scenario

| Scenario | Transaction |
|---|---|
| **Description** | The consumer can transfer part of his balance to other adherent customer. |
| **Pre-conditions** | 1. Both consumer must exist in the system.<br>2. The amount to be transferred must be inferior or equal to the first consumer balance. |
| **Post-conditions** | 1. First consumer's balance decreases the amount corresponding to the transaction.<br>2. Second consumer's balance increases the amount corresponding to the transaction. |
| **Steps** | 1. The first consumer selects the second consumer that is going to receive the money.<br>2. The first consumer transferred the amount selected.<br>3. The second consumer receives the amount transferred. |
| **Exceptions** | 1. The first consumer doesn't have the selected amount.<br>2. The second consumer is not the in system. |

Table 9: Transaction scenario

## 2.2 Class model

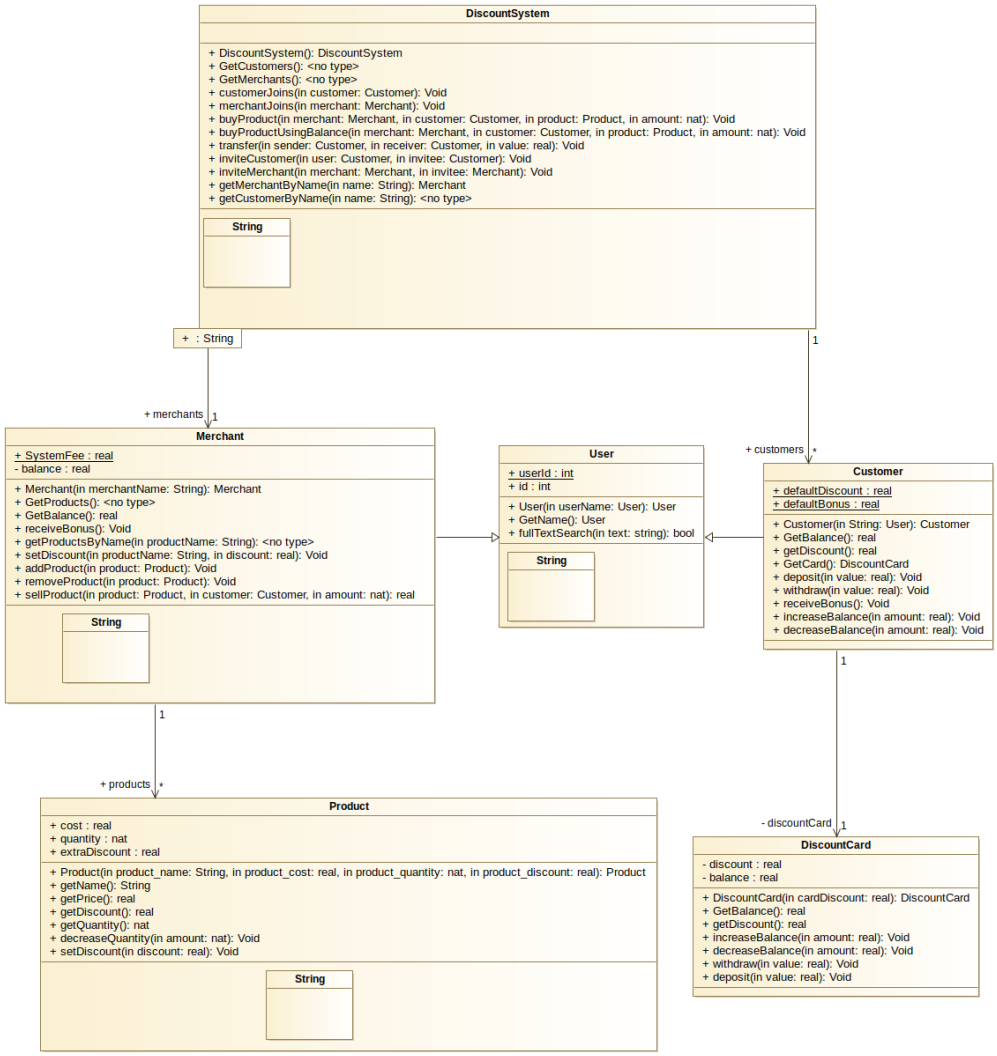


Figure 5: Class Diagram for DiscountSystem

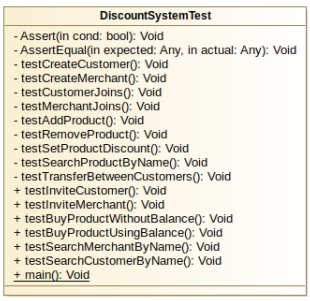


Figure 6: Class Diagram for DiscountSystemTest

| Class | Description |
|---|---|
| Customer | Defines a customer with a discount card |
| DiscountCard | Defines the discountCard to be used by the consumer |
| DiscountSystem | System that contains all the merchants and consumers |
| DiscountSystemTest | Defines the test cases for the discount system |
| Merchant | Defines a merchant which can have several products |
| Product | Defines a product at sale by a merchant |
| User | Superclass for defining a user which can be a merchant or a consumer. |

Table 10: Classes' Descriptions

# 3 Formal VDM++ model

## 3.1 Customer Class

```
class Customer is subclass of User
types

values
 public defaultDiscount: real = 5;
 public defaultBonus: real = 5;

instance variables
  private discountCard: DiscountCard := new DiscountCard(
      defaultDiscount);

operations

--Customer constructor, initializes the custumer and creates a
    discount card

public Customer : String ==> Customer
Customer(String) == ( discountCard := new DiscountCard(0.10);
    User(String););

-- Returns customer discount card balance

public GetBalance :  () ==> real
GetBalance() == return discountCard.GetBalance();

-- Returns customer discount value

public getDiscount : () ==> real
