U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Formal Modeling of an App Demand Service: Glovo - Purchases, Delivers and Picks-up

Course: Master in Informatics and Computing Engineering
Subject : Formal Methods in Software Engineering

Group:
Bernardo José Coelho Leite
Vítor Emanuel Fernandes Magalhães

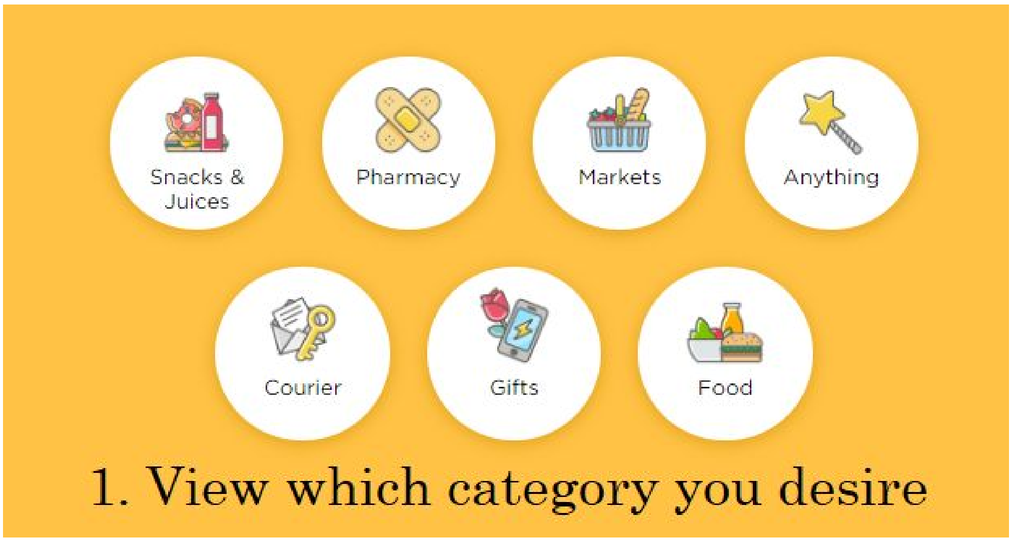
7, January 2019

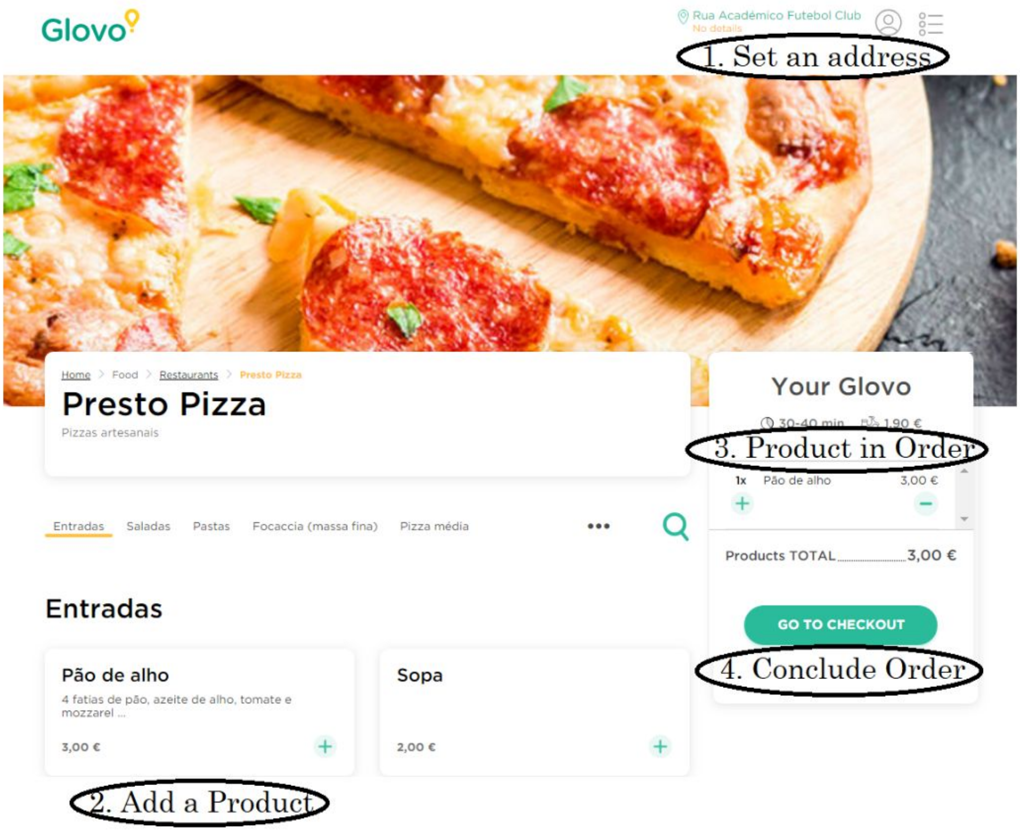
Contents

1. Informal system description and list of requirements

1.1 Informal system description

Glovo is an app that allows anyone to get desired products in minutes. All a user needs to do is tell the app what they want and the order is delivered to the usee, wherever they are.





1.2 List of requirements

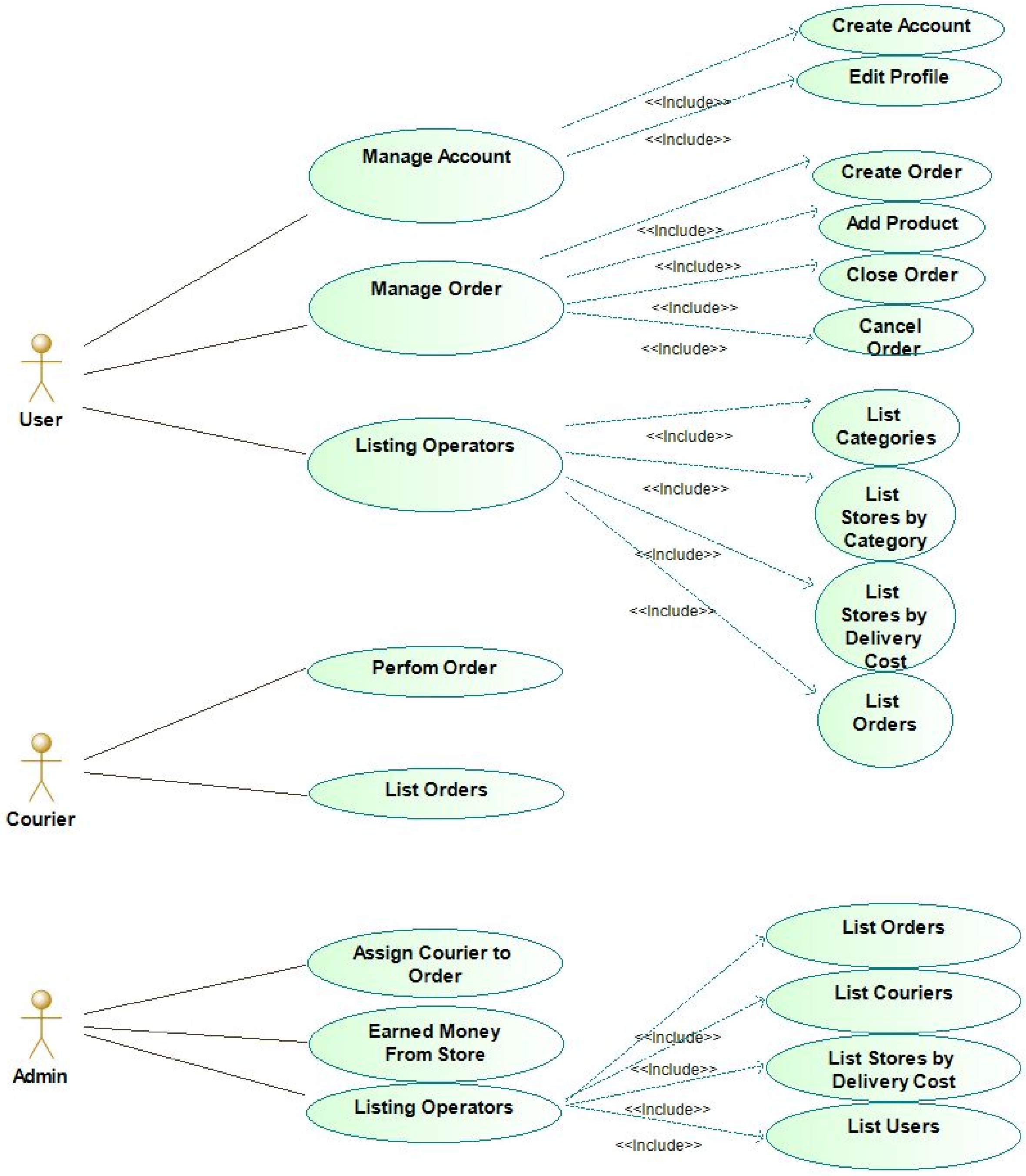
Id	Priority	Description
R1	Mandatory	The User Client can Manage an Order (create order, add products, close order, cancel

		order). The User Client can Manage Account (create and edit) The User Client can use List operators(categories,stores,etc,..)
R2	Mandatory	The Courier can Perform an Order and use List operators.
R3	Mandatory	The Admin can assign a Courier to an Order. The Admin can add Couriers. The Admin can use List operators
R4	Mandatory	The System App should generate an Order ID adding it in the list.

These requirements are directly translated onto use cases as shown next.

2. Visual UML model

2.1 Use case model



The major use case scenarios are described next.

Scenario	**Create an order**
Description	Normal ordering scenario for creating an order for a user.
Pre-conditions	1. The email belongs to a registered user. *(input)*
Post-conditions	(unspecified)
Steps	1. Input email. 2. New Order is created with a randomly generated ID.
Exceptions	(unspecified)

Scenario	**Add a product to an order**
Description	Normal addition scenario for adding a Product to an Order.
Pre-conditions	1. The order exists *(input)*. 2. The order is in the initial state. *(initial system state)* 3. The stock of the product is be greater than the indicated units. (input) 4. The email belongs to a registered user. *(input)* 5. The store exists. *(input)* 6. The product exists. *(input)*
Post-conditions	(unspecified)
Steps	1. Input email, orderID, store name, product index and number of units. 2. New Order is created with a randomly generated ID.
Exceptions	(unspecified)

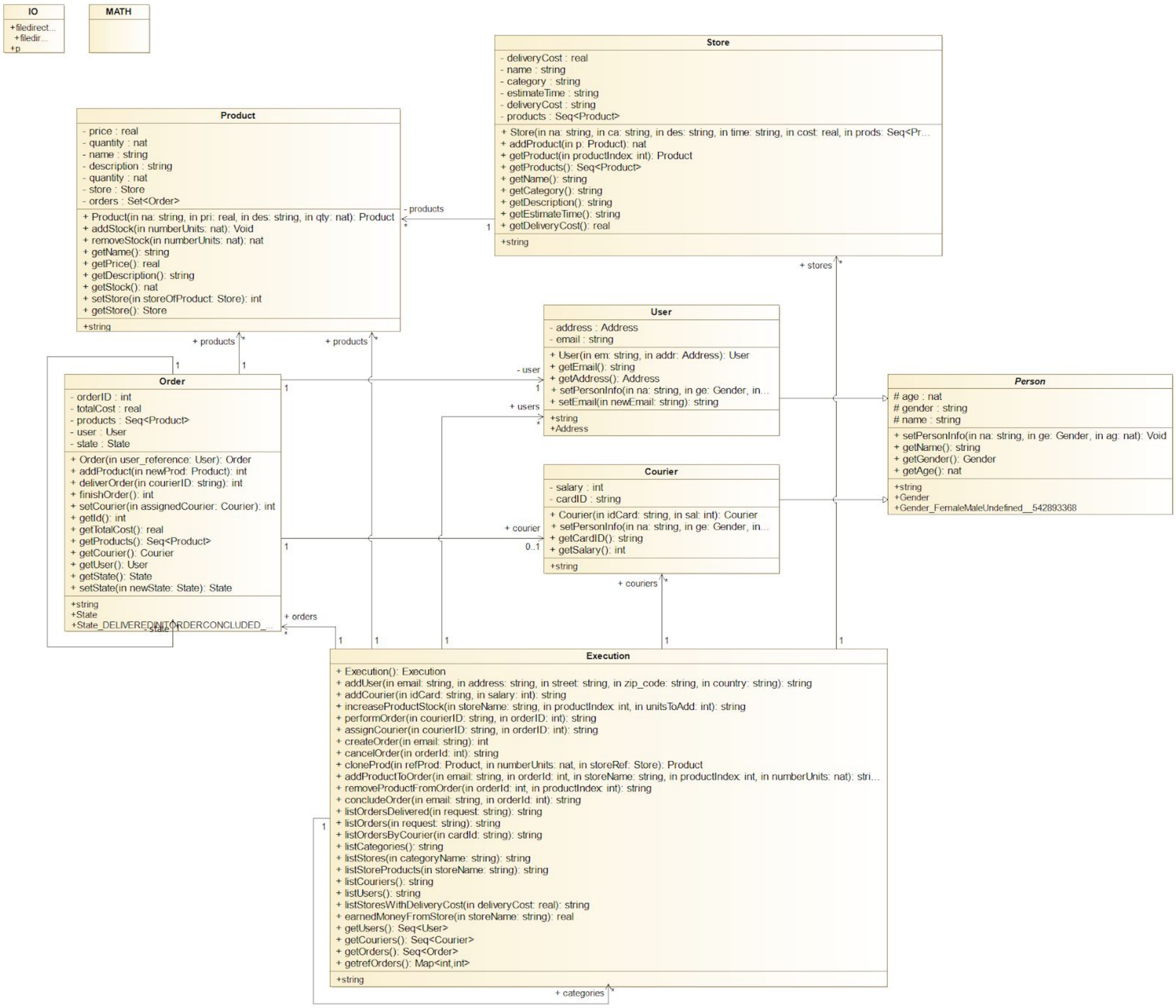
Scenario	**Conclude order**
Description	Normal addition scenario for concluding an order.
Pre-conditions	1. The order exists *(input)*. 2. The order is in the initial state. *(initial system state)* 3. The email belongs to a registered user. *(input)*
Post-conditions	1. The order is in the concluded state. *(intermediate system state)*

