Formal Modeling of ShopAdvizor in VDM++

Report

Integrated Master in Informatics and Computing Engineering

Formal Methods in Software Engineering

Class 3 - Group 3:

Daniel Filipe Santos Marques, up201503822@fe.up.pt

Maria Eduarda Santos Cunha, up201506524@fe.up.pt

Porto, 7th January 2019

Table of Contents

Formal Methods in Software Engineering....1
1 Informal System Description and List of Requirements....3
1.1 Informal System Description....3
1.2 List of Requirements....3
2 Visual UML Model....4
2.1 Use Case Model....4
2.2 Class Model....10
3 Formal VDM++ Model....11
3.1 Class Activity....11
3.2 Class Brand....12
3.3 Class Competition....13
3.4 Class Product....13
3.5 Class Retailer....15
3.6 Class ShopAdvizor....16
3.7 Class User....23
4 Model Validation....23
4.1 Class ShopAdvizorTest....24
5 Model Verification....30
5.1 Example of Domain Verification....30
5.2 Example of Invariant Verification....31
6 Code Generation....31
7 Conclusions....32
8 References....32

1 Informal System Description and List of Requirements

1.1 Informal System Description

Our system consists of an app where users can review products and consult reviews done by others. Furthermore, users may also participate in activities started by a brand in order to attempt to win a monetary prize set out for each particular activity.

In addition to this, users may also see listings of different system information, such as existing products, brands, retailers and competitions.

1.2 List of Requirements

The following table states all the system's requirements, which can directly translate into use cases.

Note that a user may be of type Normal, Retailer, Brand or Admin. There is also the concept of a guest user, which would be an unauthenticated visitor.

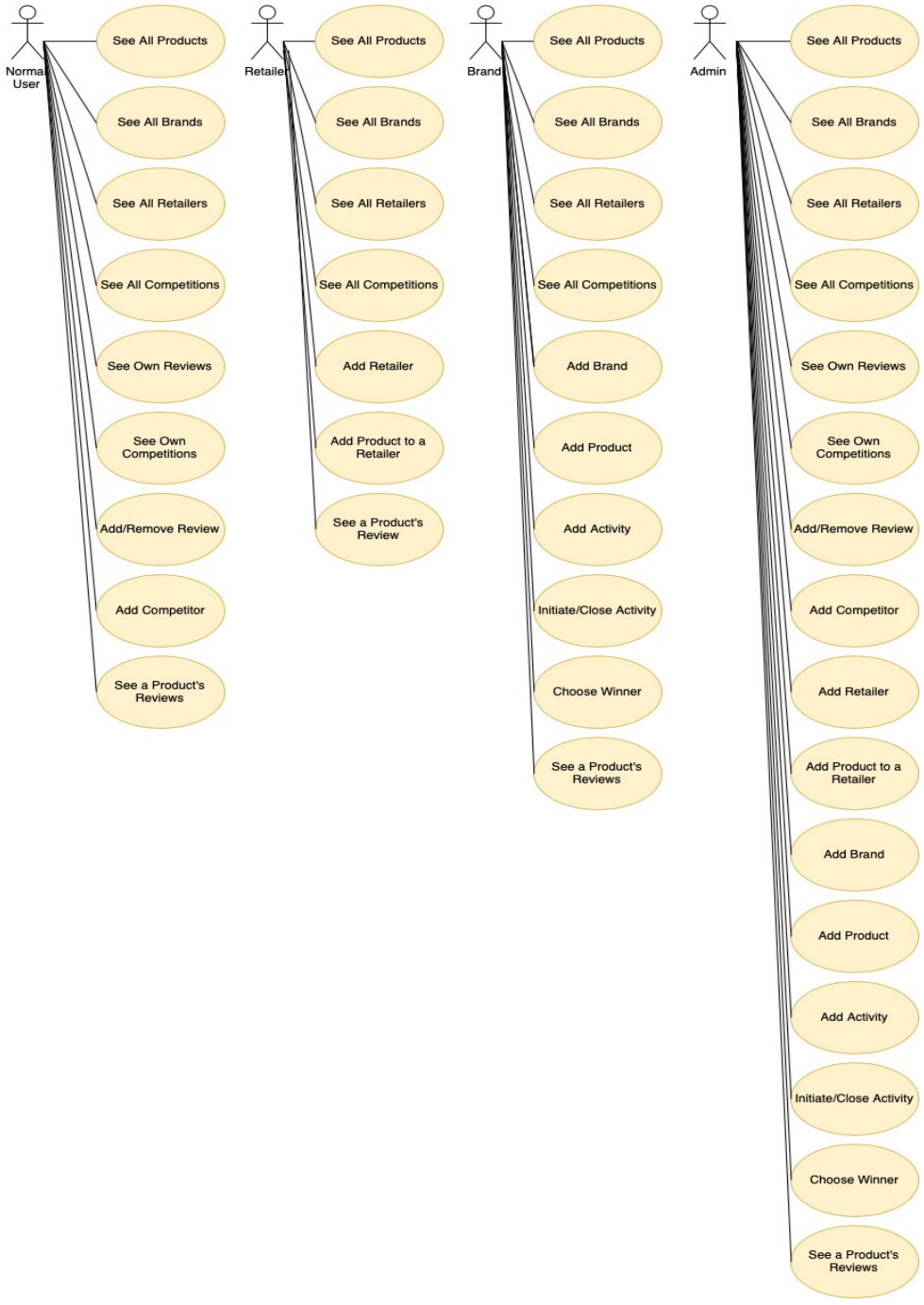
Requirements R1 to R5 are common to all users. Admin users have permission to do everything the other users can do.

Id	Priority	Description
R1	Mandatory	A user may see all products in the system.
R2	Optional	A user may see all brands in the system.
R3	Optional	A user may see all retailers in the system.
R4	Mandatory	A user may see all competitions in the system.
R5	Mandatory	A user may see a product's reviews.
R6	Mandatory	A normal user may see the reviews they have made.
R7	Mandatory	A normal user may see the competitions they are in.
R8	Mandatory	A normal user may add a review to a product.
R9	Optional	A normal user may remove a review they have made.
R10	Mandatory	A normal user may join a competition.
R11	Mandatory	A retailer user may add themselves as a retailer.
R12	Mandatory	A retailer user may add a product to their corresponding retailer.
R13	Mandatory	A brand user may add themselves as a brand.
R14	Mandatory	A brand user may add a product under their brand.
R15	Mandatory	A brand user may add an activity.
R16	Mandatory	A brand user may initiate an activity.
R17	Mandatory	A brand user may close an activity.
R18	Mandatory	A brand user may choose a winner in a competition.

2 Visual UML Model

2.1 Use Case Model

The original image can be found in */UML/useCases.png*.



