

# feup-mfes

January 2, 2018

## Contents

<b>1</b>	<b>BuyInPortugal</b>	<b>1</b>
<b>2</b>	<b>Client</b>	<b>6</b>
<b>3</b>	<b>Manufacturer</b>	<b>8</b>
<b>4</b>	<b>MyTestCase</b>	<b>9</b>
<b>5</b>	<b>Product</b>	<b>10</b>
<b>6</b>	<b>TestBuyInPortugal</b>	<b>11</b>

## 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) == (
  decl 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};

-- 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;

/** 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;
  );
);

-- 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 resultProducts: set of Product := {};
  resultProducts := rng manufacturers(man).products;
  return resultProducts;
)
pre man in set dom manufacturers;

-- Get products by subcategory

public searchProductBySubcategory: Subcategory ==> set of Product
searchProductBySubcategory(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 searchProductByCategory: Category ==> set of Product
searchProductByCategory(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%	57
addCategory	49	100.0%	26
addClient	227	100.0%	28
addManufacturer	43	100.0%	31
addProduct	87	100.0%	20
addSubcategory	65	100.0%	15
addToCart	146	100.0%	26
addToStock	103	100.0%	26
addToWishlist	128	100.0%	39
buy	187	100.0%	13
convertWishlist	205	100.0%	13
getTotalCart	173	100.0%	13
registerClient	219	100.0%	28
registerManufacturer	34	100.0%	31
removeFromCart	164	100.0%	26
removeFromStock	113	0.0%	0
removeFromWishlist	137	100.0%	13
searchProductByCategory	264	0.0%	0
searchProductBySubcategory	250	0.0%	0
searchProductsByManufacturer	241	0.0%	0
searchProductsByTitle	233	0.0%	0
setAdminCode	27	0.0%	0
setCategories	58	100.0%	13
setQtyInCart	155	100.0%	13
setSubcategories	75	100.0%	13
BuyInPortugal.vdmpp		77.3%	444

## 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%	54
addToCart	36	100.0%	39
addToWishlist	22	100.0%	39
convertWishlist	68	100.0%	13
pushCartToHistory	58	100.0%	13
removeFromCart	43	100.0%	26
removeFromWishlist	29	100.0%	13
setQtyInCart	50	100.0%	13
Client.vdmpp		100.0%	210

### 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;

end Manufacturer

```

Function or operation	Line	Coverage	Calls
Manufacturer	10	100.0%	57
addProduct	17	100.0%	26
Manufacturer.vdmpp		100.0%	83

## 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 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 assertEquals: ? * ? ==> ()
  assertEquals(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
assertEquals	20	0.0%	0
assertTrue	12	0.0%	0

MyTestCase.vdmpp		0.0%	0
------------------	--	------	---

## 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 > 1 <=> <None> not in set colors;
  inv dom quantities = colors;

operations
  -- Create product without color

  public Product : Title * Description * Subcategory * Price ==> Product
  Product(tit, des, cat, pr) == (
    subcategory := cat;
    title := tit;
    description := des;
    price := pr;
    return self;
  );

  -- 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;
  );

  -- 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	26	100.0%	26
addToStock	62	100.0%	26
getPriceWithDiscount	69	39.2%	26
removeFromStock	54	100.0%	26
setVolumeDiscounts	48	0.0%	0
Product.vdmpp		71.0%	104

## 6 TestBuyInPortugal

```

class TestBuyInPortugal
/* Test cases for BuyInPortugal model*/

types
  public Category = seq of char;

instance variables
  public category: Category := "Real Estate";
  public manufacturer : Manufacturer := new Manufacturer("RENOVA");
  public product: 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 client : Client := new Client("exemplo@gmail.com");

```

## operations

```

/** TEST CASES WITH VALID INPUTS **/

```

```

public static test: () ==> ()
test() == (
  decl 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);
);

