feup-mfes

January 3, 2018

Contents

1 BuyInPortugal

```
class BuyInPortugal
types
 public Category = seq of char;
 public Subcategory = seq of char;
 public AdminCode = seq of char;

instance variables
 public categories : set of Category := {};
 public subcategories : map Subcategory to Category := { |-> };
 public manufacturers : map Manufacturer`Name to Manufacturer := { |-> };
 public products : map Product`Title to Product := { |-> };
 public clients : map Client`Email to Client := { |-> };

 public adminCode : AdminCode := [];

 -- subcategories should be associated with category
  inv rng subcategories subset categories;

operations
 /** ADMIN OPERATIONS **/


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

 -- Change admin password

 public setAdminCode: AdminCode * AdminCode ==> ()
```

```
setAdminCode(curCode, nextCode) ==
 adminCode := nextCode
pre curCode <> nextCode
 and curCode = adminCode;

-- Register manufacturer

public registerManufacturer: Manufacturer`Name * AdminCode ==> ()
registerManufacturer(name, code) == (
 dcl m:Manufacturer := new Manufacturer(name);
 addManufacturer(m);
)
pre name not in set dom manufacturers
 and code = adminCode
post dom manufacturers = dom manufacturers~ union {name};


private addManufacturer: Manufacturer ==> ()
addManufacturer(m) == (
 manufacturers := manufacturers munion { m.name |-> m };
);

-- Add category

public addCategory: Category * AdminCode ==> ()
addCategory(category, code) == (
 categories := categories union {category};
)
pre category not in set categories
 and code = adminCode
post categories = categories~ union {category};

-- Set categories

public setCategories: set of Category * AdminCode ==> ()
setCategories(cats, code) == (
 categories := cats;
)
pre code = adminCode;

-- Add subCategories

public addSubcategory: Subcategory * Category * AdminCode ==> ()
addSubcategory(subcategory, category, code) == (
 subcategories := subcategories munion {subcategory |-> category};
)
pre subcategory not in set dom subcategories
 and category in set categories
 and code = adminCode
post dom subcategories = dom subcategories~ union {subcategory};

-- Set subCategories

public setSubcategories: map Subcategory to Category * AdminCode ==> ()
setSubcategories(subcats, code) == (
 subcategories := subcats;
)
pre code = adminCode;

/** ADMIN OPERATIONS END **/


/** MANUFACTURER OPERATIONS **/

-- Add product
```

```
public addProduct: Manufacturer`Name * Product`Title * Product`Description * Product`Subcategory
     * Product`Price * map Product`Color to Product`Quantity ==> ()
addProduct(manName, tit, des, cat, pr, qties) == (
 dcl product : Product := new Product(tit, des, cat, pr, qties);
 let manufacturer = manufacturers(manName)
  in (
   manufacturer.addProduct(product);
   products := products munion {tit |-> product};
  );
  return;
)
pre manName in set dom manufacturers
 and tit not in set dom products
 and cat in set dom subcategories
post dom products = dom products~ union {tit};

-- Remove product


public removeProduct: Manufacturer`Name * Product`Title ==> ()
removeProduct(manName, title) == (
 let manufacturer = manufacturers(manName)
  in (
   manufacturer.removeProduct(title);
   products := {title} <-: products;
   for all client in set rng clients
   do (
    for all mk_(t, color) in set client.wishlist
    do (

     if t = title then client.removeFromWishlist(t, color);
    );
    for all mk_(t, color) in set dom client.cart
    do (
     if t = title then client.removeFromCart(t, color);
    );
   );
 );
)
pre manName in set dom manufacturers

 and title in set dom products
post dom products = dom products~ \ {title};

-- Add to stock of a product
public addToStock: Manufacturer`Name * Product`Title * Product`Color * Product`Quantity ==> ()

addToStock(manName, title, color, qty) == (
 let product = products(title)
 in (
  product.addToStock(color, qty);
 );
)
pre title in set dom manufacturers(manName).products;

-- Remove from stock of a product

public removeFromStock: Manufacturer`Name * Product`Title * Product`Color * Product`Quantity ==>
     ()
removeFromStock(manName, title, color, qty) == (
 let product = products(title)
 in (
  product.removeFromStock(color, qty);
 );
```

```
)
pre title in set dom manufacturers(manName).products;


 -- Set volume discounts
public setVolumeDiscounts : Manufacturer`Name * Product`Title * Product`VolumeDiscounts ==> ()
setVolumeDiscounts(manName, title, volDiscs) == (
let product = products(title)
 in (
  product.setVolumeDiscounts(volDiscs);
 );
)
pre title in set dom manufacturers(manName).products;


/** MANUFACTURER OPERATIONS END **/


/** CLIENT OPERATIONS **/

-- Add to wishlist of a client
public addToWishlist: Client`Email * Product`Title * Product`Color ==> ()
addToWishlist(email, title, color) == (

 let client = clients(email)
 in (
  client.addToWishlist(title, color);
 );
);

-- Remove from wishlist of a client
public removeFromWishlist: Client`Email * Product`Title * Product`Color ==> ()
removeFromWishlist(email, title, color) == (

 let client = clients(email)
 in (
  client.removeFromWishlist(title, color);
 );
);

-- Add to cart of a client
public addToCart: Client`Email * Product`Title * Product`Color ==> ()
addToCart(email, title, color) == (
 let client = clients(email)
 in (
  client.addToCart(title, color);
 );
);


-- Set quantity from cart product of a client
public setQtyInCart: Client`Email * Product`Title * Product`Color * Product`Quantity ==> ()
setQtyInCart(email, title, color, qty) == (
 let client = clients(email)
 in (
  client.setQtyInCart(title, color, qty);
 );
);

-- Remove from cart of a client
public removeFromCart: Client`Email * Product`Title * Product`Color ==> ()
removeFromCart(email, title, color) == (
 let client = clients(email)
 in (
  client.removeFromCart(title, color);
```

```
 );
);


-- Get total cart value
public getTotalCart: Client`Email ==> rat
getTotalCart(email) == (
 dcl sum: rat := 0;
 let client = clients(email), cart = client.cart
 in (
  for all mk_(title, color) in set dom cart
   do (let qty = cart(mk_(title, color))
    in sum := sum + products(title).getPriceWithDiscount(qty) * qty;
   );
  return sum;
 );
)
pre email in set dom clients;


-- Buy cart from client
public buy: Client`Email ==> ()
buy(email) == (
 let client = clients(email), cart = client.cart
 in (
  for all mk_(title, color) in set dom cart
   do (

    products(title).removeFromStock(color, cart(mk_(title, color)))
   );
  client.pushCartToHistory();
 );
)
pre let client = clients(email), cart = client.cart

 in (
  forall mk_(title, color) in set dom cart
   & cart(mk_(title, color)) <= products(title).quantities(color)
 );

-- Convert wishlist of a client
public convertWishlist: Client`Email ==> ()
convertWishlist(email) == (

 let client = clients(email)
 in (
  client.convertWishlist();
 );
);

/** CLIENT OPERATIONS END **/



/** VISITOR OPERATIONS **/

-- Register client
public registerClient: Client`Email ==> ()
registerClient(email) == (
dcl c:Client := new Client(email);
 addClient(c);
)
pre email not in set dom clients
post dom clients = dom clients~ union {email};
```

```
 private addClient: Client ==> ()
 addClient(c) == (
  clients := clients munion { c.email |-> c };

 );

 -- Get product by title
 public searchProductsByTitle: Product`Title ==> Product
 searchProductsByTitle(title) == (
  dcl product: Product;
  product := products(title);
  return product;
 );

  -- Get products by manufacturer
 public searchProductsByManufacturer: Manufacturer`Name ==> set of Product
 searchProductsByManufacturer(man) == (
  dcl resultPoducts: set of Product := {};
  resultPoducts := rng manufacturers(man).products;
  return resultPoducts;
 )
 pre man in set dom manufacturers;

 -- Get products by subcategory

 public searchProductsBySubcategory: Subcategory ==> set of Product
 searchProductsBySubcategory(subcat) == (
  dcl resultProducts: set of Product := {};
  for all product in set rng products
   do (
    if product.subcategory = subcat
    then resultProducts := resultProducts union {product};
   );
  return resultProducts;
 )
 pre subcat in set dom subcategories;


 -- Get products by category

 public searchProductsByCategory: Category ==> set of Product
 searchProductsByCategory(cat) == (
  dcl resultProducts: set of Product := {};
  for all product in set rng products
   do (
    if subcategories(product.subcategory) = cat
    then resultProducts := resultProducts union {product};
   );
  return resultProducts;
 )
 pre cat in set categories;
 /** VISITOR OPERATIONS END **/

end BuyInPortugal
```