Scenario	See all products
Description	Normal scenario for a user who wants a listing of all products.
Pre-conditions	(unspecified)
Post-conditions	(unspecified)
Steps	1. Get all products.
Exception	(unspecified)

Scenario	See all brands
Description	Normal scenario for a user who wants a listing of all brands.
Pre-conditions	(unspecified)
Post-conditions	(unspecified)
Steps	1. Get all brands.
Exception	(unspecified)

Scenario	See all retailers
Description	Normal scenario for a user who wants a listing of all retailers.
Pre-conditions	(unspecified)
Post-conditions	(unspecified)
Steps	1. Get all retailers.
Exception	(unspecified)

Scenario	See all competitions
Description	Normal scenario for a user who wants a listing of all competitions.
Pre-conditions	(unspecified)
Post-conditions	(unspecified)
Steps	1. Get all competitions.
Exception	(unspecified)

Scenario	See a product's reviews
Description	Normal scenario for a user who wants a listing of all reviews of a product.
Pre-conditions	(unspecified)
Post-conditions	(unspecified)
Steps	1. State product name and brand. 2. Get all reviews on that product and brand combination.
Exception	(unspecified)

Scenario	See own competitions
Description	Normal scenario for a user who wants a listing of the competitions they are in.
Pre-conditions	1. User must be normal.
Post-conditions	(unspecified)
Steps	1. Get all competitions the user is in.
Exception	(unspecified)

Scenario	See own reviews
Description	Normal scenario for a user who wants a listing of the reviews they have made.
Pre-conditions	1. User must be normal.
Post-conditions	(unspecified)
Steps	1. Get all reviews the user has made.
Exception	(unspecified)

Scenario	Add review
Description	Normal scenario for a user who wants to add a review to a product.
Pre-conditions	1. User must be normal. 2. Product (and brand combination) is in the system. 3. User has no review on that product.
Post-conditions	1. Review is under that product.
Steps	1. State product name, brand name, rating and feedback. 2. Insert review.
Exception	(unspecified)

Scenario	Remove review
Description	Normal scenario for a user who wants to remove a review they have made on a product.
Pre-conditions	1. User must be normal. 2. User must have a review on that product.
Post-conditions	1. Review is no longer under that product.
Steps	1. State product name and brand name. 2. Remove review.
Exception	(unspecified)

Scenario	Add competitor
Description	Normal scenario for a user who wants to join a competition.
Pre-conditions	1. User must be normal. 2. User is not in that competition. 3. Competition has not started.
Post-conditions	1. User is in competition.
Steps	1. State activity title. 2. Add user to competition of that activity.
Exception	(unspecified)

Scenario	Add retailer
Description	Normal scenario for a user who wants to add a retailer to the system.
Pre-conditions	1. User must be a retailer. 2. Retailer is not in the system.
Post-conditions	1. Retailer is in the system.
Steps	1. Create retailer with retailer name. 2. Add retailer.
Exception	(unspecified)

Scenario	Add product to a retailer
Description	Normal scenario for a user who wants to add a product to a certain retailer.
Pre-conditions	1. User must be a brand. 2. Product (and brand combination) is not registered under that retailer.
Post-conditions	1. Product is under that retailer.
Steps	1. State retailer name, product name, brand name, stock and price. 2. Insert product to retailer with retailer name.
Exception	(unspecified)

Scenario	Add brand
Description	Normal scenario for a user who wants to add a brand to the system.
Pre-conditions	1. User must be a brand. 2. Brand is not in the system.
Post-conditions	1. Brand is in the system.
Steps	1. Create brand with brand name. 2. Add brand.
Exception	(unspecified)

Scenario	Add product
Description	Normal scenario for a user who wants to add a product to the system.
Pre-conditions	1. User must be a brand. 2. Product (and brand combination) is not in the system.
Post-conditions	1. Product is in the system.
Steps	1. Create product with product name, brand name and description. 2. Insert product.
Exception	(unspecified)

Scenario	Add activity
Description	Normal scenario for a user who wants to add an activity to the system.
Pre-conditions	1. User must be a brand. 2. Activity is not in the system.
Post-conditions	1. Activity is in the system.
Steps	1. Create activity with title, description, prize and brand. 2. Insert activity.
Exception	(unspecified)

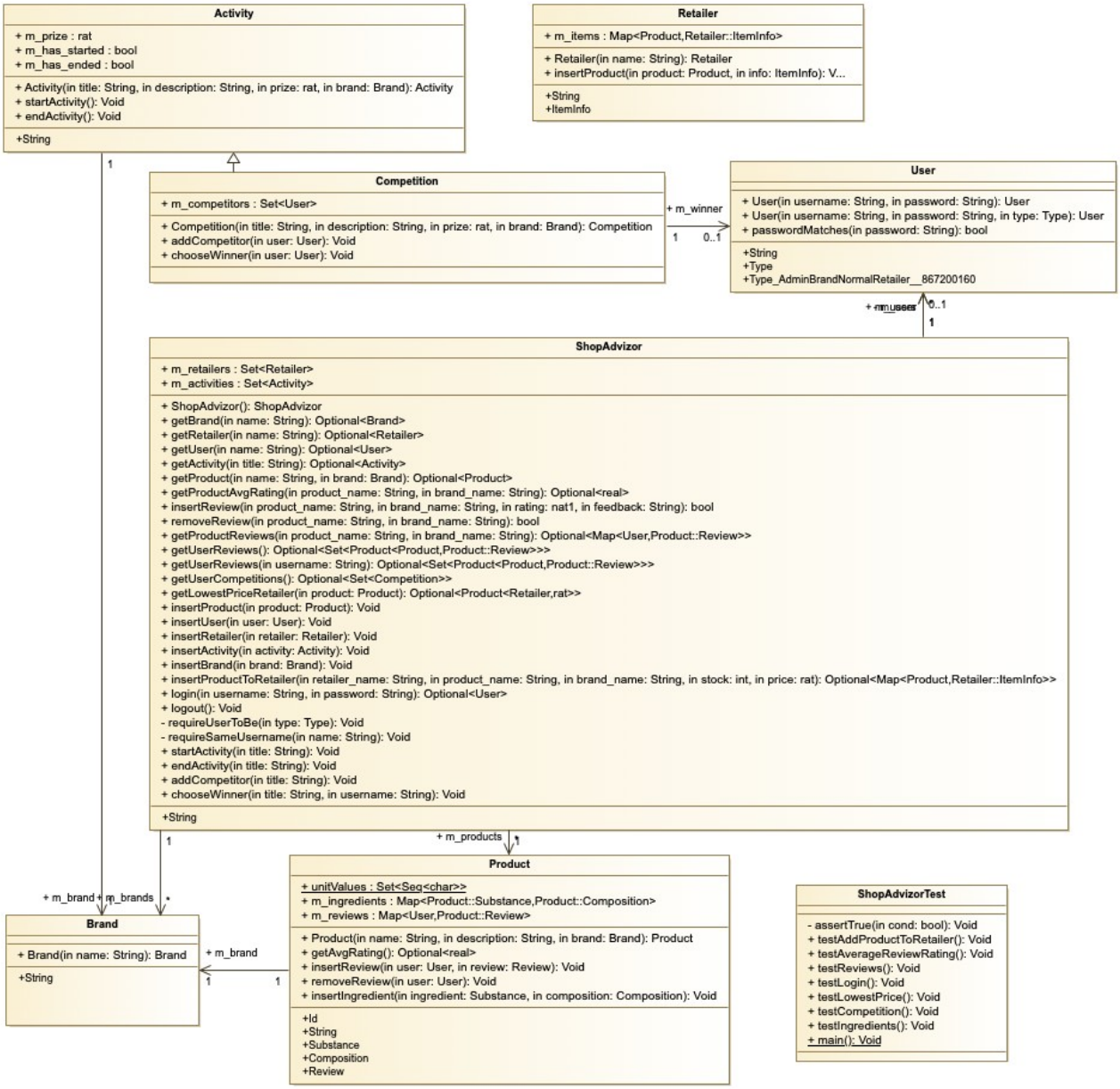
Scenario	Initiate activity
Description	Normal scenario for a user who wants to initiate an activity.
Pre-conditions	1. User must be a brand. 2. Activity has not started.
Post-conditions	1. Activity has started.
Steps	1. Start activity (set m_has_started attribute to true).
Exception	(unspecified)