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

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

-- Test Create Client

private testCreateClient: () ==> ()
testCreateClient() == (
  dcl client: Client := new Client("exemplo@gmail.com");
  Assert(client.email = "exemplo@gmail.com");
  Assert(client.wishlist = {});
);

-- Test Create Manufacturer

private testCreateManufacturer: () ==> ()
testCreateManufacturer() == (
  dcl manufacturer: Manufacturer := new Manufacturer("RENOVA");
  Assert(manufacturer.name = "RENOVA");
);

-- Test Add Category

private testAddCategory: () ==> ()
testAddCategory() == (
  dcl bip : BuyInPortugal := new BuyInPortugal();
  Assert(category not in set bip.categories);

```

```

    bip.addCategory(category,"");
    Assert(category in set bip.categories);
);

-- Test Add SubCategory

private testAddSubCategory: () ==> ()
testAddSubCategory() == (
    dcl bip : BuyInPortugal := new BuyInPortugal();
    bip.addSubcategory("Investment", "Real Estate","");
    if "Real Estate" not in set bip.categories then
        bip.addCategory("Real Estate","");
    Assert("Investment" not in set dom bip.subcategories);
    bip.addSubcategory("Investment", "Real Estate","");
    Assert("Investment" in set dom bip.subcategories);
);

-- Test Register Manufacturer

private testRegisterManufacturer: () ==> ()
testRegisterManufacturer() == (
    dcl bip : BuyInPortugal := new BuyInPortugal();

    Assert(manufacturer.name not in set dom bip.manufacturers);
    bip.registerManufacturer(manufacturer.name,"");
    Assert(manufacturer.name in set dom bip.manufacturers);
);

-- Test Create Product

private testCreateProduct: () ==> ()
testCreateProduct() == (
    Assert(product.title = "Pocket Tissues");
    Assert(product.description = "- 3-ply base sheet\n- 36x9 tissues per pack\n- Tissue size: 21
    x21cm");
    Assert(product.subcategory = "Health & Personal Care");
    Assert(product.price = 1.23);
);

-- Test Add Product

private testAddProduct: () ==> ()
testAddProduct() == (
    dcl bip : BuyInPortugal := new BuyInPortugal();
    Assert(product.title not in set dom bip.products);
    bip.addProduct("",product.title,product.description, product.subcategory, product.price,
    product.quantities);
    Assert(product.title in set dom bip.products);
);

-- Test Register Client

private testRegisterClient: () ==> ()
testRegisterClient() == (
    dcl bip : BuyInPortugal := new BuyInPortugal();

    Assert(client.email not in set dom bip.clients);
    bip.registerClient(client.email);
    Assert(client.email in set dom bip.clients);
);

-- Test Add WishList

```

```

private testAddWishList: () ==> ()
testAddWishList() == (
  dcl bip : BuyInPortugal := new BuyInPortugal();
  bip.registerManufacturer(manufacturer.name, "");
  bip.registerClient("logistica@fe.up.pt");
  bip.addProduct(manufacturer.name, product.title, product.description, product.subcategory,
    product.price, product.quantities);
  let client1 = bip.clients("logistica@fe.up.pt")
  in (
    Assert(mk_("Pocket Tissues", <Red>) not in set client1.wishlist);
  );
  bip.addToWishlist("logistica@fe.up.pt", "Pocket Tissues", <Red>);
  let client1 = bip.clients("logistica@fe.up.pt")
  in (
    Assert(mk_("Pocket Tissues", <Red>) in set client1.wishlist);
  );
);

-- Test Add Stock

private testAddToStock: () ==> ()
testAddToStock() == (
  dcl bip : BuyInPortugal := new BuyInPortugal();
  bip.registerManufacturer(manufacturer.name, "");
  bip.addProduct(manufacturer.name, product.title, product.description, product.subcategory,
    product.price, product.quantities);
  let product1 = bip.products("Pocket Tissues")
  in (
    Assert(product1.quantities(<Blue>) = 0);
  );
  bip.addToStock("RENOVA", "Pocket Tissues", <Blue>, 36);
  let product1 = bip.products("Pocket Tissues")
  in (
    Assert(product1.quantities(<Blue>) = 36);
  );
);