Function or operation	Line	Coverage	Calls
BuyInPortugal	22	100.0%	336
addCategory	49	100.0%	274
addClient	228	100.0%	128
addManufacturer	43	100.0%	234

addProduct	87	100.0%	200
addSubcategory	65	100.0%	216
addToCart	146	100.0%	84
addToStock	103	100.0%	179
addToWishlist	128	100.0%	54
buy	188	100.0%	18
convertWishlist	206	100.0%	22
getTotalCart	173	100.0%	55
registerClient	220	100.0%	128
registerManufacturer	34	100.0%	234
removeFromCart	164	100.0%	11
removeFromStock	113	100.0%	11
removeFromWishlist	137	100.0%	11
removeProduct	103	100.0%	10
searchProductByCategory	265	100.0%	3
searchProductBySubcategory	251	100.0%	1
searchProductsByCategory	299	100.0%	3
searchProductsByManufacturer	242	100.0%	1
searchProductsBySubcategory	285	100.0%	1
searchProductsByTitle	234	100.0%	1
setAdminCode	27	100.0%	9
setCategories	58	100.0%	212
setQtyInCart	155	100.0%	84
setSubcategories	75	100.0%	190
setVolumeDiscounts	123	100.0%	33
BuyInPortugal.vdmpp		100.0%	2743