Scenario	Close activity
Description	Normal scenario for a user who wants to close an activity.
Pre-conditions	1. User must be a brand. 2. Activity has not ended. 3. Activity has started.
Post-conditions	1. Activity has ended.
Steps	1. End activity (set m_has_ended attribute to true).
Exception	(unspecified)

Scenario	Choose winner
Description	Normal scenario for a user who wants to choose a winner in a competition.
Pre-conditions	1. User must be a brand. 2. Competition has started and ended. 3. Winner has not be chosen yet. 4. User to be chosen is in competition.
Post-conditions	1. User is that competition’s winner. 2. Winner is in competition.
Steps	1. State activity title and user’s username. 2. Set winner as user with that username.
Exception	(unspecified)

2.2 Class Model

The following class diagram represents all the implemented classes in VDM++, their attributes and operations and the associations between them. This class diagram was generated by Overture. The original image can be found in */UML/classDiagram.png*.



Class	Description
Activity	Defines an activity with a begin and end date. It is created by a brand user and has a description which defines its objective and rules. Each activity has an associated monetary prize.
Brand	Defines a brand which is attributed to a product. A brand can also create activities.
Competition	Subclass of Activity; Defines a competition by adding users to compete and a winner once the activity is finished.
Product	Defines a product in the system. Products can have user reviews composed by a rating and a review. It also stores its nutritional composition (ingredients) and its quantity.
Retailer	Defines a retailer responsible for the sale of products. Each retailer has information regarding stock and price of its products.
ShopAdvizor	Main class; Defines the system where all elements are initialized, added, modified, removed or queued.
ShopAdvizorTest	Class where all tests are implemented. This class tests all the system requirements with perfect coverage.
User	Defines a user who may view products' reviews, manage their own reviews and partake in competitions.

3 Formal VDM++ Model

3.1 Class Activity

```
class Activity
/*
  Defines an activity composed of a begin and end date, a description which explains
  what is consists of and a prize.
  Daniel Marques & Eduarda Cunha, FEUP, MFES, 2018/19.
*/

types
	public String = seq1 of char;

instance variables
	public m_title : String;
	public m_description : String;
	public m_prize: rat;
	public m_brand : Brand;
	public m_has_started : bool := false;
	public m_has_ended : bool := false;

	inv m_has_ended => m_has_started;
	inv m_prize >= 0.0;

operations
	-- Activity constructor
	public Activity : String * String * rat * Brand ==> Activity
	Activity(title, description, prize, brand) ==
	(
		m_title := title;
		m_description := description;
		m_prize := prize;
		m_brand := brand;
		m_has_started := false;
		m_has_ended := false;
		return self;
	);

	-- Start the activity
	public startActivity : () ==> ()
	startActivity() ==
	(
		m_has_started := true;
	)
	pre not m_has_started
	post m_has_started = true;

	-- End the activity
	public endActivity : () ==> ()
	endActivity() ==
	(
		m_has_ended := true;
```

```
        )
        pre not m_has_ended and m_has_started
        post m_has_ended = true;

end Activity
```

3.2 Class Brand

```
class Brand
/*
  Defines a brand which can manufacture a product.
  Daniel Marques & Eduarda Cunha, FEUP, MFES, 2018/19.
*/

types
        public String = seq1 of char;

instance variables
        public m_name: String;

operations
        -- Brand constructor
        public Brand : String ==> Brand
        Brand(name) ==
        (
                m_name := name;
                return self;
        );

end Brand
```

3.3 Class Competition

```
class Competition is subclass of Activity
/*
  Defines a competition which is an activity with users as competitors
  and a winner once the activity is completed.
  Daniel Marques & Eduarda Cunha, FEUP, MFES, 2018/19.
*/

instance variables
        public m_competitors : set of User;
        public m_winner : [User];

        inv forall u1, u2 in set m_competitors & u1 <> u2 => u1.m_username <> u2.m_username;
        inv m_winner = nil or (m_has_ended and m_winner in set m_competitors); -- Winner must
be a competitor

operations
        -- Competition constructor
        public Competition : String * String * rat * Brand ==> Competition
        Competition(title, description, prize, brand) ==
        (
```

```
		m_competitors := {};
		m_winner := nil;
		Activity(title, description, prize, brand);
	);

	-- Add a competitor
	public addCompetitor : User ==> ()
	addCompetitor(user) ==
	(
		m_competitors := m_competitors union {user}
	)
	pre user not in set m_competitors and not m_has_started
	post m_competitors = m_competitors~ union {user};

	-- Choose winner
	public chooseWinner : User ==> ()
	chooseWinner(user) ==
	(
		m_winner := user;
	)
	pre m_has_started and m_has_ended and m_winner = nil and user in set m_competitors
	post m_winner = user and m_winner in set m_competitors;

end Competition
```

