CartSystem.java:

```
package shoppingCart.system;

import java.math.BigDecimal;

import shoppingCart.gui.UI;
import shoppingCart.model.Inventory;
import shoppingCart.model.UserList;

/** The entry point to the Shopping Cart application.
 *  The application performs different functions depending on who logs in.
 *  It allows a seller to maintain an inventory of items available for sale
 *  and customers to browse and add items to their cart, and purchase the contents of their cart.
 *
 *  CartSystem manages interactions between the UI and the DBManager, PaymentValidator, and UserList.
 *  It also creates the same.
 *  @authorNewmanSouza
 *  @authorSethMoore
 */
publicclass CartSystem {

/** Constructs a CartSystem object.
    *  @precondition                Appropriate files are available for DBManager
    *  @postcondition               object created
    *  @postcondition               UI created and running
    */
public CartSystem() {
            dbManager = new DBManager();
            paymentValidator = new PaymentValidator();
            UI ui = new UI(this);
            userList = new UserList();
                        userList = dbManager.loadUserList();
   }

/** Creates a CartSystem
    *  @param args not used
    */
            publicstaticvoid main(String[] args) {
                        CartSystem cartSystem = new CartSystem();
            }

/** Takes a user's username and password, and, if they are a valid username/password pair,
    *  returns the User's type to the caller and loads the Inventory.
    *  Otherwise returns null.
    *  @param username              the User's username
    *  @param password              the User's password
    *  @return                      the User's type or null
    *  @precondition                username and password are valid references
    */
public String login(String username, String password) {
            String type = userList.validate(username, password);
            if (type != null) {
                Inventory inventory = Inventory.getInstance();   // Inventory is a singleton, getInstance() guarantees that
                inventory = dbManager.loadInventory();        //   this same instance will always be used when it gets called.
            }
            return type;
   }
```

```
/** Processes payment for for items in Customer's cart.
   * @param                    the customer's payment information
   * @return                   true if successful and false otherwise
   * @precondition             none
   * @postcondition            payment processed
   */
publicboolean pay(String cardNumber, BigDecimal total) {
        boolean result = paymentValidator.validate(cardNumber, total);
        if (result) {
                saveInventory();
        }
        return result;
  }

/** Saves the current state of the Inventory.
   * @precondition             appropriate file permissions are available to DBManager.
   */
publicvoid saveInventory() {
        Inventory inventory = Inventory.getInstance();
                dbManager.saveInventory(inventory);
  }

private DBManager dbManager;
private PaymentValidator paymentValidator;
private UserList userList;

}
```

DBManager.java:

```
package shoppingCart.system;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

import javax.swing.JOptionPane;

import shoppingCart.model.Inventory;
import shoppingCart.model.UserList;

/** A class that manages the Shopping Cart database.
 * @authorNewmanSouza
 * @authorSethMoore
 */
publicclass DBManager {

        private String inventorySaveFile = "database\\Inventory.dat";
        private String userListSaveFile = "database\\UserList.dat";

/** Loads Inventory from database.
   * @return                   The Seller's inventory
   * @precondition             Database file is valid and available in file system
   * @postcondition            The Seller's inventory is loaded
```

```
   */
public Inventory loadInventory() {
        Inventory inventory = null;
        try {
                ObjectInputStream in = new ObjectInputStream (new FileInputStream(inventorySaveFile));
                inventory = (Inventory)in.readObject();
                in.close();
            } catch (IOException | ClassNotFoundException e) {
                JOptionPane.showMessageDialog(null, "Inventory database not found.");
                System.exit(0);
            }
            return inventory;
  }

/** Saves Inventory to database.
   *  @param inventory      The Seller's inventory
   *  @precondition         inventory is a valid reference
   *  @postcondition        The Seller's inventory is saved to file
   */
publicvoid saveInventory(Inventory inventory) {
        try {
      ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(inventorySaveFile));
      out.writeObject(inventory);
      out.close();
            } catch (IOException e) {
                JOptionPane.showMessageDialog(null, "Inventory database not found.");
                System.exit(0);
            }
  }

/** Loads the list of users from database.
   *  @return               The list of users
   *  @precondition         Database file is valid and available in file system
   *  @postcondition        The list of users is loaded
   */
public UserList loadUserList() {
        UserList fromFile = null;
        try {
                ObjectInputStream in = new ObjectInputStream (new FileInputStream(userListSaveFile));
                fromFile = (UserList)in.readObject();
                in.close();
            } catch (IOException | ClassNotFoundException e) {
                JOptionPane.showMessageDialog(null, "UserList database not found.");
                System.exit(0);
            }
            return fromFile;
  }

/** Saves the list of users to database.
   *  @param userList       The list of users
   *  @precondition         userList is a valid reference
   *  @postcondition        The list of users is saved to file
   */
publicvoid saveUserList(UserList userList) {
        try {
            ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(userListSaveFile));
            out.writeObject(userList);
            out.close();
```

```
		} catch (IOException e) {
			JOptionPane.showMessageDialog(null, "UserList database not found.");
			System.exit(0);
		}
	}

/** Method used for testing purposes. Calls loadInventory().
	* @param filename		the filename to be loaded from
	* @return			the Inventory saved in filename
	*/
public Inventory loadInventory(String filename) {
		inventorySaveFile = filename;
		return loadInventory();
	}

/** Method used for testing purposes. Calls saveInventory(Inventory)
	*@param inventory		the Inventory to be saved
	*@param filename		the filename to be saved to
	*/
publicvoid saveInventory(Inventory inventory, String filename) {
		inventorySaveFile = filename;
		saveInventory(inventory);
	}

/** Method used for testing purposes. Calls loadUserList()
	* @param filename		the filename to be loaded from
	* @return				the UserList saved in filename
	*/
public UserList loadUserList(String filename) {
		userListSaveFile = filename;
		return loadUserList();
	}

/** Method used for testing purposes. Calls saveUserList(UserList)
	* @param userList		the UserList to be saved
	* @param filename		the filename to be saved to
	*/
publicvoid saveUserList(UserList userList, String filename) {
		userListSaveFile = filename;
		saveUserList(userList);
	}

}
```

PaymentValidator.java:

```
package shoppingCart.system;

import java.math.BigDecimal;

/** A class that validates payments for the Shopping Cart application.
 * @authorNewmanSouza
 * @authorSethMoore
 */
```

```
publicclass PaymentValidator {

/** Validates a payment.
    *  @param cardNumber    the customer's credit card number.
    *  @param total         the monetary amount being validated
    *  @return true         if the payment is approved and false otherwise
    *  @precondition        none
    *  @postcondition       payment validated
    */
publicboolean validate(String cardNumber, BigDecimal total) {
            returntrue;
   }

}
```

ProductList.java:

```
package shoppingCart.model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/** A class that manages a list of products.
 *
 * Subject in Observer Pattern
 *
 *  @authorNewmanSouza
 *  @authorSethMoore
 */
@SuppressWarnings("serial")
publicabstractclass ProductList implements Iterable<Product>, Serializable{

/** Constructs a ProductList object.
    *  @precondition                    none
    *  @postcondition                   object created
    */
public ProductList() {
            products = new ArrayList<>();
            listeners = new ArrayList<>();
   }

/** Removes all the Products from this ProductList.
    *  @precondition                    none
    *  @postcondition                   iterator().hasNext() == false
    */
publicvoid clear() {

            products.clear();
            notifyListeners();
   }
```

```
/** Increments, by one, the quantity of a Product that equals the supplied Product.
   *  @param product                  the Product whose matching Product will be incremented.
   *  @precondition                   none
   *  @postcondition                  getMatchingProduct(product).getQuantity() > 0
   */
publicvoid increment(Product product) {

        for (Product p : products) {
                if (p.equals(product)){
                        p.increment();
                notifyListeners();
                        return;
                }
        }
        // If product wasn't already in this ProductList, create a clone
        // (which will have quantity == 0), increment the clone, and add
        // to this ProductList.
        Product p = (Product)product.clone();
        p.increment();
        add(p);
        notifyListeners();
  }

/** Decrements, by one, the quantity of a Product that equals the supplied Product.
   *  @param product                  the Product whose matching Product will be decremented.
   *  @precondition                   getMatchingProduct(product) != null
   *  @precondition                   getMatchingProduct(product).getQuantity() > 0
   *  @postcondition                  getMatchingProduct(product).getQuantity() >= 0
   */
publicvoid decrement(Product product) {

        for (Product p : products) {
                if (p.equals(product)){
                        p.decrement();
                        break;
                }
        }
        notifyListeners();
  }

/**
   * Gets the Product in this ProductList that equals the supplied Product,
   * if it exists, else returns null.
   *
   * @param product                  the Product whose matching product will be retrieved
   * @return                                 the matching Product, or null if matching product not found
   */
public Product getMatchingProduct(Product product){
        for (Product p : products) {
                if (p.equals(product)){
                        return p;
                }
        }
        returnnull;
  }
```

```
/**
   * Adds a copy (including quantity) of the supplied Product to the ProductList.