2 Client

```
class Client

types
 public Email = seq of char;
 public Cart = map (Product`Title * Product`Color) to nat1;
 public Wishlist = set of (Product`Title * Product`Color);

instance variables
  public cart: Cart  := { |-> };
  public wishlist: Wishlist := {};
  public email: Email;
  public buyHistory : seq of Cart := [];

operations

 public Client : Email ==> Client
 Client(e) == (
  email := e;
   return self
 );

 -- Add product to wishlist

 public addToWishlist: Product`Title * Product`Color ==> ()
```

```
addToWishlist(title, color) == (
 wishlist := wishlist union { mk_(title, color) };
)
pre mk_(title, color) not in set wishlist;

-- Remove product from wishlist

public removeFromWishlist: Product`Title * Product`Color ==> ()
removeFromWishlist(title, color) == (
 wishlist := wishlist \ {mk_(title, color)};
)
pre mk_(title, color) in set wishlist;

-- Add product to cart

public addToCart: Product`Title * Product`Color ==> ()
addToCart(title, color) == (
 cart := cart munion { mk_(title, color) |-> 1 };
)
pre mk_(title, color) not in set dom cart;

-- Remove product from cart specific color

public removeFromCart: Product`Title * Product`Color ==> ()
removeFromCart(title, color) == (
 cart := {mk_(title, color)} <-: cart;
)
pre mk_(title, color) in set dom cart;

-- Set product quantity in cart

public setQtyInCart: Product`Title * Product`Color * Product`Quantity ==> ()
setQtyInCart(title, color, qty) == (
 cart := cart ++ { mk_(title, color) |-> qty };
)
pre mk_(title, color) in set dom cart
 and qty > 0;

-- Push cart to buy history

public pushCartToHistory: () ==> ()
pushCartToHistory() == (
 buyHistory := [cart] ^ buyHistory;
 cart := { |-> };
)
pre card dom cart > 0
post cart = { |-> }
 and len buyHistory = len buyHistory~ + 1;

-- Convert wishlist to cart

public convertWishlist: () ==> ()
convertWishlist() == (
 for all mk_(title, color) in set wishlist
  do (
   if mk_(title, color) not in set dom cart
   then addToCart(title, color);
  );
 wishlist := { };
)
pre card wishlist > 0
post wishlist = { }
 and card dom cart = card (dom cart~ union wishlist~);
```

```
end Client
```

Function or operation	Line	Coverage	Calls
Client	15	100.0%	256
addToCart	36	100.0%	153
addToWishlist	22	100.0%	111
convertWishlist	68	100.0%	41
pushCartToHistory	58	100.0%	25
removeFromCart	43	100.0%	88
removeFromWishlist	29	100.0%	38
setQtyInCart	50	100.0%	111
Client.vdmpp		100.0%	823

3 Manufacturer

```
class Manufacturer
types
 public Name = seq of char;

instance variables
 public name : Name;
 public products : map Product`Title to Product := { |-> };

operations

 public Manufacturer : Name ==> Manufacturer
 Manufacturer(n) == (
  name := n;
   return self
 );

 -- Add product

 public addProduct: Product ==> ()
 addProduct(p) == (
  products := products munion { p.title |-> p };
 )
 pre p.title not in set dom products;

 -- Remove product

 public removeProduct: Product`Title ==> ()
 removeProduct(title) == (
  products := {title} <-: products;
 )
 pre title in set dom products;

end Manufacturer
```

Function or operation	Line	Coverage	Calls

Manufacturer	10	100.0%	288
addProduct	17	100.0%	384
removeProduct	24	100.0%	9
Manufacturer.vdmpp		100.0%	681

4 MyTestCase

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  JPF, FEUP, MFES, 2014/15.
*/

operations

 -- Simulates assertion checking by reducing it to pre-condition checking.
 -- If 'arg' does not hold, a pre-condition violation will be signaled.

 protected static 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 static 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	20	100.0%	5
assertTrue	12	100.0%	460
MyTestCase.vdmpp		100.0%	465