-- Test Remove WishList

private testRemoveWishList: () ==> ()
testRemoveWishList() == (
  dcl bip : BuyInPortugal := new BuyInPortugal();
  bip.registerManufacturer(manufacturer.name, "");
  bip.registerClient("logistica@fe.up.pt");
  bip.addProduct(manufacturer.name, product.title, product.description, product.subcategory,
    product.price, product.quantities);
  bip.addToWishlist("logistica@fe.up.pt", "Pocket Tissues", <Red>);
  let client1 = bip.clients("logistica@fe.up.pt")
  in (
    if mk_("Pocket Tissues", <Red>) not in set client1.wishlist then
      bip.addToWishlist("logistica@fe.up.pt", "Pocket Tissues", <Red>);
  );
  bip.removeFromWishlist("logistica@fe.up.pt", "Pocket Tissues", <Red>);
  let client1 = bip.clients("logistica@fe.up.pt")
  in (
    Assert(mk_("Pocket Tissues", <Red>) not in set client1.wishlist);
  );
);

-- Test Add to Cart

private testAddToCart: () ==> ()
testAddToCart() == (
  dcl bip : BuyInPortugal := new BuyInPortugal();
  bip.registerManufacturer(manufacturer.name, "");

```

```

bip.registerClient("logistica@fe.up.pt");
bip.addProduct(manufacturer.name,product.title,product.description, product.subcategory,
    product.price, product.quantities);
let client1 = bip.clients("logistica@fe.up.pt")
in(
    Assert(mk_("Pocket Tissues", <Red>) not in set dom client1.cart);
);
bip.addToCart("logistica@fe.up.pt", "Pocket Tissues", <Red>);
let client1 = bip.clients("logistica@fe.up.pt")
in(
    Assert(mk_("Pocket Tissues", <Red>) in set dom client1.cart);
);
);

-- Test Add Quantity to Cart

private testQntAddToCart: () ==> ()
testQntAddToCart() == (
    decl bip : BuyInPortugal := new BuyInPortugal();
    bip.registerManufacturer(manufacturer.name, "");
    bip.registerClient("logistica@fe.up.pt");
    bip.addProduct(manufacturer.name,product.title,product.description, product.subcategory,
        product.price, product.quantities);
    bip.addToCart("logistica@fe.up.pt", "Pocket Tissues", <Blue>);
    let client1 = bip.clients("logistica@fe.up.pt")
    in(
        Assert(client1.cart(mk_("Pocket Tissues", <Blue>)) = 1);
    );
    bip.setQtyInCart("logistica@fe.up.pt", "Pocket Tissues", <Blue>, 35);
    let client1 = bip.clients("logistica@fe.up.pt")
    in(
        Assert(client1.cart(mk_("Pocket Tissues", <Blue>)) = 35);
    );
);

-- Test Remove from Cart

private testRemoveFromCart: () ==> ()
testRemoveFromCart() == (
    decl bip : BuyInPortugal := new BuyInPortugal();
    bip.registerManufacturer(manufacturer.name, "");
    bip.registerClient("logistica@fe.up.pt");
    bip.addProduct(manufacturer.name,product.title,product.description, product.subcategory,
        product.price, product.quantities);
    let client1 = bip.clients("logistica@fe.up.pt")
    in(
        if mk_("Pocket Tissues", <Red>) not in set dom client1.cart then
            bip.addToCart("logistica@fe.up.pt", "Pocket Tissues", <Red>);
    );
    bip.removeFromCart("logistica@fe.up.pt", "Pocket Tissues", <Red>);
    let client1 = bip.clients("logistica@fe.up.pt")
    in(
        Assert(mk_("Pocket Tissues", <Red>) not in set dom client1.cart);
    );
);