getDiscount() == return discountCard.getDiscount();

-- Returns customer discount card

public GetCard : () ==> DiscountCard
GetCard() == return discountCard;

-- Value is deposited on card

public deposit: real ==> ()
deposit(value) ==
 discountCard.deposit(value)
pre value >= 0;

-- Value is withdrawn from card

public withdraw : real ==> ()
withdraw(value) ==
 discountCard.withdraw(value)
pre value >= 0;

-- Customer receives a bonus

public receiveBonus: () ==> ()
receiveBonus() ==
 discountCard.deposit(defaultBonus);

-- Customer receives a part of each purchase

public increaseBalance : real ==> ()
increaseBalance(amount) ==
```

```
 discountCard.increaseBalance(amount);

-- Deacreses customer balance

public decreaseBalance : real ==> ()
decreaseBalance(amount) ==
 discountCard.decreaseBalance(amount);

functions
traces

end Customer
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Customer | 14 | 100.0% | 4 |
| GetBalance | 18 | 100.0% | 14 |
| GetCard | 26 | 100.0% | 1 |
| decreaseBalance | 52 | 100.0% | 2 |
| deposit | 30 | 100.0% | 2 |
| getDiscount | 22 | 100.0% | 14 |
| increaseBalance | 47 | 100.0% | 5 |
| receiveBonus | 42 | 100.0% | 1 |
| withdraw | 36 | 100.0% | 1 |
| Customer.vdmpp | | 100.0% | 44 |

## 3.2 DiscountCard Class

```
class DiscountCard
types

values

instance variables
 private discount: real;
 private balance: real := 0;

 inv balance >= 0;
operations

-- Discount Card constructor

public DiscountCard: real ==> DiscountCard
DiscountCard(cardDiscount) == (
 discount := cardDiscount;
 return self
);

-- Returns card balance

public GetBalance: () ==> real
GetBalance() == return balance;

-- Returns card discount

public getDiscount : () ==> real
getDiscount() == return discount;

-- Inscreses card Balance

public increaseBalance : real ==> ()
increaseBalance(amount) ==
 balance := balance + amount
 post balance = balance~ + amount;

-- Decreases balance from customer to a max minimum of 0

public decreaseBalance : real ==> ()
decreaseBalance(amount) ==
 if (balance - amount) > 0
 then
   balance := balance - amount
 else
  balance := 0
 post if (balance - amount) > 0
 then
   balance = balance~ - amount
 else
  balance = 0;

-- Withdraws from card

public withdraw : real ==> ()
withdraw(value) ==
 balance := balance - value
post balance = balance~ - value;