3.4 Class Product

```
class Product
/*
  Defines a product to be reviewed.
  Since a product is composed of substances in certain proportions
  the units in which they are measured are also defined.
  Daniel Marques & Eduarda Cunha, FEUP, MFES, 2018/19.
*/

types
	public Id = nat1;
	public String = seq1 of char;
	public Substance = seq1 of char;
	public Composition :: quantity: real
	                      unit: seq1 of char
		inv c == c.unit in set unitValues;
	public Review :: rating: nat1
	                 feedback: seq1 of char
		inv review == review.rating >= 1 and review.rating <= 5;

values
	public unitValues : set of seq1 of char = {"Kg", "g", "l", "ml", "%", "units"};

instance variables
	public m_id: Id;
	private static m_next_id : Id := 1;
	public m_name: String;
	public m_description: String;
	public m_brand: Brand;
```

```
	public m_ingredients: map Substance to Composition;
	public m_reviews: map User to Review;

operations
	-- Product constructor
	public Product : String * String * Brand ==> Product
	Product(name, description, brand) ==
	(
		m_id := m_next_id;
		m_next_id := m_next_id + 1;
		m_name := name;
		m_description := description;
		m_brand := brand;
		m_ingredients := { |-> };
		m_reviews := { |-> };
		return self;
	);

	-- Returns the average review rating
	public getAvgRating : () ==> [real]
	getAvgRating() ==
	(
	 dcl sum : int := 0;
	 dcl num_reviews : int := card rng m_reviews;

	 if num_reviews = 0
		then return nil;

	 for all review in set rng m_reviews do sum := sum + review.rating;
	 return sum / num_reviews;
	);

	--Adds a review and the user who did it to reviews map
	public insertReview : User * Review ==> ()
	insertReview(user, review) ==
	(
		m_reviews := m_reviews ++ {user |-> review}
	)
	pre user not in set dom m_reviews
	post m_reviews = m_reviews~ ++ {user |-> review};

	--Removes a review from the reviews map
	public removeReview : User ==> ()
	removeReview(user) ==
	(
		m_reviews := {user} <-: m_reviews
	)
	pre user in set dom m_reviews
	post {user} <: m_reviews = {|->};

	--Adds an igredient and its quantity and unit to ingredients map
	public insertIngredient : Substance * Composition ==> ()
	insertIngredient(ingredient, composition) ==
	(
		m_ingredients := m_ingredients ++ {ingredient |-> composition}
	)
	pre ingredient not in set dom m_ingredients
```

```
        post m_ingredients = m_ingredients~ ++ {ingredient |-> composition};

end Product
```

3.5 Class Retailer

```
class Retailer
/*
  Defines a retailer which can sell a product.
  Daniel Marques & Eduarda Cunha, FEUP, MFES, 2018/19.
*/

types
        public String = seq1 of char;
        public ItemInfo :: stock: int
                           price: rat;

instance variables
        public m_name: String;
        public m_items: map Product to ItemInfo := { |-> };

operations
        -- Retailer constructor
        public Retailer : String ==> Retailer
        Retailer(name) ==
        (
                m_name := name;
                return self;
        );

        --Adds a product and its info (stock and price) to items map
        public insertProduct : Product * ItemInfo ==> ()
        insertProduct(product, info) ==
        (
                m_items := m_items ++ {product |-> info}
        )
        pre product not in set dom m_items
        post m_items = m_items~ ++ {product |-> info};

end Retailer
```