   *  @param product                  the Product whose copy will be added to this ProductList
   *  @precondition                   getMatchingProduct(product) == null
   *  @postcondition                  getMatchingProduct(product).equals(product) == true
   *  @postcondition                  getMatchingProduct(product).getQuantity() == product.getQuantity()
   */
publicvoid add(Product product) {

            Product p = new Product(product.getID(),
            product.getName(),
            product.getDescription(),
            product.getInvoicePrice(),
            product.getSellPrice(),
            product.getQuantity());
            products.add(p);
            notifyListeners();
  }

/** Removes a Product that matches supplied Product from the ProductList
   *  @param product                  whose match will be removed from this ProductList
   *  @precondition                   getMatchingProduct(product) != null
   *  @postcondition                  getMatchingProduct(product) == null
   */
publicvoid remove(Product product) {

            Iterator<Product> it = products.iterator();
            while (it.hasNext()) {
                        Product p = (Product) it.next();
                        if (p.equals(product)){
                                    it.remove();
                        }
            }
            notifyListeners();
  }

/** Gets an iterator over the Products in this ProductList.
   *  @return                         an iterator over the Products in an umodifiableList
   *                                  created from the Products in this ProductList
   */
public Iterator<Product> iterator() {
                        return Collections.unmodifiableList(products).iterator();
  }

/** Adds a ChangeListener to the ChangeListeners that will be notified
 *  whenever this ProductList changes state.
*  @param listener              the ChangeListener to add
 */
publicvoid addListener(ChangeListener listener){
            listeners.add(listener);
}

/** Notifies the listeners that the state has changed.
 *  @precondition               none
 *  @postcondition              all ChangeListeners in listeners have been notified
 *                                                      that the state has changed.
 */
```

```
protectedvoid notifyListeners(){
        for (ChangeListener listener : listeners){
                listener.stateChanged(new ChangeEvent(this));
        }
}

private ArrayList<Product>products;
privatetransient ArrayList<ChangeListener>listeners;
}
```

Inventory.java:

```
package shoppingCart.model;

import java.math.BigDecimal;
import java.util.Iterator;

/**
 * A ProductList that manages the Products in the Seller's inventory and the
 * seller's financial state.
 *
 * ConcreteAggregate in Iterator Pattern
 * Singleton in Singleton Pattern
 *
 * @authorSethMoore
 * @authorNewmanSouza
 */
@SuppressWarnings("serial")
publicclass Inventory extends ProductList {

        /**
         * Constructor
         */
        private Inventory(){
                super();
                costs = new BigDecimal("0.00");
                revenues = new BigDecimal("0.00");
        }

        /**
         * The accessor method for obtaining the single (there can
         * only be one) instance of Inventory.
         *
         * @return the Inventory instance
         */
        publicstatic Inventory getInstance(){
                if (instance == null){
                        instance = new Inventory();
                }
                returninstance;
        }
```

```
		/**
		 * Accessor method for costs
		 *
		 * @return the seller's costs
		 */
		public BigDecimal getCosts(){
			returncosts;
		}

		/**
		 * Accessor method for revenues
		 *
		 * @return the seller's revenues
		 */
		public BigDecimal getRevenues(){
			returnrevenues;
		}

		/**
		 * Calculates and returns the seller's profits, based on revenues - costs
		 *
		 * @return the seller's profits
		 */
		public BigDecimal getProfits(){
			returnrevenues.subtract(costs);
		}

/**
 * Decrements, by one, the quantity of a Product that equals the
   * supplied Product, and increases revenues by the Product's sellPrice.
   * @param product the Product whose matching Product will be incremented.
   * @precondition none
   * @postcondition getMatchingProduct(product).getQuantity() > 0
		 */
		publicvoid decrement(Product product){
			super.decrement(product);
			revenues = revenues.add(product.getSellPrice());
		}

/**
 * Increments, by one, the quantity of a Product that equals the
   * supplied Product, and decreases revenues by the Product's sellPrice.
   *
   *  @param product                the Product whose matching Product will be decremented.
   *  @precondition                 getMatchingProduct(product) != null
   *  @precondition                 getMatchingProduct(product).getQuantity() > 0
   *  @postcondition                getMatchingProduct(product).getQuantity() >= 0
		 */
		publicvoid increment(Product product){
			super.increment(product);
			revenues = revenues.subtract(product.getSellPrice());
		}

/**
   * Creates and returns a new integer that is equal to p.getID()+1,
   * where p is the Product in Inventory with greatest ID.
   *
   * @return the largest ID in the inventory plus 1.
   */
```

```
publicint getNewID() {
        int newID = 0;
        Iterator<Product> iterator = iterator();
        while (iterator.hasNext()) {
                Product p = (Product) iterator.next();
                if (p.getID() > newID) {
                        newID = p.getID();
                }
        }
        return (newID + 1);
  }

        /**
         * Adds a copy (including quantity) of the supplied Product to the
   * ProductList, and increases costs by the invoicePrice times quantity
   * of the Product.
   *
   *  @param product the Product whose copy will be added to this ProductList
   *
   *  @precondition getMatchingProduct(product) == null
   *  @postcondition getMatchingProduct(product).equals(product) == true
   *  @postcondition getMatchingProduct(product).getQuantity() == product.getQuantity()
         */
        @Override
        publicvoid add(Product product){
                costs = costs.add(product.getInvoicePrice().multiply(BigDecimal.valueOf(product.getQuantity())));
                super.add(product);
        }

        /**
         * Updates all the fields of the product in Inventory whose ID matches
         * the ID of product to match the fields in product. If product's quantity
         * is greater than the quantity of the match, costs is increased by the
         * difference times the InvoicePrice of product.
         *
         * @param product the Product whose match will be found and updated
         * @precondition getMatchingProduct(product) != null
         */
        publicvoid update(Product product){
                Product p = getMatchingProduct(product);
                int oldQuantity = p.getQuantity();
                int newQuantity = product.getQuantity();
                if (oldQuantity < newQuantity){
                costs = costs.add(product.getInvoicePrice().multiply(BigDecimal.valueOf(newQuantity - oldQuantity)));
                }
                p.update(product.getID(), product.getName(), product.getDescription(), product.getSellPrice(),
                        product.getInvoicePrice(), product.getQuantity());
                notifyListeners();
        }

        /**
         * Removes all the Products in Inventory, and sets costs and revenues to 0.00
         *
         * @postconditioniterater().hasNext == false
         * @postcondition getCosts().equals(new BigDecimal("0.00")
         * @postcondition getRevenues().equals(new BigDecimal("0.00")
         */
```

```
	publicvoid clear() {
		super.clear();
		costs = new BigDecimal("0.00");
		revenues = new BigDecimal("0.00");
	}

	/**
	 * This method makes the Singleton Pattern play nicely with
	 * deserialization in our application. Since deserialization creates
	 * a new object, we clear the old instance, and add all the Products
	 * from the deserialized object (this) to the instance, then return
	 * the instance, rather than this.
	 *
	 * @return the Inventory instance
	 */
	private Object readResolve(){
		if (instance == null){
			instance = this;
		}
		else{
			instance.clear();
			for (Product p : this) {
				instance.add(p);
			}
			instance.costs = this.costs;
			instance.revenues = this.revenues;
		}
		instance.notifyListeners();
		returninstance;
	}

	private BigDecimal costs;
	private BigDecimal revenues;
	privatestatic Inventory instance = null;
}
```

Cart.java:

```
package shoppingCart.model;

import java.math.BigDecimal;

/**
 * A ProductList that manages the Products in a customer's shopping cart.
 *
 * ConcreteAggregate in Iterator Pattern
 * Singleton in Singleton Pattern
 *
 *  @authorNewmanSouza
 *  @authorSethMoore
 */
@SuppressWarnings("serial")
publicclass Cart extends ProductList {
```

```
	/**
	 * Constructor
	 */
	private Cart(){
		super();
	}

	/**
	 * The accessor method for obtaining the single (there can
	 * only be one) instance of Cart.
	 *
	 * @return the Cart instance
	 */
	publicstatic Cart getInstance(){
		if (instance == null){
			instance = new Cart();
		}
		returninstance;
	}

	/**
	 * Gets the Cart's total, which is the sum of the sell price times quantity
	 * of each Product in the cart.
	 *
	 * @return the Cart's total
	 */
	public BigDecimal getTotal(){
		BigDecimal total = new BigDecimal("0.00");
		for (Product p : this) {
			total = total.add(p.getSellPrice().multiply(BigDecimal.valueOf(p.getQuantity())));
		}
		return total;
	}

	/**
	 * Gets the sum of each Product's quantity.
	 *
	 * @return the number of items in the Cart.
	 */
	publicint getQuantity(){
		int quantity = 0;
		for (Product p : this) {
			quantity += p.getQuantity();
		}
		return quantity;
	}

	privatestatic Cart instance;

}
```

Product.java:

```
package shoppingCart.model;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.ArrayList;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
 * A class that manages Product state.
 *
 * Subject in Observer Pattern
 *
 * @authorNewmanSouza
 * @authorSethMoore
 */
@SuppressWarnings("serial")
publicclass Product implements Cloneable, Serializable{

	/**
	 * Constructs a Product object.
	 *
	 * @param ID
	 *		Product's ID
	 * @param name
	 *		Product's name
	 * @param description
	 *		Product's description
	 * @param sellPrice
	 *		Product's selling price
	 * @param invoicePrice
	 *		Product's invoice price
	 * @param quantity
	 *		Product's quantity
	 * @precondition none
	 * @postcondition object created
	 */
	public Product(int ID, String name, String description,
				BigDecimal invoicePrice, BigDecimal sellPrice, int quantity) {

		this.ID = ID;
		this.name = name;
		this.description = description;
		this.sellPrice = sellPrice;
		this.invoicePrice = invoicePrice;
		this.quantity = quantity;
		listeners = new ArrayList<>();

	}


	/**
	 * Updates the Product.