-- Deposits on card

public deposit : real ==> ()
```

```
deposit(value) ==
 balance := balance + value
post balance = balance~ +value;


functions

traces

end DiscountCard
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| DiscountCard | 14 | 100.0% | 8 |
| GetBalance | 21 | 100.0% | 15 |
| decreaseBalance | 35 | 100.0% | 2 |
| deposit | 55 | 100.0% | 3 |
| getDiscount | 25 | 100.0% | 14 |
| increaseBalance | 29 | 100.0% | 5 |
| withdraw | 49 | 100.0% | 1 |
| DiscountCard.vdmpp | | 100.0% | 48 |

## 3.3 DiscountSystem Class

```
class DiscountSystem
types
 public String = seq of char;
values
-- TODO Define values here
instance variables
 public customers: set of Customer := {};
 public merchants: map String to Merchant := {|->};

operations


 public DiscountSystem: () ==> DiscountSystem
 DiscountSystem() == return self;


 public GetCustomers: () ==> set of Customer
 GetCustomers() == return customers;


 public GetMerchants: () ==> map String to Merchant
 GetMerchants() == return merchants;

 -- Adds a customer to the system

 public customerJoins : Customer ==> ()
 customerJoins(customer) == customers := customers union {
     customer}
 pre customer not in set customers
 post customers = customers~ union {customer};

 -- Adds a merchant to the system

 public merchantJoins : Merchant ==> ()
 merchantJoins(merchant) == merchants := merchants munion {
     merchant.GetName() |-> merchant}
 pre merchant.GetName() not in set dom merchants
 post merchants = merchants~ munion {merchant.GetName() |->
     merchant};

 -- Customer buys a product from a merchant

 public buyProduct : Merchant * Customer * Product * nat ==> ()
 buyProduct(merchant, customer, product, amount) ==
  customer.increaseBalance(merchant.sellProduct(product, customer
      , amount))
 pre product.getQuantity() >= amount;

 -- Customer buys a product from a merchant using as much of is
     balance as possible

 public buyProductUsingBalance : Merchant * Customer * Product *
     nat ==> ()
 buyProductUsingBalance(merchant, customer, product, amount) ==
 (
   customer.decreaseBalance(product.getPrice());
   customer.increaseBalance(merchant.sellProduct(product,
       customer, amount));
 )
 pre product.getQuantity() >= amount;

 -- Customer transfers an amount of is balance to another
```

```
    customer

public transfer : Customer * Customer * real ==> ()
transfer(sender, receiver, value) == (sender.withdraw(value);
    receiver.deposit(value);)
pre sender in set customers and receiver in set customers and
    value >=0
post sender in set customers and receiver in set customers;


-- Custumer invites another customer to the system and receives
    a bonus

public inviteCustomer : Customer * Customer ==> ()
inviteCustomer(user, invitee) == (customerJoins(invitee); user.
    receiveBonus())
pre user in set customers and invitee not in set customers
post user in set customers and invitee in set customers;

-- Merchant invites another merchant to thte system and receives
     a bonus

public inviteMerchant : Merchant * Merchant ==> ()
inviteMerchant(merchant, invitee) == (merchantJoins(invitee);
    merchant.receiveBonus())
pre merchant.GetName() in set dom merchants and invitee.GetName
    () not in set dom merchants
post merchant.GetName() in set dom merchants and invitee.GetName
    () in set dom merchants;

-- Returns a list of merchants by name

public getMerchantByName : String ==> Merchant
getMerchantByName(name) == return merchants(name)
pre name in set dom merchants;

-- Returns a list of merchants by name

public getCustomerByName : String ==> set of Customer
getCustomerByName(name) ==
return {customer | customer in set customers & customer.
    fullTextSearch(name)};

functions
-- TODO Define functiones here

traces
-- TODO Define Combinatorial Test Traces here
end DiscountSystem
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| DiscountSystem | 12 | 100.0% | 9 |
| GetCustomers | 15 | 100.0% | 1 |
| GetMerchants | 18 | 100.0% | 1 |
| buyProduct | 34 | 100.0% | 3 |
| buyProductUsingBalance | 40 | 100.0% | 2 |
| customerJoins | 22 | 100.0% | 9 |
| getCustomerByName | 73 | 100.0% | 1 |
| getMerchantByName | 68 | 100.0% | 1 |
| inviteCustomer | 56 | 100.0% | 1 |
| inviteMerchant | 62 | 100.0% | 3 |
| merchantJoins | 28 | 100.0% | 6 |
| transfer | 49 | 100.0% | 1 |
| DiscountSystem.vdmpp | | 100.0% | 38 |

## 3.4 Merchant Class