3.6 Class ShopAdvizor

```
class ShopAdvizor
/*
  Contains the core model of the ShopAdvizor app.
  Defines the state variables and operations available to the users.
  Daniel Marques & Eduarda Cunha, FEUP, MFES, 2018/19.
*/

types
        public String = seq1 of char;
```

```
instance variables
	public m_products: set of Product;
	public m_users: set of User;
	public m_retailers: set of Retailer;
	public m_brands: set of Brand;
	public m_activities: set of Activity;
	private m_user: [User];

	inv forall b1, b2 in set m_brands & b1 <> b2 => b1.m_name <> b2.m_name;
	inv forall r1, r2 in set m_retailers & r1 <> r2 => r1.m_name <> r2.m_name;
	inv forall u1, u2 in set m_users & u1 <> u2 => u1.m_username <> u2.m_username;
	inv forall p1, p2 in set m_products & p1 <> p2 => p1.m_name <> p2.m_name or
p1.m_brand <> p2.m_brand;
	inv forall a1, a2 in set m_activities & a1 <> a2 => a1.m_title <> a2.m_title;
	inv m_user = nil or m_user in set m_users;

operations
	-- ShopAdvizor constructor
	public ShopAdvizor : () ==> ShopAdvizor
	ShopAdvizor() ==
	(
		m_products := {};
		m_users := {};
		m_retailers := {};
		m_brands := {};
		m_activities := {};
		m_user := nil;
		return self;
	);

	-- Get brand by name
	public getBrand : String ==> [Brand]
	getBrand(name) ==
	(
		dcl brand : Brand;
		if exists1 b in set m_brands & b.m_name = name
		then (
			brand := iota b in set m_brands & b.m_name = name;
			return brand;
		)
		else return nil;
	);

	-- Get retailer by name
	public getRetailer : String ==> [Retailer]
	getRetailer(name) ==
	(
		dcl retailer : Retailer;
		if exists1 r in set m_retailers & r.m_name = name
		then (
			retailer := iota r in set m_retailers & r.m_name = name;
			return retailer;
		)
		else return nil;
	);

	-- Get user by name
```

```
	public getUser : String ==> [User]
	getUser(name) ==
	(
		dcl user : User;
		if exists1 u in set m_users & u.m_username = name
		then (
			user := iota u in set m_users & u.m_username = name;
			return user;
		)
		else return nil;
	);

	-- Get activity by title
	public getActivity : String ==> [Activity]
	getActivity(title) ==
	(
		dcl activity : Activity;
		if exists1 a in set m_activities & a.m_title = title
		then (
			activity := iota a in set m_activities & a.m_title = title;
			return activity;
		)
		else return nil;
	);

	-- Get product by name and brand
	public getProduct : String * Brand ==> [Product]
	getProduct(name, brand) ==
	(
		dcl product : Product;
		if exists1 p in set m_products & p.m_name = name and p.m_brand = brand
		then (
			product := iota p in set m_products & p.m_name = name and p.m_brand =
brand;
			return product;
		)
		else return nil;
	);

	-- Gets the average rating of a product
	public getProductAvgRating : String * String ==> [real]
	getProductAvgRating(product_name, brand_name) ==
	(
		dcl brand : [Brand] := getBrand(brand_name);
		dcl product : [Product];
		if brand = nil then return nil;
		product := getProduct(product_name, brand);
		if product = nil then return nil
		else return product.getAvgRating();
	);

	-- Insert a user review on a product
	public insertReview : String * String * nat1 * String ==> bool
	insertReview(product_name, brand_name, rating, feedback) ==
	(
		dcl brand : [Brand] := getBrand(brand_name);
		dcl product : [Product];
```

```
		requireUserToBe(<Normal>);
		if brand = nil then return false;
		product := getProduct(product_name, brand);
		if product = nil then return false
		else (
			product.insertReview(m_user, mk_Product`Review(rating, feedback));
			return true;
		)
	);

	-- Remove a user review on a product
	public removeReview : String * String ==> bool
	removeReview(product_name, brand_name) ==
	(
		dcl brand : [Brand] := getBrand(brand_name);
		dcl product : [Product];
		requireUserToBe(<Normal>);
		if brand = nil then return false;
		product := getProduct(product_name, brand);
		if product = nil then return false
		else (
			product.removeReview(m_user);
			return true;
		)
	);

	-- Gets all reviews of a product
	public getProductReviews : String * String ==> [map User to Product`Review]
	getProductReviews(product_name, brand_name) ==
	(
		dcl brand : [Brand] := getBrand(brand_name);
		dcl product : [Product];
		if brand = nil then return nil;
		product := getProduct(product_name, brand);
		if product = nil then return nil
		else return product.m_reviews
	);

	-- Gets all reviews given by the logged in user
	public getUserReviews : () ==> [set of (Product * Product`Review)]
	getUserReviews() ==
	(
		return getUserReviews(m_user.m_username);
	)
	pre m_user <> nil;

	-- Gets all reviews given by a user
	public getUserReviews : String ==> [set of (Product * Product`Review)]
	getUserReviews(username) ==
	(
		dcl user : [User] := getUser(username);
		dcl reviews : set of (Product * Product`Review);
		if user = nil
			then return nil
		else (
			reviews := dunion { { mk_(r, r.m_reviews(user)) | u in set dom r.m_reviews &
u = user} | r in set m_products};
```

```
                return reviews;
            )
        );

        -- Gets the competitions the user is competing on
        public getUserCompetitions : () ==> [set of Competition]
        getUserCompetitions() ==
        (
            dcl competitions : set of Competition;
            requireUserToBe(<Normal>);
            competitions := { c | c in set m_activities & isofclass(Competition,c) and m_user in
set narrow_(c,Competition).m_competitors};
            return competitions;
        );

        -- Gets the retailer with the lowest price on a product
        public getLowestPriceRetailer : Product ==> [Retailer * rat]
        getLowestPriceRetailer(product) ==
        (
            dcl retailer : Retailer * rat; -- Retailer that sells for the lowest price
            dcl retailers : set of (Retailer * rat); -- Retailers that sell the product
            if product not in set m_products
                then return nil
            else (
                retailers := { mk_(r, r.m_items(product).price) | r in set m_retailers & product
in set dom r.m_items };
                if exists r1 in set retailers & (forall r2 in set retailers & r1.#2 <= r2.#2)
                    then (
                        retailer := iota r1 in set retailers & (forall r2 in set retailers &
r1.#2 <= r2.#2);
                        return retailer;
                    )
                else return nil;
            )
        );

        --Adds a product to products set
        public insertProduct : Product ==> ()
        insertProduct(product) ==
        (
            requireUserToBe(<Brand>);
            requireSameUsername(product.m_brand.m_name);
            m_products := m_products union {product}
        )
        pre product not in set m_products and product.m_brand in set m_brands
        post m_products = m_products~ union {product};

        --Adds a user to users set
        public insertUser : User ==> ()
        insertUser(user) ==
        (
            m_users := m_users union {user}
        )
        pre user not in set m_users
        post m_users = m_users~ union {user};

        --Adds a retailer to retailers set
```

```
	public insertRetailer : Retailer ==> ()
	insertRetailer(retailer) ==
	(
		requireUserToBe(<Retailer>);
		requireSameUsername(retailer.m_name);
		m_retailers := m_retailers union {retailer}
	)
	pre retailer not in set m_retailers
	post m_retailers = m_retailers~ union {retailer};

	--Adds an activity to activities set
	public insertActivity : Activity ==> ()
	insertActivity(activity) ==
	(
		requireUserToBe(<Brand>);
		requireSameUsername(activity.m_brand.m_name);
		m_activities := m_activities union {activity}
	)
	pre activity not in set m_activities
	post m_activities = m_activities~ union {activity};

	--Adds a brand to brands set
	public insertBrand : Brand ==> ()
	insertBrand(brand) ==
	(
		requireUserToBe(<Brand>);
		requireSameUsername(brand.m_name);
		m_brands := m_brands union {brand}
	)
	pre brand not in set m_brands
	post m_brands = m_brands~ union {brand};

	-- Adds a product to a retailer
	public insertProductToRetailer : String * String * String * int * rat ==> [ map Product to
Retailer`ItemInfo]
	insertProductToRetailer(retailer_name, product_name, brand_name, stock, price) ==
	(
		dcl brand : [Brand] := getBrand(brand_name);
		dcl retailer: [Retailer] := getRetailer(retailer_name);
		dcl product : [Product];
		requireUserToBe(<Retailer>);
		if brand = nil then return nil;
		product := getProduct(product_name, brand);
		if product = nil or retailer = nil then return nil
		else (
			requireSameUsername(retailer.m_name);
			retailer.insertProduct(product, mk_Retailer`ItemInfo(stock, price));
			return retailer.m_items;
		)
	);

	--Login
	public login : String * String ==> [User]
	login(username, password) ==
	(
		if exists1 u in set m_users & u.m_username = username and
u.passwordMatches(password)
```

```
                then (
                        m_user := iota u in set m_users & u.m_username = username and
u.passwordMatches(password);
                        return m_user;
                )
        else return nil;
)
pre m_user = nil
post (RESULT <> nil and m_user <> nil) or (RESULT = nil and m_user = nil);

--Logout
public logout : () ==> ()
logout() ==
(
        m_user := nil;
)
pre m_user <> nil
post m_user = nil;

-- Enforce user permissions
private requireUserToBe : User`Type ==> ()
requireUserToBe(type) == return
pre m_user <> nil and m_user.m_type = type or m_user.m_type = <Admin>;

-- Enforce same username
private requireSameUsername : String ==> ()
requireSameUsername(name) == return
pre m_user <> nil and m_user.m_username = name or m_user.m_type = <Admin>;

-- Start activity
public startActivity : String ==> ()
startActivity(title) ==
(
        dcl activity : [Activity] := getActivity(title);
        requireUserToBe(<Brand>);
        if activity <> nil
                then (
                        requireSameUsername(activity.m_brand.m_name);
                        activity.startActivity();
                )
);

-- End activity
public endActivity : String ==> ()
endActivity(title) ==
(
        dcl activity : [Activity] := getActivity(title);
        requireUserToBe(<Brand>);
        if activity <> nil
                then (
                        requireSameUsername(activity.m_brand.m_name);
                        activity.endActivity();
                )
);

-- Add competitor to competition
public addCompetitor : String ==> ()
```

```
        addCompetitor(title) ==
        (
                dcl activity : [Activity] := getActivity(title);
                dcl competition : Competition;
                requireUserToBe(<Normal>);
                if activity <> nil and isofclass(Competition, activity)
                        then (
                                competition := narrow_(activity, Competition);
                                competition.addCompetitor(m_user)
                        );
        );

        -- Choose winner of a competition
        public chooseWinner : String * String ==> ()
        chooseWinner(title, username) ==
        (
                dcl activity : [Activity] := getActivity(title);
                dcl competition : Competition;
                dcl user : [User] := getUser(username);
                requireUserToBe(<Brand>);
                if user <> nil and activity <> nil and isofclass(Competition, activity)
                        then (
                                requireSameUsername(activity.m_brand.m_name);
                                competition := narrow_(activity, Competition);
                                competition.chooseWinner(user);
                        )
        );

end ShopAdvizor
```

3.7 Class User

```
class User
/*
  Defines a user to can use the app.
  Since with different permitions are allowed to do different things,
  those types are defined here.
  Daniel Marques & Eduarda Cunha, FEUP, MFES, 2018/19.
*/

        types
                public String = seq1 of char;
                public Type = <Normal> | <Retailer> | <Brand> | <Admin>

        instance variables
                public m_username: String;
                public m_type: Type;
                private m_password: String;

        operations
                -- User constructor
                public User : String * String ==> User
                User(username, password) ==
                (
                        m_username := username;
```

```
            m_password := password;
            m_type := <Normal>;
            return self;
        );

        -- User constructor
        public User : String * String * Type ==> User
        User(username, password, type) ==
        (
            m_username := username;
            m_password := password;
            m_type := type;
            return self;
        );

        -- Checks if the password matches
        public passwordMatches : String ==> bool
        passwordMatches(password) == return m_password = password;

end User
```

4 Model Validation

4.1 Class ShopAdvizorTest

Our project presently has 100% coverage as can be seen all throughout section 3 Formal VDM++ Model, as all classes have green highlighting to represent the code covered by our tests. Also, all our tests are currently passing.