Steps	1. Input email and orderID. 2. The Total Cost of the Order is calculated and saved. 3. The order is in the concluded state. *(intermediate system state)*
Exceptions	(unspecified)

Scenario	**Assign a Courier**
Description	Normal assingment scenario for setting a courier to a Order.
Pre-conditions	1. The order exists *(input).* 2. The courier exists. *(input)* 3. The courier is available. *(system state)* 4. The order is in the concluded state. *(intermediate system state)*
Post-conditions	1. The courier has the order assigned to him/her.
Steps	(unspecified)
Exceptions	(unspecified)

Scenario	**Perform Order**
Description	Normal assingment scenario for setting a courier to a Order.
Pre-conditions	1. The order exists. *(input)* 2. The courier exists. *(input)* 3. The order is in the concluded state. *(intermediate system state)* 4. The order has the courier assigned to it.
Post-conditions	1. The order is in the delivered state. *(final system state)*
Steps	(unspecified)
Exceptions	(unspecified)

2.2 Class model


IO
+filedirect...
+filedir...
+p
MATH
Store
- deliveryCost : real
- name : string
- category : string
- estimateTime : string
- deliveryCost : string
- products : Seq<Product>
+ Store(in na: string, in ca: string, in des: string, in time: string, in cost: real, in prods: Seq<Pr...
+ addProduct(in p: Product): nat
+ getProduct(in productIndex: int): Product
+ getProducts(): Seq<Product>
+ getName(): string
+ getCategory(): string
+ getDescription(): string
+ getEstimateTime(): string
+ getDeliveryCost(): real
+string
Product
- price : real
- quantity : nat
- name : string
- description : string
- quantity : nat
- store : Store
- orders : Set<Order>
+ Product(in na: string, in pri: real, in des: string, in qty: nat): Product
+ addStock(in numberUnits: nat): Void
+ removeStock(in numberUnits: nat): nat
+ getName(): string
+ getPrice(): real
+ getDescription(): string
+ getStock(): nat
+ setStore(in storeOfProduct: Store): int
+ getStore(): Store
+string
- products
*
1
+ stores
*
+ products
*
+ products
*
1
1
User
- address : Address
- email : string
+ User(in em: string, in addr: Address): User
+ getEmail(): string
+ getAddress(): Address
+ setPersonInfo(in na: string, in ge: Gender, in...
+ setEmail(in newEmail: string): string
+string
+Address
- user
1
1
+ users
*
Person
# age : nat
# gender : string
# name : string
+ setPersonInfo(in na: string, in ge: Gender, in ag: nat): Void
+ getName(): string
+ getGender(): Gender
+ getAge(): nat
+string
+Gender
+Gender_FemaleMaleUndefined__542893368
Order
- orderID : int
- totalCost : real
- products : Seq<Product>
- user : User
- state : State
+ Order(in user_reference: User): Order
+ addProduct(in newProd: Product): int
+ deliverOrder(in courierID: string): int
+ finishOrder(): int
+ setCourier(in assignedCourier: Courier): int
+ getId(): int
+ getTotalCost(): real
+ getProducts(): Seq<Product>
+ getCourier(): Courier
+ getUser(): User
+ getState(): State
+ setState(in newState: State): State
+string
+State
+State_DELIVEREDINITORDERCONCLUDED_...
- state
1
Courier
- salary : int
- cardID : string
+ Courier(in idCard: string, in sal: int): Courier
+ setPersonInfo(in na: string, in ge: Gender, in...
+ getCardID(): string
+ getSalary(): int
+string
+ courier
1
0..1
+ couriers
*
+ orders
*
1
1
1
1
1
Execution
+ Execution(): Execution
+ addUser(in email: string, in address: string, in street: string, in zip_code: string, in country: string): string
+ addCourier(in idCard: string, in salary: int): string
+ increaseProductStock(in storeName: string, in productIndex: int, in unitsToAdd: int): string
+ performOrder(in courierID: string, in orderID: int): string
+ assignCourier(in courierID: string, in orderID: int): string
+ createOrder(in email: string): int
+ cancelOrder(in orderId: int): string
+ cloneProd(in refProd: Product, in numberUnits: nat, in storeRef: Store): Product
+ addProductToOrder(in email: string, in orderId: int, in storeName: string, in productIndex: int, in numberUnits: nat): stri...
+ removeProductFromOrder(in orderId: int, in productIndex: int): string
+ concludeOrder(in email: string, in orderId: int): string
+ listOrdersDelivered(in request: string): string
+ listOrders(in request: string): string
+ listOrdersByCourier(in cardId: string): string
+ listCategories(): string
+ listStores(in categoryName: string): string
+ listStoreProducts(in storeName: string): string
+ listCouriers(): string
+ listUsers(): string
+ listStoresWithDeliveryCost(in deliveryCost: real): string
+ earnedMoneyFromStore(in storeName: string): real
+ getUsers(): Seq<User>
+ getCouriers(): Seq<Courier>
+ getOrders(): Seq<Order>
+ getrefOrders(): Map<int,int>
+string
1
+ categories
*

Class	Description
Courier	Defines a courier in the app, which accepts and delivers Orders. Is a subclass of Person.
Execution	Core model; defines the state variables, such as products, users and orders and operations available to the any type of Person.
MyTestCase	Superclass for test classes; defines assertEquals and assertTrue.
Order	Defines an Order, composed by products and created by a User.

Person	Superclass that defines a person that uses the application.
Product	Defines a product at sale in a store.
Store	Defines a store with products to be added to the store.
TestGlovo	Defines the test/usage scenarios and test cases for the Glovo application
User	Defines a user in the app, which creates, edits, deletes and orders Orders. Is a subclass of Person.

3. Formal VDM++ model

3.1 Class Courier

```
class Courier is subclass of Person

types
public string = seq of char;

instance variables
private cardID : string;
private salary : int;

inv salary > 0;

operations

public Courier: string * int ==> Courier
Courier(idCard, sal) == (cardID := idCard; salary:=sal; return self);

public setPersonInfo: string * Gender * nat ==> ()
setPersonInfo(na, ge, ag) == (name := na; gender := ge; age:=ag;)
```

```
post name = na and gender = ge and age = ag;

pure public getCardID: () ==> string
getCardID() == return cardID
post RESULT = cardID;

pure public getSalary: () ==> int
getSalary() == return salary
post RESULT = salary;
end Courier
```