```
class Merchant is subclass of User
types
 public String = seq of char;

values
 public SystemFee: real = 3;

instance variables
 public products : set of Product := {};
 private balance: real := 0;

operations

-- Merchant constructor

public Merchant : String ==> Merchant
Merchant(merchantName) == User(merchantName
);

-- Returns merchant's products

public GetProducts: () ==> set of Product
GetProducts() == return products;

-- Returns merchant's balance

public GetBalance : () ==> real
GetBalance() == return balance;

-- Merchant receives a bonus

public receiveBonus: () ==> ()
receiveBonus() == ( balance:= balance + 5)
post balance = balance~ + 5;

-- Returns a product searched by name

public getProductsByName : String ==> set of Product
getProductsByName(productName) ==
return {product | product in set products & product.name =
    productName};

-- Modify discounts on selected products

public setDiscount : String * real ==> ()
setDiscount(productName, discount) ==
 for all product in set getProductsByName(productName) do
  product.setDiscount(discount);

-- Adds a product to merchant's products list

public addProduct : Product ==> ()
addProduct(product) ==
 products := products union {product}
 pre product not in set products
 post products = products~ union {product};

-- Removes a product from merchant's products list

public removeProduct : Product ==> ()
removeProduct(product) ==
 products := products  \ {product}
```

```
 pre product in set products
 post products~ = products union {product} and product not in set
      products;


-- Merchant sell a product, decreasing the product quantity and
    increasing his balance

public sellProduct : Product * Customer * nat ==> real
sellProduct(product, customer, amount) ==
 (
  balance := balance + ((product.getPrice() * (1-product.
      getDiscount()/100)) * (1-customer.getDiscount()/100) * (
      SystemFee/100)) * amount;
  product.decreaseQuantity(amount);
  return (product.getPrice() - (product.getPrice() * (1-product.
      getDiscount()/100)) * (1-customer.getDiscount()/100)) *
      amount;
 )
 pre (
  product.getQuantity() >= amount and
  product in set products
 );

functions

traces

end Merchant
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| GetBalance | 24 | 100.0% | 7 |
| GetProducts | 20 | 100.0% | 3 |
| Merchant | 15 | 100.0% | 4 |
| addProduct | 44 | 100.0% | 5 |
| getProductsByName | 33 | 100.0% | 4 |
| receiveBonus | 28 | 100.0% | 1 |
| removeProduct | 51 | 100.0% | 1 |
| sellProduct | 59 | 100.0% | 5 |
| setDiscount | 38 | 100.0% | 1 |
| Merchant.vdmpp | | 100.0% | 31 |

## 3.5 Product Class

```
class Product

types
 public String = seq of char;

values

instance variables
 public name : String;
 public cost : real;
 public quantity: nat;
 public extraDiscount : real;
operations

-- Product constructor

public Product : String * real * nat * real ==> Product
Product(product_name, product_cost, product_quantity,
    product_discount) ==
(
 name := product_name;
 cost := product_cost;
 quantity := product_quantity;
 extraDiscount := product_discount;
 return self;
);

-- Returns product name

public getName : ()  ==> String
getName() == return name;

-- Returns product price

public getPrice : () ==> real
getPrice()  == return cost;

-- Returns product discount

public getDiscount : () ==> real
getDiscount() == return extraDiscount;

-- Returns the existing quantity for the product

public pure getQuantity : () ==> nat
getQuantity() == return quantity;

-- Descreases product quantity

public decreaseQuantity : nat ==> ()
decreaseQuantity(amount) ==
 quantity := quantity - amount
 pre (quantity - amount) >= 0
 post quantity~ = quantity + amount;

-- Modifies product disccount

public setDiscount : real ==> ()
setDiscount(discount) ==
 extraDiscount := discount
 post extraDiscount = discount;
```

```
functions

traces

end Product
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Product | 16 | 100.0% | 3 |
| decreaseQuantity | 43 | 100.0% | 5 |
| getDiscount | 35 | 100.0% | 17 |
| getName | 27 | 100.0% | 4 |
| getPrice | 31 | 100.0% | 24 |
| getQuantity | 39 | 100.0% | 15 |
| setDiscount | 50 | 100.0% | 1 |
| Product.vdmpp | | 100.0% | 69 |

## 3.6 User Class

