Métodos Formais em Engenharia de Software

MIEIC

Universidade do Porto

Faculdade de Engenharia

FEUP

THEME: SHOPADVIZOR

JANUARY 7, 2019

FRANCISCO MARIA FERNANDES MACHADO SANTOS
UP201607928
4MIEIC02 – GROUP 3

Table of Contents

1. Informal System Description

The project is intended to be a collaborative platform between people and brands where products are exposed and can be reviewed as well as rated.

I decided to take these requirements further and conceived a project to implement the requested base operations together with a shop functionality.

This way this platform can be seen as a shop with the following operations:

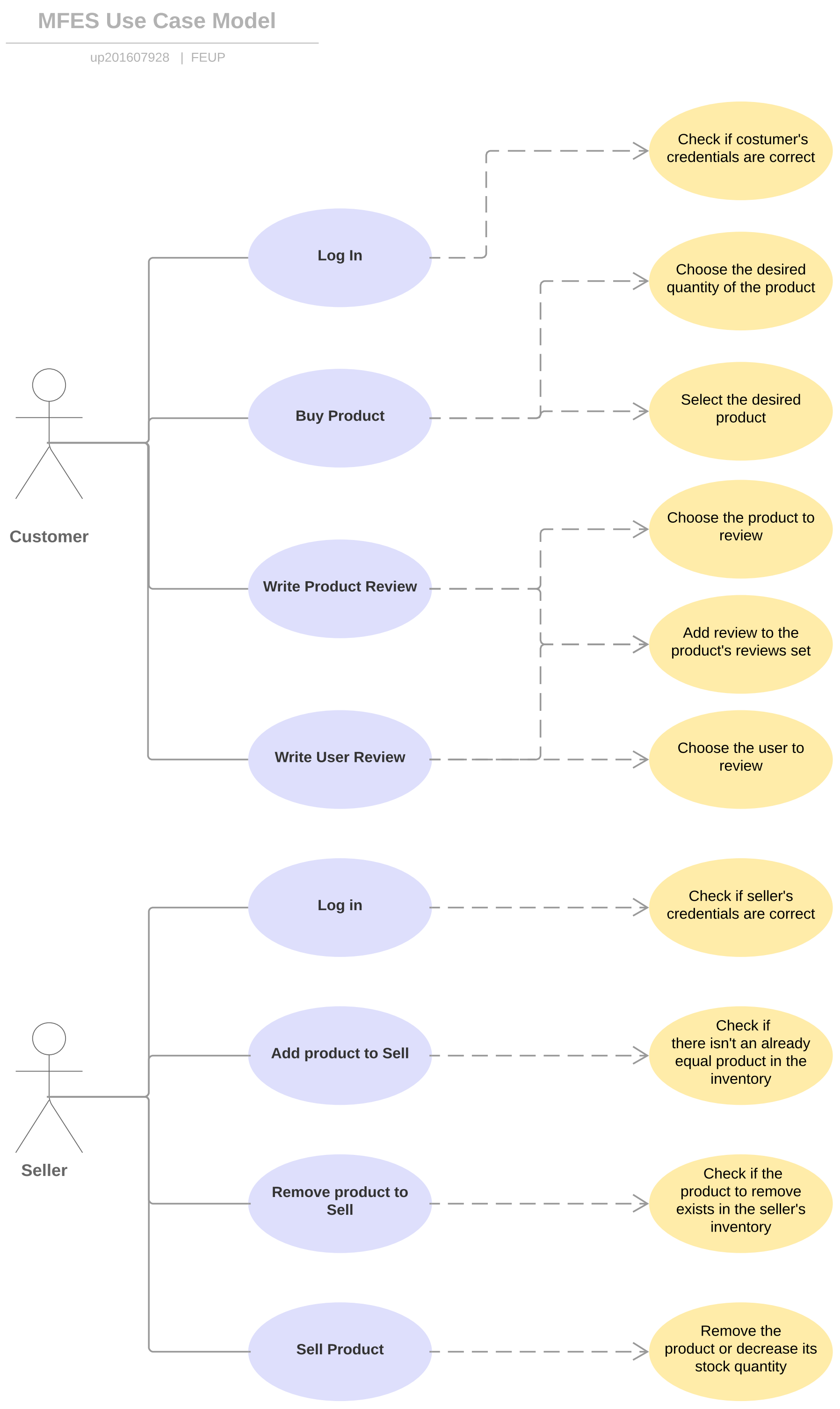
- Register new users support
- Two types of users can be registered to the shop (Sellers and Costumers)
- Authorization operations (Log in and Log out of users)
- Products can be assigned to sellers
- All products and users can be reviewed and rated
- Customers can buy products from seller

2. List of Requirements

ID	*Priority*	*Description*
R1	Mandatory	A costumer may be able to buy products from a seller
R2	Mandatory	A costumer may review and rate products
R3	Mandatory	A seller can add as much products to his/hers inventory as he/she wants
R4	Mandatory	A seller can remove a product from his/hers inventory
R5	Mandatory	A seller can change the stock (quantity) of a product in his/hers inventory
R6	Mandatory	The platform has to keep track of all users and products that exist
R7	Optional	The platform (shop) has to have a register support to add new users to it
R8	Optional	The platform (shop) has to have an authentication support for the users log in and log out.

3. Visual UML Model

3.1 Use Case Model:


MFES Use Case Model
up201607928 | FEUP
Customer
Log In
Buy Product
Write Product Review
Write User Review
Check if costumer's credentials are correct
Choose the desired quantity of the product
Select the desired product
Choose the product to review
Add review to the product's reviews set
Choose the user to review
Seller
Log in
Add product to Sell
Remove product to Sell
Sell Product
Check if seller's credentials are correct
Check if there isn't an already equal product in the inventory
Check if the product to remove exists in the seller's inventory
Remove the product or decrease its stock quantity

Below are represented the specified descriptions of the major use cases:

Scenario	*Costumer Buys Product*
Description	A costumer buys a product listed from his/hers home page from a determined seller.
Pre-conditions	1. The product to buy has to have in stock a quantity bigger or equal than the one that the costumer wants.
Post-conditions	1. The seller's inventory is updated with the bought product alteration.

Scenario	*Costumer Writes a Review*
Description	Costumer writes a review to a product or to a user.
Pre-conditions	1. The rating of the review must be a number comprehended between 1 and 5.
Post-conditions	1. The review is added to the respective seller's product reviews set.