3.2 Class Execution

```
class Execution

types
public string = seq of char;
public Gender = <Male> | <Female> | <Undefined>;

values
address1 : User`Address = mk_User`Address("as","as","as","as");
address2 : User`Address = mk_User`Address("as","bl","44","z9");

instance variables
public stores: seq of Store;
public products: seq of Product;
public couriers: seq of Courier;
public orders: seq of Order;
public refOrders: map int to int;
public users: seq of User;
public categories: set of string;

-- Users
user0 : User := new User("blankuser", address1);
user1 : User := new User("bernas@hotmail.com", address1);
user2 : User := new User("vitorino@hotmail.com", address2);

-- Couriers
courier1 : Courier := new Courier("maria@glovo.com", 3);

-- Products
prod1A : Product := new Product("Rabanada", 2.5, "Rabanada de Convento", 5);
prod2A : Product := new Product("Bolina", 1.5, "Docinho de Bolina", 10);
prod3A : Product := new Product("Bolo Rei", 3.99, "Bolo Rei sem frutos", 6);
prodsA : seq of Product := [prod1A,prod2A,prod3A];

prod1B : Product := new Product("Pizza Tropical", 10, "Pizza com ananas", 3);
prod2B : Product := new Product("Pizza Funghi", 7, "Pizza com cogumelos", 8);
prod3B : Product := new Product("Pizza Margherita", 6.5, "Pizza normal", 2);
prodsB : seq of Product := [prod1B,prod2B,prod3B];

prod1C : Product := new Product("Sashimi", 30, "Peixe Sashimi", 3);
prod2C : Product := new Product("Hosomaki", 41, "Peixe Hosomaki", 8);
prod3C : Product := new Product("Uramaki", 90, "Peixe com Arrroz", 2);
prod4C : Product := new Product("Hot Roll", 19, "Sushi Nuggets", 2);
```

```
prodsC : seq of Product := [prod1C,prod2C,prod3C,prod4C];

prod1D : Product := new Product("Penso Rápido", 10, "Pensos", 7);
prod2D : Product := new Product("Ben-u-ron", 20, "Benuron 1000mg", 8);
prod3D : Product := new Product("Akiliever", 50, "Melhor creme para as mãos", 18);
prod4D : Product := new Product("Nivea", 19, "Creme corporal", 9);
prodsD : seq of Product := [prod1D,prod2D,prod3D,prod4D];

prod1E : Product := new Product("Croissant", 4, "Croissa Simples", 4);
prod2E : Product := new Product("Lanche", 11, "Lanche Misto", 6);
prod3E : Product := new Product("Tapioca", 2, "Sobremesa", 5);
prod4E : Product := new Product("Mini Francesinha", 9, "Francesinha tapa", 11);
prodsE : seq of Product := [prod1E,prod2E,prod3E,prod4E];

prod1F : Product := new Product("Bolo Chocolate", 5, "O melhor bolo do mundo", 5);
prod2F : Product := new Product("Dolce Bolinho", 12, "Especialidade da Casa", 10);
prod3F : Product := new Product("Camarao", 21, "Camarao lindo", 13);
prod4F : Product := new Product("Tosta Alentejana", 91, "Tosta Alentejana com
oregaos", 6);
prodsF : seq of Product := [prod1F,prod2F,prod3F,prod4F];

-- Stores
store1: Store := new Store("store1", "candies", "Candies Store", "10-15min", 1.5,
prodsA);
store2: Store := new Store("store2", "pizza", "Papa Pizza", "10-30min", 5,
prodsB);
store3: Store := new Store("store3", "sushi", "Sushi For Life", "10-15min", 9,
prodsC);
store4: Store := new Store("store4", "pharmacy", "Pharmacy Feupy", "10-40min", 20,
prodsD);
store5: Store := new Store("store5", "snacks", "Divino Rosa", "6-12min", 6,
prodsE);
store6: Store := new Store("store6", "snacks", "Divino Rosalino", "3-6min", 7,
prodsF);

-- Orders
order1: Order := new Order(user1);

inv stores <> [];
inv couriers <> [];

operations

public Execution: () ==> Execution
Execution() == (
stores := []; products := []; couriers := []; users := []; orders := [];
stores := stores ^ [store1,store2,store3,store4,store5,store6];
products := products ^ [prod1A, prod2A, prod3A];
couriers := couriers ^ [courier1];
refOrders := {0 |-> -1 };
users := users ^ [user0,user1,user2];
categories :=  {"pizza", "snacks", "candies", "pharmacy", "sushi"};
IO`println("Welcome to Feup-Glovo App! Made by MFES Brothers.");
IO`println("Here is what you can do:");
IO`println("Add a User: print t.addUser(email, address, street, zip-code,
country)");
IO`println("Add a Courier: print t.addCourier(cardID,salary)");
IO`println("Create an Order : print t.createOrder(name@mail.com)");
IO`println("Add Product to Order : print
t.addProductToOrder(name@mail.com,OrderID,storeName,productIndex,numberUnits)");
IO`println("Conclude your order: print t.concludeOrder(name@mail.com,OrderID)");
IO`println("Assign a Courier to your order: print
t.assignCourier(courier@glovo.com,OrderID)");
IO`println("Perform an Order : print t.performOrder(courier@glovo.com,OrderID)");
IO`println("Set Person Info: print t.user1(name, gender, age)");
IO`println("List Categories : print t.listCategories()");
IO`println("List Stores by Category : print t.listStores(CategoryName)");
IO`println("List Stores by DeliveryCost : print
t.listStoresWithDeliveryCost(CategoryName)");
IO`println("List Products of Store : print t.listStoreProducts(StoreName)");
IO`println("List Orders : print t.listOrders(request) -> request can be all or
UserEmail");
IO`println("List Delivered Orders : print t.listOrdersDelivered(request) ->
request can be all or UserEmail");
IO`println("List Users : print t.listUsers()");
IO`println("List Couriers : print t.listCouriers()");
IO`println("Check earned money from a store: print
t.earnedMoneyFromStore(storeName)");
return self
)
post RESULT = self;

public addUser: string * string * string * string * string ==> string
addUser(email, address, street, zip_code, country) == (
dcl newAddress : User`Address := mk_User`Address(address,street,zip_code,country);
dcl newUser : User := new User(email, newAddress);
users := users ^ [newUser];
return "Result: User added to Users list.";
)
pre not exists user in set elems users & user.getEmail()= email;

public addCourier: string * int ==> string
addCourier(idCard, salary) == (
dcl newCourier : Courier := new Courier(idCard, salary);
couriers := couriers ^ [newCourier];
return "Result: Courier added to Couriers list.";
)
pre not exists courier in set elems couriers & courier.getCardID()= idCard
post exists courier in set elems couriers & courier.getCardID()= idCard;

public increaseProductStock: string * int * int ==> string
increaseProductStock(storeName, productIndex, unitsToAdd) == (
for store in stores do (
       if store.getName()=storeName then (
          dcl product : Product;
          product := store.getProduct(productIndex);
              product.addStock(unitsToAdd);
              IO`print("Product: "); IO`println(product.getName());
              IO`print("Added Units: "); IO`println(unitsToAdd);
              IO`print("Current Stock: "); IO`println(product.getStock());
              return "Result: success";
              );
);
return "No store/product found.";
)
pre unitsToAdd > 0;
```

```
public performOrder: string * int ==> string
performOrder(courierID, orderID) == (
 dcl auxCourier : Courier ;
 dcl auxReturn: int ;
 for courier in couriers do  (if(courier.getCardID()=courierID) then auxCourier :=
courier; );
 for order in orders do (
       if order.getId() = orderID then auxReturn := order.deliverOrder(courierID);
       );
IO`print("Order: "); IO`println(orderID);
IO`println("Order State: DELIVERED");
IO`println("By Courier: " ^ courierID);
return "Result: success";
)
pre (orderID in set dom refOrders
 and exists1 order in set elems orders & order.getId() = orderID
 and exists1 courier in set elems couriers & courier.getCardID() = courierID
 and order.getState() = <ORDER_CONCLUDED>
 and order.getCourier().getCardID() = courierID
 )
post exists order in set elems orders & (order.getId() = orderID and
order.getState() = <DELIVERED>);


public assignCourier: string * int ==> string
assignCourier(courierID, orderID) == (
 dcl auxCourier : Courier ;
 dcl auxReturn: int ;
 for courier in couriers do  (if(courier.getCardID()=courierID) then auxCourier :=
courier; );
 for order in orders do (
       if order.getId() = orderID then auxReturn:=order.setCourier(auxCourier);
       );
IO`print("Order: "); IO`println(orderID);
IO`print("Assigned Courier: "); IO`println(courierID);
return "Result: success";
)
pre exists1 order in set elems orders & (order.getId() = orderID and
(order.courier = nil or order.getCourier().getCardID() <> courierID) and
order.getState() = <ORDER_CONCLUDED>)
 and exists1 courier in set elems couriers & courier.getCardID() = courierID
post exists1 order in set elems orders & (order.getId() = orderID and
order.getCourier().getCardID() = courierID);


public createOrder: string ==> int
createOrder(email) == (
for user in users do (
             if user.getEmail() = email then
                   (
                   dcl newOrder : Order := new Order(user);
                   orders := orders ^ [newOrder];
                   refOrders := refOrders ++ {newOrder.getId() |-> len orders };
                   IO`println("New Order for " ^ user.getEmail());
                   IO`print("Order ID "); IO`println(newOrder.getId());
                   IO`println("Please add Products");
```

```
                        return newOrder.getId();
                        )
                );
        return -1;
)
pre exists user in set elems users & user.getEmail() = email;


public cancelOrder: int ==> string
cancelOrder(orderId) == (

dcl newRefOrders : map int to int;
dcl newOrders : seq of Order;
newOrders := [];
newRefOrders := {0 |-> -1 };

for order in orders do (
                if(order.getId() <> orderId) then (
                newOrders := newOrders^[order];
                newRefOrders := newRefOrders ++ {order.getId() |-> len newOrders };
                );
        );

orders := newOrders;
refOrders := newRefOrders;
return "Success: Order Canceled";
)
pre exists order in set elems orders & order.getId() = orderId;


public cloneProd: Product * nat * Store ==> Product
cloneProd(refProd, numberUnits, storeRef) == (
dcl auxRet : int ;
dcl newProd : Product := new Product(refProd.getName(), refProd.getPrice(),
refProd.getDescription(), numberUnits);
auxRet := newProd.setStore(storeRef);
return newProd;
)
post refProd.getName() = RESULT.getName() and refProd.getPrice() =
RESULT.getPrice() and refProd.getDescription() = RESULT.getDescription();


public addProductToOrder: string * int * string * int * nat ==> string
addProductToOrder(email, orderId, storeName, productIndex, numberUnits) == (
for store in stores do (
 if store.getName() = storeName then
 (
        dcl auxRet : int ;
        dcl refProd : Product := store.getProduct(productIndex);
        dcl newProd : Product := cloneProd(refProd, numberUnits, store);
        dcl newStock : nat := refProd.removeStock(numberUnits);
        for order in orders do (
                if order.getId() = orderId then
                        (
                        if order.getUser().getEmail() = email then (auxRet :=
order.addProduct(newProd););
                        IO`println("Product(s) added. Current Order:");
```

```
                    for prod in order.getProducts() do (IO`print(prod.getName() ^
". Quantity: "); IO`println(prod.getStock()));
                    return "Result: success";
                    )
             );
 ) );
      return "error";
)
pre exists1 order in set elems orders & (order.getId() = orderId and
order.getState() = <INIT>)
      and products(productIndex).getStock() >= numberUnits
      and exists user in set elems users & user.getEmail() = email
      and exists store in set elems stores & store.getName() = storeName
      and products(productIndex) in set elems products;


public removeProductFromOrder: int  * int  ==> string
removeProductFromOrder(orderId, productIndex) == (
dcl refOrder : Order ;
dcl numberUnits : nat ;
dcl refProduct : Product ;
dcl productName : string ;
dcl orderProducts : seq of Product ;
dcl newProdsList : seq of Product ;
newProdsList := [];

      for order in orders do (
             if order.getId() = orderId then
                    (
                    refOrder  := order;
                    orderProducts  := order.getProducts();
                    productName  := order.products(productIndex).getName();
                    refProduct  := order.products(productIndex);
                    numberUnits  := products(productIndex).getStock();
                    )
             );

      for prod in orderProducts do (
             if prod.getName() <> productName then
                    (
                          newProdsList := newProdsList ^ [prod];
                          refOrder.products := newProdsList;
                    )
             );

      for store in stores do (
             if store.getName() = refProduct.getStore().getName() then
                    (
                          for currProd in store.getProducts() do (if
currProd.getName() = productName then currProd.addStock(numberUnits));
                    )
             );

return "Result: success";
)
pre products(productIndex) in set elems products
      and exists order in set elems orders & order.getId() = orderId;
```

```
public concludeOrder: string * int ==> string
concludeOrder(email, orderId) == (
for order in orders do (
             if order.getId() = orderId then
                    (
                    dcl auxRet : int ;
                    if order.getUser().getEmail() = email then (auxRet :=
order.finishOrder());
                    IO`println("Order Concluded. Bill:");
                    for prod in order.getProducts() do (IO`print(prod.getName() ^
". Quantity: "); IO`print(prod.getStock()); IO`print(" Unit Price: ");
IO`println(prod.getPrice()));
                    IO`print("Total: "); IO`println(order.getTotalCost());
                    return "Result: success";
                    )
             );

      return "error";
)
pre exists1 order in set elems orders & (order.getId() = orderId and
order.getState() = <INIT>)
      and exists1 user in set elems users & user.getEmail() = email
post exists order in set elems orders & (order.getId() = orderId and
order.getState() = <ORDER_CONCLUDED>);



             /***** LISTING OPERATORS ******/
-- "all" for all orders and "email" for user orders
public listOrdersDelivered: string  ==> string
listOrdersDelivered(request) ==
(
if request = "all" then for order in orders do
(
 if order.getState() = <DELIVERED> then
(
IO`print("Order ID: "); IO`println(order.getId());
IO`print("State of Order: "); IO`println(order.getState());
IO`print("Client: "); IO`println(order.getUser().getEmail());
IO`println("Products (if any): ");
if len order.getProducts() > 0 then
(for product in order.getProducts() do (IO`print("Product: " ^ product.getName() ^
". Units: "); IO`println(product.getStock()));););
)
);

if request <> "all" then for order in orders do
(
 if order.getState() = <DELIVERED> then
(
if (order.getUser().getEmail() = request) then (
IO`print("Order ID: "); IO`println(order.getId());
IO`print("State of Order: "); IO`println(order.getState());
IO`print("Client: "); IO`println(order.getUser().getEmail());
IO`println("Products (if any): ");
if len order.getProducts() > 0 then
(for product in order.getProducts() do (IO`print("Product: " ^ product.getName() ^
". Units: "); IO`println(product.getStock()););););
)
);

IO`println("\n");
return "Result: success";
)
pre orders <> [];


-- "all" for all orders and "email" for user orders
public listOrders: string  ==> string
listOrders(request) ==
(
if request = "all" then for order in orders do
(
IO`print("Order ID: "); IO`println(order.getId());
IO`print("State of Order: "); IO`println(order.getState());
IO`print("Client: "); IO`println(order.getUser().getEmail());
IO`println("Products (if any): ");
if len order.getProducts() > 0 then
(for product in order.getProducts() do (IO`print("Product: " ^ product.getName() ^
". Units: "); IO`println(product.getStock())););););
);

if request <> "all" then for order in orders do
(
if (order.getUser().getEmail() = request) then (
IO`print("Order ID: "); IO`println(order.getId());
IO`print("State of Order: "); IO`println(order.getState());
IO`print("Client: "); IO`println(order.getUser().getEmail());
IO`println("Products (if any): ");
if len order.getProducts() > 0 then
(for product in order.getProducts() do (IO`print("Product: " ^ product.getName() ^
". Units: "); IO`println(product.getStock()););););););
);

IO`println("\n");
return "Result: success";
)
pre orders <> [];

public listOrdersByCourier: string  ==> string
listOrdersByCourier(cardId) ==
(
for order in orders do (if order.getCourier().getCardID() = cardId then
        (
        (IO`print("Order: "); IO`println(order.getId()););
        (IO`print("State: "); IO`println(order.getState()););
        (IO`print("Client: "); IO`println(order.getUser()););
        (IO`print("Total Cost: "); IO`println(order.getTotalCost()););
        IO`println("\n");
        )
);

IO`println("\n");
return "Result: success";
```

```
)
pre orders <> [];


public listCategories: () ==> string
listCategories() ==
(
for all category in set categories do (
IO`println("Category: " ^ category);
);
return "Result: success";
)
pre categories <> {};

public listStores: string ==> string
listStores(categoryName) ==
(
IO`println("Stores with Category: " ^ categoryName);
for store in stores do
        (
        if store.getCategory() = categoryName then (IO`println("Store: " ^
store.getName() ^ ". Description: " ^ store.getDescription()));
        );
return "Result: success";
)
pre categoryName in set categories and stores <> [];


public listStoreProducts: string ==> string
listStoreProducts(storeName) ==
(
dcl auxProducts : seq of Product ;
dcl countProds : int := 1;

IO`println("Products of: " ^ storeName);
        for store in stores do (
                if store.getName() = storeName then auxProducts :=
store.getProducts();
        );

        for auxProd in auxProducts do (
                IO`print(countProds); IO`println(" Name: " ^ auxProd.getName() ^ ".
Description: " ^ auxProd.getDescription());
                IO`print("Unit Price: " ); IO`println(auxProd.getPrice());
                IO`print("Stock: " ); IO`println(auxProd.getStock());
                countProds:=countProds+1;
        );

return "Result: success";
)
pre exists1 store in set elems stores & store.getName() = storeName and stores <>
[];

public listCouriers: () ==> string
listCouriers() ==
(
for courier in couriers do (IO`println("Courier ID: " ^ courier.getCardID()););
return "Result: success";
```

```
)
pre couriers <> [];


public listUsers: () ==> string
listUsers() ==
(
for user in users do (IO`println("User Email: " ^ user.getEmail()););
return "Result: success";
)
pre users <> [];


public listStoresWithDeliveryCost: real ==> string
listStoresWithDeliveryCost(deliveryCost) ==
(
for store in stores do
      (
      if store.getDeliveryCost() <= deliveryCost then (IO`println("Store: " ^
store.getName() ^ ". Description: " ^ store.getDescription()));
      );
return "Result: success";
)
pre stores <> [];


public earnedMoneyFromStore: (string) ==> real
earnedMoneyFromStore(storeName) ==
(
for store in stores do(
      if store.getName() = storeName then(
             for all order in set elems orders do
             (
                    if order.getState() = <DELIVERED> then(
                    dcl aux : map Product to Product := {x|-> y | x in set elems
order.getProducts() , y in set elems store.getProducts() & x.getName() =
y.getName()};
                    dcl commonProducts : set of Product := dom aux;
                    dcl earned : real := 0;
                    for all product in set commonProducts do(
                                product.getPrice() *
product.getStock();
                     );
                     earned := earned -order.getCourier().getSalary();
                     return earned;
                     )
             );
             );
      );
      return 0;
)
pre orders <> [];

public editProfile: string * string * Gender * nat ==> string
editProfile(email, name, gender, age) == (

for user in users do (
      if user.getEmail() = email then user.setPersonInfo(name,gender,age)
);
return "Result: success";

)

pre exists user in set elems users & user.getEmail()= email;

             /***** GET OPERATORS ******/

public getUsers: () ==> seq of User
getUsers() == return users
post RESULT = users;

public getCouriers: () ==> seq of Courier
getCouriers() == return couriers
post RESULT = couriers;

public getOrders: () ==> seq of Order
getOrders() == return orders
post RESULT = orders;

public getrefOrders: () ==> map int to int
getrefOrders() == return refOrders
post RESULT = refOrders;

end Execution
```