	 *
	 * @param ID
	 *		Product's ID
```

```
     * @param name
     *          Product's name
     * @param description
     *          Product's description
     * @param sellPrice
     *          Product's selling price
     * @param invoicePrice
     *          Product's invoice price
     * @param quantity
     *          Product's quantity
     * @precondition none
     * @postcondition  All the Product's fields have been
     *                                                  updated to the supplied parameters.
     */
    publicvoid update(int ID, String name, String description,
                        BigDecimal invoicePrice, BigDecimal sellPrice, int quantity) {

                this.ID = ID;
                this.name = name;
                this.description = description;
                this.sellPrice = sellPrice;
                this.invoicePrice = invoicePrice;
                this.quantity = quantity;
                notifyListeners();
    }

    /**
     * Increments the Product's quantity by 1.
     *
     * @precondition none
     * @postcondition quantity > 0
     */
    publicvoid increment(){
                quantity++;
                notifyListeners();
    }

    /**
     * Decrements the Product's quantity by 1.
     *
     * @precondition quantity > 0
     * @postcondition quantity >= 0
     */
    publicvoid decrement(){
                quantity--;
                notifyListeners();
    }

    /**
     * Getter for ID
     *
     * @return the ID
     */
    publicint getID() {
                returnID;
    }
```

```
/**
 * Getter for name
 *
 * @return the name
 */
public String getName() {
	returnname;
}

/**
 * Getter for description
 *
 * @return the description
 */
public String getDescription() {
	returndescription;
}

/**
 * Getter for sellPrice
 *
 * @return the sellPrice
 */
public BigDecimal getSellPrice() {
	returnsellPrice;
}

/**
 * Getter for invoicePrice
 *
 * @return the invoicePrice
 */
public BigDecimal getInvoicePrice() {
	returninvoicePrice;
}

/**
 * Getter for quantity
 *
 * @return the quantity
 */
publicint getQuantity() {
	returnquantity;
}

/**
 * Creates and returns an Object whose fields match the fields of this
 * Product, but with quantity set to 0.
 *
 * @return the clone of this object.
 * @precondition none
 */
@Override
public Object clone(){
	Product cloned = null;
	try {
		cloned = (Product)super.clone();
		cloned.quantity = 0;
		cloned.listeners = new ArrayList<>();
```

```
            } catch (CloneNotSupportedException e) {
            }
            return cloned;
}

/**
 * Adds a ChangeListener to the ChangeListeners that will be notified
 * whenever this Product changes state.
 *
 * @param listener the ChangeListener to add
 */
publicvoid addListener(ChangeListener listener){
            listeners.add(listener);
}

/**
 * Calculates and returns the Product's hash code.
 *
 * @return the Product's hash code.
 */
@Override
publicint hashCode() {
            int hash = 1;
            int prime = 31;
            return (hash * prime + ID);
}

/**
 * Checks for equality between this Product and the obj parameter, and
 * returns the result. Equality is based on ID.
 *
 * @param obj the Object that is being compared to this Product.
 * @return true if obj and this are equivalent (does not fields other
 * than ID into account), otherwise false
 */
@Override
publicboolean equals(Object obj) {
            if (this == obj) {
                        returntrue;
            }
            if (obj == null) {
                        returnfalse;
            }
            if (!(obj instanceof Product)) {
                        returnfalse;
            }
            Product other = (Product) obj;
            if (ID != other.ID) {
                        returnfalse;
            }
            returntrue;
}

/**
 * Notifies the listeners that the state has changed.
 * @precondition none
 * @postcondition all ChangeListeners in listeners have been notified
 * that the state has changed.
 */
```

```
	privatevoid notifyListeners(){
		for (ChangeListener listener : listeners){
			listener.stateChanged(new ChangeEvent(this));
		}
	}

	/**
	 * Removes all the ChangeListeners.
	 *
	 * @postcondition no change listeners that had been added to this
	 * Product will receive notifications of this Product's state changes.
	 */
	publicvoid removeListeners() {
		listeners.clear();
	}

	privateintID;
	private String name;
	private String description;
	private BigDecimal sellPrice;
	private BigDecimal invoicePrice;
	privateintquantity;
	privatetransient ArrayList<ChangeListener>listeners;

}
```

PrunningIterator.java:

```
package shoppingCart.model;

import java.util.Iterator;

/**
 * A decorator for a Iterator<Product>, which weeds out Products
 * that have quantity < 1.
 *
 * Decorator in Decorator Pattern
 * ConcreteIterator in Iterator Pattern
 *
 * @authorSethMoore
 * @authorNewmanSouza
 */
publicclass PrunningIterator implements Iterator<Product> {

	/**
	 * PrunningIterator constructor.
	 *
	 * @param iter the Iterator<Product> being decorated
	 */
	public PrunningIterator(Iterator<Product> iter){
		this.iter = iter;
	}
```

```
	/**
	 * Checks the decorated iterator for the next Product with
	 * quantity > 0, and returns true if one is found, otherwise
	 * returns false.
	 *
	 * @return answer to whether there is another Product with quantity > 0
	 */
	@Override
	publicboolean hasNext() {
		if (nextProduct != null){
			returntrue;
		}
		while (iter.hasNext()){
			Product p = iter.next();
			if (p.getQuantity() > 0){
				nextProduct = p;
				returntrue;
			}
		}
		returnfalse;
	}

	/**
	 * Returns the next Product whose quantity > 0 from the
	 * decorated iterator.
	 *
	 * @return the next Product whose quantity > 0
	 * @precondition hasNext() == true
	 */
	@Override
	public Product next() {
		if (nextProduct != null){
			Product temp = nextProduct;
			nextProduct = null;
			return temp;
		}
		elseif (hasNext()){
			Product temp = nextProduct;
			nextProduct = null;
			return temp;
		}
		else{
			returnnull;
		}
	}

	/**
	 * remove() is not supported.
	 */
	@Override
	publicvoid remove() throws UnsupportedOperationException {
		thrownew UnsupportedOperationException();
	}

	Iterator<Product>iter;
	Product nextProduct = null;
}
```

UserList.java:

```
package shoppingCart.model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

/** A class that manages a list of users.
 *  @authorNewmanSouza
 *  @authorSethMoore
 */
@SuppressWarnings("serial")
publicclass UserList implements Serializable {

/** Constructs a UserList object.
    *  @precondition                    none
    *  @postcondition                   object created
    */
public UserList() {
            users = new ArrayList<User>();
   }

/**
    * Creates and adds a new user to the UserList.
    *
    * @param username the User's username
    * @param password the User's password
    * @param type the User's type
    */
publicvoid addUser(String username, String password, String type) {
                        User user = new User(username, password, type);
                        users.add(user);
   }

/** Validates user credentials.
             *  @param username          User's username
             *  @param password          User's password
             *  @return                  The User's type if user is in the list and null otherwise
             *  @precondition            username and password are valid references
             *  @postcondition User's type is determined
             */
            public String validate(String username, String password) {
                        Iterator<User> userList = Collections.unmodifiableList(users).iterator();
            while (userList.hasNext()) {
                        User user = (User)userList.next();
                        if (user.getUsername().equals(username)) {
                                    if (user.checkPassword(password)) {
                                                return user.getType();
                                    }
                        }
            }
                        returnnull;
            }

private ArrayList<User>users;

}
```

User.java:

```
package shoppingCart.model;

import java.io.Serializable;

/** A class that manages User state.
 *  @authorNewmanSouza
 *  @authorSethMoore
 */
@SuppressWarnings("serial")
publicclass User implements Serializable {

/** Constructs a User object.
    *  @param username              User's username
    *  @param password              User's password
    *  @param type                  User's type
    *  @precondition                none
    *  @postcondition               object created
    */
public User(String username, String password, String type) {
            this.username = username;
            this.password = password;
            this.type = type;
   }

/** Retrieves the User's username.
    *  @return                      The User's username
    *  @precondition                none
    */
public String getUsername() {
            returnthis.username;
   }

/** Accessor- Retrieves the User's type.
    *  @return                      The User's type
    *  @precondition                none
    */
public String getType() {
            returnthis.type;
   }

/** Validates the User's password.
    *  @param password              The User's password
    *  @return                      True if password is correct and False otherwise
    *  @precondition                password is a valid reference
    */
publicboolean checkPassword(String password) {
            return (this.password.equals(password));
   }



/** The User's username.
    */
private String username;
/** The User's password.
    */
private String password;
```

```
/** The User's type.
   */
private String type;

}
```

UI.java:

```
package shoppingCart.gui;

import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Iterator;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import javax.swing.border.EtchedBorder;

import shoppingCart.model.Cart;
import shoppingCart.model.Inventory;
import shoppingCart.model.Product;
import shoppingCart.model.PrunningIterator;
import shoppingCart.system.CartSystem;

/** A class that manages interaction with user, receives input and displays screens.
 *  @authorNewmanSouza
 *  @authorSethMoore
 */
@SuppressWarnings("serial")
publicclass UI extends JFrame{

        finalstatic String LOGINPANEL = "LoginScreen";
        finalstatic String CUSTOMERPANEL = "CustomerScreen";
        finalstatic String SELLERPANEL = "SellerScreen";
        finalstatic String CHECKOUTPANEL = "CheckoutScreen";

/** Constructs a UI object (as well as all its screens),
   *  and sets itself visible.