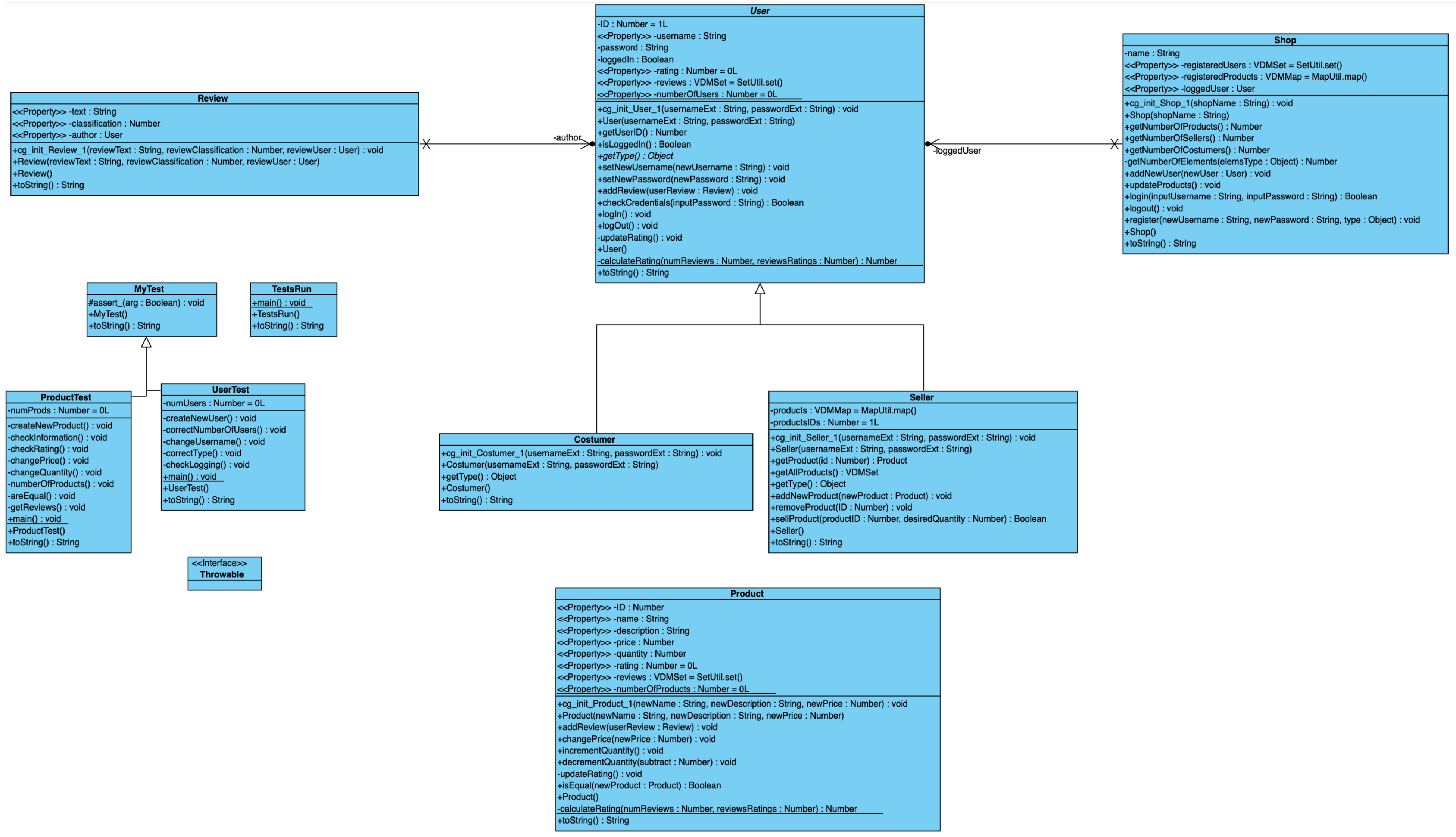
Scenario	*Seller Adds New Product*
Description	A seller adds a new product to sell in the shop so it may be available for any costumer to buy.
Pre-conditions	1. The product information may not be null and its price shall not be lesser than 0. 2. The product has to have a stock quantity bigger or equal than 1.
Post-conditions	1. The product is added to the seller's inventory. 2. The number of products in the shop is incremented.

Scenario	*Seller Removes Product*
Description	A seller decides to remove a certain product from its inventory
Pre-conditions	1. The product to remove exists in the inventory of the seller.
Post-conditions	1. The products ceases to exist in the sellers inventory.

Scenario	*Seller Sells Product*
Description	A seller sells a certain product from its inventory.
Pre-conditions	1. The product to sell exists in the inventory of the seller. 2. The desired quantity to sell of the product has to be in stock.
Post-conditions	1. The product inventory is updated with the removal of the product if the desired quantity is as much as of the stock or with the decreased quantity of the stock of the product.

3.2 Class Diagram of the Model:

OBS: Unfortunately, due to the wide width of the generated UML, the only way it could fit into the report was if it was pasted sideways. This way it is presented in the next page the UML class diagram of the created model.


Review
<<Property>> -text : String
<<Property>> -classification : Number
<<Property>> -author : User
+cg_init_Review_1(reviewText : String, reviewClassification : Number, reviewUser : User) : void
+Review(reviewText : String, reviewClassification : Number, reviewUser : User)
+Review()
+toString() : String
-author
User
-ID : Number = 1L
<<Property>> -username : String
-password : String
-loggedIn : Boolean
<<Property>> -rating : Number = 0L
<<Property>> -reviews : VDMSet = SetUtil.set()
<<Property>> -numberOfUsers : Number = 0L
+cg_init_User_1(usernameExt : String, passwordExt : String) : void
+User(usernameExt : String, passwordExt : String)
+getUserID() : Number
+isLoggedIn() : Boolean
+getType() : Object
+setNewUsername(newUsername : String) : void
+setNewPassword(newPassword : String) : void
+addReview(userReview : Review) : void
+checkCredentials(inputPassword : String) : Boolean
+logIn() : void
+logOut() : void
-updateRating() : void
+User()
-calculateRating(numReviews : Number, reviewsRatings : Number) : Number
+toString() : String
-loggedUser
Shop
-name : String
<<Property>> -registeredUsers : VDMSet = SetUtil.set()
<<Property>> -registeredProducts : VDMMap = MapUtil.map()
<<Property>> -loggedUser : User
+cg_init_Shop_1(shopName : String) : void
+Shop(shopName : String)
+getNumberOfProducts() : Number
+getNumberOfSellers() : Number
+getNumberOfCostumers() : Number
-getNumberOfElements(elemsType : Object) : Number
+addNewUser(newUser : User) : void
+updateProducts() : void
+login(inputUsername : String, inputPassword : String) : Boolean
+logout() : void
+register(newUsername : String, newPassword : String, type : Object) : void
+Shop()
+toString() : String
MyTest
#assert_(arg : Boolean) : void
+MyTest()
+toString() : String
TestsRun
+main() : void
+TestsRun()
+toString() : String
ProductTest
-numProds : Number = 0L
-createNewProduct() : void
-checkInformation() : void
-checkRating() : void
-changePrice() : void
-changeQuantity() : void
-numberOfProducts() : void
-areEqual() : void
-getReviews() : void
+main() : void
+ProductTest()
+toString() : String
UserTest
-numUsers : Number = 0L
-createNewUser() : void
-correctNumberOfUsers() : void
-changeUsername() : void
-correctType() : void
-checkLogging() : void
+main() : void
+UserTest()
+toString() : String
<<Interface>>
Throwable
Costumer
+cg_init_Costumer_1(usernameExt : String, passwordExt : String) : void
+Costumer(usernameExt : String, passwordExt : String)
+getType() : Object
+Costumer()
+toString() : String
Seller
-products : VDMMap = MapUtil.map()
-productsIDs : Number = 1L
+cg_init_Seller_1(usernameExt : String, passwordExt : String) : void
+Seller(usernameExt : String, passwordExt : String)
+getProduct(id : Number) : Product
+getAllProducts() : VDMSet
+getType() : Object
+addNewProduct(newProduct : Product) : void
+removeProduct(ID : Number) : void
+sellProduct(productID : Number, desiredQuantity : Number) : Boolean
+Seller()
+toString() : String
Product
<<Property>> -ID : Number
<<Property>> -name : String
<<Property>> -description : String
<<Property>> -price : Number
<<Property>> -quantity : Number
<<Property>> -rating : Number = 0L
<<Property>> -reviews : VDMSet = SetUtil.set()
<<Property>> -numberOfProducts : Number = 0L
+cg_init_Product_1(newName : String, newDescription : String, newPrice : Number) : void
+Product(newName : String, newDescription : String, newPrice : Number)
+addReview(userReview : Review) : void
+changePrice(newPrice : Number) : void
+incrementQuantity() : void
+decrementQuantity(subtract : Number) : void
-updateRating() : void
+isEqual(newProduct : Product) : Boolean
+Product()
-calculateRating(numReviews : Number, reviewsRatings : Number) : Number
+toString() : String