3.3 Class Order

```
class Order

types
public string = seq of char;
public State = <INIT> | <ORDER_CONCLUDED> | <DELIVERED> ;

instance variables
private orderID:  int;
private totalCost : real;
public products : seq of Product;
public courier : [Courier] :=nil;
private user : User;
private state: State;

operations
public Order: User ==> Order
Order(user_reference) == (
products := [];
state := <INIT>;
orderID := MATH`rand(1000) + MATH`rand(1000);
user := user_reference;
totalCost := 0;
return self
);

public addProduct: Product ==> int
addProduct(newProd) == (
for prod in products do
(
if prod.getName() = newProd.getName() then (prod.addStock(newProd.getStock());
return 1;)
);
products := products ^ [newProd];
return 1;
)
pre newProd.getStock() > 0;

public deliverOrder: string ==> int
deliverOrder(courierID) == (
if courier.getCardID() = courierID then (state := <DELIVERED>;)
else return -1;
return 1;
)
pre state = <ORDER_CONCLUDED>;

public finishOrder: () ==> int
finishOrder() == (
for prod in products do(totalCost := totalCost + prod.getPrice()*prod.getStock());
state := <ORDER_CONCLUDED>;
return 1;
)
pre state = <INIT>
post totalCost > 0 and state = <ORDER_CONCLUDED>;

public setCourier: Courier ==> int
setCourier(assignedCourier) == (courier:=assignedCourier; return 1;)
pre courier <> assignedCourier
post courier=assignedCourier;

pure public getId: () ==> int
getId() == return orderID
post RESULT = orderID;

pure public getTotalCost: () ==> real
getTotalCost() == return totalCost
post RESULT = totalCost;

pure public getProducts: () ==> seq of Product
getProducts() == return products
post RESULT = products;

pure public getCourier: () ==> Courier
getCourier() == return courier
post RESULT = courier;

pure public getUser: () ==> User
getUser() == return user
post RESULT = user;

pure public getState: () ==> State
getState() == return state
post RESULT = state;

public setState: State ==> State
setState(newState) == (state:=newState; return state;)
post state = newState and RESULT = state;
```

```
end Order
```

3.4 Class Person

```
class Person

types
public string = seq of char;
public Gender = <Male> | <Female> | <Undefined>;

instance variables
protected name : string := "";
protected gender: Gender:= <Undefined>;
protected age : nat := 18;

operations
public setPersonInfo: string * Gender * nat ==> ()
setPersonInfo(na, ge, ag) == is subclass responsibility
pre ag >= 18
post age = ag;

pure public getName: () ==> string
getName() == return name
post RESULT = name;

pure public getGender: () ==> Gender
getGender() == return gender
post RESULT = gender;

pure public getAge: () ==> nat
getAge() == return age
post RESULT = age;

end Person
```

3.5 Class Product

```
class Product

types
public string = seq of char;

instance variables
private name : string;
private price : real := 0.1;
private description : string;
private quantity: nat;
private store: Store;
private orders : set of Order := {};

inv price >= 0.0 and quantity > 0;

operations
```

```
public Product: string * real * string * nat ==> Product
Product(na, pri, des, qty) == (name := na; price := pri; description:=des;
quantity := qty; return self)
pre price > 0;

public addStock: nat ==> ()
addStock(numberUnits) == (
quantity := quantity + numberUnits;
)
pre numberUnits >= 0;

public removeStock: nat ==> nat
removeStock(numberUnits) == (
quantity := quantity - numberUnits;
return quantity;
)
pre numberUnits > 0
post RESULT = quantity;

pure public getName: () ==> string
getName() == return name
post RESULT = name;

pure public getPrice: () ==> real
getPrice() == return price
post RESULT = price;

pure public getDescription: () ==> string
getDescription() == return description
post RESULT = description;

pure public getStock: () ==> nat
getStock() == return quantity
post RESULT = quantity;

pure public getStore: () ==> Store
getStore() == return store
post RESULT = store;

public setStore: Store ==> int
setStore(storeOfProduct) == (store := storeOfProduct; return 1;);


functions

end Product
```

3.6 Class Store

```
class Store
types
public string = seq of char;

instance variables
private name : string;
private category : string;
private description : string;
private estimateTime : string;
private deliveryCost : real;
private products : seq of Product;

inv estimateTime(2) = '-' or estimateTime(3) = '-' ;

operations
public Store: string * string * string * string * real * seq of Product ==> Store
Store(na, ca, des, time, cost, prods) == (name := na; category:=ca; description :=
des; estimateTime:=time; deliveryCost := cost; products := prods; return self)
pre cost > 0
post deliveryCost = cost;

public addProduct: Product ==> nat
addProduct(p) == (
products := products ^ [p];
return len products;
)
pre p not in set elems products
post p in set elems products;

pure public getProduct: int ==> Product
getProduct(productIndex) == return products(productIndex)
post RESULT = products(productIndex);

pure public getProducts: () ==> seq of Product
getProducts() == return products
post RESULT = products;

pure public getName: () ==> string
getName() == return name
post RESULT = name;

pure public getCategory: () ==> string
getCategory() == return category
post RESULT = category;

pure public getDescription: () ==> string
getDescription() == return description
post RESULT = description;

pure public getEstimateTime: () ==> string
getEstimateTime() == return estimateTime
post RESULT = estimateTime;

pure public getDeliveryCost: () ==> real
getDeliveryCost() == return deliveryCost
post RESULT = deliveryCost;

end Store
```

3.7 Class User

```
class User is subclass of Person
types
public string = seq of char;
public Address :: address : string
```

```
                                                          street : string
                                                      zip_code: string
                                                      country : string;

values
instance variables
private email : string;
private address: Address;

operations
public User: string * User`Address ==> User
User(em, addr) == (email:=em; address:=addr; return self);

pure public getEmail: () ==> string
getEmail() == return email
post RESULT = email;

public setPersonInfo: string * Gender * nat ==> ()
setPersonInfo(na, ge, ag) == (name := na; gender := ge; age:=ag;)
post name = na and gender = ge and age = ag;


end User
```

4. Model validation

4.1 Class MyTestCase

```
class MyTestCase
/*
 Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
*/
operations
-- Simulates assertion checking by reducing it to pre-condition checking.
-- If 'arg' does not hold, a pre-condition violation will be signaled.
protected assertTrue: bool ==> ()
assertTrue(arg) ==
return
pre arg;

-- Simulates assertion checking by reducing it to post-condition checking.
-- If values are not equal, prints a message in the console and generates
-- a post-conditions violation.
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

end MyTestCase
```