5 Product

```
class Product

types
```

```
 public Title = seq of char;
 public Description = seq of char;
 public Subcategory = seq of char;
 public VolumeDiscounts =  map Quantity to Price;
 public Quantity = nat;
 public Price = rat;
 public Color = <White> | <Blue> | <Pink> | <Yellow> | <Orange> | <Black> | <Purple> | <Brown> |
     <Green> | <Gray> | <Red> | <None> ;

instance variables
  public title: Title;
  public description: Description;
  public price: Price;
  public subcategory: Subcategory;
  public quantities: map Color to Quantity := { <None> |-> 0};
  public volumeDiscounts: VolumeDiscounts := { |-> };
  public colors: set of Color := {<None>};

  inv card colors > 0;
  inv <None> in set colors => card colors = 1;
  inv dom quantities = colors;

operations


 -- Create product with color
 public Product : Title * Description * Subcategory * Price * map Color to Quantity ==> Product
 Product(tit, des, cat, pr, qties) == (
  subcategory := cat;
   title := tit;
   description := des;
   price := pr;
   quantities := qties;
   colors := dom qties;
   return self;
 )
 pre
  if <None> in set (dom qties)
  then (dom qties) = {<None>}
  else qties <> { |-> };

 -- Set volume discounts
 public setVolumeDiscounts : VolumeDiscounts ==> ()
 setVolumeDiscounts(volDiscs) == (
  volumeDiscounts := volDiscs;
 );

 -- Remove from stock in products with color
 public removeFromStock: Color * Quantity ==> ()
 removeFromStock(color, qty) == (
  quantities := quantities ++ {color |-> (quantities(color) - qty)};

 )
 pre color in set colors
  and qty <= quantities(color);

 -- Add to stock in products with color
 public addToStock: Color * Quantity ==> ()

 addToStock(color, qty) == (
  quantities := quantities ++ {color |-> (quantities(color) + qty)};
 )
 pre color in set colors;

 -- Get price with discount applied
```

```
 public getPriceWithDiscount: Quantity ==> Price
 getPriceWithDiscount(qty) == (

  dcl discountedPrice : Price := price;
  if volumeDiscounts <> { |-> }
  then (
   for all quantity in set dom volumeDiscounts
    do (
     if (qty >= quantity and discountedPrice > volumeDiscounts(quantity))
     then discountedPrice := volumeDiscounts(quantity);

    );
   );
  return discountedPrice;
 )
 post RESULT <= price;

end Product
```

Function or operation	Line	Coverage	Calls
Product	27	100.0%	214
addToStock	67	100.0%	134
getPriceWithDiscount	74	100.0%	32
removeFromStock	59	100.0%	52
setVolumeDiscounts	53	100.0%	8
Product.vdmpp		100.0%	440