```
class User
types
 public String = seq of char;

values

instance variables
 private name : String;
 public static userId : int := 0;
 public id : int := userId;

operations

-- User Constructor

public User: String ==> User
User(userName) == (
    name := userName;
    id := userId;
    userId := userId + 1;
    return self
);

-- Returns user name

public pure GetName : () ==> String
GetName() == return name;


public fullTextSearch : seq of char ==> bool
fullTextSearch(text) == (
 dcl temp_text: seq of char := name;
 dcl match: bool := false;

 while len temp_text >= len text and not match do(
  match := true;

  for index = 1 to len text do
   if match and text(index) <> temp_text(index) then (
    match := false;
   );

  if match then
   return true
  else (
   temp_text := tl temp_text;
   match := false;
  );

 );

 return false;

)
pre len text > 0;

functions
traces

end User
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| GetName | 24 | 100.0% | 33 |
| User | 15 | 100.0% | 8 |
| fullTextSearch | 27 | 100.0% | 4 |
| User.vdmpp | | 100.0% | 45 |

# 4 Model validation

## 4.1 DiscountSystemTest Class

```
class DiscountSystemTest
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
customer1 : Customer := new Customer("Rui");
customer2 : Customer := new Customer("Joana");
merchant1 : Merchant := new Merchant("zara");
product: Product := new Product("botas",24.99, 5, 5)

operations


private Assert : bool ==> ()
Assert(cond) == return
pre cond;


private AssertEqual: ? * ? ==> ()
 AssertEqual(expected, actual) ==
  if expected <> actual then (
     IO`print("Actual value (");
     IO`print(actual);
     IO`print(") different from expected (");
     IO`print(expected);
     IO`println(")\n")
  )
 post expected = actual;



private testCreateCustomer: () ==> ()
testCreateCustomer() ==
(
 dcl customer: Customer := new Customer("Rui");
 Assert(customer.GetName() = "Rui");
 Assert(customer.GetBalance() = 0);

 Assert(customer.GetCard().GetBalance() = 0);


);

private testCreateMerchant: () ==> ()
testCreateMerchant() ==
(
dcl merchant: Merchant := new Merchant("zara");
Assert(merchant.GetName() = "zara");
Assert(merchant.GetProducts() = {});

Assert(merchant.GetBalance() = 0);
);

private testCustomerJoins: () ==> ()
testCustomerJoins() ==
(
 dcl customer : Customer := new Customer("Rui");
 dcl system: DiscountSystem := new DiscountSystem();
```

```
 Assert(customer not in set system.customers);
 system.customerJoins(customer);
 Assert(customer in set system.GetCustomers());
);

private testMerchantJoins: () ==> ()
testMerchantJoins() ==
(
 dcl merchant : Merchant := new Merchant("zara");

 dcl system: DiscountSystem := new DiscountSystem();
 Assert(merchant.GetName() not in set dom system.merchants);
 system.merchantJoins(merchant);
 Assert(merchant.GetName() in set dom system.GetMerchants());
);

private testAddProduct: () ==> ()
testAddProduct() ==
(
 Assert(product not in set merchant1.GetProducts());
 merchant1.addProduct(product);
 Assert(product in set merchant1.GetProducts());

 Assert(product.getName() = "botas");
 Assert(product.getPrice() = 24.99 );
 Assert(product.getQuantity() = 5);
 Assert(product.getDiscount() = 5)
);

private testRemoveProduct: () ==> ()
testRemoveProduct() ==
(
 if product not in set merchant1.products then

  merchant1.addProduct(product);

 merchant1.removeProduct(product);
 Assert( product not in set merchant1.products);
);

private testSetProductDiscount: () ==> ()
testSetProductDiscount() ==
(
 Assert(product.getDiscount() = 5);

 merchant1.addProduct(product);
 merchant1.setDiscount(product.getName(), 10);
 Assert(product.getDiscount() = 10);
);


private testSearchProductByName: () ==> ()
testSearchProductByName() ==
(
 dcl merchant: Merchant:= new Merchant("new merchant");

 AssertEqual({}, merchant.getProductsByName(product.getName()));
 merchant.addProduct(product);
 Assert(product in set merchant.getProductsByName(product.getName
     ()))
);

private testTransferBetweenCustomers: () ==> ()
```

```
testTransferBetweenCustomers() ==
(
 dcl customer : Customer := new Customer("Rui");
 dcl system: DiscountSystem := new DiscountSystem();

 system.customerJoins(customer);
 system.customerJoins(customer2);

 --sender
 AssertEqual(0,customer.GetBalance());
 customer.deposit(10);
 AssertEqual(10,customer.GetBalance());

 --receiver
 AssertEqual(0, customer2.GetBalance());


 --transfer 10 from customer1 to customer2
 system.transfer(customer, customer2, 10);