4.2 Class TestGlovo

```
class TestGlovo is subclass of MyTestCase
/*
 Contains the test cases for the Glovo app.
 Illustrates a scenario-based testing approach.
*/
types
public string = seq of char;
public State = <INIT> | <ORDER_CONCLUDED> | <DELIVERED> ;

values
address : User`Address = mk_User`Address("as","as","as","Portugal");

instance variables
       exec : Execution := new Execution();
       orderId: int := 0;
       order: Order ;

operations
/***** USE CASE SCENARIOS ******/

 private test_CreateOrder: () ==> ()
 test_CreateOrder() ==
 (
 dcl currentRefOrders: map int to int;
 dcl orderIndex: nat;
 dcl auxState: State;

 dcl currentNumOrders: int :=  len exec.getOrders();
 orderId := exec.createOrder("blankuser");

 currentRefOrders :=  exec.getrefOrders();
 orderIndex := currentRefOrders(orderId);
 order :=  exec.getOrders()(orderIndex);
  auxState := order.setState(<INIT>);

 assertEqual(currentNumOrders+1,len exec.getOrders());
 )
 pre exists user in set elems exec.users & user.getEmail() = "blankuser"
 post exists userOrder in set elems exec.orders & (userOrder.getId() = orderId and
order.getState() = <INIT>);

  private test_cancelOrder: () ==> ()
 test_cancelOrder() ==
 (
 dcl currentRefOrders: map int to int;
 dcl orderIndex: nat;
 dcl auxRet: string;

 dcl currentNumOrders: int :=  len exec.getOrders();
 orderId := exec.createOrder("blankuser");
 currentRefOrders  :=  exec.getrefOrders();
 orderIndex  := currentRefOrders(orderId);
 order  :=  exec.getOrders()(orderIndex);

 assertEqual(currentNumOrders+1,len exec.getOrders());

 auxRet  := exec.cancelOrder(order.getId());

 currentNumOrders  := len exec.getOrders();

 assertEqual(currentNumOrders,1);
 )
 pre exists testOrder in set elems exec.orders & testOrder.getId() = orderId;


  private test_AddProductToOrder: () ==> ()
 test_AddProductToOrder() ==
 (

 dcl numberProducts: int :=  len order.getProducts();

 dcl auxRet : string := exec.addProductToOrder("blankuser", orderId, "store1", 1,
1);
 dcl newNumberProducts: int :=  len order.getProducts();
 assertEqual(newNumberProducts,numberProducts+1);

 numberProducts  :=  len order.getProducts();

 auxRet  := exec.addProductToOrder("blankuser", orderId, "store1", 1, 1);
 newNumberProducts  :=  len order.getProducts();

 )
 pre exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and
userOrder.getState() = <INIT>)
       and exists user in set elems exec.users & user.getEmail() = "blankuser"
       and exists store in set elems exec.stores & store.getName() = "store1"
       and exec.products(1) in set elems exec.products;

         private test_removeProductFromOrder: () ==> ()
 test_removeProductFromOrder() ==
 (


 dcl numberProducts: int :=  len order.getProducts();

 dcl auxRet : string := exec.removeProductFromOrder(orderId, 1);
 dcl newNumberProducts: int :=  len order.getProducts();
 IO`print("numberProducts: "); IO`print(numberProducts);

 IO`print("newNumberProducts: "); IO`print(newNumberProducts);
 assertEqual(auxRet,"Result: success");

 )
 pre exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and
userOrder.getState() = <INIT>)
       and exists user in set elems exec.users & user.getEmail() = "blankuser"
       and exists store in set elems exec.stores & store.getName() = "store1"
```

```
      and exec.products(1) in set elems exec.products;

  private test_concludeOrder: () ==> ()
 test_concludeOrder() ==
 (
      dcl auxRet : string;
      assertTrue(order.getState() = <INIT>);

      auxRet := exec.concludeOrder("blankuser", orderId);

      assertTrue(len order.getProducts() > 0);
      assertTrue(order.getState() = <ORDER_CONCLUDED>);

 )
 pre exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and
userOrder.getState() = <INIT>)
      and exists1 user in set elems exec.users & user.getEmail() = "blankuser"
post exists userOrder in set elems exec.orders & (userOrder.getId() = orderId and
userOrder.getState() = <ORDER_CONCLUDED>);

  private test_assignCourier: string ==> ()
 test_assignCourier(cardId) ==
 (
      dcl auxRet : string;

      auxRet := exec.assignCourier(cardId,orderId);
      assertTrue(order.getCourier().getCardID() = cardId);

 )
pre exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and
(order.courier = nil or userOrder.getCourier().getCardID() <> cardId) and
userOrder.getState() = <ORDER_CONCLUDED>)
 and exists1 courier in set elems exec.couriers & courier.getCardID() = cardId
post exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and
userOrder.getCourier().getCardID() = cardId);

   private test_performOrder: string ==> ()
 test_performOrder(cardId) ==
 (
      dcl auxRet : string;
      dcl totalMoney : real;

      assertTrue(order.getCourier().getCardID() = cardId);
      assertTrue(order.getState() = <ORDER_CONCLUDED>);

      auxRet := exec.performOrder(cardId,orderId);
      totalMoney := exec.earnedMoneyFromStore("store1");

      assertTrue(order.getState() = <DELIVERED>);

 )
 pre (orderId in set dom exec.refOrders
 and exists1 userOrder in set elems exec.orders & userOrder.getId() = orderId
 and exists1 courier in set elems exec.couriers & courier.getCardID() = cardId
 );

   private test_addUser: string ==> ()
 test_addUser(email) ==
 (
        dcl auxRet : string;
        dcl hasUser : bool := false;
        for user in exec.getUsers() do ( if user.getEmail() = email then hasUser :=
true);

        assertTrue(hasUser = false);

        auxRet := exec.addUser(email, "Number", "Liberty", "3310-119",
"Feuplandia");

        for user in exec.getUsers() do ( if user.getEmail() = email then hasUser :=
true);

        assertTrue(hasUser = true);

 );

     private test_setCourierInfo: string ==> ()
 test_setCourierInfo(cardID) ==
 (
        dcl foundCourier : Courier;
        for courier in exec.getCouriers() do ( if courier.getCardID() = cardID then
foundCourier := courier);

   foundCourier.setPersonInfo("name", <Male>, 25);

        assertTrue(foundCourier.getGender() = <Male>);


 );
   private test_addCourier: string ==> ()
 test_addCourier(cardId) ==
 (
        dcl auxRet : string;
        dcl hasCourier : bool := false;
        for courier in exec.getCouriers() do ( if courier.getCardID() = cardId then
hasCourier := true);

        assertTrue(hasCourier = false);

        auxRet := exec.addCourier(cardId, 1000);

        for courier in exec.getCouriers() do ( if courier.getCardID() = cardId then
hasCourier := true);

        assertTrue(hasCourier = true);


 );

    private test_editProfile: string ==> ()
 test_editProfile(userEmail) ==
 (
        dcl auxRet : string;
        dcl refUser : User;

        auxRet := exec.editProfile(userEmail, "newName", <Female>, 99);
```

```
       for user in exec.getUsers() do ( if user.getEmail() = userEmail then
refUser := user; );

       assertEqual(refUser.getName(), "newName");
       assertEqual(refUser.getGender(), <Female>);
       assertEqual(refUser.getAge(), 99);

 )
 pre exists user in set elems exec.users & user.getEmail() = userEmail;

    private test_increaseProductStock: () ==> ()
 test_increaseProductStock() ==
 (
       dcl auxRet : string;
       dcl prodStock : int;
       dcl refStores : seq of Store;
       dcl estimateTime : string;

       refStores := exec.stores;
       estimateTime := exec.stores(1).getEstimateTime();
       prodStock := refStores(1).getProducts()(1).getStock();

       auxRet := exec.increaseProductStock(refStores(1).getName(), 1, 10);

       assertEqual(refStores(1).getProducts()(1).getStock(), prodStock+10);

 );


    private test_lists: () ==> ()
 test_lists() ==
 (
 dcl auxRet : string;

 auxRet := exec.listCategories();
 auxRet := exec.listStores("pizza");
 auxRet := exec.listStoreProducts("store1");
 auxRet := exec.listOrders("all");
 auxRet := exec.listOrders("blankuser");
 auxRet := exec.listOrdersDelivered("all");
 auxRet := exec.listOrdersDelivered("blankuser");
 auxRet := exec.listOrdersDelivered("blankuser");
 auxRet := exec.listUsers();
 auxRet := exec.listCouriers();
 auxRet := exec.listStoresWithDeliveryCost(3);
 auxRet := exec.listOrdersByCourier("maria@glovo.com");


 );

      private test_createAddProductToStore: () ==> ()
 test_createAddProductToStore() ==
 (
  dcl prod1A : Product := new Product("Pipoca", 2.5, "Bem boa", 5);
       dcl store1: Store := new Store("testStore", "candies", "Candies Store",
"10-15min", 1.5, [prod1A] );
```

```
 dcl prod1Test : Product := new Product("test", 1, "test 2", 6);

      dcl productsNum : nat := store1.addProduct(prod1Test);
      assertTrue(store1.getName() = "testStore");
      assertTrue(len store1.getProducts() = productsNum);


);


public static main: () ==> ()
main() ==
(
dcl testGlovo: TestGlovo :=  new TestGlovo();

      testGlovo.test_increaseProductStock();
      testGlovo.test_CreateOrder();
      testGlovo.test_AddProductToOrder();
      testGlovo.test_removeProductFromOrder();
      testGlovo.test_concludeOrder();
      testGlovo.test_assignCourier("maria@glovo.com");
      testGlovo.test_performOrder("maria@glovo.com");
      testGlovo.test_cancelOrder();
      testGlovo.test_addUser("newUser@mail.com");
      testGlovo.test_addCourier("newCourier@glovo.com");
      testGlovo.test_setCourierInfo("newCourier@glovo.com");
      testGlovo.test_editProfile("blankuser");
      testGlovo.test_createAddProductToStore();
      testGlovo.test_lists();

);
traces
-- test cases will be generated in possible combinations
testaddUsers :
(let email in set {"email1@hotmail.com","email2@hotmail.com"} in
exec.addUser(email,"address","street","1440-100","PT")){1,2};

 testCreateOrders :
(let email in set {"email1@hotmail.com","email2@hotmail.com"} in
exec.createOrder(email)){1,5};

end TestGlovo
```

5. Model verification

5.1 Example of domain verification

One of the proof obligations generated by Overture is:

No.	**PO Name**	**Type**
88	Store`addProduct	sequence domain verification

The code under analysis(with the relevant sequence application underlined) is:

```
public addProduct: Product ==> nat
addProduct(p) == (
        products := products ^ [p];
        return len products;
)
pre p not in set elems products
post p in set elems products;
```

Verifying the Proof Obligation View, the following PO is returned:

```
(forall p:Product & ((p not in set (elems products)) => (p in set (elems (products ^ [p])))))
```

In this case, since a new Product is being added to a store, the Product musn't be duplicated in the seq of Product, *products*. Thus, the precondition is used to assure that the argument *p* isn't in the seq of Products.

5.2 Example of invariant verification

Another proof obligation generated by Overture is:

No.	**PO Name**	**Type**
2	Product`removeStock	state invariant holds

The code under analysis (with the relevant state changes underlined) is:

```
public removeStock: nat ==> nat
removeStock(numberUnits) == (
        quantity := quantity - numberUnits;
        return quantity;
)
pre numberUnits > 0
post RESULT = quantity;
```

The relevant invariant under analysis is:

inv price >= 0 **or quantity** > 0;

Verifying the Proof Obligation View, the following PO is returned:

(**forall** numberUnits:**nat** & ((numberUnits > 0) => ((quantity - numberUnits) >= 0)))

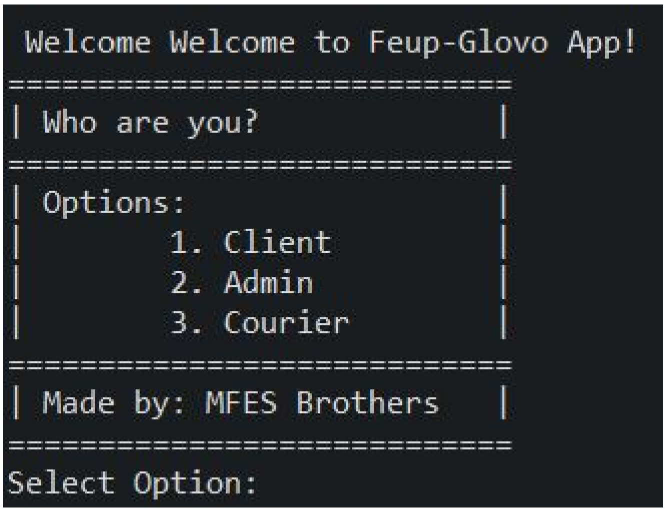
When executing the first line of codee, the value of quantity is subtracted by numberUnits.

quantity := quantity - numberUnits;

After executing the subtraction, the invariant verifies if it holds and throwing an error if if doesn't.

6. Code Generation

In the VDM Explorer window, with the mouse over the project folder, we selected the option *Code Generation-> Generate Java (Configurarion based).* A Java project with source files, libraries and settings was generated in the folder "../generated/java".
Afterwards, the project was imported to Eclipse.
With the project on Eclipse, the AppInit.java class was created and the User Interface was implemented there.



```
 Welcome Welcome to Feup-Glovo App!
============================
| Who are you?              |
============================
| Options:                  |
|        1. Client          |
|        2. Admin           |
|        3. Courier         |
============================
| Made by: MFES Brothers    |
============================
Select Option:
```

7. Conclusions

We consider the final result of the work quite positive. The created model fulfills all requirements.

Although the project has enough features, we would like, as future work, to complete it with extra features that are inherent to the *Glovo* application.

The project took about a week and a half to be implemented.

Both members of the group worked equally, i.e, 50%.

8. References

1. MFES-VDM++.ppt, Ana Paiva
2. VDM-10 Language Manual, http://overturetool.org/documentation/manuals.html
3. Overture VDM-10 Tool Support: User Guide, http://overturetool.org/documentation/manuals.html

MFES Glovo Coverage

Bernardo Leite, Vtor Magalhes

January 5, 2019

Contents

1 Courier

```
class Courier is subclass of Person

types
public string = seq of char;

instance variables
private cardID : string;
private salary : int;

inv salary > 0;

operations


public Courier: string * int ==> Courier
Courier(idCard, sal) == (cardID := idCard; salary:=sal; return self);


public setPersonInfo: string * Gender * nat ==> ()
setPersonInfo(na, ge, ag) == (name := na; gender := ge; age:=ag;)
post name = na and gender = ge and age = ag;
```

```
pure public getCardID: () ==> string
getCardID() == return cardID
post RESULT = cardID;


pure public getSalary: () ==> int
getSalary() == return salary
post RESULT = salary;
end Courier
```

Function or operation	Line	Coverage	Calls
Courier	14	100.0%	41
getCardID	21	100.0%	245
getSalary	25	100.0%	13
setPersonInfo	17	100.0%	1
Courier.vdmpp		100.0%	300