The following class, ShopAdvizorTest, tests all the system's requirements.

```
class ShopAdvizorTest
/*
  Contains the test cases for the ShopAdvizor app.
  Illustrates a scenario-based testing approach.
  The test cases cover all usage scenarios.
  Daniel Marques & Eduarda Cunha, FEUP, MFES, 2018/19.
*/

 values
        private admin : User = new User("admin", "admin", <Admin>);

 operations
        private assertTrue: bool ==> ()
        assertTrue(cond) == return
        pre cond;

        public testAddProductToRetailer: () ==> ()
        testAddProductToRetailer() ==
        (
            dcl shopAdvizor : ShopAdvizor := new ShopAdvizor();
            dcl brand : Brand := new Brand("Chip Mix");
```

```
		dcl product : Product := new Product("Bolachas", "Bolachas de Chocolate", brand);
		dcl info : Retailer`ItemInfo := mk_Retailer`ItemInfo(20, 2.1);
		dcl retailer : Retailer := new Retailer("Pingo Doce");
		dcl user : User := new User("Pingo Doce", "pingo_doce", <Retailer>);
		shopAdvizor.insertUser(admin);
		assertTrue(shopAdvizor.login("admin", "admin") = admin);
		shopAdvizor.insertUser(user);
		shopAdvizor.insertBrand(brand);
		shopAdvizor.insertProduct(product);
		shopAdvizor.insertRetailer(retailer);
		shopAdvizor.logout();
		assertTrue(shopAdvizor.login("Pingo Doce", "pingo_doce") = user);
		assertTrue(shopAdvizor.insertProductToRetailer("Pingo Azedo", "Bolachas", "Chip
Mix", info.stock, info.price) = nil);
		assertTrue(shopAdvizor.insertProductToRetailer("Pingo Doce", "Bolachas", "Oreo",
info.stock, info.price) = nil);
		assertTrue(shopAdvizor.insertProductToRetailer("Pingo Doce", "Detergente da
Roupa", "Chip Mix", info.stock, info.price) = nil);
		assertTrue(shopAdvizor.insertProductToRetailer("Pingo Doce", "Bolachas", "Chip
Mix", info.stock, info.price) = {product |-> info});
	);

	public testAverageReviewRating : () ==> ()
	testAverageReviewRating() ==
	(
		dcl shopAdvizor : ShopAdvizor := new ShopAdvizor();
		dcl brand : Brand := new Brand("Chip Mix");
		dcl product : Product := new Product("Bolachas", "Bolachas de Chocolate", brand);
		dcl u1 : User := new User("u1", "p1");
		dcl u2 : User := new User("u2", "p2");
		shopAdvizor.insertUser(admin);
		shopAdvizor.insertUser(u1);
		shopAdvizor.insertUser(u2);
		assertTrue(shopAdvizor.login("admin", "admin") = admin);

		assertTrue(shopAdvizor.getProductAvgRating("Bolachas", "Chip Mix") = nil);
		shopAdvizor.insertBrand(brand);
		assertTrue(shopAdvizor.getProductAvgRating("Bolachas", "Chip Mix") = nil);
		shopAdvizor.insertProduct(product);

		shopAdvizor.logout();
		assertTrue(shopAdvizor.getProductAvgRating("Bolachas", "Chip Mix") = nil);
		assertTrue(shopAdvizor.login("u1", "p1") = u1); -- Insert review with user u1
		assertTrue(shopAdvizor.insertReview("Bolachas", "Oreos", 1, "mau") = false); --
Oreos does not exist
		assertTrue(shopAdvizor.insertReview("Bolachas", "Chip Mix", 1, "mau") = true);
		assertTrue(shopAdvizor.getProductAvgRating("Bolachas", "Chip Mix") = 1); --
Average of {1} = 1
		shopAdvizor.logout();
		assertTrue(shopAdvizor.login("u2", "p2") = u2); -- Insert review with user u2
		assertTrue(shopAdvizor.insertReview("Ketchup", "Chip Mix", 1, "mau") = false); --
Chip Mix does not sell Ketchup
		assertTrue(shopAdvizor.insertReview("Bolachas", "Chip Mix", 4, "bom") = true);
		assertTrue(shopAdvizor.getProductAvgRating("Bolachas", "Chip Mix") = 2.5); --
Average of {1, 4} = 2.5

		-- Remove review
```

```
		assertTrue(shopAdvizor.removeReview("Bolachas", "Oreos") = false); -- Oreos does
not exist
		assertTrue(shopAdvizor.removeReview("Ketchup", "Chip Mix") = false); -- Chip Mix
does not sell Ketchup
		assertTrue(shopAdvizor.removeReview("Bolachas", "Chip Mix") = true); -- Remove
review of user u2
		assertTrue(shopAdvizor.getProductAvgRating("Bolachas", "Chip Mix") = 1); --
Average of {1} = 1
	);

	public testReviews : () ==> ()
	testReviews() ==
	(
		dcl shopAdvizor : ShopAdvizor := new ShopAdvizor();
		dcl b1 : Brand := new Brand("b1");
		dcl b2 : Brand := new Brand("b2");
		dcl p1 : Product := new Product("p1", "pd1", b1);
		dcl p2 : Product := new Product("p2", "pd2", b2);
		dcl u1 : User := new User("n1", "np1");
		dcl u2 : User := new User("n2", "np2");
		shopAdvizor.insertUser(admin);
		assertTrue(shopAdvizor.login("admin", "admin") = admin);
		shopAdvizor.insertBrand(b1);
		shopAdvizor.insertBrand(b2);
		shopAdvizor.insertProduct(p1);
		shopAdvizor.insertProduct(p2);
		shopAdvizor.insertUser(u1);
		shopAdvizor.insertUser(u2);

		assertTrue(shopAdvizor.getUserReviews("n3") = nil); -- User n3 is not in the system
		assertTrue(shopAdvizor.getUserReviews("n1") = {}); -- User n1 does not have any
reviews
		assertTrue(shopAdvizor.getProductReviews("p3", "b1") = nil); -- Product p3 does not
exist
		assertTrue(shopAdvizor.getProductReviews("p1", "b3") = nil); -- Brand b3 does not
exist
		assertTrue(shopAdvizor.getProductReviews("p1", "b1") = {|->}); -- Product p3 of
brand b3 has no reviews

		p1.insertReview(u1, mk_Product`Review(1, "fb1")); -- User1 has a review on
Product1
		assertTrue(shopAdvizor.getUserReviews("n1") = { mk_(p1, mk_Product`Review(1,
"fb1")) });
		assertTrue(shopAdvizor.getProductReviews("p1", "b1") = { u1 |->
mk_Product`Review(1, "fb1") });
		p1.insertReview(u2, mk_Product`Review(2, "fb2")); -- User2 has a review on
Product1
		assertTrue(shopAdvizor.getUserReviews("n2") = { mk_(p1, mk_Product`Review(2,
"fb2")) });
		assertTrue(shopAdvizor.getProductReviews("p1", "b1") = { u1 |->
mk_Product`Review(1, "fb1"), u2 |-> mk_Product`Review(2, "fb2") });