4. Formal VDM++ Model

4.1 Class User:

```
class User
--abstract class

types
	public String = seq of char;
	public Float = real;

	-- sets the type of user: the seller and the costumer
	public static UserType = <SELLER> | <COSTUMER>;

values
	--constantes
instance variables
	private ID : nat1 := 1;
	private username : String;
	private password : String;
	private loggedIn : bool;
	private rating : Float := 0;
	private reviews : set of Review := {};

	private static numberOfUsers : int := 0;

	inv ID <= numberOfUsers and ID > 0;
	inv rating >= 0 and rating <= 5;

operations
	-------------------------------------------
	-- CONSTRUCTOR
	-------------------------------------------

	public User : String * String ==> User
	User(usernameExt, passwordExt) ==
		(
			username := usernameExt;
			password := passwordExt;
			numberOfUsers := numberOfUsers + 1;
			ID := numberOfUsers;
			loggedIn := false;
		)
		pre usernameExt <> "" and passwordExt <> "";

	-------------------------------------------
	-- ACCESSOR OPERATIONS
	-------------------------------------------

	public getUserID: () ==> nat1
	getUserID() ==
	(
		return ID;
	);
```

```
public getUsername: () ==> String
getUsername() ==
(
	return username;
);

public getRating: () ==> Float
getRating() ==
(
	return rating;
);

public getReviews: () ==> set of Review
getReviews() ==
(
	return reviews;
);

public isLoggedIn: () ==> bool
isLoggedIn() ==
(
	return loggedIn;
);

-- abstract function to check what type the user is (seller or customer)
public getType: () ==> UserType
getType() == is subclass responsibility;

public static getNumberOfUsers: () ==> int
getNumberOfUsers() ==
(
	return numberOfUsers;
);

-----------------------------------------
-- MODIFIER OPERATIONS
-----------------------------------------
-- changes the user's username
public setNewUsername: String ==> ()
setNewUsername(newUsername) ==
(
	username := newUsername;
)
pre newUsername <> "" and loggedIn = true
post username = newUsername;

-- changes the user's password
public setNewPassword: String ==> ()
setNewPassword(newPassword) ==
(
	password := newPassword;
)
pre newPassword <> "" and loggedIn = true
post password = newPassword;

-- adds a review to the user
public addReview: Review ==> ()
addReview(userReview) ==
        (
                reviews := reviews union {userReview};
                updateRating();
        )
        pre userReview.getAuthor() <> self;

        -- checks if the credentials are correct:
        -- if true the user automatically logs in
        public checkCredentials: String ==> bool
        checkCredentials(inputPassword) ==
        (
                if password = inputPassword
                then (
                        loggedIn := true;
                        return true;
                );
                return false;
        );

        -- log in operation
        public logIn: () ==> ()
        logIn() ==
        (
                loggedIn := true;
        )
        pre loggedIn = false
        post loggedIn = true;

        -- log out operation
        public logOut: () ==> ()
        logOut() ==
        (
                loggedIn := false;
        )
        pre loggedIn = true
        post loggedIn = false;

        -- This operation updates the average rating of the user
        -- taking in account all the ratings it had so far
        private updateRating: () ==> ()
        updateRating() ==
        (
                dcl allRatingsSum : nat := 0, numRatings : nat := 0;
                for all r in set reviews do
                (
                        allRatingsSum := allRatingsSum + r.getClassification();
                        numRatings := numRatings + 1;
                );
                rating := calculateRating(numRatings, allRatingsSum);
        );

functions

        --this function calculates the average product rating
        private calculateRating: nat * nat -> Float
        calculateRating(numReviews, reviewsRatings) ==
        (
                if numReviews = 0 then 0
```

```
             else
                    reviewsRatings / numReviews
        );

Traces

end User
```

4.2 Class Costumer:

```
class Costumer is subclass of User
types
values
instance variables
operations
        ------------------------------------------
        -- CONSTRUCTOR
        ------------------------------------------
        public Costumer : String * String ==> Costumer
        Costumer(usernameExt, passwordExt) ==
        (
              User`User(usernameExt, passwordExt);
        );

        -- gets the type of the user
        public getType: () ==> UserType
        getType() ==
        (
              return <COSTUMER>;
        );

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Costumer
```

4.3 Class Seller:

```
class Seller is subclass of User
types
  public Products = set of Product;
  public ProductName = seq of char;

values
instance variables

        -- product set representing the inventory of the items the seller is
selling.
        -- each product within the sellers product inventory is characterized by an
ID.
        private products : map nat1 to Product := { |-> };
        -- var to keep track of the seller products IDS
        private productsIDs : nat1 := 1;


operations
        ------------------------------------------
        -- CONSTRUCTOR
        ------------------------------------------

        public Seller : String * String ==> Seller
        Seller(usernameExt, passwordExt) ==
        (
                User`User(usernameExt, passwordExt);
        );

        ------------------------------------------
        -- ACCESSOR OPERATIONS
        ------------------------------------------

        -- Get the product by its ID in the inventory.
        public getProduct: nat1 ==> Product
        getProduct(id) ==
        (
                return products(id);
        )
        pre id in set dom products;

        -- returns the range of products in the map
        public getAllProducts: () ==> Products
        getAllProducts() ==
        (
                return rng products;
        );

        public getType: () ==> UserType
        getType() ==
        (
                return <SELLER>;
        );

        ------------------------------------------
        -- MODIFIER OPERATIONS
        ------------------------------------------

        -- This operation adds a new product to the seller's inventory.
        -- If the product already exists in the inventory then it just increments
its quantity.
        -- Otherwise, it adds the new product to the inventory.
        public addNewProduct: Product ==> ()
        addNewProduct(newProduct) ==
        (
                dcl existingProducts : set of Product := rng products, productExists :
bool := false;

                for all p in set existingProducts do
                        if newProduct.isEqual(p)
                        then (
                        p.incrementQuantity();
                        productExists := true;
                    );

                if productExists = false then (
                    products := products munion { productsIDs |-> newProduct };
                    productsIDs := productsIDs + 1;
                )
        );

        -- removes a product from the seller inventory given the product ID in the
inventory
        public removeProduct: nat1 ==> ()
        removeProduct(ID) ==
        (
                products := {ID} <-: products;
        )
        pre ID in set dom products
        post ID not in set dom products;

        -- sells a product (decreases its quantity)
        -- if the desired quantity is bigger than the stock then it does not sell
the product.
        public sellProduct: nat1 * nat1 ==> bool
        sellProduct(productID, desiredQuantity) ==
        (
                dcl product : Product := products(productID);

                if product.getQuantity() = 1 and desiredQuantity = 1
                then (
                    products := products :-> {product};
                    return true;
                )
                elseif product.getQuantity() >= desiredQuantity
                then (
                    product.decrementQuantity(desiredQuantity);
                    return true;
                )
                else
                    return false;
        )
        pre productID in set dom products and desiredQuantity >= 1;

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here

end Seller
```

4.4 Class Product:

```
class Product
types
	public String = seq of char;
	public Float = real;

values
instance variables

	private ID : nat1;
	private name : String;
	private description : String;
	private price : Float;
	private quantity : int;
	private rating : Float := 0;
	private reviews : set of Review := {};

	private static numberOfProducts : int := 0;

	inv ID <= numberOfProducts and ID > 0;
	inv rating >= 0 and rating <= 5;
	inv price >= 0.00;

operations
	-----------------------------------------
	-- CONSTRUCTOR
	-----------------------------------------

	public Product: String * String * Float ==> Product
	Product(newName, newDescription, newPrice) ==
	(
		name := newName;
		description := newDescription;
		price := newPrice;
		quantity := 1;
		numberOfProducts := numberOfProducts + 1;
		ID := numberOfProducts;
	)
	pre newName <> "" and newDescription <> "" and newPrice >= 0;

	-----------------------------------------
	-- ACCESSOR OPERATIONS
	-----------------------------------------

	public getID: () ==> nat1
	getID() ==
	(
		return ID;
	);

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

```
public getDescription: () ==> String
getDescription() ==
(
        return description;
);

public getPrice: () ==> Float
getPrice() ==
(
        return price;
);

public getQuantity: () ==> int
getQuantity() ==
(
        return quantity;
);

public getRating: () ==> Float
getRating() ==
(
        return rating;
);

public getReviews: () ==> set of Review
getReviews() ==
(
        return reviews;
);

public static getNumberOfProducts: () ==> int
getNumberOfProducts() ==
(
        return numberOfProducts;
);

------------------------------------------
-- MODIFIER OPERATIONS
------------------------------------------

-- adds a review to the products
public addReview: Review ==> ()
addReview(userReview) ==
(
        reviews := reviews union {userReview};
        updateRating();
);

-- changes product price
public changePrice: Float ==> ()
changePrice(newPrice) ==
(
        price := newPrice;
)
pre price <> newPrice;

-- increments the user quantity
```

```
        public incrementQuantity: () ==> ()
        incrementQuantity() ==
        (
                quantity := quantity + 1;
        )
        post quantity > 0;

        -- decreases product quantity
        public decrementQuantity: nat1 ==> ()
        decrementQuantity(subtract) ==
        (
                quantity := quantity - subtract;
        )
        pre quantity > 0 and subtract <= quantity
        post quantity >= 0;

        -- This operation updates the average rating of the product
        private updateRating: () ==> ()
        updateRating() ==
        (
                dcl allRatingsSum : nat := 0, numRatings : nat := 0;
                for all r in set reviews do
                (
                        allRatingsSum := allRatingsSum + r.getClassification();
                        numRatings := numRatings + 1;
                );
                rating := calculateRating(numRatings, allRatingsSum);
        );