2 Execution

```
class Execution

types
public string = seq of char;
public Gender = <Male> | <Female> | <Undefined>;

values
address1 : User`Address = mk_User`Address("as","as","as","as");
address2 : User`Address = mk_User`Address("as","bl","44","z9");

instance variables
public stores: seq of Store;
public products: seq of Product;
public couriers: seq of Courier;
public orders: seq of Order;
public refOrders: map int to int;
public users: seq of User;
public categories: set of string;

-- Users
user0 : User := new User("blankuser", address1);
user1 : User := new User("bernas@hotmail.com", address1);
user2 : User := new User("vitorino@hotmail.com", address2);

-- Couriers
courier1 : Courier := new Courier("maria@glovo.com", 3);

-- Products
prod1A : Product := new Product("Rabanada", 2.5, "Rabanada de Convento", 5);
prod2A : Product := new Product("Bolina", 1.5, "Docinho de Bolina", 10);
prod3A : Product := new Product("Bolo Rei", 3.99, "Bolo Rei sem frutos", 6);
prodsA : seq of Product := [prod1A,prod2A,prod3A];

prod1B : Product := new Product("Pizza Tropical", 10, "Pizza com ananas", 3);
prod2B : Product := new Product("Pizza Funghi", 7, "Pizza com cogumelos", 8);
prod3B : Product := new Product("Pizza Margherita", 6.5, "Pizza normal", 2);
```

```
prodsB : seq of Product := [prod1B,prod2B,prod3B];

prod1C : Product := new Product("Sashimi", 30, "Peixe Sashimi", 3);
prod2C : Product := new Product("Hosomaki", 41, "Peixe Hosomaki", 8);
prod3C : Product := new Product("Uramaki", 90, "Peixe com Arrroz", 2);
prod4C : Product := new Product("Hot Roll", 19, "Sushi Nuggets", 2);
prodsC : seq of Product := [prod1C,prod2C,prod3C,prod4C];

prod1D : Product := new Product("Penso R pido", 10, "Pensos", 7);
prod2D : Product := new Product("Ben-u-ron", 20, "Benuron 1000mg", 8);
prod3D : Product := new Product("Akiliever", 50, "Melhor creme para as m os", 18);
prod4D : Product := new Product("Nivea", 19, "Creme corporal", 9);
prodsD : seq of Product := [prod1D,prod2D,prod3D,prod4D];

prod1E : Product := new Product("Croissant", 4, "Croissa Simples", 4);
prod2E : Product := new Product("Lanche", 11, "Lanche Misto", 6);
prod3E : Product := new Product("Tapioca", 2, "Sobremesa", 5);
prod4E : Product := new Product("Mini Francesinha", 9, "Francesinha tapa", 11);
prodsE : seq of Product := [prod1E,prod2E,prod3E,prod4E];

prod1F : Product := new Product("Bolo Chocolate", 5, "O melhor bolo do mundo", 5);
prod2F : Product := new Product("Dolce Bolinho", 12, "Especialidade da Casa", 10);
prod3F : Product := new Product("Camarao", 21, "Camarao lindo", 13);
prod4F : Product := new Product("Tosta Alentejana", 91, "Tosta Alentejana com oregaos", 6);
prodsF : seq of Product := [prod1F,prod2F,prod3F,prod4F];

-- Stores
store1: Store := new Store("store1", "candies", "Candies Store", "10-15min", 1.5, prodsA);
store2: Store := new Store("store2", "pizza", "Papa Pizza", "10-30min", 5, prodsB);
store3: Store := new Store("store3", "sushi", "Sushi For Life", "10-15min", 9, prodsC);
store4: Store := new Store("store4", "pharmacy", "Pharmacy Feupy", "10-40min", 20, prodsD);
store5: Store := new Store("store5", "snacks", "Divino Rosa", "6-12min", 6, prodsE);
store6: Store := new Store("store6", "snacks", "Divino Rosalino", "3-6min", 7, prodsF);

-- Orders
order1: Order := new Order(user1);

inv stores <> [];
inv couriers <> [];

operations


public Execution: () ==> Execution
Execution() == (
stores := []; products := []; couriers := []; users := []; orders := [];
stores := stores ^ [store1,store2,store3,store4,store5,store6];
products := products ^ [prod1A, prod2A, prod3A];
couriers := couriers ^ [courier1];
refOrders := {0 |-> -1 };
users := users ^ [user0,user1,user2];
categories :=  {"pizza", "snacks", "candies", "pharmacy", "sushi"};
IO`println("Welcome to Feup-Glovo App! Made by MFES Brothers.");
IO`println("Here is what you can do:");
IO`println("Add a User: print t.addUser(email, address, street, zip-code, country)");
IO`println("Add a Courier: print t.addCourier(cardID,salary)");
IO`println("Create an Order : print t.createOrder(name@mail.com)");
IO`println("Add Product to Order : print t.addProductToOrder(name@mail.com,OrderID,storeName,
    productIndex,numberUnits)");
IO`println("Conclude your order: print t.concludeOrder(name@mail.com,OrderID)");
IO`println("Assign a Courier to your order: print t.assignCourier(courier@glovo.com,OrderID)");
IO`println("Perform an Order : print t.performOrder(courier@glovo.com,OrderID)");
IO`println("Set Person Info: print t.user1(name, gender, age)");
IO`println("List Categories : print t.listCategories()");
IO`println("List Stores by Category : print t.listStores(CategoryName)");
IO`println("List Stores by DeliveryCost : print t.listStoresWithDeliveryCost(CategoryName)");
IO`println("List Products of Store : print t.listStoreProducts(StoreName)");
IO`println("List Orders : print t.listOrders(request) -> request can be all or UserEmail");
IO`println("List Delivered Orders : print t.listOrdersDelivered(request) -> request can be all or
     UserEmail");
IO`println("List Users : print t.listUsers()");
IO`println("List Couriers : print t.listCouriers()");
IO`println("Check earned money from a store: print t.earnedMoneyFromStore(storeName)");
return self
)
post RESULT = self;


public addUser: string * string * string * string * string ==> string
addUser(email, address, street, zip_code, country) == (
dcl newAddress : User`Address := mk_User`Address(address,street,zip_code,country);
dcl newUser : User := new User(email, newAddress);
users := users ^ [newUser];
return "Result: User added to Users list.";
)
pre not exists user in set elems users & user.getEmail()= email;


public addCourier: string * int ==> string
addCourier(idCard, salary) == (
dcl newCourier : Courier := new Courier(idCard, salary);
couriers := couriers ^ [newCourier];
return "Result: Courier added to Couriers list.";
)
pre not exists courier in set elems couriers & courier.getCardID()= idCard
post exists courier in set elems couriers & courier.getCardID()= idCard;


public increaseProductStock: string * int * int ==> string
increaseProductStock(storeName, productIndex, unitsToAdd) == (
for store in stores do (
 if store.getName()=storeName then (
    dcl product : Product;
    product := store.getProduct(productIndex);
   product.addStock(unitsToAdd);
   IO`print("Product: "); IO`println(product.getName());
   IO`print("Added Units: "); IO`println(unitsToAdd);
   IO`print("Current Stock: "); IO`println(product.getStock());
   return "Result: success";
   );
);
return "No store/product found.";
)
pre unitsToAdd > 0;



public performOrder: string * int ==> string
performOrder(courierID, orderID) == (
 dcl auxCourier : Courier ;
 dcl auxReturn: int ;
 for courier in couriers do  (if(courier.getCardID()=courierID) then auxCourier := courier; );
 for order in orders do (
  if order.getId() = orderID then auxReturn := order.deliverOrder(courierID);
  );
IO`print("Order: "); IO`println(orderID);
IO`println("Order State: DELIVERED");
IO`println("By Courier: " ^ courierID);
return "Result: success";
)
```

```
pre (orderID in set dom refOrders
 and exists1 order in set elems orders & order.getId() = orderID
 and exists1 courier in set elems couriers & courier.getCardID() = courierID
 and order.getState() = <ORDER_CONCLUDED>
 and order.getCourier().getCardID() = courierID
 )
post exists order in set elems orders & (order.getId() = orderID and order.getState() = <
    DELIVERED>);



public assignCourier: string * int ==> string
assignCourier(courierID, orderID) == (
 dcl auxCourier : Courier ;
 dcl auxReturn: int ;
 for courier in couriers do  (if(courier.getCardID()=courierID) then auxCourier := courier; );
 for order in orders do (
  if order.getId() = orderID then auxReturn:=order.setCourier(auxCourier);
  );
IO`print("Order: "); IO`println(orderID);
IO`print("Assigned Courier: "); IO`println(courierID);
return "Result: success";
)
pre exists1 order in set elems orders & (order.getId() = orderID and (order.courier = nil or
    order.getCourier().getCardID() <> courierID) and order.getState() = <ORDER_CONCLUDED>)
 and exists1 courier in set elems couriers & courier.getCardID() = courierID
post exists1 order in set elems orders & (order.getId() = orderID and order.getCourier().
    getCardID() = courierID);



public createOrder: string ==> int
createOrder(email) == (
for user in users do (
  if user.getEmail() = email then
   (
   dcl newOrder : Order := new Order(user);
   orders := orders ^ [newOrder];
   refOrders := refOrders ++ {newOrder.getId() |-> len orders };
   IO`println("New Order for " ^ user.getEmail());
   IO`print("Order ID "); IO`println(newOrder.getId());
   IO`println("Please add Products");
   return newOrder.getId();
   )
  );
 return -1;
)
pre exists user in set elems users & user.getEmail() = email;



public cancelOrder: int ==> string
cancelOrder(orderId) == (

dcl newRefOrders : map int to int;
dcl newOrders : seq of Order;
newOrders := [];
newRefOrders := {0 |-> -1 };

for order in orders do (
  if(order.getId() <> orderId) then (
  newOrders := newOrders^[order];
  newRefOrders := newRefOrders ++ {order.getId() |-> len newOrders };
  );
 );
```

```
orders := newOrders;
refOrders := newRefOrders;
return "Success: Order Canceled";
)
pre exists order in set elems orders & order.getId() = orderId;



public cloneProd: Product * nat * Store ==> Product
cloneProd(refProd, numberUnits, storeRef) == (
dcl auxRet : int ;
dcl newProd : Product := new Product(refProd.getName(), refProd.getPrice(), refProd.
    getDescription(), numberUnits);
auxRet := newProd.setStore(storeRef);
return newProd;
)
post refProd.getName() = RESULT.getName() and refProd.getPrice() = RESULT.getPrice() and refProd.
    getDescription() = RESULT.getDescription();



public addProductToOrder: string * int * string * int * nat ==> string
addProductToOrder(email, orderId, storeName, productIndex, numberUnits) == (
for store in stores do (
 if store.getName() = storeName then
 (
  dcl auxRet : int ;
  dcl refProd : Product := store.getProduct(productIndex);
  dcl newProd : Product := cloneProd(refProd, numberUnits, store);
  dcl newStock : nat := refProd.removeStock(numberUnits);
  for order in orders do (
  if order.getId() = orderId then
   (
   if order.getUser().getEmail() = email then (auxRet := order.addProduct(newProd););
   IO`println("Product(s) added. Current Order:");
   for prod in order.getProducts() do (IO`print(prod.getName() ^ ". Quantity: "); IO`println(prod
       .getStock()));
   return "Result: success";
   )
  );
 ) );
 return "error";
)
pre exists1 order in set elems orders & (order.getId() = orderId and order.getState() = <INIT>)
 and products(productIndex).getStock() >= numberUnits
 and exists user in set elems users & user.getEmail() = email
 and exists store in set elems stores & store.getName() = storeName
 and products(productIndex) in set elems products;



public removeProductFromOrder: int  * int  ==> string
removeProductFromOrder(orderId, productIndex) == (
dcl refOrder : Order ;
dcl numberUnits : nat ;
dcl refProduct : Product ;
dcl productName : string ;
dcl orderProducts : seq of Product ;
dcl newProdsList : seq of Product ;
newProdsList := [];

  for order in orders do (
  if order.getId() = orderId then
   (
   refOrder := order;
   orderProducts := order.getProducts();
   productName := order.products(productIndex).getName();
   refProduct := order.products(productIndex);
   numberUnits := products(productIndex).getStock();
   )
  );

 for prod in orderProducts do (
  if prod.getName() <> productName then
   (
    newProdsList := newProdsList ^ [prod];
    refOrder.products := newProdsList;
   )
  );

 for store in stores do (
  if store.getName() = refProduct.getStore().getName() then
   (
    for currProd in store.getProducts() do (if currProd.getName() = productName then currProd.
        addStock(numberUnits));
   )
  );

return "Result: success";
)
pre products(productIndex) in set elems products
 and exists order in set elems orders & order.getId() = orderId;



public concludeOrder: string * int ==> string
concludeOrder(email, orderId) == (
for order in orders do (
  if order.getId() = orderId then
   (
   dcl auxRet : int ;
   if order.getUser().getEmail() = email then (auxRet := order.finishOrder());
   IO`println("Order Concluded. Bill:");
   for prod in order.getProducts() do (IO`print(prod.getName() ^ ". Quantity: "); IO`print(prod.
       getStock()); IO`print(" Unit Price: "); IO`println(prod.getPrice()));
   IO`print("Total: "); IO`println(order.getTotalCost());
   return "Result: success";
   )
  );

 return "error";
)
pre exists1 order in set elems orders & (order.getId() = orderId and order.getState() = <INIT>)
 and exists1 user in set elems users & user.getEmail() = email
post exists order in set elems orders & (order.getId() = orderId and order.getState() = <
    ORDER_CONCLUDED>);


                          /***** LISTING OPERATORS ******/
-- "all" for all orders and "email" for user orders

public listOrdersDelivered: string  ==> string
listOrdersDelivered(request) ==
(
if request = "all" then for order in orders do
(
 if order.getState() = <DELIVERED> then
(
IO`print("Order ID: "); IO`println(order.getId());
```

```
IO`print("State of Order: "); IO`println(order.getState());
IO`print("Client: "); IO`println(order.getUser().getEmail());
IO`println("Products (if any): ");
if len order.getProducts() > 0 then
(for product in order.getProducts() do (IO`print("Product: " ^ product.getName() ^ ". Units: ");
    IO`println(product.getStock()));););
)
);

if request <> "all" then for order in orders do
(
 if order.getState() = <DELIVERED> then
(
if (order.getUser().getEmail() = request) then (
IO`print("Order ID: "); IO`println(order.getId());
IO`print("State of Order: "); IO`println(order.getState());
IO`print("Client: "); IO`println(order.getUser().getEmail());
IO`println("Products (if any): ");
if len order.getProducts() > 0 then
(for product in order.getProducts() do (IO`print("Product: " ^ product.getName() ^ ". Units: ");
    IO`println(product.getStock())););););
)
);

IO`println("\n");
return "Result: success";
)
pre orders <> [];


-- "all" for all orders and "email" for user orders

public listOrders: string  ==> string
listOrders(request) ==
(
if request = "all" then for order in orders do
(
IO`print("Order ID: "); IO`println(order.getId());
IO`print("State of Order: "); IO`println(order.getState());
IO`print("Client: "); IO`println(order.getUser().getEmail());
IO`println("Products (if any): ");
if len order.getProducts() > 0 then
(for product in order.getProducts() do (IO`print("Product: " ^ product.getName() ^ ". Units: ");
    IO`println(product.getStock()));););
);

if request <> "all" then for order in orders do
(
if (order.getUser().getEmail() = request) then (
IO`print("Order ID: "); IO`println(order.getId());
IO`print("State of Order: "); IO`println(order.getState());
IO`print("Client: "); IO`println(order.getUser().getEmail());
IO`println("Products (if any): ");
if len order.getProducts() > 0 then
(for product in order.getProducts() do (IO`print("Product: " ^ product.getName() ^ ". Units: ");
    IO`println(product.getStock())););););
);

IO`println("\n");
return "Result: success";
)
pre orders <> [];


public listOrdersByCourier: string  ==> string
```

```
listOrdersByCourier(cardId) ==
(
for order in orders do (if order.getCourier().getCardID() = cardId then
 (
 (IO`print("Order: "); IO`println(order.getId()););
 (IO`print("State: "); IO`println(order.getState()););
 (IO`print("Client: "); IO`println(order.getUser()););
 (IO`print("Total Cost: "); IO`println(order.getTotalCost()););
 IO`println("\n");
 )
);

IO`println("\n");
return "Result: success";
)
pre orders <> [];



public listCategories: () ==> string
listCategories() ==
(
for all category in set categories do (
IO`println("Category: " ^ category);
);
return "Result: success";
)

pre categories <> {};

public listStores: string ==> string
listStores(categoryName) ==
(
IO`println("Stores with Category: " ^ categoryName);
for store in stores do
 (
 if store.getCategory() = categoryName then (IO`println("Store: " ^ store.getName() ^ ".
     Description: " ^ store.getDescription()));
 );
return "Result: success";
)
pre categoryName in set categories and stores <> [];



public listStoreProducts: string ==> string
listStoreProducts(storeName) ==
(
dcl auxProducts : seq of Product ;
dcl countProds : int := 1;

IO`println("Products of: " ^ storeName);
 for store in stores do (
  if store.getName() = storeName then auxProducts := store.getProducts();
 );

 for auxProd in auxProducts do (
  IO`print(countProds); IO`println(" Name: " ^ auxProd.getName() ^ ". Description: " ^ auxProd.
      getDescription());
  IO`print("Unit Price: " ); IO`println(auxProd.getPrice());
  IO`print("Stock: " ); IO`println(auxProd.getStock());
  countProds:=countProds+1;
 );

return "Result: success";
)

pre exists1 store in set elems stores & store.getName() = storeName and stores <> [];

public listCouriers: () ==> string
listCouriers() ==
(
for courier in couriers do (IO`println("Courier ID: " ^ courier.getCardID()););
return "Result: success";
)
pre couriers <> [];



public listUsers: () ==> string
listUsers() ==
(
for user in users do (IO`println("User Email: " ^ user.getEmail()););
return "Result: success";
)
pre users <> [];



public listStoresWithDeliveryCost: real ==> string
listStoresWithDeliveryCost(deliveryCost) ==
(
for store in stores do
 (
 if store.getDeliveryCost() <= deliveryCost then (IO`println("Store: " ^ store.getName() ^ ".
     Description: " ^ store.getDescription()));
 );
return "Result: success";
)
pre stores <> [];



public earnedMoneyFromStore: (string) ==> real
earnedMoneyFromStore(storeName) ==
(
for store in stores do(
 if store.getName() = storeName then(
  for all order in set elems orders do
  (
    if order.getState() = <DELIVERED> then(
    dcl aux : map Product to Product := {x|-> y | x in set elems order.getProducts() , y in set
        elems store.getProducts() & x.getName() = y.getName()};
    dcl commonProducts : set of Product := dom aux;
    dcl earned : real := 0;
    for all product in set commonProducts do(
      earned := earned + product.getPrice() * product.getStock();
     );
     earned := earned -order.getCourier().getSalary();
     return earned;
     )
  );
  );
 );
 return 0;
)

pre orders <> [];

public editProfile: string * string * Gender * nat ==> string
```

```
editProfile(email, name, gender, age) == (

for user in users do (
 if user.getEmail() = email then user.setPersonInfo(name,gender,age)
);
return "Result: success";

);


                          /***** GET OPERATORS ******/

public getUsers: () ==> seq of User
getUsers() == return users

post RESULT = users;

public getCouriers: () ==> seq of Courier
getCouriers() == return couriers

post RESULT = couriers;

public getOrders: () ==> seq of Order
getOrders() == return orders

post RESULT = orders;

public getrefOrders: () ==> map int to int
getrefOrders() == return refOrders
post RESULT = refOrders;

-- !USER --
-- see user information
-- list category
-- list stores by category
-- create order
-- cancel order
-- add product to order
-- close order
-- remove Product from Order

-- !ADMIN/ROBOT --
-- list all users
-- list all orders
-- list couriers
-- assign current orders to courier (closed orders)

-- !Courier --
-- list Courier orders
-- Perform Order (Close order)
-- close order


functions
-- TODO Define functiones here
 traces
-- TODO Define Combinatorial Test Traces here
end Execution
-- create t := new Execution()
-- print t.createOrder("bernas@hotmail.com")
-- print t.addProductToOrder("bernas@hotmail.com",XXXX,"store1",1,1)
-- print t.concludeOrder("bernas@hotmail.com",XXXX)
-- print t.assignCourier("maria@glovo.com",XXXX)
-- print t.performOrder("maria@glovo.com",XXXX)
-- print t.increaseProductStock("store1",1,2)
```