6 TestBuyInPortugal

```
class TestBuyInPortugal is subclass of MyTestCase
/* Test cases for BuyInPortugal model*/

instance variables
 public static testManufacturer : Manufacturer := new Manufacturer("RENOVA");
 public static testProduct: Product := new Product("Pocket Tissues",
  "- 3-ply base sheet\n- 36x9 tissues per pack\n- Tissue size: 21x21cm",
  "Health & Personal Care",
  1.23,
  {<White> |-> 0,
   <Blue> |-> 0,
   <Pink> |-> 0,
   <Yellow> |-> 0,
   <Orange> |-> 0,
   <Purple> |-> 0,
   <Green> |-> 0,
   <Red> |-> 0
  });
  public static testClient : Client := new Client("logistica@fe.up.pt");



operations

 -- Pre configuration 1 that returns a BuyInPortugal model with categories, subcategories and a
     manufacturer
```

```
public static configuredBuyInPortugal1: () ==> BuyInPortugal
configuredBuyInPortugal1() == (
 dcl bip : BuyInPortugal := new BuyInPortugal();

 bip.setCategories({
  "Agriculture & Food",
  "Beauty & Health",
  "Books & Audible",
  "Clothes, Shoes & Jewellery",
  "Car & Motorbike",
  "Fresh Products, Drinks & Grocery",
  "Home, Garden, Pets & DIY",
  "Electronics & Computers",
  "Metallurgy, Chemicals, Rubber & Plastics",
  "Movies, TV, Music & Games",
  "Machinery, Industrial Parts & Tools",
  "Toys, Children & Baby",
  "Sports & Outdoors"
  },"");
 bip.addCategory("Real Estate","");

 bip.setSubcategories({
 "Vanilla Beans" |-> "Agriculture & Food",
 "Plant Seeds & Bulbs" |-> "Agriculture & Food",
 "Nuts & Kernels" |-> "Agriculture & Food",
 "Health & Personal Care" |-> "Beauty & Health",
 "Breads & Bakery" |-> "Fresh Products, Drinks & Grocery",
 "Dairy & Eggs" |-> "Fresh Products, Drinks & Grocery"
 },"");
 bip.addSubcategory("Investment", "Real Estate","");

 bip.registerManufacturer(testManufacturer.name,"");

 return bip;
);


-- Pre configuration 2 that returns a BuyInPortugal model containing all pre config 1 contained
    plus a product and a client
public static configuredBuyInPortugal2: () ==> BuyInPortugal
configuredBuyInPortugal2() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal1();

 bip.addProduct(testManufacturer.name, testProduct.title, testProduct.description, testProduct.
     subcategory, testProduct.price, testProduct.quantities);

 bip.registerClient(testClient.email);

 return bip;
);


-- Pre configuration 2 that returns a BuyInPortugal model containing all pre config 2 contained
    plus stock of the product
public static configuredBuyInPortugal3: () ==> BuyInPortugal
configuredBuyInPortugal3() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal2();

 bip.addToStock(testManufacturer.name, testProduct.title, <Blue>, 36);
 bip.addToStock(testManufacturer.name, testProduct.title, <White>, 2);

 return bip;
);
```

```
-- Pre configuration 2 that returns a BuyInPortugal model containing all pre config 2 contained
    plus more products and manufacturers examples
public static configuredBuyInPortugal4: () ==> BuyInPortugal
configuredBuyInPortugal4() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal2();

 bip.registerManufacturer("DANONE","");

 bip.addProduct("DANONE","YOGURT PURE AROMA TUTTI - FRUTTI 4X120G", "", "Dairy & Eggs", 2.45, {<
     None> |-> 0});
 bip.addProduct("DANONE","YOGURT CHILD ONE BONGO 8 FRUITS 4X155G", "", "Dairy & Eggs", 2, {<None
     > |-> 0});

 bip.addToStock(testManufacturer.name, testProduct.title, <Blue>, 36);
 bip.addToStock(testManufacturer.name, testProduct.title, <White>, 2);

 return bip;
);


/** TEST CASES WITH VALID INPUTS **/
/*public static test: () ==> ()
test() == (
 dcl bip : BuyInPortugal := new BuyInPortugal();

 bip.setCategories({
  "Agriculture & Food",
  "Beauty & Health",
  "Books & Audible",
  "Clothes, Shoes & Jewellery",
  "Car & Motorbike",
  "Fresh Products, Drinks & Grocery",
  "Home, Garden, Pets & DIY",
  "Electronics & Computers",
  "Metallurgy, Chemicals, Rubber & Plastics",
  "Movies, TV, Music & Games",
  "Machinery, Industrial Parts & Tools",
  "Toys, Children & Baby",
  "Sports & Outdoors"
  },"");
 bip.addCategory("Real Estate","");

 bip.setSubcategories({
 "Vanilla Beans" |-> "Agriculture & Food",
 "Plant Seeds & Bulbs" |-> "Agriculture & Food",
 "Nuts & Kernels" |-> "Agriculture & Food",
 "Health & Personal Care" |-> "Beauty & Health"
 },"");
 bip.addSubcategory("Investment", "Real Estate","");

 bip.registerManufacturer("RENOVA","");

 bip.addProduct(
  "RENOVA",
 "Pocket Tissues",
 "- 3-ply base sheet\n- 36x9 tissues per pack\n- Tissue size: 21x21cm",
 "Health & Personal Care",
 1.23,
 {<White> |-> 0,
  <Blue> |-> 0,
  <Pink> |-> 0,
  <Yellow> |-> 0,
  <Orange> |-> 0,
  <Purple> |-> 0,
```

```
  <Green> |-> 0,
  <Red> |-> 0
 });