   *
   *  @precondition none
   *  @postcondition LoginScreen is displayed
   */
public UI(CartSystem cartSystem) {
        super("Shopping Cart");
```

```
            this.cartSystem = cartSystem;
            inventory = Inventory.getInstance();
            cart = Cart.getInstance();
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setPreferredSize(new Dimension(900, 600));
            screenCards = new JPanel(new CardLayout());

            JPanel loginScreen = createLoginScreen();
            screenCards.add(loginScreen, LOGINPANEL);

            customerScreen = new CustomerScreen(this);
            customerScreen.createScreen();
            screenCards.add(customerScreen, CUSTOMERPANEL);

            sellerScreen = new SellerScreen(this);
            sellerScreen.createScreen();
            screenCards.add(sellerScreen, SELLERPANEL);

            checkoutScreen = new CheckoutScreen(this);
            checkoutScreen.createScreen();
            screenCards.add(checkoutScreen, CHECKOUTPANEL);

            add(screenCards);

            pack();
            setLocationRelativeTo(null);
            setVisible(true);

            displayLoginScreen();
    }

/**
     * Switches to the LoginScreen.
     */
publicvoid displayLoginScreen() {
            ((CardLayout)screenCards.getLayout()).show(screenCards, LOGINPANEL);
    }

/**
     * Constructs the LoginScreen.
     *
     * @return the LoginScreen.
     */
private JPanel createLoginScreen() {
            JPanel loginScreen = new JPanel();
                        loginScreen.setLayout(new BorderLayout());
                        JPanel topPanel = new JPanel();
            JLabel label = new JLabel("Login Screen");
                        label.setFont(label.getFont().deriveFont(16.0f));
                        topPanel.add(label);
                        topPanel.setBorder(new EtchedBorder());
                        loginScreen.add(topPanel, BorderLayout.NORTH);

                        final JTextField usernameField = new JTextField(10);
                        final JPasswordField passwordField = new JPasswordField(10);
            JButton loginButton = new JButton("Login");
            loginButton.addActionListener(new
                                    ActionListener() {
                                                publicvoid actionPerformed(ActionEvent e) {
```

```
                                        String userType = cartSystem.login(usernameField.getText(),
                                                String.valueOf(passwordField.getPassword()));
                                        if (userType == null){
                                                JOptionPane.showMessageDialog(screenCards, "Invalid username /
                                                        password.\n\nPlease try again.\n");
                                                usernameField.setText("");
                                                passwordField.setText("");
                                        }
                                        elseif (userType.equals("Customer")) {
                                                cart.clear();
                                                displayCustomerScreen();
                                        }
                                        elseif (userType.equals("Seller")){
                                                displaySellerScreen();
                                        }

                                };
                        }
                );

        JPanel centerPanel = new JPanel();
        centerPanel.setMaximumSize(new Dimension(200, 300));
        centerPanel.setLayout(new GridBagLayout());
        GridBagConstraints loginC = new GridBagConstraints();
        label = new JLabel();
        label.setText("Username:");
        loginC.anchor = GridBagConstraints.LINE_END;
                loginC.insets = new Insets(10,10,10,10);
                loginC.weightx = 0.5;
                loginC.gridx = 0;
                loginC.gridy = 0;
                centerPanel.add(label, loginC);
        label = new JLabel();
        label.setText("Password:");
                loginC.gridy = 1;
                centerPanel.add(label, loginC);
                usernameField.setFont(usernameField.getFont().deriveFont(16.0f));
        loginC.anchor = GridBagConstraints.LINE_START;
                loginC.gridx = 1;
                loginC.gridy = 0;
                centerPanel.add(usernameField, loginC);
                passwordField.setFont(passwordField.getFont().deriveFont(16.0f));
                loginC.gridy = 1;
                centerPanel.add(passwordField, loginC);
                loginC.gridx = 1;
                loginC.gridy = 2;
        centerPanel.add(loginButton, loginC);

        loginScreen.add(centerPanel, BorderLayout.CENTER);
                return loginScreen;
        }

/**
   * Passes iterator over entire Inventory to SellerScreen to populate
   * itself, and switches to the SellerScreen.
   */
```

```
publicvoid displaySellerScreen() {
		Iterator<Product> iter = inventory.iterator();
		sellerScreen.populate(iter);
			GridLayout grid = (GridLayout)sellerScreen.browsePanel.getLayout();
		if (sellerScreen.browsePanel.getComponentCount() < 14) {
			grid.setRows(14);
		} else {
			grid.setRows(0);
		}
		((CardLayout)(screenCards.getLayout())).show(screenCards, SELLERPANEL);
  }

/**
   * Passes iterator over Products in Inventory with quantity > 0 (using PrunningIterator decorator)
   * to CustomerScreen to populate itself, and switches to the CustomerScreen.
   */
publicvoid displayCustomerScreen() {
		PrunningIterator pIter = new PrunningIterator(inventory.iterator());
		customerScreen.populate(pIter);
			GridLayout grid = (GridLayout)customerScreen.browsePanel.getLayout();
		if (customerScreen.browsePanel.getComponentCount() < 14) {
			grid.setRows(14);
		} else {
			grid.setRows(0);
		}
		((CardLayout)(screenCards.getLayout())).show(screenCards, CUSTOMERPANEL);
  }

/**
   * Passes iterator over Products in Cart with quantity > 0 (using PrunningIterator decorator)
   * to CheckoutScreen to populate itself, and switches to the CheckoutScreen.
   */
publicvoid displayCheckoutScreen() {
		PrunningIterator pIter = new PrunningIterator(cart.iterator());
		if (!pIter.hasNext()) {
				JOptionPane.showMessageDialog(screenCards, "Cart is empty.");
		} else {
		checkoutScreen.populate(pIter);
			GridLayout grid = (GridLayout)checkoutScreen.browsePanel.getLayout();
		if (checkoutScreen.browsePanel.getComponentCount() < 9) {
			grid.setRows(9);
		} else {
			grid.setRows(0);
		}
		((CardLayout)(screenCards.getLayout())).show(screenCards, CHECKOUTPANEL);
		}
  }

/**
   * Gets reference to the CartSystem that created this UI.
   *
   * @return the CartSystem that created this UI
   */
		public CartSystem getCartSystem() {
			returncartSystem;
		}
```

```
private CartSystem cartSystem;
private Inventory inventory;
private Cart cart;
private AbstractScreen customerScreen;
private AbstractScreen sellerScreen;
private AbstractScreen checkoutScreen;
   JPanel screenCards;

}
```

AbstractScreen.java:

```
package shoppingCart.gui;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.util.Iterator;

import javax.swing.JPanel;
import javax.swing.JScrollPane;

import shoppingCart.model.Product;

/**
 * A class that assembles JPanels for the UI.
 * AbstractClass in Template Method Pattern
 *
 * @authorNewmanSouza
 * @authorSethMoore
 */
@SuppressWarnings("serial")
publicabstractclass AbstractScreen extends JPanel {

/**
            * Constructs a AbstractScreen object.
            *
            * @param ui                        a reference to the UI that created this screen.
            * @precondition            none
    * @postcondition                   object created
            */
public AbstractScreen(UI ui) {
            super();
            headerPanel = new JPanel();
            browsePanel = new JPanel();
            //sidePanel = new JPanel(); // Moved creation into createSidePanel().
            this.ui = ui;
   }

/**
    * Calls methods that fill Header, Browse, and Side panels with components and listeners.
    * This is a Template Method in the Template Method Pattern,
    * with createheaderPanel() and createSidePanel() being abstract primitive methods.
    */
```

```
publicvoid createScreen() {

		this.setLayout(new BorderLayout());
		createHeaderPanel();
		createBrowsePanel();
		createSidePanel();

  }

/**
   * Sets up the BrowsePanel, in preparation for being populated
   * with lines of Product information.
   */
		publicvoid createBrowsePanel() {
			GridLayout grid = new GridLayout();
			grid.setColumns(1);
			browsePanel.setLayout(grid);
			JScrollPane scrollPane = new JScrollPane(browsePanel);
			scrollPane.getVerticalScrollBar().setUnitIncrement(12);
			this.add(scrollPane, BorderLayout.CENTER);
		}

/** Populates a browsePanel with a list of products.
   * This is another Template Method, with addLine() being
   * the abstract primitive method.
   *
		 *  @param iterator		An Product iterator
		 *  @precondition		iterator is a valid reference
		 *  @postcondition	browsePanel is populated with lines of Products
		 */
publicvoid populate(Iterator<Product> iterator) {
		browsePanel.removeAll();
		while (iterator.hasNext()) {
		Product product = iterator.next();
		product.removeListeners();
		browsePanel.add(addLine(product));
		}
  }

/**
   * Shows the user a pre-populated product form, which is
   * editable for a seller.
   *
   * @param product		the Product whose information will be
   *				displayed in the product form.
   */
publicabstractvoid displayProductForm(Product product);

/**
   * Primitive method called by createScreen() to create the appropriate header
   * for the sub-class
   */
publicabstractvoid createHeaderPanel();

/**
   * Primitive method called by createScreen() to create the appropriate side panel
   * for the sub-class
   */
```

```
publicabstractvoid createSidePanel();

/** Primitive method called by populate(). Assembles a line for
   *       each Product in the Inventory.
   *
            * @param product                    The Product to be displayed in the line
            * @return                          a JPanel containing the appropriate components/listeners
            * @precondition            product is a valid reference
            * @postcondition Line assembled
            */
publicabstract JPanel addLine(Product product);

protected JPanel headerPanel;
protected JPanel browsePanel;
protected JPanel sidePanel;
protected UI ui;
}
```

SellerScreen.java:

```
package shoppingCart.gui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.math.BigDecimal;
import java.text.NumberFormat;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.BevelBorder;
import javax.swing.border.EtchedBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import shoppingCart.model.Cart;
import shoppingCart.model.Inventory;
import shoppingCart.model.Product;

/**
 *A JPanel containing all the panels, buttons, listeners, etc. for
 *  the SellerScreen.