        ------------------------------------------
        -- COMPARISION OPERATIONS
        ------------------------------------------
        -- checks if two products are the same
        public isEqual: Product ==> bool
        isEqual(newProduct) ==
        (
                return name = newProduct.getName()
                        and description = newProduct.getDescription()
                        and price = newProduct.getPrice();
        );

functions

        --this function calculates the average product rating
        private calculateRating: nat * nat -> Float
        calculateRating(numReviews, reviewsRatings) ==
        (
                if numReviews = 0 then 0
                else
                        reviewsRatings / numReviews
        );

traces
-- TODO Define Combinatorial Test Traces here

end Product
```

4.5 *Class Review:*

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

values
instance variables
	private text : String;
	private classification : nat1;
	private author : User;

	inv classification > 0 and classification <= 5;

operations
	------------------------------------------
	-- CONSTRUCTOR
	------------------------------------------
	public Review: String * nat1 * User ==> Review
	Review(reviewText, reviewClassification, reviewUser) ==
	(
		text := reviewText;
		classification := reviewClassification;
		author := reviewUser;
	)
	pre reviewClassification > 0 and reviewClassification <= 5;

	------------------------------------------
	-- ACCESSOR OPERATIONS
	------------------------------------------
	-- gets the review text
	pure public getText: () ==> String
	getText() ==
	(
		return text;
	);

	-- gets the rating of the review
	pure public getClassification: () ==> nat1
	getClassification() ==
	(
		return classification;
	);

	-- gets the user who wrote the review
	pure public getAuthor: () ==> User
	getAuthor() ==
	(
		return author;
	);

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Review
```

4.6 Class Shop:

```
class Shop

types
	public String = seq of char;
	public Users = set of User;
	public Products = set of Product;

values

instance variables
	private name : String;
	private registeredUsers : Users := {};
	private registeredProducts : map User`String to Products := {|->};

	--variable to keep track of the current logged user
	private loggedUser : User;

operations
	------------------------------------------
	-- CONSTRUCTOR
	------------------------------------------

	public Shop: String ==> Shop
	Shop(shopName) ==
	(
		name := shopName;
	);

	------------------------------------------
	-- ACCESSOR OPERATIONS
	------------------------------------------

	public getLoggedUser: () ==> User
	getLoggedUser() ==
	(
		return loggedUser;
	);

	public getRegisteredUsers: () ==> Users
	getRegisteredUsers() ==
	(
		return registeredUsers;
	);

	public getRegisteredProducts: () ==> map User`String to Products
	getRegisteredProducts() ==
	(
		return registeredProducts;
	);

	public getNumberOfProducts: () ==> int
	getNumberOfProducts() ==
	(
		return Product`getNumberOfProducts();
	);

	public getNumberOfSellers: () ==> int
	getNumberOfSellers() ==
	(
		return getNumberOfElements(<SELLER>);
	);

	public getNumberOfCostumers: () ==> int
	getNumberOfCostumers() ==
	(
		return getNumberOfElements(<COSTUMER>);
	);

	-- Gets the number of users of a certain type (SELLER or COSTUMER)
	private getNumberOfElements: User`UserType ==> int
	getNumberOfElements(elemsType) ==
	(
		dcl num : int := 0;

		for all user in set registeredUsers do
			if user.getType() = elemsType
			then num := num+1;

		return num;
	);

	------------------------------------------
	-- MODIFIER OPERATIONS
	------------------------------------------

	public addNewUser: User ==> ()
	addNewUser(newUser) ==
	(
		registeredUsers := registeredUsers union {newUser};
	)
	pre newUser not in set registeredUsers
	post newUser in set registeredUsers;


	-- Updates all the products that exist in the shop
	-- It keeps all products listed with the respective association to the
product's seller by it's username.
	public updateProducts: () ==> ()
	updateProducts() ==
	(
		for all user in set registeredUsers do
			if user.getType() = <SELLER>
			then (
					dcl seller : Seller := user, existsUserProducts :
map User`String to Products := {|->};
					existsUserProducts := {user.getUsername()} <:
registeredProducts;

					if existsUserProducts = {|->}
					then (
						registeredProducts := registeredProducts
munion { seller.getUsername() |-> seller.getAllProducts() };
```

```
                                    )
                                    else (
                                          registeredProducts := registeredProducts ++
{ seller.getUsername() |-> seller.getAllProducts() };
                                    )
                        );
        );

        ------------------------------------------
        -- AUTHORIZATION OPERATIONS
        ------------------------------------------
        -- if the entered credentials are correct then the user "logs in"
        public login: String * String ==> bool
        login(inputUsername, inputPassword) ==
        (
                for all user in set registeredUsers do
                        if user.getUsername() = inputUsername
                        then (
                                if user.checkCredentials(inputPassword)
                                then (
                                        loggedUser := user;
                                        return true
                                )
                                else return false
                        );
                return false;
        );

        -- logs current user out
        public logout: () ==> ()
        logout() ==
        (
                loggedUser.logOut();
        );
        -- registers a new user in the shop
        public register: String * String * User`UserType ==> ()
        register(newUsername, newPassword, type) ==
        (
                if type = <SELLER>
                then (
                        dcl newSeller : Seller := new Seller(newUsername, newPassword);
                        self.addNewUser(newSeller);
                        loggedUser := newSeller;
                        loggedUser.logIn();
                )
                else (
                        dcl newCostumer : Costumer := new Costumer(newUsername,
newPassword);
                        self.addNewUser(newCostumer);
                        loggedUser := newCostumer;
                        loggedUser.logIn();
                )
        )
        pre type = <SELLER> or type = <COSTUMER>;

functions
traces
end Shop
```

5. Model Validation

5.1 Class MyTest:

```
class MyTest
types
values
instance variables
operations

   -- Assert if something is true
   protected assert: bool ==> ()
   assert(arg) == return
   pre arg;

   protected 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

functions
traces
end MyTest
```

5.2 Class TestsRun:

```
class TestsRun
-- This class is just a simple test runner.
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
      public static main: () ==> ()
      main() ==
      (
            UserTest`main();
            ProductTest`main();
            ShopTest`main();
      );
functions
traces
-- TODO Define Combinatorial Test Traces here
end TestsRun
```