 bip.addToStock("RENOVA", "Pocket Tissues", <Blue>, 36);
 bip.addToStock("RENOVA", "Pocket Tissues", <White>, 2);

 bip.registerClient("logistica@fe.up.pt");

 bip.addToWishlist("logistica@fe.up.pt", "Pocket Tissues", <Red>);
 bip.removeFromWishlist("logistica@fe.up.pt", "Pocket Tissues", <Red>);
 bip.addToWishlist("logistica@fe.up.pt", "Pocket Tissues", <Blue>);
 bip.addToWishlist("logistica@fe.up.pt", "Pocket Tissues", <White>);

 bip.addToCart("logistica@fe.up.pt", "Pocket Tissues", <Blue>);
 bip.setQtyInCart("logistica@fe.up.pt", "Pocket Tissues", <Blue>, 35);
 bip.addToCart("logistica@fe.up.pt", "Pocket Tissues", <Red>);
 bip.removeFromCart("logistica@fe.up.pt", "Pocket Tissues", <Red>);

 bip.convertWishlist("logistica@fe.up.pt");

 IO`print(bip.getTotalCart("logistica@fe.up.pt"));
 IO`print("\n");

 bip.buy("logistica@fe.up.pt");
 IO`print(bip);
);*/

-- Test Create Client
/*private testCreateClient: () ==> ()
testCreateClient() == (
  dcl client: Client := new Client("exemplo@gmail.com");
  assertTrue(client.email = "exemplo@gmail.com");
  assertEqual({}, client.wishlist);
);*/



-- Test Create Manufacturer
/*private testCreateManufacturer: () ==> ()
testCreateManufacturer() == (
  dcl manufacturer: Manufacturer := new Manufacturer("RENOVA");
  assertEqual("RENOVA", manufacturer.name);
  assertEqual({ |-> }, manufacturer.products);
);*/

-- Test Create Product
/*private testCreateProduct: () ==> ()
testCreateProduct() == (
  assertTrue(product.title = "Pocket Tissues");

  assertTrue(product.description = "- 3-ply base sheet\n- 36x9 tissues per pack\n- Tissue size:
      21x21cm");
  assertTrue(product.subcategory = "Health & Personal Care");
  assertTrue(product.price = 1.23);
);*/

-- Test Set Admin Code
private static testSetAdminCode: () ==> ()
testSetAdminCode() == (
 dcl bip : BuyInPortugal := new BuyInPortugal();
 bip.setAdminCode("","1234");

 assertEqual("1234", bip.adminCode);
);
```

```
-- Test Category
private static testCategories: () ==> ()
testCategories() == (
 dcl bip : BuyInPortugal := new BuyInPortugal();
 assertEqual({}, bip.categories);
 bip.addCategory("Agriculture & Food","");

  assertTrue("Agriculture & Food" in set bip.categories);
  bip.setCategories({"Beauty & Health", "Books & Audible"},"");
 bip.addCategory("Real Estate","");
 assertEqual({"Beauty & Health", "Books & Audible", "Real Estate"}, bip.categories);
);

-- Test Add SubCategory
private static testAddSubCategory: () ==> ()
testAddSubCategory() == (
 dcl bip : BuyInPortugal := new BuyInPortugal();
 bip.addCategory("Real Estate","");
 assertTrue("Investment" not in set dom bip.subcategories);
 bip.addSubcategory("Investment", "Real Estate","");
  assertEqual("Real Estate", bip.subcategories("Investment"));

);

-- Test Register Manufacturer
private static testRegisterManufacturer: () ==> ()
testRegisterManufacturer() == (
 dcl bip : BuyInPortugal := new BuyInPortugal();
 assertTrue(testManufacturer.name not in set dom bip.manufacturers);
 bip.registerManufacturer(testManufacturer.name,"");
 assertTrue(testManufacturer.name in set dom bip.manufacturers);
);


-- Test Add Product
private static testAddProduct: () ==> ()
testAddProduct() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal1();
 assertTrue(testProduct.title not in set dom bip.products);
 bip.addProduct(testManufacturer.name, testProduct.title, testProduct.description, testProduct.
     subcategory, testProduct.price, {<None> |-> 0});
  assertTrue(testProduct.title in set dom bip.products);
  assertEqual(testProduct.title, bip.products(testProduct.title).title);
  assertEqual(testProduct.description, bip.products(testProduct.title).description);
  assertEqual(testProduct.subcategory, bip.products(testProduct.title).subcategory);
  assertEqual(testProduct.price, bip.products(testProduct.title).price);
  assertEqual({<None> |-> 0}, bip.products(testProduct.title).quantities);

  assertEqual({<None>}, bip.products(testProduct.title).colors);
);

-- Test Register Client
private static testRegisterClient: () ==> ()
testRegisterClient() == (
 dcl bip : BuyInPortugal := new BuyInPortugal();

 assertTrue(testClient.email not in set dom bip.clients);
 bip.registerClient(testClient.email);
 assertTrue(testClient.email in set dom bip.clients);

);


-- Test Add WishList
```

```
private static testAddWishList: () ==> ()
testAddWishList() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal2();
 let client = bip.clients(testClient.email)
 in(
  assertTrue(mk_(testProduct.title, <Red>) not in set client.wishlist);
  bip.addToWishlist(testClient.email, testProduct.title, <Red>);
  assertTrue(mk_(testProduct.title, <Red>) in set client.wishlist);
 );
);