*  ConcreteClass in Template Method Pattern
*  @authorNewmanSouza
 *  @authorSethMoore
 */
```

```
@SuppressWarnings("serial")
publicclass SellerScreen extends AbstractScreen {

		/**
		 * Constructs a SellerScreen object.
		 *
		 * @param ui a reference to the UI that created this screen.
		 */
public SellerScreen(UI ui) {
		super(ui);
  }

/**
   * Creates the header panel for the SellerScreen.
   */
		@Override
publicvoid createHeaderPanel() {
		headerPanel.setBorder(new EtchedBorder());
			headerPanel.setLayout(new GridBagLayout());
		GridBagConstraints headerC = new GridBagConstraints();
			headerC.anchor = GridBagConstraints.LINE_END;
			headerC.weightx = 0.5;
			headerC.gridx = 0;
		JLabel label = new JLabel("Seller Screen");
			label.setFont(label.getFont().deriveFont(16.0f));
		headerPanel.add(label, headerC);
			JButton logoutButton = new JButton("Logout");
		logoutButton.addActionListener(new
					ActionListener() {
						publicvoid actionPerformed(ActionEvent e) {
							Cart.getInstance().clear();
							ui.displayLoginScreen();
						}
					}
			);
			headerC.gridx = 1;
			headerC.insets = new Insets(3,0,4,4);
			headerC.anchor = GridBagConstraints.LINE_END;
		headerPanel.add(logoutButton, headerC);
		this.add(headerPanel, BorderLayout.NORTH);
  }

		/**
		 * Creates the side panel for the SellerScreen,
		 * which includes Seller's financial information,
		 * and an Add Product button.
		 */
		@Override
		publicvoid createSidePanel() {
			final JLabel revenuesLabel = new
		JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(Inventory.getInstance().getRevenues())));
			final JLabel costsLabel = new
		JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(Inventory.getInstance().getCosts())));
			final JLabel profitsLabel = new JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(
					Inventory.getInstance().getRevenues().subtract(Inventory.getInstance().getCosts()))));
			sidePanel = new
					JPanel() {
						publicvoid repaint() {
```

```
costsLabel.setText(String.valueOf(NumberFormat.getCurrencyInstance().format(Inventory.getInstance().getCosts())));
            revenuesLabel.setText(String.valueOf(NumberFormat.getCurrencyInstance().format(Inventory.getInstance()
            .getRevenues())));

profitsLabel.setText(String.valueOf(NumberFormat.getCurrencyInstance().format(

Inventory.getInstance().getRevenues().subtract(Inventory.getInstance().getCosts())));
                                                        super.repaint();
                                                }
                                        };
            sidePanel.setBorder(new EtchedBorder());
            sidePanel.setPreferredSize(new Dimension(240, 600));
            JPanel panel = new JPanel();
            panel.setSize(sidePanel.getPreferredSize());
            panel.setLayout(new GridBagLayout());
            JLabel titleLabel = new JLabel("Financial Information");
            titleLabel.setFont(titleLabel.getFont().deriveFont(16.0f));
GridBagConstraints panelC = new GridBagConstraints();
            panelC.insets = new Insets(10,10,10,10);
            panelC.weightx = 1;
            panelC.gridx = 0;
            panelC.gridy = 0;
            panel.add(titleLabel, panelC);
            JPanel sellerFinancials = new JPanel();
            sellerFinancials.setLayout(new GridBagLayout());
GridBagConstraints financialsC = new GridBagConstraints();
sellerFinancials.setPreferredSize(new Dimension(210, 140));
sellerFinancials.setBorder(new EtchedBorder());

            JLabel label;
            label = new JLabel("Revenues:");
            label.setFont(label.getFont().deriveFont(15.0f));
            financialsC.anchor = GridBagConstraints.LINE_END;
            financialsC.insets = new Insets(10,0,10,0);
            financialsC.weightx = 0.5;
            financialsC.gridx = 0;
            financialsC.gridy = 0;
            sellerFinancials.add(label, financialsC);
            label = new JLabel("Costs:");
            label.setFont(label.getFont().deriveFont(15.0f));
            financialsC.gridy = 1;
            sellerFinancials.add(label, financialsC);
            label = new JLabel("Profit:");
            label.setFont(label.getFont().deriveFont(15.0f));
            financialsC.gridy = 2;
            sellerFinancials.add(label, financialsC);
            financialsC.insets = new Insets(10,0,10,20);
            financialsC.gridwidth = 2;
            financialsC.gridx = 1;
            financialsC.gridy = 0;
            revenuesLabel.setFont(label.getFont().deriveFont(15.0f));
            sellerFinancials.add(revenuesLabel, financialsC);
            financialsC.gridy = 1;
            costsLabel.setFont(label.getFont().deriveFont(15.0f));
            sellerFinancials.add(costsLabel, financialsC);
            financialsC.gridy = 2;
            profitsLabel.setFont(label.getFont().deriveFont(15.0f));
            sellerFinancials.add(profitsLabel, financialsC);
```

```
                panelC.gridy = 1;
                panel.add(sellerFinancials, panelC);
                JButton addProductButton = new JButton("Add Product");
                addProductButton.addActionListener(new
                        ActionListener() {
                                publicvoid actionPerformed(ActionEvent e) {
                                displayProductForm();
                                }
                        }
                        );
                panelC.gridy = 2;
                panel.add(addProductButton, panelC);
                Inventory.getInstance().addListener(new
                                ChangeListener() {
                                        @Override
                                        publicvoid stateChanged(ChangeEvent arg0) {
                                                sidePanel.repaint();
                                        }

                });
                sidePanel.add(panel);
                this.add(sidePanel, BorderLayout.EAST);
        }

        /**
         * Displays a blank Product form, which can be filled in
         * and saved in order to add new products to Inventory.
         */
publicvoid displayProductForm() {
        JPanel productForm = new JPanel();
        productForm.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;

                JLabel label;
        label = new JLabel("ID:");
        c.insets = new Insets(10,20,10,0);
        c.weightx = 0.5;
        c.gridx = 0;
        c.gridy = 0;
        productForm.add(label, c);
        label = new JLabel("Name:");
        c.gridy = 1;
        productForm.add(label, c);
        label = new JLabel("Description: ");
        c.gridy = 2;
        productForm.add(label, c);
        label = new JLabel("Invoice Price:");
        c.gridy = 3;
        productForm.add(label, c);
        label = new JLabel("Sell Price:");
        c.gridy = 4;
        productForm.add(label, c);
        label = new JLabel("Quantity:");
        c.gridy = 5;
        productForm.add(label, c);

        int newID = Inventory.getInstance().getNewID();
```

```
		label = new JLabel(String.valueOf(newID));
		c.fill = GridBagConstraints.NONE;
		c.anchor = GridBagConstraints.LINE_START;
		c.insets = new Insets(10,20,10,20);
		c.weightx = 0.5;
		c.gridx = 2;
		c.gridy = 0;
		productForm.add(label, c);
		JTextField nameTextField = new JTextField();
		nameTextField.setPreferredSize(new Dimension(200, 25));
		nameTextField.requestFocus();
		c.gridwidth = 3;
		c.gridy = 1;
		productForm.add(nameTextField, c);
		JTextField descriptionTextField = new JTextField();
		descriptionTextField.setPreferredSize(new Dimension(300, 25));
		c.gridy = 2;
		productForm.add(descriptionTextField, c);
		JTextField invoicePriceTextField = new JTextField();
		invoicePriceTextField.setPreferredSize(new Dimension(70, 25));
		c.gridy = 3;
		productForm.add(invoicePriceTextField, c);
		JTextField sellPriceTextField = new JTextField();
		sellPriceTextField.setPreferredSize(new Dimension(70, 25));
		c.gridy = 4;
		productForm.add(sellPriceTextField, c);
		JTextField quantityTextField = new JTextField();
		quantityTextField.setPreferredSize(new Dimension(50, 25));
		c.gridy = 5;
		productForm.add(quantityTextField, c);

		Object[] options = {"Save", "Cancel"};
		int button = 2;
		while (button == 2) {
		button = JOptionPane.showOptionDialog(ui, productForm, "New Product",
				JOptionPane.YES_NO_OPTION,
				JOptionPane.QUESTION_MESSAGE, null, options, null);
		if (button == 0) {
			if (validateFields(nameTextField, descriptionTextField, invoicePriceTextField, sellPriceTextField,
				quantityTextField)) {
			Product p = new Product(newID, nameTextField.getText(), descriptionTextField.getText(),
					new BigDecimal(invoicePriceTextField.getText()), new
					BigDecimal(sellPriceTextField.getText()), Integer.parseInt(quantityTextField.getText()));
			Inventory.getInstance().add(p);
			ui.getCartSystem().saveInventory();
			ui.displaySellerScreen();
		ui.validate();
			} else {
				button = 2;
			}
		}
		}
	}

privateboolean validateFields(JTextField nameTextField, JTextField descriptionTextField,
			JTextField invoicePriceTextField, JTextField sellPriceTextField, JTextField quantityTextField) {
		if (!nameTextField.getText().matches(".+")) {
			JOptionPane.showMessageDialog(ui, "Invalid Name:\n\nName cannot be blank.");
			returnfalse;
```

```
            }
            if (!descriptionTextField.getText().matches(".+")) {
                        JOptionPane.showMessageDialog(ui, "Invalid Description:\n\nDescription cannot be blank.");
                        returnfalse;
            }
            if (!invoicePriceTextField.getText().matches("[0-9]+(?:\\.[0-9]{1,2})?")) {
                        JOptionPane.showMessageDialog(ui, "Invalid Invoice Price:\n\n" + invoicePriceTextField.getText());
                        returnfalse;
            }
            if (!sellPriceTextField.getText().matches("[0-9]+(?:\\.[0-9]{1,2})?")) {
                        JOptionPane.showMessageDialog(ui, "Invalid Sell Price:\n\n" + sellPriceTextField.getText());
                        returnfalse;
            }
            if (!quantityTextField.getText().matches("[0-9]+")) {
                        JOptionPane.showMessageDialog(ui, "Invalid Quantity:\n\n" + quantityTextField.getText());
                        returnfalse;
            }
            returntrue;
  }

/**
             * Displays a pre-filled Product form, which can be changed
             * in order to update a product in Inventory, or the Product
             * can be deleted from Inventory.