Function or operation	Line	Coverage	Calls
Execution	79	100.0%	28
addCourier	120	100.0%	26
addProductToOrder	237	97.9%	24
addUser	111	100.0%	13
assignCourier	169	93.5%	13
cancelOrder	205	100.0%	13
cloneProd	227	100.0%	24
concludeOrder	307	97.5%	13
createOrder	186	94.1%	36
earnedMoneyFromStore	488	96.7%	3
editProfile	512	100.0%	1
getCouriers	528	100.0%	3
getOrders	532	100.0%	7
getUsers	524	100.0%	3
getrefOrders	536	100.0%	2
increaseProductStock	129	94.5%	13
listCategories	413	100.0%	1
listCouriers	458	100.0%	2
listOrders	367	100.0%	3
listOrdersByCourier	396	100.0%	1
listOrdersDelivered	330	100.0%	6
listStoreProducts	436	100.0%	1
listStores	423	100.0%	1
listStoresWithDeliveryCost	476	100.0%	1
listUsers	467	100.0%	1
performOrder	147	100.0%	26
removeProductFromOrder	265	90.3%	9
Execution.vdmpp		98.3%	274

3 MyTestCase

```
class MyTestCase
/*
 Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
*/
operations
-- Simulates assertion checking by reducing it to pre-condition checking.
-- If 'arg' does not hold, a pre-condition violation will be signaled.

protected assertTrue: bool ==> ()
assertTrue(arg) ==
return
pre arg;

-- Simulates assertion checking by reducing it to post-condition checking.
-- If values are not equal, prints a message in the console and generates
-- a post-conditions violation.

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

end MyTestCase
```

Function or operation	Line	Coverage	Calls
assertEqual	16	100.0%	123
assertTrue	8	100.0%	292
MyTestCase.vdmpp		100.0%	415

4 Order

```
class Order

types
public string = seq of char;
public State = <INIT> | <ORDER_CONCLUDED> | <DELIVERED> ;

instance variables
private orderID:  int;
private totalCost : real;
public products : seq of Product;
public courier : [Courier] :=nil;
private user : User;
private state: State;

operations

public Order: User ==> Order
Order(user_reference) == (
products := [];
state := <INIT>;
orderID := MATH`rand(1000) + MATH`rand(1000);
user := user_reference;
totalCost := 0;
return self
);


public addProduct: Product ==> int
addProduct(newProd) == (
for prod in products do
(
if prod.getName() = newProd.getName() then (prod.addStock(newProd.getStock()); return 1;)
);
products := products ^ [newProd];
return 1;
)
pre newProd.getStock() > 0;


public deliverOrder: string ==> int
deliverOrder(courierID) == (
if courier.getCardID() = courierID then (state := <DELIVERED>;)
else return -1;
return 1;
)
pre state = <ORDER_CONCLUDED>;


public finishOrder: () ==> int
finishOrder() == (
for prod in products do(totalCost := totalCost + prod.getPrice()*prod.getStock());
state := <ORDER_CONCLUDED>;
return 1;
)
pre state = <INIT>
post totalCost > 0 and state = <ORDER_CONCLUDED>;


public setCourier: Courier ==> int
setCourier(assignedCourier) == (courier:=assignedCourier; return 1;)
pre courier <> assignedCourier
post courier=assignedCourier;


pure public getId: () ==> int
getId() == return orderID
post RESULT = orderID;


pure public getTotalCost: () ==> real
getTotalCost() == return totalCost
post RESULT = totalCost;


pure public getProducts: () ==> seq of Product
getProducts() == return products
post RESULT = products;


pure public getCourier: () ==> Courier
getCourier() == return courier
post RESULT = courier;


pure public getUser: () ==> User
getUser() == return user
post RESULT = user;


pure public getState: () ==> State
getState() == return state
post RESULT = state;


public setState: State ==> State
setState(newState) == (state:=newState; return state;)
post state = newState and RESULT = state;

end Order
```

Function or operation	Line	Coverage	Calls
Order	16	100.0%	64
addProduct	26	100.0%	2
deliverOrder	37	82.3%	13
finishOrder	45	100.0%	13
getCourier	71	100.0%	81
getId	59	100.0%	517
getProducts	67	100.0%	164
getState	79	100.0%	247
getTotalCost	63	100.0%	16
getUser	75	100.0%	64
setCourier	54	100.0%	13
setState	83	100.0%	3
Order.vdmpp		97.8%	1197

5 Person

```
class Person

types
public string = seq of char;
public Gender = <Male> | <Female> | <Undefined>;

instance variables
protected name : string := "";
protected gender: Gender:= <Undefined>;
protected age : nat := 18;

operations

public setPersonInfo: string * Gender * nat ==> ()
setPersonInfo(na, ge, ag) == is subclass responsibility
pre ag >= 18
post age = ag;


pure public getName: () ==> string
getName() == return name
post RESULT = name;


pure public getGender: () ==> Gender
getGender() == return gender
post RESULT = gender;


pure public getAge: () ==> nat
getAge() == return age
post RESULT = age;

end Person
```

Function or operation	Line	Coverage	Calls
getAge	26	100.0%	1
getGender	22	100.0%	14
getName	18	100.0%	13
setPersonInfo	13	14.2%	0
Person.vdmpp		76.0%	28

6 Product

```
class Product

types
public string = seq of char;

instance variables
private name : string;
private price : real := 0.1;
private description : string;
private quantity: nat;
private store: Store;
private orders : set of Order := {};

inv price >= 0.0 and quantity > 0;

operations

public Product: string * real * string * nat ==> Product
Product(na, pri, des, qty) == (name := na; price := pri; description:=des; quantity := qty;
    return self)
pre price > 0;


public addStock: nat ==> ()
addStock(numberUnits) == (
quantity := quantity + numberUnits;
)
pre numberUnits >= 0;


public removeStock: nat ==> nat
removeStock(numberUnits) == (
quantity := quantity - numberUnits;
return quantity;
)
pre numberUnits > 0
post RESULT = quantity;


pure public getName: () ==> string
getName() == return name
post RESULT = name;


pure public getPrice: () ==> real
getPrice() == return price
post RESULT = price;


pure public getDescription: () ==> string
```

```
getDescription() == return description
post RESULT = description;


pure public getStock: () ==> nat
getStock() == return quantity
post RESULT = quantity;


pure public getStore: () ==> Store
getStore() == return store
post RESULT = store;


public setStore: Store ==> int
setStore(storeOfProduct) == (store := storeOfProduct; return 1;);


functions

end Product
```

Function or operation	Line	Coverage	Calls
Product	17	100.0%	642
addStock	21	100.0%	3
getDescription	43	100.0%	81
getName	35	100.0%	271
getPrice	39	100.0%	120
getStock	47	100.0%	185
getStore	54	100.0%	6
removeStock	27	100.0%	2
setStore	51	100.0%	2
Product.vdmpp		100.0%	1312