-- Test Remove WishList
private static testRemoveWishList: () ==> ()
testRemoveWishList() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal2();
 let client = bip.clients(testClient.email)
 in(
  bip.addToWishlist(testClient.email, testProduct.title, <Red>);
  assertEqual({mk_(testProduct.title, <Red>)}, client.wishlist);
  bip.removeFromWishlist(testClient.email, testProduct.title, <Red>);
  assertTrue(mk_(testProduct.title, <Red>) not in set client.wishlist);
 );
);


-- Test Add Stock
private static testAddToStock: () ==> ()
testAddToStock() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal2();
 let product = bip.products(testProduct.title)
 in(
  assertTrue(product.quantities(<Blue>) = 0);
  bip.addToStock(testManufacturer.name, testProduct.title, <Blue>, 36);
  assertTrue(product.quantities(<Blue>) = 36);
 );
);


-- Test Remove from Stock
private static testRemoveFromStock: () ==> ()
testRemoveFromStock() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal2();
 let product = bip.products(testProduct.title)
 in(
  assertTrue(product.quantities(<Blue>) = 0);
  bip.addToStock(testManufacturer.name, testProduct.title, <Blue>, 36);
  bip.removeFromStock(testManufacturer.name, testProduct.title, <Blue>, 30);
  assertTrue(product.quantities(<Blue>) = 6);
 );
);


-- Test Add to Cart
private static testAddToCart: () ==> ()
testAddToCart() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal3();
 let client = bip.clients(testClient.email)
 in(
  assertTrue(mk_(testProduct.title, <Red>)  not in set dom client.cart);
  bip.addToCart(testClient.email, testProduct.title, <Red>);
  assertTrue(mk_(testProduct.title, <Red>) in set dom client.cart);
 );
);
```

```
-- Test Add Quantity to Cart
private static testQntAddToCart: () ==> ()
testQntAddToCart() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal3();
 let client = bip.clients(testClient.email)
 in(
  bip.addToCart(testClient.email, testProduct.title, <Blue>);
  assertTrue(client.cart(mk_(testProduct.title,<Blue>)) = 1);
  bip.setQtyInCart(testClient.email, testProduct.title, <Blue>, 35);
  assertTrue(client.cart(mk_(testProduct.title,<Blue>)) = 35);
 );
);

-- Test Remove from Cart
private static testRemoveFromCart: () ==> ()
testRemoveFromCart() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal3();
 let client = bip.clients(testClient.email)
 in(
  bip.addToCart(testClient.email, testProduct.title, <Red>);

  bip.removeFromCart(testClient.email, testProduct.title, <Red>);
  assertTrue(mk_(testProduct.title, <Red>) not in set dom client.cart);
 );
);

-- Test Convert WishList
private static testConvertWishList: () ==> ()
testConvertWishList() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal3();
 let client = bip.clients(testClient.email)

 in(
  bip.addToWishlist(testClient.email, testProduct.title, <Red>);
   bip.convertWishlist(testClient.email);
   assertTrue(mk_(testProduct.title, <Red>) in set dom client.cart);
   assertTrue(client.cart(mk_(testProduct.title,<Red>)) = 1);
   assertTrue(mk_(testProduct.title, <Red>) not in set client.wishlist);

  bip.setQtyInCart(testClient.email, testProduct.title, <Red>, 35);
  bip.addToWishlist(testClient.email, testProduct.title, <Red>);

   bip.convertWishlist(testClient.email);
   assertTrue(mk_(testProduct.title, <Red>) in set dom client.cart);
   assertTrue(client.cart(mk_(testProduct.title,<Red>)) = 35);
   assertTrue(mk_(testProduct.title, <Red>) not in set client.wishlist);
 );
);

-- Test Get Total Cart
private static testGetTotalCart: () ==> ()
testGetTotalCart() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal3();
 bip.addToCart(testClient.email, testProduct.title, <Blue>);
 bip.setQtyInCart(testClient.email, testProduct.title, <Blue>, 2);
 assertTrue(bip.getTotalCart(testClient.email) = 2.46);
 bip.addToStock(testManufacturer.name, testProduct.title, <Blue>, 1000);
 bip.setVolumeDiscounts(testManufacturer.name, testProduct.title, { 500 |-> 1, 750 |-> 0.5, 1000
      |-> 0.25});
 bip.setQtyInCart(testClient.email, testProduct.title, <Blue>, 500);
 assertEqual(500, bip.getTotalCart(testClient.email));
 bip.setQtyInCart(testClient.email, testProduct.title, <Blue>, 600);
 assertEqual(600, bip.getTotalCart(testClient.email));
 bip.setQtyInCart(testClient.email, testProduct.title, <Blue>, 800);
```

```
 assertEqual(400, bip.getTotalCart(testClient.email));

 bip.setQtyInCart(testClient.email, testProduct.title, <Blue>, 1000);
 assertEqual(250, bip.getTotalCart(testClient.email));
);

-- Test Remove Product
private static testRemoveProduct: () ==> ()
testRemoveProduct() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal3();
 bip.addToCart(testClient.email, testProduct.title, <Blue>);