             *
             * @param product the Product that can be updated or deleted.
             */
publicvoid displayProductForm(Product product) {
            JPanel productForm = new JPanel();
            productForm.setLayout(new GridBagLayout());
            GridBagConstraints c = new GridBagConstraints();
            c.fill = GridBagConstraints.HORIZONTAL;

            JLabel label;
            label = new JLabel("ID:");
            c.insets = new Insets(10,20,10,0);
            c.weightx = 0.5;
            c.gridx = 0;
            c.gridy = 0;
            productForm.add(label, c);
            label = new JLabel("Name:");
            c.gridy = 1;
            productForm.add(label, c);
            label = new JLabel("Description: ");
            c.gridy = 2;
            productForm.add(label, c);
            label = new JLabel("Invoice Price:");
            c.gridy = 3;
            productForm.add(label, c);
            label = new JLabel("Sell Price:");
            c.gridy = 4;
            productForm.add(label, c);
            label = new JLabel("Quantity:");
            c.gridy = 5;
            productForm.add(label, c);

            label = new JLabel(String.valueOf(product.getID()));
            c.fill = GridBagConstraints.NONE;
            c.anchor = GridBagConstraints.LINE_START;
```

```
c.insets = new Insets(10,20,10,20);
c.weightx = 0.5;
c.gridx = 2;
c.gridy = 0;
productForm.add(label, c);
JTextField nameTextField = new JTextField(product.getName());
nameTextField.setPreferredSize(new Dimension(200, 25));
nameTextField.requestFocus();
c.gridwidth = 3;
c.gridy = 1;
productForm.add(nameTextField, c);
JTextField descriptionTextField = new JTextField(product.getDescription());
descriptionTextField.setPreferredSize(new Dimension(300, 25));
c.gridy = 2;
productForm.add(descriptionTextField, c);
JTextField invoicePriceTextField = new JTextField(String.valueOf(product.getInvoicePrice()));
invoicePriceTextField.setPreferredSize(new Dimension(70, 25));
c.gridy = 3;
productForm.add(invoicePriceTextField, c);
JTextField sellPriceTextField = new JTextField(String.valueOf(product.getSellPrice()));
sellPriceTextField.setPreferredSize(new Dimension(70, 25));
c.gridy = 4;
productForm.add(sellPriceTextField, c);
JTextField quantityTextField = new JTextField(String.valueOf(product.getQuantity()));
quantityTextField.setPreferredSize(new Dimension(50, 25));
c.gridy = 5;
productForm.add(quantityTextField, c);

Object[] options = {"Update", "Delete", "Cancel"};

int button = 3;
while (button == 3) {
button = JOptionPane.showOptionDialog(ui, productForm, "Update Product",
                        JOptionPane.YES_NO_CANCEL_OPTION,
                        JOptionPane.QUESTION_MESSAGE, null, options, null);
if (button == 0) {
            if (validateFields(nameTextField, descriptionTextField, invoicePriceTextField, sellPriceTextField,
                        quantityTextField)) {
            Product p = new Product(product.getID(), nameTextField.getText(), descriptionTextField.getText(),
                                    new BigDecimal(invoicePriceTextField.getText()), new
                                    BigDecimal(sellPriceTextField.getText()), Integer.parseInt(quantityTextField.getText()));
            Inventory.getInstance().update(p);
            ui.getCartSystem().saveInventory();
} else {
                        button = 3;
            }
} elseif (button == 1) {
button = JOptionPane.showConfirmDialog(ui,"Are you sure?", "Confirm Delete",
                        JOptionPane.YES_NO_OPTION,
                        JOptionPane.QUESTION_MESSAGE, null);
if (button == 0) {
            Inventory.getInstance().remove(product);
            ui.getCartSystem().saveInventory();
            ui.displaySellerScreen();
ui.validate();
            } else {
                                    button = 3;
                        }
}
```

```
            }
        }

/**
    * Assembles and returns a JPanel for the supplied Product,
    * with product summary. The Product name is clickable in
    * order to update or delete the Product.
    *
            * @param product                The Product represented by the JPanel.
            * @return                       a JPanel filled with appropriate labels, buttons, etc.
            * @precondition          product is a valid reference
            */
@Override
public JPanel addLine(final Product product) {
            GridBagLayout grid = new GridBagLayout();
            GridBagConstraints headerC = new GridBagConstraints();
            if (browsePanel.getComponentCount() == 0) {
                        JPanel headerLine = new JPanel();
            headerLine.setLayout(grid);
                        JLabel idHeaderLabel = new JLabel("ID");
            idHeaderLabel.setPreferredSize(new Dimension(10, 25));
            headerC.fill = GridBagConstraints.HORIZONTAL;
            headerC.anchor = GridBagConstraints.CENTER;
            headerC.insets = new Insets(10,5,0,5);
            headerC.weightx = 0.1;
            headerC.gridx = 0;
            headerC.gridy = 0;
            headerLine.add(idHeaderLabel, headerC);
                        JLabel nameHeaderLabel = new JLabel("Name");
            nameHeaderLabel.setPreferredSize(new Dimension(120, 25));
            headerC.gridwidth = 4;
            headerC.weightx = 0.3;
            headerC.gridx = 1;
            headerLine.add(nameHeaderLabel, headerC);
                        JLabel invoicePriceHeaderLabel = new JLabel("Invoice Price");
            invoicePriceHeaderLabel.setPreferredSize(new Dimension(74, 25));
            invoicePriceHeaderLabel.setHorizontalAlignment(JLabel.RIGHT);
            headerC.anchor = GridBagConstraints.LINE_END;
            headerC.fill = GridBagConstraints.NONE;
            headerC.gridwidth = 1;
            headerC.weightx = 0.2;
            headerC.gridx = 5;
            headerLine.add(invoicePriceHeaderLabel, headerC);
                        JLabel sellPriceHeaderLabel = new JLabel("Sell Price");
            sellPriceHeaderLabel.setPreferredSize(new Dimension(70, 25));
            sellPriceHeaderLabel.setHorizontalAlignment(JLabel.RIGHT);
            headerC.anchor = GridBagConstraints.LINE_END;
            headerC.fill = GridBagConstraints.NONE;
            headerC.gridx = 6;
            headerLine.add(sellPriceHeaderLabel, headerC);
            headerC.anchor = GridBagConstraints.CENTER;
            headerC.gridx = 7;
                        JLabel quantityHeaderLabel = new JLabel("Quantity");
                        quantityHeaderLabel.setPreferredSize(new Dimension(50, 25));
            quantityHeaderLabel.setHorizontalAlignment(JLabel.RIGHT);
            headerLine.add(quantityHeaderLabel, headerC);
                        browsePanel.add(headerLine);
            }
            final JLabel nameLabel = new JLabel(product.getName());
```

```
final JLabel invoicePriceLabel = new
JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(product.getInvoicePrice())));
        final JLabel sellPriceLabel = new
JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(product.getSellPrice())));
final JLabel quantityLabel = new JLabel(String.valueOf(product.getQuantity()));
        final JPanel line = new
                        JPanel() {
                                publicvoid repaint() {
                                        nameLabel.setText(product.getName());

        invoicePriceLabel.setText(String.valueOf(NumberFormat.getCurrencyInstance().format(product.getInvoicePrice())));

        sellPriceLabel.setText(String.valueOf(NumberFormat.getCurrencyInstance().format(product.getSellPrice())));
                                        quantityLabel.setText(String.valueOf(product.getQuantity()));
                                        super.repaint();
                                }
                        };
        line.setLayout(grid);

    JLabel idLabel = new JLabel(String.valueOf(product.getID()));
        idLabel.setPreferredSize(new Dimension(10, 25));
        GridBagConstraints lineC = new GridBagConstraints();
        lineC.fill = GridBagConstraints.HORIZONTAL;
        lineC.insets = new Insets(10,5,0,5);
        lineC.gridwidth = 1;
        lineC.weightx = 0.1;
        lineC.gridx = 0;
        lineC.gridy = 0;
        line.add(idLabel, lineC);

    nameLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
    nameLabel.addMouseListener(new
                        MouseAdapter(){
                                publicvoid mouseClicked(MouseEvent e) {
                                        displayProductForm(product);
                                };
                        }
        );
        nameLabel.setPreferredSize(new Dimension(120, 25));
        lineC.gridwidth = 4;
        lineC.weightx = 0.3;
        lineC.gridx = 1;
    line.add(nameLabel, lineC);

        invoicePriceLabel.setPreferredSize(new Dimension(70, 25));
        invoicePriceLabel.setHorizontalAlignment(JLabel.RIGHT);
        lineC.anchor = GridBagConstraints.LINE_END;
        lineC.fill = GridBagConstraints.NONE;
        lineC.gridwidth = 1;
        lineC.weightx = 0.2;
        lineC.gridx = 5;
    line.add(invoicePriceLabel, lineC);

        sellPriceLabel.setPreferredSize(new Dimension(74, 25));
        sellPriceLabel.setHorizontalAlignment(JLabel.RIGHT);
        lineC.gridx = 6;
    line.add(sellPriceLabel, lineC);

                quantityLabel.setPreferredSize(new Dimension(50, 25));
```

```
            quantityLabel.setHorizontalAlignment(JLabel.RIGHT);
            lineC.anchor = GridBagConstraints.CENTER;
            lineC.gridx = 7;
      line.add(quantityLabel, lineC);

            product.addListener(new ChangeListener() {
                                    @Override
                                    publicvoid stateChanged(ChangeEvent e) {
                                            line.repaint();
                                    }
                            });
            return line;
    }

}
```

CustomerScreen.java:

```
package shoppingCart.gui;


import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.text.NumberFormat;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.border.BevelBorder;
import javax.swing.border.EtchedBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import shoppingCart.model.Cart;
import shoppingCart.model.Inventory;
import shoppingCart.model.Product;

/**
 *          A JPanel containing all the panels, buttons, listeners, etc. for
 *  the CustomerScreen.