		p2.insertReview(u1, mk_Product`Review(5, "fb1")); -- User1 has a review on
Product1 and Product2
		assertTrue(shopAdvizor.getUserReviews("n1") = { mk_(p1, mk_Product`Review(1,
"fb1")), mk_(p2, mk_Product`Review(5, "fb1")) });
		assertTrue(shopAdvizor.getProductReviews("p2", "b2") = { u1 |->
mk_Product`Review(5, "fb1") });
```

```
        p2.removeReview(u1);
        assertTrue(shopAdvizor.getProductReviews("p2", "b2") = {|->});

        shopAdvizor.logout();
        assertTrue(shopAdvizor.login("n1","np1") = u1);
        assertTrue(shopAdvizor.getUserReviews() = shopAdvizor.getUserReviews("n1"));
);

public testLogin : () ==> ()
testLogin() ==
(
                dcl shopAdvizor : ShopAdvizor := new ShopAdvizor();
                dcl u1 : User := new User("user", "pass");
                shopAdvizor.insertUser(u1);
                assertTrue(shopAdvizor.login("user", "123") = nil);
                assertTrue(shopAdvizor.login("123", "pass") = nil);
                assertTrue(shopAdvizor.login("user", "pass") = u1);
                shopAdvizor.logout();
);

public testLowestPrice : () ==> ()
testLowestPrice() ==
(
        dcl shopAdvizor : ShopAdvizor := new ShopAdvizor();
        dcl b1 : Brand := new Brand("b1");
        dcl b2 : Brand := new Brand("b2");
        dcl p1 : Product := new Product("p1", "pd1", b1);
        dcl p2 : Product := new Product("p2", "pd2", b1);
        dcl p3 : Product := new Product("p1", "pd3", b2);
        dcl p4 : Product := new Product("p2", "pd4", b2);
        dcl r1 : Retailer := new Retailer("r1");
        dcl r2 : Retailer := new Retailer("r2");
        shopAdvizor.insertUser(admin);
        assertTrue(shopAdvizor.login("admin", "admin") = admin);
        shopAdvizor.insertBrand(b1);
        shopAdvizor.insertBrand(b2);
        shopAdvizor.insertProduct(p1);
        shopAdvizor.insertProduct(p2);
        shopAdvizor.insertProduct(p3);
        shopAdvizor.insertRetailer(r1);
        shopAdvizor.insertRetailer(r2);
        r1.insertProduct(p1, mk_Retailer`ItemInfo(25, 2.5)); -- R1 sells P1 for 2.5
        r1.insertProduct(p2, mk_Retailer`ItemInfo(30, 1.5)); -- R1 sells P2 for 1.5
        r2.insertProduct(p1, mk_Retailer`ItemInfo(25, 2.0)); -- R2 sells P1 for 2.0
        assertTrue(shopAdvizor.getLowestPriceRetailer(p1) = mk_(r2, 2.0));
        assertTrue(shopAdvizor.getLowestPriceRetailer(p2) = mk_(r1, 1.5));
        assertTrue(shopAdvizor.getLowestPriceRetailer(p3) = nil); -- Not sold by any retailer
        assertTrue(shopAdvizor.getLowestPriceRetailer(p4) = nil); -- Not in the system
);

public testCompetition : () ==> ()
testCompetition() ==
(
        dcl shopAdvizor : ShopAdvizor := new ShopAdvizor();
        dcl u1 : User := new User("u1", "p1");
        dcl u2 : User := new User("u2", "p2");
        dcl u3 : User := new User("u3", "p3");
```

```
		dcl bu : User := new User("Oreo", "oreo_pass", <Brand>);
		dcl b : Brand := new Brand("Oreo");
		dcl b2 : Brand := new Brand("ChipMix");
		dcl c : Competition := new Competition("Competitive eating", "Eat as much cookies as possible", 20.0, b);
		dcl c2 : Competition := new Competition("Biggest buyer", "Buy the biggest ammount of cookies", 100.0, b2);
		assertTrue(shopAdvizor.getActivity("Competitive eating") = nil);
		shopAdvizor.insertUser(admin);
		shopAdvizor.insertUser(bu); -- Oreo brand user
		assertTrue(shopAdvizor.login("admin", "admin") = admin); -- Admin has all permissions
		shopAdvizor.insertBrand(b);
		shopAdvizor.insertBrand(b2);
		shopAdvizor.insertUser(u1);
		shopAdvizor.insertUser(u2);
		shopAdvizor.insertUser(u3);
		shopAdvizor.insertActivity(c);
		shopAdvizor.insertActivity(c2);

		-- Users can only enter themselves in the competition
		shopAdvizor.logout();
		assertTrue(shopAdvizor.login("u1","p1") = u1);
		shopAdvizor.addCompetitor("Competitive eating");
		shopAdvizor.addCompetitor("Biggest buyer");
		assertTrue(shopAdvizor.getUserCompetitions() = {c,c2});
		shopAdvizor.logout();
		assertTrue(shopAdvizor.login("u2","p2") = u2);
		shopAdvizor.addCompetitor("Competitive eating");
		shopAdvizor.logout();
		assertTrue(shopAdvizor.login("u3","p3") = u3);
		shopAdvizor.addCompetitor("Competitive eating");
		assertTrue(c.m_competitors = {u1, u2, u3});

		-- The brand user can start, end and choose the winner of a competition
		shopAdvizor.logout();
		assertTrue(shopAdvizor.login("Oreo","oreo_pass") = bu);
		shopAdvizor.startActivity("Competitive eating");
		shopAdvizor.endActivity("Competitive eating");
		shopAdvizor.chooseWinner("Competitive eating", "u1");
		assertTrue(c.m_winner = u1);
	);

	public testIngredients : () ==> ()
	testIngredients() ==
	(
		dcl shopAdvizor : ShopAdvizor := new ShopAdvizor();
		dcl b : Brand := new Brand("Oreo");
		dcl p : Product := new Product("Cookies", "Chocolate cookies", b);
		p.insertIngredient("milk", mk_Product`Composition(20.0, "ml"));
		p.insertIngredient("chocolate", mk_Product`Composition(50.0, "g"));
		p.insertIngredient("sugar", mk_Product`Composition(5.0, "g"));
		shopAdvizor.insertUser(admin);
		assertTrue(shopAdvizor.login("admin", "admin") = admin);
		shopAdvizor.insertBrand(b);
		shopAdvizor.insertProduct(p);
	);
```

```
public static main: () ==> ()
main() ==
(
        dcl test : ShopAdvizorTest := new ShopAdvizorTest();
        test.testAddProductToRetailer();
        test.testAverageReviewRating();
        test.testReviews();
        test.testLogin();
        test.testLowestPrice();
        test.testCompetition();
        test.testIngredients();
);

end ShopAdvizorTest
```