 bip.addToCart(testClient.email, testProduct.title, <Red>);
 bip.addToWishlist(testClient.email, testProduct.title, <Red>);
 bip.removeProduct(testManufacturer.name, testProduct.title);
 assertEqual({ |-> }, bip.products);
 assertEqual({ |-> }, bip.manufacturers(testManufacturer.name).products);
 assertEqual({ }, bip.clients(testClient.email).wishlist);
 assertEqual({ |-> }, bip.clients(testClient.email).cart);
);

-- Test Remove Product
private static testBuy: () ==> ()
testBuy() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal3();
 bip.addToCart(testClient.email, testProduct.title, <Blue>);
 bip.setQtyInCart(testClient.email, testProduct.title, <Blue>, 35);
 bip.addToCart(testClient.email, testProduct.title, <White>);
 bip.buy(testClient.email);
 assertEqual([{mk_(testProduct.title, <Blue>) |-> 35, mk_(testProduct.title, <White>) |-> 1}],
     bip.clients(testClient.email).buyHistory);
 assertEqual({ |-> }, bip.clients(testClient.email).cart);
 bip.addToCart(testClient.email, testProduct.title, <Blue>);
 bip.buy(testClient.email);
 assertEqual([{mk_(testProduct.title, <Blue>) |-> 1},{mk_(testProduct.title, <Blue>) |-> 35, mk_
     (testProduct.title, <White>) |-> 1}], bip.clients(testClient.email).buyHistory);
);


-- Test Remove Product

private static testSearch: () ==> ()
testSearch() == (
 dcl bip : BuyInPortugal := configuredBuyInPortugal4();
 let product = bip.searchProductsByTitle(testProduct.title)
 in (
  assertEqual(testProduct.description, product.description);
 );
 let products = bip.searchProductsByCategory("Beauty & Health")
 in (
  assertTrue(card products = 1);
 );
 let products = bip.searchProductsBySubcategory("Dairy & Eggs")
 in (
  assertTrue(card products = 2);
 );
 let products = bip.searchProductsByManufacturer("DANONE")
 in (
  assertTrue(card products = 2);
 );
);
-- Entry point that runs all tests with valid inputs
public static testAll: () ==> ()
testAll() ==
(
```

```
    /*IO‘print("Create Client: " );
     testCreateClient();
     IO‘println("Finish");

     IO‘print("Create Manufacturer: " );
     testCreateManufacturer();
     IO‘println("Finish");

     IO‘print("Create Product: " );
     testCreateProduct();
     IO‘println("Finish");*/

     IO‘print("Set admin code: " );
     testSetAdminCode();
     IO‘println("Finish");

     IO‘print("Add Category: " );
     testCategories();
     IO‘println("Finish");

     IO‘print("Add SubCategory: " );
     testAddSubCategory();
     IO‘println("Finish");

     IO‘print("Register Manufacturer: " );
     testRegisterManufacturer();
     IO‘println("Finish");

    IO‘print("Add Product: " );
     testAddProduct();
     IO‘println("Finish");

     IO‘print("Register Client: " );
     testRegisterClient();
     IO‘println("Finish");

     IO‘print("Add to Stock: " );
     testAddToStock();
     IO‘println("Finish");

     IO‘print("Remove from Stock: " );
     testRemoveFromStock();
     IO‘println("Finish");

     IO‘print("Add Wish List: " );
     testAddWishList();
     IO‘println("Finish");

    IO‘print("Add Remove Wish List: " );
     testRemoveWishList();
     IO‘println("Finish");

     IO‘print("Add To Cart: " );
     testAddToCart();
     IO‘println("Finish");

     IO‘print("Add Qty To Cart: " );
     testQntAddToCart();
     IO‘println("Finish");

    IO‘print("Remove from Cart: " );
     testRemoveFromCart();
     IO‘println("Finish");

     IO‘print("Convert Wish List: " );
```

```
    testConvertWishList();
    IO`println("Finish");

   IO`print("Get Total Cart: " );
    testGetTotalCart();
    IO`println("Finish");

   IO`print("Buy: " );
    testBuy();
    IO`println("Finish");

   IO`print("Remove Product: " );
    testRemoveProduct();
    IO`println("Finish");

   IO`print("Search: " );
    testSearch();
    IO`println("Finish");
  );


 /** TEST CASES WITH VALID INPUTS END **/

end TestBuyInPortugal
```

Function or operation	Line	Coverage	Calls
configuredBuyInPortugal1	26	100.0%	13
configuredBuyInPortugal2	61	100.0%	126
configuredBuyInPortugal3	73	100.0%	14
configuredBuyInPortugal4	84	100.0%	2
testAddProduct	213	100.0%	1
testAddSubCategory	194	100.0%	2
testAddToCart	288	100.0%	3
testAddToStock	263	100.0%	1
testAddWishList	238	100.0%	1
testAll	356	100.0%	1
testBuy	387	100.0%	1
testCategories	182	100.0%	1
testConvertWishList	325	100.0%	1
testGetTotalCart	346	100.0%	8
testQntAddToCart	300	100.0%	1
testRegisterClient	227	100.0%	1
testRegisterManufacturer	204	100.0%	1
testRemoveFromCart	313	100.0%	2
testRemoveFromStock	275	100.0%	4
testRemoveProduct	365	100.0%	1
testRemoveWishList	250	100.0%	8
testSearch	422	100.0%	1
testSetAdminCode	182	100.0%	1
TestBuyInPortugal.vdmpp		100.0%	195