 *
 *  ConcreteClass in Template Method Pattern
 *
 *  @authorNewmanSouza
 *  @authorSethMoore
 */
@SuppressWarnings("serial")
```

```
publicclass CustomerScreen extends AbstractScreen {

		/**
		 * Constructs a CustomerScreen object.
		 * @param ui a reference to the UI that created this screen.
		 */
public CustomerScreen(UI ui) {
		super(ui);
  }

/**
   * Creates the header panel for the CustomerScreen.
   */
		@Override
publicvoid createHeaderPanel() {
		headerPanel.setBorder(new EtchedBorder());
			headerPanel.setLayout(new GridBagLayout());
		GridBagConstraints headerC = new GridBagConstraints();
			headerC.anchor = GridBagConstraints.LINE_END;
			headerC.weightx = 0.5;
			headerC.gridx = 0;
		JLabel label = new JLabel("Customer Screen");
			label.setFont(label.getFont().deriveFont(16.0f));
		headerPanel.add(label, headerC);
			JButton logoutButton = new JButton("Logout");
		logoutButton.addActionListener(new
				ActionListener() {
					publicvoid actionPerformed(ActionEvent e) {
						Cart.getInstance().clear();
						ui.displayLoginScreen();
					}
				}
			);
			headerC.gridx = 1;
			headerC.insets = new Insets(3,0,4,4);
			headerC.anchor = GridBagConstraints.LINE_END;
		headerPanel.add(logoutButton, headerC);
		this.add(headerPanel, BorderLayout.NORTH);
  }

		/**
		 * Creates the side panel for the CustomerScreen,
		 * which includes a cart summary.
		 */
		@Override
		publicvoid createSidePanel() {
			final JLabel itemsLabel = new JLabel(String.valueOf(Cart.getInstance().getQuantity()));
			final JLabel totalLabel = new
JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(Cart.getInstance().getTotal())));
			sidePanel = new
					JPanel() {
						publicvoid repaint() {
							itemsLabel.setText(String.valueOf(Cart.getInstance().getQuantity()));

		totalLabel.setText(String.valueOf(NumberFormat.getCurrencyInstance().format(Cart.getInstance().getTotal())));
							super.repaint();
						}

					};
```

```
		sidePanel.setBorder(new EtchedBorder());
		sidePanel.setPreferredSize(new Dimension(240, 600));
		JPanel panel = new JPanel();
		panel.setSize(sidePanel.getPreferredSize());
		panel.setLayout(new GridBagLayout());
		JLabel titleLabel = new JLabel("Cart Summary");
		titleLabel.setFont(titleLabel.getFont().deriveFont(16.0f));
GridBagConstraints panelC = new GridBagConstraints();
		panelC.insets = new Insets(10,10,3,10);
		panelC.weightx = 1;
		panelC.gridx = 0;
		panelC.gridy = 0;
		panel.add(titleLabel, panelC);
		JPanel cartSummary = new JPanel();
		cartSummary.setLayout(new GridBagLayout());
GridBagConstraints summaryC = new GridBagConstraints();
		cartSummary.setPreferredSize(new Dimension(210, 100));
		cartSummary.setBorder(new EtchedBorder());

		JLabel label;
		label = new JLabel("Items:");
		label.setFont(label.getFont().deriveFont(16.0f));
		summaryC.anchor = GridBagConstraints.LINE_END;
		summaryC.insets = new Insets(10,0,10,0);
		summaryC.weightx = 0.5;
		summaryC.gridx = 0;
		summaryC.gridy = 0;
		cartSummary.add(label, summaryC);
		label = new JLabel("Total:");
		label.setFont(label.getFont().deriveFont(16.0f));
		summaryC.gridy = 1;
		cartSummary.add(label, summaryC);
		summaryC.insets = new Insets(10,0,10,20);
		summaryC.gridwidth = 2;
		summaryC.gridx = 1;
		summaryC.gridy = 0;
		itemsLabel.setFont(label.getFont().deriveFont(16.0f));
		cartSummary.add(itemsLabel, summaryC);
		summaryC.gridy = 1;
		totalLabel.setFont(label.getFont().deriveFont(16.0f));
		cartSummary.add(totalLabel, summaryC);
		panelC.insets = new Insets(0,10,10,10);
		panelC.gridy = 1;
		panel.add(cartSummary, panelC);
		JButton checkoutButton = new JButton("Checkout");
		checkoutButton.addActionListener(new
			ActionListener() {
				publicvoid actionPerformed(ActionEvent e) {
					ui.displayCheckoutScreen();
				}
			}
		);
		panelC.gridy = 2;
		panel.add(checkoutButton, panelC);
		Cart.getInstance().addListener(new
			ChangeListener() {
				@Override
				publicvoid stateChanged(ChangeEvent arg0) {
					sidePanel.repaint();
```

```
                                        }

                });
                sidePanel.add(panel);
                this.add(sidePanel, BorderLayout.EAST);
        }

        /**
         * Displays a Product's details.
         * @param product
         *          the Product whose details are to be displayed.
         */
publicvoid displayProductForm(Product product) {
        JPanel productForm = new JPanel();
        productForm.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;

        JLabel label;
        label = new JLabel("ID:");
        c.insets = new Insets(10,20,10,0);
        c.weightx = 0.5;
        c.gridx = 0;
        c.gridy = 0;
        productForm.add(label, c);
        label = new JLabel("Name:");
        c.gridy = 1;
        productForm.add(label, c);
        label = new JLabel("Description: ");
        c.gridy = 2;
        productForm.add(label, c);
        label = new JLabel("Price:");
        c.gridy = 3;
        productForm.add(label, c);
        label = new JLabel("Quantity:");
        c.gridy = 4;
        productForm.add(label, c);

        label = new JLabel(String.valueOf(product.getID()));
        c.insets = new Insets(10,20,10,20);
        c.weightx = 0.5;
        c.gridx = 2;
        c.gridy = 0;
        productForm.add(label, c);
        label = new JLabel(product.getName());
        c.gridy = 1;
        productForm.add(label, c);
        label = new JLabel(product.getDescription());
        c.gridy = 2;
        productForm.add(label, c);
        label = new JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(product.getSellPrice())));
        c.gridy = 3;
        productForm.add(label, c);
        label = new JLabel(String.valueOf(product.getQuantity()));
        c.gridy = 4;
        productForm.add(label, c);
        JOptionPane.showMessageDialog(ui, productForm, "Product Detail", JOptionPane.INFORMATION_MESSAGE);
  }
```

```
/**
    * Assembles and returns a JPanel for the supplied Product,
    * with product summary and Add button.
    *
            * @param product                The Product represented by the JPanel.
            * @return                       a JPanel filled with appropriate labels, buttons, etc.