 AssertEqual(0, customer.GetBalance());
 AssertEqual(10, customer2.GetBalance());
);

private testInviteCustomer: () ==> ()
testInviteCustomer() ==
(
 dcl invitee: Customer := new Customer("Invitee User");
 dcl balance: real := customer1.GetBalance();
 dcl system: DiscountSystem := new DiscountSystem();

 -- Customer 1 is part of the system
 system.customerJoins(customer1);
 Assert(customer1 in set system.customers);

 -- Invitee is new on the system
  Assert(invitee not in set system.customers);

 -- Customer1 invites invitee
  system.inviteCustomer(customer1, invitee);

 -- Invitee becames a member of the system

  Assert(invitee in set system.customers);

  -- Customer Receives a bonus
  AssertEqual(balance + 5,customer1.GetBalance());

);

private testInviteMerchant: () ==> ()
testInviteMerchant() ==
(
 dcl merchant : Merchant := new Merchant("massimo");
 dcl system: DiscountSystem := new DiscountSystem();
 dcl invitee: Merchant := new Merchant("Invitee Merchant");
 dcl balance: real := merchant1.GetBalance();


  -- Merchant1 is part of the system
 system.merchantJoins(merchant);
 Assert(merchant.GetName() in set dom system.merchants);

  -- Invitee is new on the system
  Assert(invitee.GetName() not in set dom system.merchants);
```

```
 -- Merchant1 invites invitee

  system.inviteMerchant(merchant, invitee);

  -- Invitee becames a member of the system
  Assert(invitee.GetName() in set dom system.merchants);

  -- Merchant1 Receives a bonus
  AssertEqual(balance + 5,merchant.GetBalance());
);

private testBuyProductWithoutBalance: () ==> ()
testBuyProductWithoutBalance() ==
(
 dcl merchant : Merchant := new Merchant("zara");
 dcl system: DiscountSystem := new DiscountSystem();
 dcl customerBalance: real := customer1.GetBalance();
 dcl merchantBalance: real := merchant.GetBalance();
 dcl perfume : Product := new Product("perfume", 13.5, 10, 5);
 dcl quantity: nat := perfume.getQuantity();
 dcl amount: nat := 1;
 dcl merchantReturn : real := ((perfume.getPrice() * (1-perfume.
     getDiscount()/100)) * (1-customer1.getDiscount()/100) * (
     merchant.SystemFee/100)) * amount;
 dcl customerReturn : real := (perfume.getPrice() - (perfume.
     getPrice() * (1-perfume.getDiscount()/100)) * (1-customer1.
     getDiscount()/100)) * amount;


 -- Prepare System
 system.merchantJoins(merchant);
 system.customerJoins(customer1);

 -- Adding product
 merchant.addProduct(perfume);

 --Customer buys a product
 system.buyProduct(merchant, customer1, perfume, amount);


 -- Merchant sells product
 AssertEqual(quantity-1, perfume.getQuantity());

 -- Merchant balance increases
 AssertEqual(merchantBalance + merchantReturn, merchant.
     GetBalance());

 -- Customer balance increases
 AssertEqual(customerBalance + customerReturn, customer1.
     GetBalance());

);

private testBuyProductUsingBalance: () ==> ()
testBuyProductUsingBalance() ==
(
 dcl merchant : Merchant := new Merchant("zara");
 dcl system: DiscountSystem := new DiscountSystem();
 dcl customerBalance: real := customer1.GetBalance();
 dcl merchantBalance: real := merchant.GetBalance();
 dcl perfume : Product := new Product("perfume", 100, 20, 50);
 dcl quantity: nat := perfume.getQuantity();
 dcl amount: nat := 1;
 dcl merchantReturn : real := ((perfume.getPrice() * (1-perfume.
     getDiscount()/100)) * (1-customer1.getDiscount()/100) * (
```

```
    merchant.SystemFee/100)) * amount;
 dcl customerReturn : real := (perfume.getPrice() - (perfume.
    getPrice() * (1-perfume.getDiscount()/100)) * (1-customer1.
    getDiscount()/100)) * amount;

 -- Prepare System
 system.merchantJoins(merchant);
 system.customerJoins(customer1);

 -- Adding product
 merchant.addProduct(perfume);

 --Customer buys a product
 system.buyProduct(merchant, customer1, perfume, amount);

 -- Merchant sells product
 AssertEqual(quantity-1, perfume.getQuantity());

 -- Merchant balance increases
 AssertEqual(merchantBalance + merchantReturn, merchant.
    GetBalance());

 -- Customer balance increases
 AssertEqual(customerBalance + customerReturn, customer1.
    GetBalance());

 system.buyProductUsingBalance(merchant, customer1, perfume,
    amount);


 -- Customer uses 50.05euros balance to buy 100euros using 50.05
    euros from the balance and 49.95euros from cash, and
    receives new bonus, keeping the same balance
 AssertEqual(customerReturn, customer1.GetBalance());

 system.buyProduct(merchant, customer1, perfume, 5);