5.3 Class UserTest:

```
class UserTest is subclass of MyTest

types
values
instance variables
	-- This variable keeps track of the number of users that were created during 
the tests.
	-- This way to check if numberOfUsers is correct it is not necessary to have 
harcoded values.
	private numUsers : int := 0;

operations

	-------------------------------------------------------------
	--
	USER
	-------------------------------------------------------------
	-------------------------------------------------------------
	-- Tests:
	--		: the creation of a user of type Seller.
	--		: the creation of a user of type Costumer.
	-------------------------------------------------------------
	private createNewUser: () ==> ()
	createNewUser() ==
	(
		dcl seller : Seller, costumer : Costumer;

		seller := new Seller("TestSeller", "password1");
		numUsers := numUsers+1;
		seller.logIn();
		seller.setNewPassword("newPassword1");
		seller.logOut();
		costumer := new Costumer("TestCostumer", "password1");
		numUsers := numUsers+1;

		assert(seller.getUsername() = "TestSeller");
		assert(costumer.getUsername() = "TestCostumer");
	);

	-------------------------------------------------------------
	-- Tests:
	--		: the correct update of the total number of users.
	--		: the correct attribuition of the user ID.
	-------------------------------------------------------------
	private correctNumberOfUsers: () ==> ()
	correctNumberOfUsers() ==
	(
		dcl seller : Seller, costumer : Costumer, costumer2 : Costumer, 
seller2 : Seller, seller3 : Seller;
		assert(User`getNumberOfUsers() = numUsers);

		seller := new Seller("TestSeller", "password1");
		numUsers := numUsers+1;
		costumer := new Costumer("TestCostumer", "password1");
		numUsers := numUsers+1;
```

```
        seller2 := new Seller("TestSeller2", "password1");
        numUsers := numUsers+1;
        seller3 := new Seller("TestSeller3", "password1");
        numUsers := numUsers+1;
        costumer2 := new Costumer("TestCostumer2", "password1");
        numUsers := numUsers+1;

        assert(User`getNumberOfUsers() = numUsers);
        assert(costumer2.getUserID() = numUsers);
);

-------------------------------------------------------------
-- Tests:
--             : the substitution of a user's username.
-------------------------------------------------------------
private changeUsername: () ==> ()
changeUsername() ==
(
        dcl seller : Seller := new Seller("TestSeller", "password1");
        numUsers := numUsers+1;

        if seller.checkCredentials("password1")
        then (
                assert(seller.getUsername() = "TestSeller");
                seller.setNewUsername("NewTestSellerUsername");
                assert(seller.getUsername() = "NewTestSellerUsername");
        )
);

-------------------------------------------------------------
-- Tests:
--             : the type of the user.
-------------------------------------------------------------
private correctType: () ==> ()
correctType() ==
(
        dcl seller : Seller, costumer : Costumer;
        seller := new Seller("TestSeller", "password1");
        numUsers := numUsers+1;
        costumer := new Costumer("TestCostumer", "password1");
        numUsers := numUsers+1;

        assert(seller.getType() = <SELLER>);
        assert(costumer.getType() = <COSTUMER>);
);

        -------------------------------------------------------------
-- Tests:
--             : the successful addition of a review to an user
--             : the correct rating of a user.
-------------------------------------------------------------
private getRating: () ==> ()
getRating() ==
(
        dcl seller : Seller, costumer1: Costumer,
                costumer2: Costumer, review1 : Review, review2 : Review;
        seller := new Seller("TestSeller", "password1");
        numUsers := numUsers+1;
        costumer1 := new Costumer("TestCostumer1", "password1");
        numUsers := numUsers+1;
        costumer2 := new Costumer("TestCostumer2", "password1");
        numUsers := numUsers+1;
        assertEqual(seller.getReviews(), {});
        review1 := new Review("Great seller", 5, costumer1);
        review2 := new Review("Mediocre seller", 3, costumer2);
        seller.addReview(review1); seller.addReview(review2);
        assertEqual(review1.getText(), "Great seller");
        assertEqual(seller.getRating(), 4);
);

-------------------------------------------------------------
--
SELLER
-------------------------------------------------------------
-------------------------------------------------------------
-- Tests:
--           : the successful addition of a new product to a
--      seller's inventory.
--           : the quantity change of a product.
--           : the selling of a product.
--           : the removal of a product.
-------------------------------------------------------------
private sellerTestNewProductOperations: () ==> ()
sellerTestNewProductOperations() ==
(
        dcl seller : Seller, prod : Product;
        seller := new Seller("TestSeller", "password1");
        numUsers := numUsers + 1;
        prod := new Product("P1", "Great!", 15.99);
        ProductTest`numProds := ProductTest`numProds + 1;

        seller.addNewProduct(prod);
        assertEqual(seller.getProduct(1), prod);
        seller.getProduct(1).incrementQuantity();
        assert(seller.sellProduct(1,1) = true);
        assertEqual(seller.getProduct(1).getQuantity(), 1);
        seller.removeProduct(1);
        assert(seller.getAllProducts() = {});
);

-------------------------------------------------------------
-- Tests:
--           : if the addition of an already exisiting product
--      increments its quantity instead of adding an equal
--      product.
--           : if the selling of an existing product with only one
--      instance in stock removes the product from the
--      seller's inventory.
-------------------------------------------------------------
private sellerTestExistingProductOperations: () ==> ()
sellerTestExistingProductOperations() ==
(
        dcl seller : Seller, prod : Product;
        seller := new Seller("TestSeller", "password1");
        numUsers := numUsers + 1;
        prod := new Product("P1", "Great!", 15.99);
        ProductTest`numProds := ProductTest`numProds + 1;

        seller.addNewProduct(prod);
        seller.addNewProduct(prod);
        assertEqual(seller.getProduct(1).getQuantity(), 2);
        assert(seller.sellProduct(1,1) = true);
        assertEqual(seller.sellProduct(1,1), true);
        assert(seller.getAllProducts() = {});
);

--------------------------------------------------------------
-- Tests:
--          : the logging in and out of an user.
--------------------------------------------------------------
private checkLogging: () ==> ()
checkLogging() ==
(
        dcl seller : Seller;
        seller := new Seller("TestSeller", "password1");
        numUsers := numUsers + 1;

        assert(seller.checkCredentials("password1") = true);
        assert(seller.isLoggedIn() = true);
        seller.logOut();
        assert(seller.isLoggedIn() = false);
);

-- Entry Point
public static main : () ==> ()
main() ==
(
        dcl test : UserTest := new UserTest();

        test.createNewUser();
        test.correctNumberOfUsers();
        test.changeUsername();
        test.correctType();
        test.checkLogging();
        test.getRating();
        --Seller
        test.sellerTestNewProductOperations();
        test.sellerTestExistingProductOperations();
);

-----------------------------------------------------------------
-- | ENTRY POINTS FOR INVALID INPUTS (TO RUN ONE AT A TIME) | --
-----------------------------------------------------------------

        --------------------------------------------------------------
-- Error:
--          : violates pre condition where a user has to be given
--      a username and a password upon its creation.
--      P.S.: one of the statements below can be uncommented to
--      get the desired result.
--------------------------------------------------------------
public static constructUserNoUsermame: () ==> ()
constructUserNoUsermame() ==
(
		dcl seller : Seller := new Seller("", "password1");
		return;
		--dcl seller : Seller := new Seller("TestUsername", "");
		--dcl seller : Seller := new Seller("TestUsername", "");
	);

	-------------------------------------------------------------
	-- Error:
	--		: violates pre condition where a user has to be logged
	--	in in order to change its password/username
	-------------------------------------------------------------
	public static unloggedPasswordChange: () ==> ()
	unloggedPasswordChange() ==
	(
		dcl seller : Seller := new Seller("TestSeller", "password1");
		seller.setNewPassword("newPassword");
	);

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end UserTest
```