-- Test Convert WishList

private testConvertWishList: () ==> ()
testConvertWishList() == (
    decl bip : BuyInPortugal := new BuyInPortugal();
    bip.registerManufacturer(manufacturer.name, "");
    bip.registerClient("logistica@fe.up.pt");
    bip.addProduct(manufacturer.name,product.title,product.description, product.subcategory,
        product.price, product.quantities);

```



```

bip.addToWishlist("logistica@fe.up.pt", "Pocket Tissues", <Red>);
let client1 = bip.clients("logistica@fe.up.pt")
in (
  if mk_("Pocket Tissues", <Red>) not in set dom client1.cart and mk_("Pocket Tissues", <Red>)
    in set client1.wishlist then
    bip.convertWishlist("logistica@fe.up.pt");
    Assert(mk_("Pocket Tissues", <Red>) in set dom client1.cart);
    Assert(client1.cart(mk_("Pocket Tissues", <Red>)) = 1);
    Assert(mk_("Pocket Tissues", <Red>) not in set client1.wishlist);
);
let client1 = bip.clients("logistica@fe.up.pt")
in (
  if mk_("Pocket Tissues", <Red>) in set dom client1.cart and mk_("Pocket Tissues", <Red>) in
    set client1.wishlist then
    let tmp = client1.cart(mk_("Pocket Tissues", <Red>))
    in (
      bip.convertWishlist("logistica@fe.up.pt");
      Assert(mk_("Pocket Tissues", <Red>) in set dom client1.cart);
      Assert(client1.cart(mk_("Pocket Tissues", <Red>)) = tmp);
      Assert(mk_("Pocket Tissues", <Red>) not in set client1.wishlist);
    );
);

-- Test Get Total Cart

private testGetTotalCart: () ==> ()
testGetTotalCart() == (
  dcl bip : BuyInPortugal := new BuyInPortugal();
  bip.registerManufacturer(manufacturer.name, "");
  bip.registerClient("logistica@fe.up.pt");
  bip.addProduct(manufacturer.name, product.title, product.description, product.subcategory,
    product.price, product.quantities);
  bip.addToCart("logistica@fe.up.pt", "Pocket Tissues", <Blue>);
  bip.setQtyInCart("logistica@fe.up.pt", "Pocket Tissues", <Blue>, 2);
  Assert(bip.getTotalCart("logistica@fe.up.pt") = 2.46);
);

-- Entry point that runs all tests with valid inputs

public static testAll: () ==> ()
testAll() == (
  dcl test: TestBuyInPortugal := new TestBuyInPortugal();

  test();
  test.testCreateClient();
  test.testCreateManufacturer();
  test.testAddCategory();
  --test.testAddSubCategory();
  test.testRegisterManufacturer();
  test.testCreateProduct();
  --test.testAddProduct();
  test.testRegisterClient();
  --test.testAddToStock();
  --test.testAddWishList();
  --test.testRemoveWishList();
  --test.testAddToCart();
  --test.testQtyAddToCart();
  --test.testRemoveFromCart();
  --test.testConvertWishList();
  --test.testGetTotalCart();
);

```

```

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

end TestBuyInPortugal

```

Function or operation	Line	Coverage	Calls
Assert	101	100.0%	302
AssertEqual	105	0.0%	0
test	30	100.0%	13
testAddCategory	132	100.0%	13
testAddProduct	173	75.8%	2
testAddSubCategory	141	22.5%	0
testAddToCart	248	0.0%	0
testAddToStock	211	42.5%	0
testAddWishList	193	44.0%	8
testAll	345	100.0%	11
testConvertWishList	304	0.0%	0
testCreateClient	117	100.0%	13
testCreateManufacturer	125	100.0%	13
testCreateProduct	164	100.0%	11
testGetTotalCart	332	0.0%	0
testQntAddToCart	266	0.0%	0
testRegisterClient	182	100.0%	18
testRegisterManufacturer	153	100.0%	11
testRemoveFromCart	285	0.0%	0
testRemoveWishList	228	37.9%	0
TestBuyInPortugal.vdmpp		44.5%	415