 system.buyProductUsingBalance(merchant, customer1, perfume,
    amount);

 Assert(customer1.GetBalance() > customerReturn);

);

private testSearchMerchantByName: () ==> ()

testSearchMerchantByName() ==
(
 dcl merchant : Merchant := new Merchant("zara");
 dcl system: DiscountSystem := new DiscountSystem();

 -- Prepare System
 system.merchantJoins(merchant);

 -- Search for merchant
 AssertEqual(system.getMerchantByName("zara").GetName(), merchant
    .GetName());

);

private testSearchCustomerByName: () ==> ()
testSearchCustomerByName() ==
(

 dcl customer3 : Customer := new Customer("Rui");
```

```
 dcl system: DiscountSystem := new DiscountSystem();

 -- Prepare System
 system.customerJoins(customer3);
 system.customerJoins(customer2);

 -- Search for customer
  for all customer in set system.getCustomerByName("Rui") do
   AssertEqual(customer.GetName(), customer3.GetName());


);

public static main: () ==> ()
main() ==
(
 dcl systemTest: DiscountSystemTest := new DiscountSystemTest();

 IO`print("Create Customer: " );
 systemTest.testCreateCustomer();
 IO`println("Success");

 IO`print("Create Merchant: ");
 systemTest.testCreateMerchant();
 IO`println("Success");

 IO`print("Customer Joins: ");
 systemTest.testCustomerJoins();
 IO`println("Success");

 IO`print("Merchant Joins: ");
 systemTest.testMerchantJoins();
 IO`println("Success");

 IO`print("Add Product: ");
 systemTest.testAddProduct();
 IO`println("Success");

 IO`print("Remove Product: ");
 systemTest.testRemoveProduct();
 IO`println("Success");

 IO`print("Set Product Discount: ");
 systemTest.testSetProductDiscount();
 IO`println("Success");

 IO`print("Search Product By Name: ");
 systemTest.testSearchProductByName();
 IO`println("Success");

 IO`print("Transfer to Customer: ");
 systemTest.testTransferBetweenCustomers();
 IO`println("Success");

 IO`print("Customer invites Customer and receives a bonus: ");
 systemTest.testInviteCustomer();
 IO`println("Success");

 IO`print("Merchant invites Merchant and receives a bonus: ");
 systemTest.testInviteMerchant();
 IO`println("Success");

 IO`print("Buy Product without discount balance: ");
 systemTest.testBuyProductWithoutBalance();
 IO`println("Success");
```

```
 IO`print("Buy Product using current balance: ");
 systemTest.testBuyProductUsingBalance();
 IO`println("Success");

 IO`print("Search Merchant by name: ");
 systemTest.testSearchMerchantByName();
 IO`println("Success");

 IO`print("Search Customer by name: ");
 systemTest.testSearchCustomerByName();
 IO`println("Success");
)


functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
CreateCustomer: testCreateCustomer();
CreateMerchant: testCreateMerchant();
CustomerJoins: testCustomerJoins();
MerchantJoins: testMerchantJoins();
AddProduct: testAddProduct();
RemoveProduct: testRemoveProduct();
SetProductDiscount: testSetProductDiscount();
SearchProductByName: testSearchProductByName();
TransferBetweenCustomers: testTransferBetweenCustomers();
InviteCustomerReceiveBonus: testInviteCustomer();
InviteMerchantReceiveBonus: testInviteMerchant();
BuyProductWithoutBalance: testBuyProductWithoutBalance();
BuyProductUsingBalance: testBuyProductUsingBalance();
SearchMerchantByName: testSearchMerchantByName();
SearchConsumerByName: testSearchCustomerByName();

end DiscountSystemTest
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Assert | 14 | 100.0% | 135 |
| AssertEqual | 18 | 38.8% | 0 |
| main | 283 | 100.0% | 5 |
| testAddProduct | 65 | 100.0% | 15 |
| testBuyProductUsingBalance | 210 | 100.0% | 5 |
| testBuyProductWithoutBalance | 177 | 100.0% | 5 |
| testCreateCustomer | 30 | 100.0% | 15 |
| testCreateMerchant | 38 | 100.0% | 5 |
| testCustomerJoins | 47 | 100.0% | 5 |
| testInviteCustomer | 128 | 100.0% | 5 |
| testInviteMerchant | 153 | 100.0% | 2 |
| testMerchantJoins | 56 | 100.0% | 5 |
| testRemoveProduct | 77 | 86.6% | 5 |
| testSearchCustomerByName | 267 | 100.0% | 1 |
| testSearchMerchantByName | 254 | 100.0% | 1 |
| testSearchProductByName | 97 | 100.0% | 5 |
| testSetProductDiscount | 87 | 100.0% | 5 |
| testTransferBetweenCustomers | 107 | 100.0% | 1 |
| DiscountSystemTest.vdmpp | | 90.0% | 220 |

# 5 Model verification

## 5.1 Example of domain verification

One of the proof obligations generated by Overture is:

| No. | PO Name | Type |
|---|---|---|
| 21 | DiscountSystem'getMerchantByName | legal map application |

Table 18: Example of proof obligation generated by Overture

The code under analysis is:

```
public getMerchantByName : String ==> Merchant
getMerchantByName(name) == return merchants(name)
pre name in set dom merchants;
```

In this case, we can easly prove the pre-condition - name **in set dom** merchants - that assures the map is only accessed inside the its domain.

## 5.2 Example of invariant verification

Another proof obligations generated by Overture is:

| No. | PO Name | Type |
|---|---|---|
| 6 | DiscountCard'decreaseBalance | state invariant holds |

Table 19: Example of proof obligation generated by Overture

The code under analysis is:

```
-- Decreases balance from customer to a max minimum of 0
public decreaseBalance : real ==> ()
decreaseBalance(amount) ==
  if (balance - amount) >= 0
  then
    balance := balance - amount
  else
    balance := 0
  post if (balance - amount) >= 0
  then
    balance = balance~ - amount
  else
    balance = 0;
```

In this case, we can easly prove the invariant holds true because we are only decreasing the value of balance as long as it is bigger or equal than 0, which is its the invariant definition.

# 6 Conclusions

The team developed all the requirements planned for the project and also developed a console interface with all the functionalities working as excepted.

With more time the team would like to specify and test the behavior of exceptional conditions and would wish to construct a swing interface, more friendly and interactive than the console.

This project took the team 40 hours to develop.

# 7 References


1. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
2. Overture tool web site, http://overturetool.org
3. Ana Paiva, MFESVDM++ Slides, https://moodle.up.pt/pluginfile.php/165033/mod_resource/content/0/VDM%2B%2B.pdfMFESVDM++.ppt