5.4 Class ProductTest:

```
class ProductTest is subclass of MyTest
types
values
instance variables
	public static numProds : int := 0;

operations
	-------------------------------------------------------------
	-- Tests:
	--		: the creation of a new product.
	--		: the retrieval of the product ID.
	-------------------------------------------------------------
	private createNewProduct: () ==> ()
	createNewProduct() ==
	(
		dcl prod : Product;
		prod := new Product("p1", "description1", 12.99);
		numProds := numProds+1;

		assert(prod.getID() = numProds);
	);

		-------------------------------------------------------------
		-- Tests:
		--		: the retrieval of the product ID.
		--		: the retrieval of the product name.
		--		: the retrieval of the product quantity.
		--		: the retrieval of the product description.
```

```
        --            : the retrieval of the product price.
        -------------------------------------------------------------
private checkInformation: () ==> ()
checkInformation() ==
(
        dcl prod : Product := new Product("p1", "description1", 12.99);
        numProds := numProds+1;

        assert(prod.getID() = numProds);
        assert(prod.getName() = "p1");
        assert(prod.getQuantity() = 1);
        assert(prod.getDescription() = "description1");
        assert(prod.getPrice() = 12.99);
);

-------------------------------------------------------------
-- Tests:
--            : the successful update of the product rating.
--      * the product rating is only updated when a new review
--      * is added to the product.
-------------------------------------------------------------
private checkRating: () ==> ()
checkRating() ==
(
        dcl prod : Product, review: Review, review2: Review,
                costumer : Costumer, costumer2 : Costumer;
        prod := new Product("p1", "description1", 12.99);
        numProds := numProds+1;

        costumer := new Costumer("TestCostumer", "password1");
        costumer2 := new Costumer("TestCostumer2", "password2");
        review := new Review("Good product", 4, costumer);
        review2 := new Review("Disappointed with this product", 2, costumer);

        assert(prod.getRating() = 0);
        prod.addReview(review);
        assert(prod.getRating() = 4.0);
        prod.addReview(review2);
        assert(prod.getRating() = 3.0);
);

-------------------------------------------------------------
-- Tests:
--            : the changing of the price of a product
-------------------------------------------------------------
private changePrice: () ==> ()
changePrice() ==
(
        dcl prod : Product := new Product("p1", "description1", 12.99);
        numProds := numProds+1;

        assert(prod.getPrice() = 12.99);
        prod.changePrice(11.99);
        assert(prod.getPrice() = 11.99);
);

-------------------------------------------------------------
-- Tests:
```

```
--            : the changing of the quantity of a product
--            : the quantity increment and decrement
-------------------------------------------------------------
private changeQuantity: () ==> ()
changeQuantity() ==
(
       dcl prod : Product := new Product("p1", "description1", 12.99);
       numProds := numProds+1;

       assert(prod.getQuantity() = 1);
       prod.incrementQuantity();
       assert(prod.getQuantity() = 2);
       prod.decrementQuantity(1);
       assert(prod.getQuantity() = 1);
);

-------------------------------------------------------------
-- Tests:
--            : if the total number of products is well updated
-------------------------------------------------------------
private numberOfProducts: () ==> ()
numberOfProducts() ==
(
       assert(numProds = Product`getNumberOfProducts());
);

-------------------------------------------------------------
-- Tests:
--            : if two products are the same
-------------------------------------------------------------
private areEqual: () ==> ()
areEqual() ==
(
       dcl prod : Product, prod2 : Product;
       prod := new Product("p1", "description1", 12.99);
       numProds := numProds + 1;
       prod2 := new Product("p1", "description1", 12.99);
       numProds := numProds + 1;

       assert(prod.isEqual(prod2));
);

-------------------------------------------------------------
-- Tests:
--            : if the reviews are being well returned
-------------------------------------------------------------
private getReviews: () ==> ()
getReviews() ==
(
       dcl prod : Product;
       prod := new Product("p1", "description1", 12.99);
       numProds := numProds + 1;

       assert(prod.getReviews() = {});
);

--ENTRY POINT
public static main: () ==> ()
```

```
        main() ==
        (
                dcl test : ProductTest := new ProductTest();

                test.createNewProduct();
                test.checkInformation();
                test.checkRating();
                test.changePrice();
                test.changeQuantity();
                test.areEqual();
                test.numberOfProducts();
                test.getReviews();
        );

        ----------------------------------------------------------------
        -- | ENTRY POINTS FOR INVALID INPUTS (TO RUN ONE AT A TIME) | --
        ----------------------------------------------------------------

        -------------------------------------------------------------
        -- Error:
        --            : violates pre condition where a product price cannot
        --      be updated with an equal price.
        -------------------------------------------------------------
        public static samePriceUpdate: () ==> ()
        samePriceUpdate() ==
        (
                dcl prod : Product := new Product("p1", "description1", 12.99);
                prod.changePrice(12.99);
        );

        -------------------------------------------------------------
        -- Error:
        --            : violates pre condition where the desired quantity to
        --      subtract is bigger than the existing quantity
        -------------------------------------------------------------
        public static decrementQuantity: () ==> ()
        decrementQuantity() ==
        (
                dcl prod : Product := new Product("p1", "description1", 12.99);
                prod.decrementQuantity(4);
        );

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end ProductTest
```

5.5 Class ShopTest:

```
class ShopTest is subclass of MyTest
types
values
instance variables

       -- Shop instance
       private shop : Shop;

operations

       -------------------------------------------------------------
       -- Tests:
       --            : the successful creation of a shop.
       -------------------------------------------------------------
       private createShop: () ==> ()
       createShop() ==
       (
              shop := new Shop("Loja Testes");
              assertEqual(shop.getRegisteredUsers(),{});
              assertEqual(shop.getRegisteredProducts(),{|->});
       );

       -------------------------------------------------------------
       -- Tests:
       --            : the number of users of type Seller that are
       --     registered in the shop.
       -------------------------------------------------------------
       private getNumberOfSellers: () ==> ()
       getNumberOfSellers() ==
       (
              shop.register("Seller1", "pswd1", <SELLER>);
              assert(shop.login("Seller1", "pswd1") = true);
              shop.logout();
              shop.register("Seller2","pswd2", <SELLER>);
              shop.logout();
              shop.register("Costumer1","pswd3", <COSTUMER>);
              shop.logout();
              assert(shop.getNumberOfSellers() = 2);
       );

       -------------------------------------------------------------
       -- Tests:
       --            : the number of users of type Seller that are
       --     registered in the shop.
       -------------------------------------------------------------
       private getNumberOfCostumers: () ==> ()
       getNumberOfCostumers() ==
       (
              assert(shop.getNumberOfCostumers() = 1);
       );

       -------------------------------------------------------------
       -- Tests:
       --            : the successful logging in and out of a user in the
```

```
        --     shop.
        --------------------------------------------------------------
        private userLogging: () ==> ()
        userLogging() ==
        (
                assertEqual(shop.login("Seller1", "pswd1"), true);
                assert(shop.getLoggedUser().getUsername() = "Seller1");
                shop.logout();
                assertEqual(shop.login("Seller2", "pswd2"), true);
                assert(shop.getLoggedUser().getUsername() = "Seller2");
        );

        --------------------------------------------------------------
        -- Tests:
        --            : the correct number of products registererd in the
        --     shop
        --------------------------------------------------------------
        private getNumberOfProducts: () ==> ()
        getNumberOfProducts() ==
        (
                dcl prod : Product := new Product("P1", "Great product! Brand new",
15.99), seller : Seller;
                assertEqual(shop.getNumberOfProducts(),
Product`getNumberOfProducts());
                assertEqual(shop.login("Seller1", "pswd1"), true);
                seller := shop.getLoggedUser();
                seller.addNewProduct(prod);
                shop.updateProducts();
                seller.addNewProduct(prod);
                assertEqual(shop.getNumberOfProducts(),
Product`getNumberOfProducts());
        );

        -- Entry point
        public static main: () ==> ()
        main() ==
        (
                dcl test : ShopTest := new ShopTest();

                test.createShop();
                test.getNumberOfSellers();
                test.getNumberOfCostumers();
                test.userLogging();
                test.getNumberOfProducts();
        );

        -----------------------------------------------------------------
        -- | ENTRY POINTS FOR INVALID INPUTS (TO RUN ONE AT A TIME) | --
        -----------------------------------------------------------------

        --------------------------------------------------------------
        -- Error:
        --            : violates pre condition where already exists the user
        --     in the dom of the registeredUsers set.
        --------------------------------------------------------------
        public static addExistingUser: () ==> ()
        addExistingUser() ==
        (
                dcl shopError : Shop := new Shop("Loja Testes Erro"), seller : Seller;
                seller := new Seller("s1","s1");
                shopError.addNewUser(seller);
                shopError.addNewUser(seller);
        );

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end ShopTest
```

6. Model Verification

6.1 Example of Domain Verification:

No.	PO Name	Type
20	Seller`getProduct(nat1)	legal map application

Above is the proof obligation generated by Overture related to the domain verification. Below is the code responsible:

```
public getProduct: nat1 ==> Product
getProduct(id) ==
(
        return products(id);
)
pre id in set dom products;
```

This example is clear as the pre-condition assures that the map is accessed only inside its domain: `pre id in set dom products`.

6.2 Example of Invariant Verification:

No.	PO Name	Type
15	Product`updateRating()	state invariant holds

Above is the proof obligation generated by Overture related to the invariant verification. Below is the code responsible:

```
private updateRating: () ==> ()
updateRating() ==
(
        dcl allRatingsSum : nat := 0, numRatings : nat := 0;
        for all r in set reviews do
        (
                allRatingsSum := allRatingsSum + r.getClassification();
                numRatings := numRatings + 1;
        );
        rating := calculateRating(numRatings, allRatingsSum);
);
```

In this example the invariant in analysis is the following:

```
inv rating >= 0 and rating <= 5;
```

This way, when the rating variable is updated it checks for the invariant acceptance.

7. Code Generation

I decided that I wanted to create a GUI (Graphical User Interface) to prove and test my model architecture.

I have generated the .java files through the Overture's "Code Generation" tool. Afterwards I imported those .java generated files to a new eclipse project where I started to implement my program GUI.

Below are some snapshots of the GUI I have created for the model.