7 Store

```
class Store
types
public string = seq of char;

instance variables
private name : string;
private category : string;
private description : string;
private estimateTime : string;
private deliveryCost : real;
private products : seq of Product;

inv estimateTime(2) = '-' or estimateTime(3) = '-' ;

operations

public Store: string * string * string * string * real * seq of Product ==> Store
```

```
Store(na, ca, des, time, cost, prods) == (name := na; category:=ca; description := des;
    estimateTime:=time; deliveryCost := cost; products := prods; return self)
pre cost > 0
post deliveryCost = cost;


public addProduct: Product ==> nat
addProduct(p) == (
products := products ^ [p];
return len products;
)
pre p not in set elems products
post p in set elems products;


pure public getProduct: int ==> Product
getProduct(productIndex) == return products(productIndex)
post RESULT = products(productIndex);


pure public getProducts: () ==> seq of Product
getProducts() == return products
post RESULT = products;


pure public getName: () ==> string
getName() == return name
post RESULT = name;


pure public getCategory: () ==> string
getCategory() == return category
post RESULT = category;


pure public getDescription: () ==> string
getDescription() == return description
post RESULT = description;


pure public getEstimateTime: () ==> string
getEstimateTime() == return estimateTime
post RESULT = estimateTime;


pure public getDeliveryCost: () ==> real
getDeliveryCost() == return deliveryCost
post RESULT = deliveryCost;

end Store
```

Function or operation	Line	Coverage	Calls
Store	16	0.0%	0
addProduct	21	100.0%	1
getCategory	41	100.0%	18
getDeliveryCost	53	100.0%	18
getDescription	45	100.0%	6
getEstimateTime	49	100.0%	2
getName	37	100.0%	252

getProduct	29	100.0%	37
getProducts	33	100.0%	54
Store.vdmpp		100.0%	388

8 TestGlovo

```
class TestGlovo is subclass of MyTestCase
/*
 Contains the test cases for the Glovo app.
 Illustrates a scenario-based testing approach.
*/
types
public string = seq of char;
public State = <INIT> | <ORDER_CONCLUDED> | <DELIVERED> ;

values
address : User`Address = mk_User`Address("as","as","as","Portugal");

instance variables
 exec : Execution := new Execution();
 orderId: int := 0;
 order: Order ;

operations
/***** USE CASE SCENARIOS ******/


 private test_CreateOrder: () ==> ()
 test_CreateOrder() ==
 (
 dcl currentRefOrders: map int to int;
 dcl orderIndex: nat;
 dcl auxState: State;

 dcl currentNumOrders: int :=  len exec.getOrders();
 orderId := exec.createOrder("blankuser");

 currentRefOrders :=  exec.getrefOrders();
 orderIndex := currentRefOrders(orderId);
 order :=  exec.getOrders()(orderIndex);
  auxState := order.setState(<INIT>);

 assertEqual(currentNumOrders+1,len exec.getOrders());
 )
 pre exists user in set elems exec.users & user.getEmail() = "blankuser"
 post exists userOrder in set elems exec.orders & (userOrder.getId() = orderId and order.getState
     () = <INIT>);


  private test_cancelOrder: () ==> ()
 test_cancelOrder() ==
 (
 dcl currentRefOrders: map int to int;
 dcl orderIndex: nat;
 dcl auxRet: string;

 dcl currentNumOrders: int :=  len exec.getOrders();
 orderId := exec.createOrder("blankuser");

 currentRefOrders :=  exec.getrefOrders();
```

```
orderIndex := currentRefOrders(orderId);
order :=  exec.getOrders()(orderIndex);

assertEqual(currentNumOrders+1,len exec.getOrders());

auxRet := exec.cancelOrder(order.getId());

currentNumOrders := len exec.getOrders();

assertEqual(currentNumOrders,1);
)
pre exists testOrder in set elems exec.orders & testOrder.getId() = orderId;



 private test_AddProductToOrder: () ==> ()
test_AddProductToOrder() ==
(

dcl numberProducts: int :=  len order.getProducts();

dcl auxRet : string := exec.addProductToOrder("blankuser", orderId, "store1", 1, 1);
dcl newNumberProducts: int :=  len order.getProducts();
assertEqual(newNumberProducts,numberProducts+1);

numberProducts :=  len order.getProducts();

auxRet := exec.addProductToOrder("blankuser", orderId, "store1", 1, 1);
newNumberProducts :=  len order.getProducts();


)
pre exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and userOrder.
    getState() = <INIT>)
and exists user in set elems exec.users & user.getEmail() = "blankuser"
and exists store in set elems exec.stores & store.getName() = "store1"
and exec.products(1) in set elems exec.products;


  private test_removeProductFromOrder: () ==> ()
test_removeProductFromOrder() ==
(


dcl numberProducts: int :=  len order.getProducts();

dcl auxRet : string := exec.removeProductFromOrder(orderId, 1);
dcl newNumberProducts: int :=  len order.getProducts();
IO`print("numberProducts: "); IO`print(numberProducts);


IO`print("newNumberProducts: "); IO`print(newNumberProducts);
assertEqual(auxRet,"Result: success");

)
pre exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and userOrder.
    getState() = <INIT>)
and exists user in set elems exec.users & user.getEmail() = "blankuser"
and exists store in set elems exec.stores & store.getName() = "store1"
and exec.products(1) in set elems exec.products;

 private test_concludeOrder: () ==> ()
test_concludeOrder() ==
(
 dcl auxRet : string;
 assertTrue(order.getState() = <INIT>);

  auxRet := exec.concludeOrder("blankuser", orderId);

 assertTrue(len order.getProducts() > 0);
 assertTrue(order.getState() = <ORDER_CONCLUDED>);

 )
 pre exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and userOrder.
     getState() = <INIT>)
 and exists1 user in set elems exec.users & user.getEmail() = "blankuser"
post exists userOrder in set elems exec.orders & (userOrder.getId() = orderId and userOrder.
    getState() = <ORDER_CONCLUDED>);

  private test_assignCourier: string ==> ()
 test_assignCourier(cardId) ==
 (
  dcl auxRet : string;

  auxRet := exec.assignCourier(cardId,orderId);
 assertTrue(order.getCourier().getCardID() = cardId);


 )
pre exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and (order.courier
    = nil or userOrder.getCourier().getCardID() <> cardId) and userOrder.getState() = <
    ORDER_CONCLUDED>)
 and exists1 courier in set elems exec.couriers & courier.getCardID() = cardId
post exists1 userOrder in set elems exec.orders & (userOrder.getId() = orderId and userOrder.
    getCourier().getCardID() = cardId);

   private test_performOrder: string ==> ()
 test_performOrder(cardId) ==
 (
  dcl auxRet : string;
  dcl totalMoney : real;

  assertTrue(order.getCourier().getCardID() = cardId);
  assertTrue(order.getState() = <ORDER_CONCLUDED>);

  auxRet := exec.performOrder(cardId,orderId);
  totalMoney := exec.earnedMoneyFromStore("store1");


 assertTrue(order.getState() = <DELIVERED>);

 )
 pre (orderId in set dom exec.refOrders
 and exists1 userOrder in set elems exec.orders & userOrder.getId() = orderId
 and exists1 courier in set elems exec.couriers & courier.getCardID() = cardId
 );

   private test_addUser: string ==> ()
 test_addUser(email) ==
 (
  dcl auxRet : string;
 dcl hasUser : bool := false;
 for user in exec.getUsers() do ( if user.getEmail() = email then hasUser := true);

 assertTrue(hasUser = false);

 auxRet := exec.addUser(email, "Number", "Liberty", "3310-119", "Feuplandia");
```

```
for user in exec.getUsers() do ( if user.getEmail() = email then hasUser := true);

assertTrue(hasUser = true);

);


    private test_setCourierInfo: string ==> ()
test_setCourierInfo(cardID) ==
(
dcl foundCourier : Courier;
for courier in exec.getCouriers() do ( if courier.getCardID() = cardID then foundCourier :=
    courier);

 foundCourier.setPersonInfo("name", <Male>, 25);

assertTrue(foundCourier.getGender() = <Male>);



);
  private test_addCourier: string ==> ()
test_addCourier(cardId) ==
(
 dcl auxRet : string;
dcl hasCourier : bool := false;
for courier in exec.getCouriers() do ( if courier.getCardID() = cardId then hasCourier := true);

assertTrue(hasCourier = false);

auxRet := exec.addCourier(cardId, 1000);

for courier in exec.getCouriers() do ( if courier.getCardID() = cardId then hasCourier := true);

assertTrue(hasCourier = true);



);

   private test_editProfile: string ==> ()
test_editProfile(userEmail) ==
(
dcl auxRet : string;
dcl refUser : User;

auxRet := exec.editProfile(userEmail, "newName", <Female>, 99);

for user in exec.getUsers() do ( if user.getEmail() = userEmail then refUser := user; );

assertEqual(refUser.getName(), "newName");
assertEqual(refUser.getGender(), <Female>);
assertEqual(refUser.getAge(), 99);

)
pre exists user in set elems exec.users & user.getEmail() = userEmail;

   private test_increaseProductStock: () ==> ()
test_increaseProductStock() ==

(
dcl auxRet : string;
dcl prodStock : int;
dcl refStores : seq of Store;
dcl estimateTime : string;
```

```
refStores := exec.stores;
estimateTime := exec.stores(1).getEstimateTime();
prodStock := refStores(1).getProducts()(1).getStock();

auxRet := exec.increaseProductStock(refStores(1).getName(), 1, 10);

assertEqual(refStores(1).getProducts()(1).getStock(), prodStock+10);

);


   private test_lists: () ==> ()
test_lists() ==
(
dcl auxRet : string;

auxRet := exec.listCategories();
auxRet := exec.listStores("pizza");
auxRet := exec.listStoreProducts("store1");
auxRet := exec.listOrders("all");
auxRet := exec.listOrders("blankuser");
auxRet := exec.listOrdersDelivered("all");
auxRet := exec.listOrdersDelivered("blankuser");
auxRet := exec.listOrdersDelivered("blankuser");
auxRet := exec.listUsers();
auxRet := exec.listCouriers();
auxRet := exec.listStoresWithDeliveryCost(3);
auxRet := exec.listOrdersByCourier("maria@glovo.com");


);


     private test_createAddProductToStore: () ==> ()
test_createAddProductToStore() ==
(
 dcl prod1A : Product := new Product("Pipoca", 2.5, "Bem boa", 5);
dcl store1: Store := new Store("testStore", "candies", "Candies Store", "10-15min", 1.5, [prod1A
    ] );


 dcl prod1Test : Product := new Product("test", 1, "test 2", 6);

dcl productsNum : nat := store1.addProduct(prod1Test);
assertTrue(store1.getName() = "testStore");
assertTrue(len store1.getProducts() = productsNum);


);


public static main: () ==> ()
main() ==
(
dcl testGlovo: TestGlovo :=  new TestGlovo();

 testGlovo.test_increaseProductStock();
testGlovo.test_CreateOrder();
testGlovo.test_AddProductToOrder();
testGlovo.test_removeProductFromOrder();
testGlovo.test_concludeOrder();
testGlovo.test_assignCourier("maria@glovo.com");
testGlovo.test_performOrder("maria@glovo.com");
testGlovo.test_cancelOrder();
```

```
 testGlovo.test_addUser("newUser@mail.com");
 testGlovo.test_addCourier("newCourier@glovo.com");
 testGlovo.test_setCourierInfo("newCourier@glovo.com");
 testGlovo.test_editProfile("blankuser");
 testGlovo.test_createAddProductToStore();
 testGlovo.test_lists();

 );
 traces
 -- test cases will be generated in possible combinations
 testaddUsers :
(let email in set {"email1@hotmail.com","email2@hotmail.com"} in exec.addUser(email,"address",
    "street","1440-100","PT")){1,2};

 testCreateOrders :
(let email in set {"email1@hotmail.com","email2@hotmail.com"} in exec.createOrder(email)){1,5};

end TestGlovo
```

Function or operation	Line	Coverage	Calls
main	222	100.0%	1
test_AddProductToOrder	65	100.0%	1
test_CreateOrder	21	100.0%	1
test_addCourier	148	94.5%	1
test_addUser	130	95.0%	1
test_assignCourier	97	91.5%	2
test_cancelOrder	41	100.0%	1
test_concludeOrder	81	100.0%	1
test_createAddProductToStore	259	100.0%	1
test_editProfile	166	100.0%	1
test_increaseProductStock	182	100.0%	1
test_lists	201	100.0%	1
test_performOrder	110	100.0%	1
test_removeProductFromOrder	86	100.0%	1
test_setCourierInfo	173	100.0%	1
TestGlovo.vdmpp		96.6%	16

9 User

```
class User is subclass of Person
types
public string = seq of char;
public Address :: address : string
         street : string
          zip_code: string
           country : string;

values
instance variables
private email : string;
private address: Address;
```

```
operations

public User: string * User`Address ==> User
User(em, addr) == (email:=em; address:=addr; return self);


pure public getEmail: () ==> string
getEmail() == return email
post RESULT = email;


public setPersonInfo: string * Gender * nat ==> ()
setPersonInfo(na, ge, ag) == (name := na; gender := ge; age:=ag;)
post name = na and gender = ge and age = ag;



end User
```

Function or operation	Line	Coverage	Calls
User	15	100.0%	48
getEmail	18	100.0%	247
setPersonInfo	26	100.0%	1
User.vdmpp		100.0%	296