The following tables' values were automatically generated in Overture for each class and display not only the full project's coverage, but also each operation, the line where it is implemented and how many times it has been called.

Function or operation	Line	Coverage	Calls
[Activity:23]Activity	23	100.0%	2
[endAactivity:45]endActivity	45	100.0%	1
[startActivity:36]startActivity	36	100.0%	1
Activity.vdmpp		100.0%	4

Function or operation	Line	Coverage	Calls
[Brand:15]Brand	15	100.0%	9
Brand.vdmpp		100.0%	9

Function or operation	Line	Coverage	Calls
[Competition:17]Competition	17	100.0%	2
[addCompetitor:26]addCompetitor	26	100.0%	4
[chooseWinner:35]chooseWinner	35	100.0%	2
Competition.vdmpp		100.0%	8

Function or operation	Line	Coverage	Calls
[Product:34]Product	34	100.0%	9
[getAvgRating:48]getAvgRating	48	100.0%	4
[insertIngredient:35]insertIngredient	80	100.0%	3
[insertReview:62]insertReview	62	100.0%	5
[removeReview:71]removeReview	71	100.0%	2
Product.vdmpp		100.0%	23

Function or operation	Line	Coverage	Calls
[Retailer:18]Retailer	18	100.0%	3
[insertProduct:26]insertProduct	26	100.0%	4
Retailer.vdmpp		100.0%	7

Function or operation	Line	Coverage	Calls
[ShopAdvizor:28]ShopAdvizor	28	100.0%	7
[addCompetitor:343]addCompetitor	343	100.0%	4
[chooseWinner:357]chooseWinner	357	100.0%	1
[endActivity:330]endActivity	330	100.0%	1
[getActivity:80]getActivity	80	100.0%	1
[getBrand:4]getBrand	41	100.0%	5
[getLowestPriceRetailer:194]getLowestPriceRetailer	194	100.0%	2
[getProduct:93]getProduct	93	100.0%	5
[getProductAvgRating:106]getProductAvgRating	106	100.0%	4
[getProductReviews:150]getProductReviews	150	100.0%	7
[getRetailer:54]getRetailer	54	100.0%	1
[getUser:67]getUser	67	100.0%	1
[getUserCompetitions:184]getUserCompetitions	184	100.0%	1
[getUserReviews:162]getUserReviews	162	100.0%	6
[insertActivity:244]insertActivity	244	100.0%	2
[insertBrand:255]insertBrand	255	100.0%	9
[insertProduct:213]insertProduct	213	100.0%	8
[insertProductToRetailer:266]insertProductToRetailer	266	100.0%	1
[insertRetailer:233]insertRetailer	233	100.0%	3
[insertReview:118]insertReview	118	100.0%	2
[insertUser:224]insertUser	224	100.0%	16
[login:284]login	284	100.0%	2
[logout:298]logout	298	100.0%	18
[removeReview:134]removeReview	134	100.0%	2
[requireSameUsername:312]requireSameUsername	312	100.0%	26
[requireUserToBe:307]requireUserToBe	307	100.0%	41
[startActivity:317]startActivity	317	100.0%	1
ShopAdvizor.vdmpp		100.0%	177

Function or operation	Line	Coverage	Calls
[User:20]User	20	100.0%	3
[passwordMatches:40]passwordMatches	40	100.0%	31
User.vdmpp		100.0%	34

5 Model Verification

5.1 Example of Domain Verification

One of the proof obligations generated by Overture is:

No.	Proof Obligation Name	Type
37	ShopAdvizor`getUserReviews(ShopAdvizor`String)	legal map application

The code under analysis is in the operation *ShopAdvizor`getUserReviews*, which returns the reviews made by the user whose *username* is the operation's parameter. The relevant map application is underlined:

```
-- Gets all reviews given by a user
public getUserReviews : String ==> [set of (Product * Product`Review)]
getUserReviews(username) ==
(
	dcl user : [User] := getUser(username);
	dcl reviews : set of (Product * Product`Review);
	if user = nil
		then return nil
	else (
		reviews := dunion { { mk_(r, r.m_reviews(user)) | u in set dom r.m_reviews
& u = user} | r in set m_products};
		return reviews;
	)
);
```

We can prove that the domain holds because the map key *user* is being accessed (r.m_reviews(user)) after it's verified that *user* is equal to *u* (u = user) which is an element of the map (u **in set dom** r.m_reviews).

5.2 Example of Invariant Verification

Another proof obligation generated by Overture is:

No.	Proof Obligation Name	Type
9	Competition`chooseWinner(User)	state invariant holds

The code under analysis is in the operation *Competition`chooseWinner*, which assigns the parameter as the winner of a competition. The relevant state change is underlined:

```
-- Choose winner
public chooseWinner : User ==> ()
chooseWinner(user) ==
(
	m_winner := user;
)
pre m_has_started and m_has_ended and m_winner = nil and user in set m_competitors
post m_winner = user and m_winner in set m_competitors;
```

The relevant invariant under analysis is:

inv m_winner = **nil or** (m_has_ended **and** m_winner **in set** m_competitors); -- Winner must be a competitor

The invariant holds because:

1. The pre condition enforces 'm_has_ended' to be true, which is not changed throughout the operation.
2. The post condition enforces 'm_winner **in set** m_competitors'.
3. The pre condition confirms there was no winner before the operation was called ('m_winner = **nil**').

6 Code Generation

The Java code was generated from our VDM++ model using the option offered by Overture as suggested. Essentially, in Overture, we selected the project folder and chose Code Generation → Generate Java (Launch Configuration Based). After following these steps, we noticed that some code elements such as pre/post conditions and invariants were not transcribed into the Java code resulting in code with very poor error and consistency checking. Also, the enumerations were converted with an invalid naming convention so the group fixed the naming of those manually.

In addition, we also implemented a console interface so that a user may use the developed app and test it out. The entry point for the console interface is in the class *MenuFactory.java*.

All the generated Java code can be found in */Java*.

7 Conclusions

The model we developed not only covers all the requirements set in the provided guideline but also accommodates additional features we created with the purpose of adding complexity to the project and allowing us to explore VDM++ in more depth.

Although we are pleased with the end result, in the future we believe we could add even more features as to fully explore all of VDM++'s potential. Considering the deadline of the project, we could have worked on this but decided not to since we already had more code pages than intended for a group of two students.

This project took approximately 30 hours to develop spread out over a 10-day period with all operations, tests and java code generation included. Each group member worked equally on this project.

8 References

1. Overture tool web site, http://overturetool.org
2. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
3. Class slides on VDM++