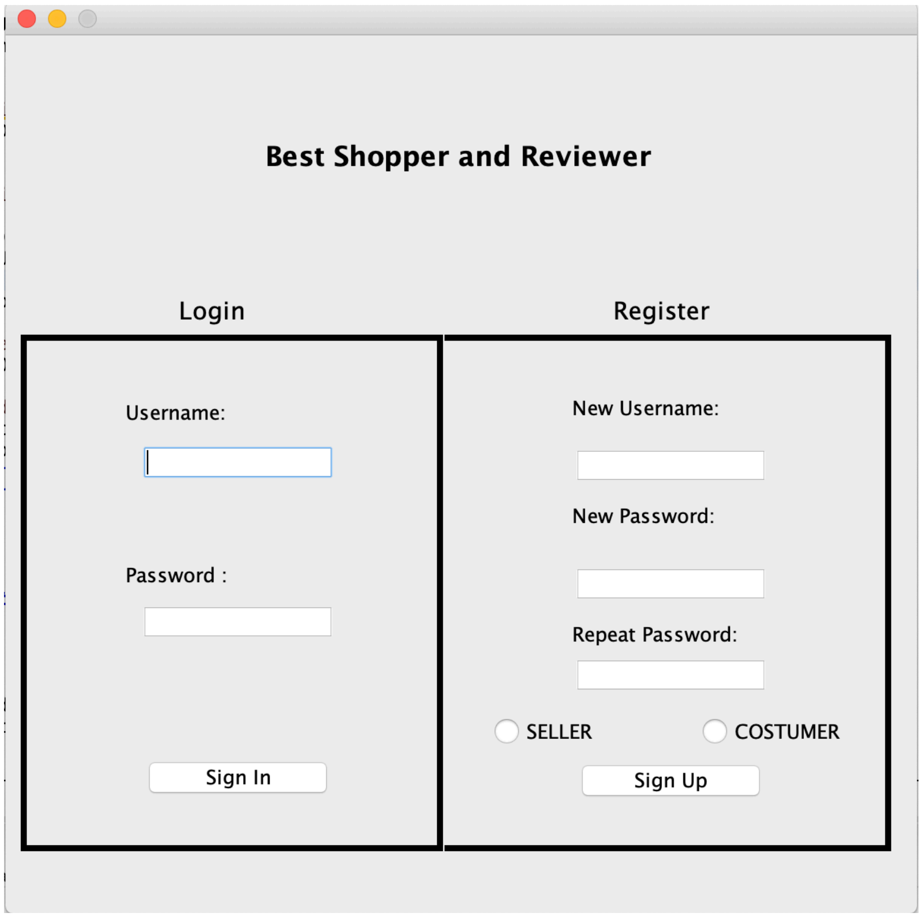
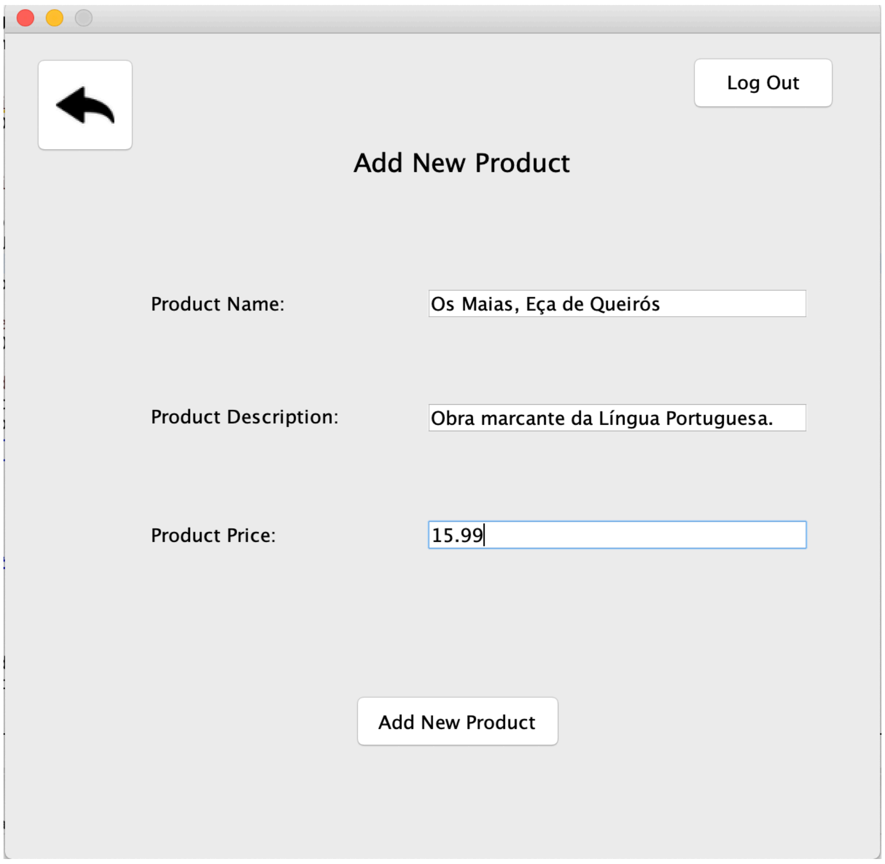


Figure 1 - Initial screen (Log in / Register)


Log Out

Add New Product

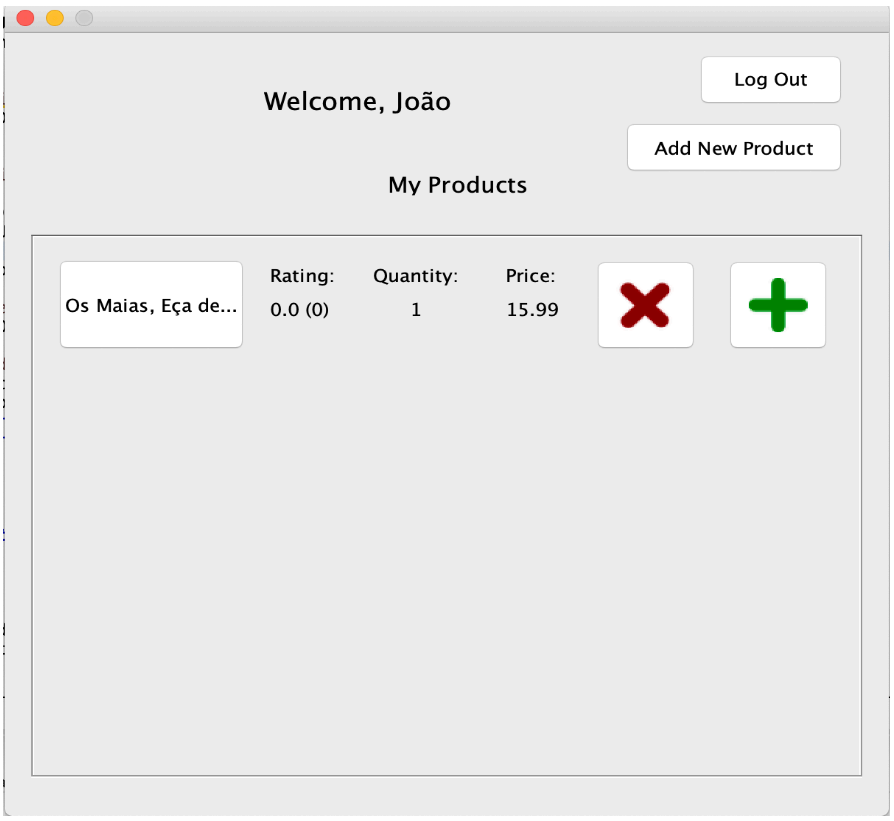
Product Name: Os Maias, Eça de Queirós

Product Description: Obra marcante da Língua Portuguesa.

Product Price: 15.99

Add New Product

Figure 2 - Add new product to seller's inventory screen


Welcome, João

Log Out

Add New Product

My Products

	Rating:	Quantity:	Price:
Os Maias, Eça de...	0.0 (0)	1	15.99

Figure 3 - Seller's home page

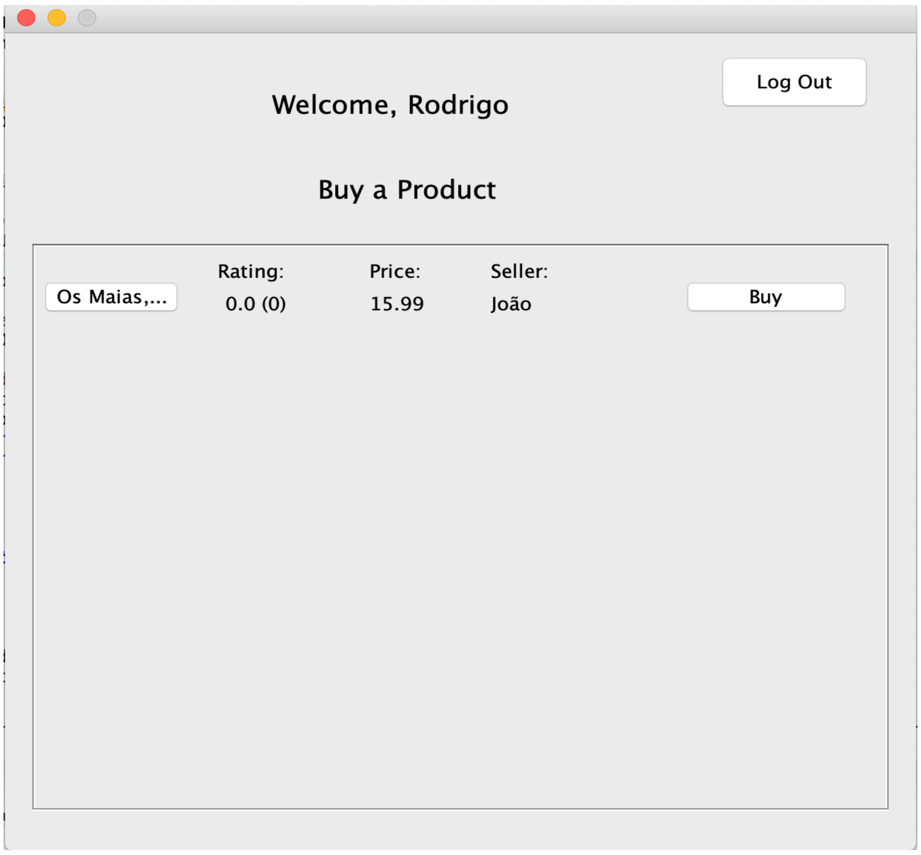


Figure 4 - Costumer home page

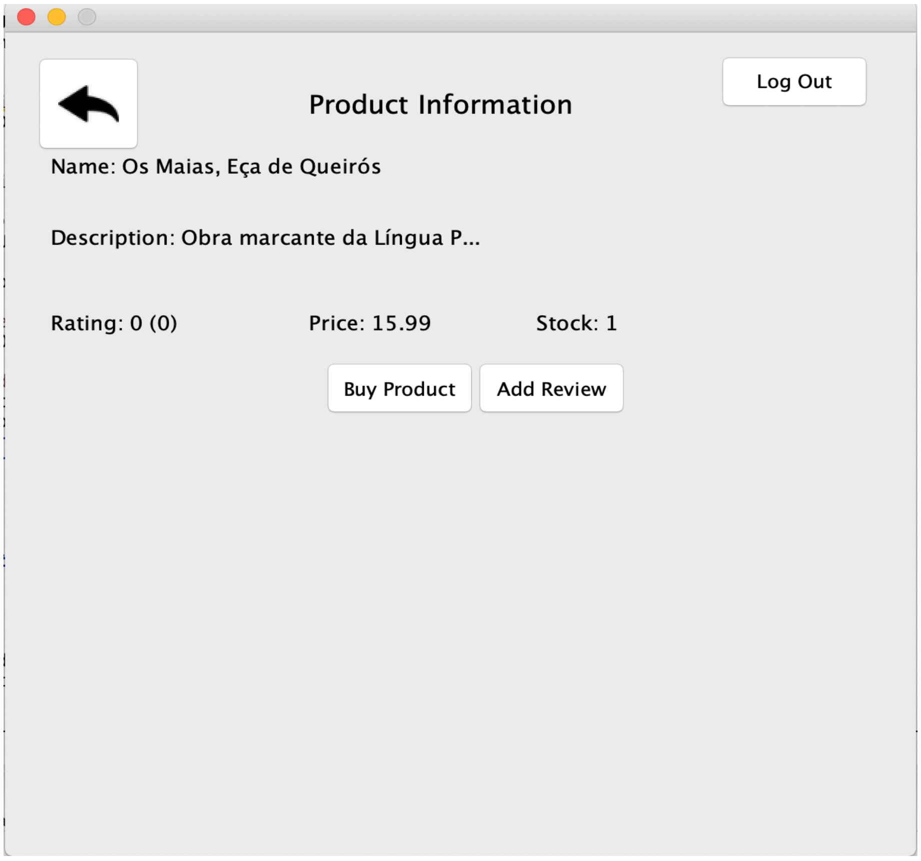
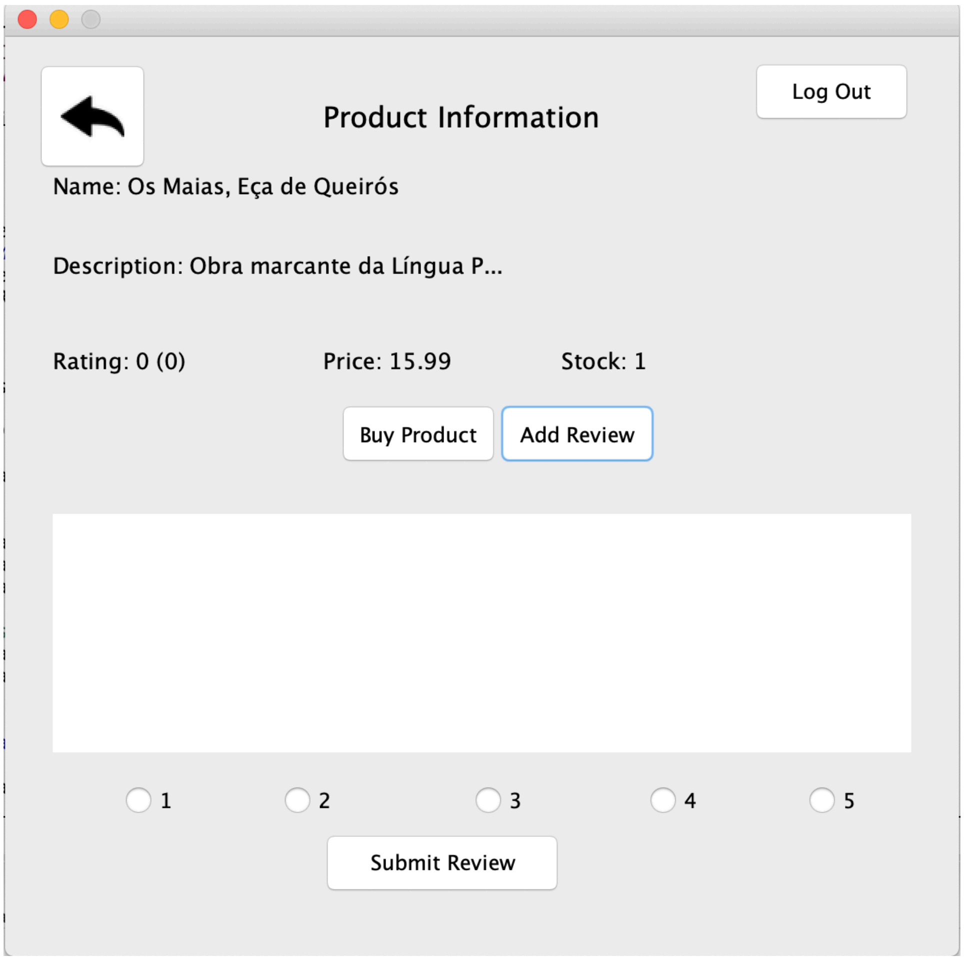


Figure 5 - Detailed product information screen



Name: Os Maias, Eça de Queirós

Description: Obra marcante da Língua P...

Rating: 0 (0) Price: 15.99 Stock: 1

Figure 6 - Add review screen

8. Conclusion

I was able to achieve my goals in the making of this project. I have created a fully operational model of an online shop with the required operations and implemented the majority of the types Overture has (*set, seq, map*, etc).

If I had more time, I could have improved the user interface to be more detailed and cover more of the model.

This project was of great benefit to me as it made me realize how important are formal methods in the development, testing and in the design of a software project.

Overall, I believe I have covered all the requirements of this project and went beyond by creating an extra functionality: the shop. I still created a user-friendly interface to the model proving that the model work and could be implemented.

As a one-man group I had some struggle with having everything done until the deadline, but I managed to accomplish it and to successfully finish the project.

9. References

- http://overturetool.org/documentation/manuals.html
- https://www.trialpanel.com/en/shopadvizor/
- https://creately.com/diagram/example/htjdqsn2/Amazon%20UML
- https://creately.com/diagram/example/i7f2vsa1/Ebay