            * @precondition          product is a valid reference
            */
@Override
public JPanel addLine(final Product product) {
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints headerC = new GridBagConstraints();
        if (browsePanel.getComponentCount() == 0) {
                JPanel headerLine = new JPanel();
        headerLine.setLayout(grid);
                JLabel nameHeaderLabel = new JLabel("Name");
        nameHeaderLabel.setPreferredSize(new Dimension(120, 25));
        headerC.fill = GridBagConstraints.HORIZONTAL;
        headerC.anchor = GridBagConstraints.CENTER;
        headerC.gridwidth = 4;
        headerC.insets = new Insets(10,15,0,5);
        headerC.weightx = 0.4;
        headerC.gridx = 0;
        headerC.gridy = 0;
        headerLine.add(nameHeaderLabel, headerC);
                JLabel priceHeaderLabel = new JLabel("Price");
        priceHeaderLabel.setPreferredSize(new Dimension(70, 25));
        headerC.anchor = GridBagConstraints.LINE_END;
        headerC.fill = GridBagConstraints.NONE;
        headerC.gridwidth = 1;
        headerC.weightx = 0.2;
        headerC.gridx = 4;
        headerLine.add(priceHeaderLabel, headerC);
        headerC.anchor = GridBagConstraints.CENTER;
        headerC.gridx = 5;
                JLabel quantityHeaderLabel = new JLabel("Quantity");
                quantityHeaderLabel.setPreferredSize(new Dimension(50, 25));
        headerLine.add(quantityHeaderLabel, headerC);
        headerC.gridx = 6;
                JLabel buttonHeaderLabel = new JLabel();
                buttonHeaderLabel.setPreferredSize(new Dimension(30, 25));
        headerLine.add(buttonHeaderLabel, headerC);
                browsePanel.add(headerLine);
        }
        final JButton addButton = new JButton("Add to Cart");
        final JLabel quantityLabel = new JLabel(String.valueOf(product.getQuantity()));
        final JPanel line = new
                        JPanel() {
                                publicvoid repaint() {
                                        quantityLabel.setText(String.valueOf(product.getQuantity()));
                                        if (product.getQuantity() < 1) {
                                                addButton.setEnabled(false);
                                        }
                                        super.repaint();
                                }
                        };
        line.setLayout(grid);

        JLabel nameLabel = new JLabel(product.getName());
```

```
		nameLabel.setPreferredSize(new Dimension(120, 25));
		nameLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
		nameLabel.addMouseListener(new
					MouseAdapter(){
						publicvoid mouseClicked(MouseEvent e) {
							displayProductForm(product);
						};
					}
		);
		GridBagConstraints lineC = new GridBagConstraints();
		lineC.fill = GridBagConstraints.HORIZONTAL;
		lineC.gridwidth = 4;
		lineC.insets = new Insets(10,15,0,5);
		lineC.weightx = 0.4;
		lineC.gridx = 0;
		lineC.gridy = 0;
		line.add(nameLabel, lineC);

		JLabel priceLabel = new JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(product.getSellPrice())));
		priceLabel.setPreferredSize(new Dimension(70, 25));
		priceLabel.setHorizontalAlignment(JLabel.RIGHT);
		lineC.anchor = GridBagConstraints.LINE_END;
		lineC.fill = GridBagConstraints.NONE;
		lineC.gridwidth = 1;
		lineC.weightx = 0.2;
		lineC.gridx = 4;
		line.add(priceLabel, lineC);

		lineC.gridx = 5;
		quantityLabel.setPreferredSize(new Dimension(50, 25));
		quantityLabel.setHorizontalAlignment(JLabel.RIGHT);
		line.add(quantityLabel, lineC);

		addButton.addActionListener(new
					ActionListener(){
						@Override
						publicvoid actionPerformed(ActionEvent arg0) {
							Inventory.getInstance().decrement(product);
							Cart.getInstance().increment(product);

						};
					}
		);
			addButton.setFont(addButton.getFont().deriveFont(10.0f));
		addButton.setMargin(new Insets(1,1,1,1));
		lineC.insets = new Insets(0,5,0,15);
		lineC.gridx = 6;
		line.add(addButton, lineC);

		product.addListener(new ChangeListener() {
					@Override
					publicvoid stateChanged(ChangeEvent e) {
						line.repaint();
					}
			});
		return line;
	}

}
```

CheckoutScreen.java:

```
package shoppingCart.gui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.math.BigDecimal;
import java.text.NumberFormat;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.BevelBorder;
import javax.swing.border.EtchedBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import shoppingCart.model.Cart;
import shoppingCart.model.Inventory;
import shoppingCart.model.Product;

/**
 * A JPanel containing all the panels, buttons, listeners, etc. for
 * the customer's Checkout Screen.
 *
 * ConcreteClass in Template Method Pattern
 *
 *  @authorNewmanSouza
 *  @authorSethMoore
 */
@SuppressWarnings("serial")
publicclass CheckoutScreen extends AbstractScreen {

	/**
	 * Constructs a CheckoutScreen object.
	 *
	 * @param ui a reference to the UI that created this screen.
	 */
public CheckoutScreen(UI ui) {
	super(ui);
  }

/**
   * Creates the header panel for the CheckoutScreen.
   */
	publicvoid createHeaderPanel() {
	headerPanel.setBorder(new EtchedBorder());
		headerPanel.setLayout(new GridBagLayout());
	GridBagConstraints headerC = new GridBagConstraints();
```

```
                headerC.anchor = GridBagConstraints.LINE_END;
                headerC.weightx = 0.5;
                headerC.gridx = 0;
        JLabel label = new JLabel("Checkout Screen");
                label.setFont(label.getFont().deriveFont(16.0f));
        headerPanel.add(label, headerC);
                JButton logoutButton = new JButton("Logout");
        logoutButton.addActionListener(new
                        ActionListener() {
                                publicvoid actionPerformed(ActionEvent e) {
                                        Cart.getInstance().clear();
                                        ui.displayLoginScreen();
                                }
                        }
                );
                headerC.gridx = 1;
                headerC.insets = new Insets(3,0,4,4);
                headerC.anchor = GridBagConstraints.LINE_END;
        headerPanel.add(logoutButton, headerC);
        this.add(headerPanel, BorderLayout.NORTH);
  }

        /**
         * Creates the side panel for the CheckoutScreen,
         * which includes a cart summary and payment form.
         */
        @Override
        publicvoid createSidePanel() {
                final JLabel itemsLabel = new JLabel(String.valueOf(Cart.getInstance().getQuantity()));
                final JLabel totalLabel = new
JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(Cart.getInstance().getTotal())));
                sidePanel = new
                                JPanel() {
                                        publicvoid repaint() {
                                                itemsLabel.setText(String.valueOf(Cart.getInstance().getQuantity()));

        totalLabel.setText(String.valueOf(NumberFormat.getCurrencyInstance().format(Cart.getInstance().getTotal())));
                                                super.repaint();
                                        }

                                };
                sidePanel.setBorder(new EtchedBorder());
                sidePanel.setPreferredSize(new Dimension(240, 600));
                JPanel panel = new JPanel();
                panel.setSize(sidePanel.getPreferredSize());
                panel.setLayout(new GridBagLayout());
                JLabel titleLabel = new JLabel("Cart Summary");
                titleLabel.setFont(titleLabel.getFont().deriveFont(16.0f));
        GridBagConstraints panelC = new GridBagConstraints();
                panelC.insets = new Insets(10,10,3,10);
                panelC.weightx = 1;
                panelC.gridx = 0;
                panelC.gridy = 0;
                panel.add(titleLabel, panelC);
                JPanel cartSummary = new JPanel();
                cartSummary.setLayout(new GridBagLayout());
        GridBagConstraints summaryC = new GridBagConstraints();
                cartSummary.setPreferredSize(new Dimension(210, 100));
                cartSummary.setBorder(new EtchedBorder());
```

```
                JLabel label;
                label = new JLabel("Items:");
                label.setFont(label.getFont().deriveFont(16.0f));
                summaryC.anchor = GridBagConstraints.LINE_END;
                summaryC.insets = new Insets(10,0,10,0);
                summaryC.weightx = 0.5;
                summaryC.gridx = 0;
                summaryC.gridy = 0;
                cartSummary.add(label, summaryC);
                label = new JLabel("Total:");
                label.setFont(label.getFont().deriveFont(16.0f));
                summaryC.gridy = 1;
                cartSummary.add(label, summaryC);
                summaryC.insets = new Insets(10,0,10,20);
                summaryC.gridwidth = 2;
                summaryC.gridx = 1;
                summaryC.gridy = 0;
                itemsLabel.setFont(label.getFont().deriveFont(16.0f));
                cartSummary.add(itemsLabel, summaryC);
                summaryC.gridy = 1;
                totalLabel.setFont(label.getFont().deriveFont(16.0f));
                cartSummary.add(totalLabel, summaryC);
                panelC.insets = new Insets(0,10,0,10);
                panelC.gridy = 1;
                panel.add(cartSummary, panelC);

                JLabel formLabel = new JLabel("Payment Information");
                formLabel.setFont(titleLabel.getFont().deriveFont(16.0f));
                panelC.insets = new Insets(30,10,3,10);
                panelC.weightx = 1;
                panelC.gridy = 3;
                panel.add(formLabel, panelC);
                JPanel paymentForm = new JPanel();
                paymentForm.setLayout(new GridBagLayout());
GridBagConstraints formC = new GridBagConstraints();
                paymentForm.setPreferredSize(new Dimension(210, 260));
                paymentForm.setBorder(new EtchedBorder());

                label = new JLabel("Type:");
                formC.anchor = GridBagConstraints.LINE_START;
                formC.insets = new Insets(5,5,0,5);
                formC.gridwidth = 2;
                formC.weightx = 0.2;
                formC.gridx = 0;
                formC.gridy = 0;
                paymentForm.add(label, formC);
                label = new JLabel("Cardholder Name:");
                formC.gridy = 2;
                paymentForm.add(label, formC);
                label = new JLabel("Card Number:");
                formC.gridy = 4;
                paymentForm.add(label, formC);
                label = new JLabel("Expiration Date:");
                formC.gridwidth = 1;
                formC.gridy = 6;
                paymentForm.add(label, formC);
                label = new JLabel("Security Code:");
                formC.gridy = 7;
```

```
                paymentForm.add(label, formC);

                JTextField cardTypeTextField = new JTextField();
                cardTypeTextField.setPreferredSize(new Dimension(195, 25));
                cardTypeTextField.setText("MasterCard");
                formC.gridwidth = 2;
                formC.insets = new Insets(5,5,3,5);
                formC.weightx = 1;
                formC.gridy = 1;
                paymentForm.add(cardTypeTextField, formC);
                JTextField cardholderTextField = new JTextField();
                cardholderTextField.setPreferredSize(new Dimension(195, 25));
                cardholderTextField.setText("John Smith");
                formC.gridy = 3;
                paymentForm.add(cardholderTextField, formC);
                final JTextField cardNumberTextField = new JTextField();
                cardNumberTextField.setPreferredSize(new Dimension(195, 25));
                cardNumberTextField.setText("1234-5678-1234-5678");
                formC.gridy = 5;
                paymentForm.add(cardNumberTextField, formC);
                JTextField expirationTextField = new JTextField();
                expirationTextField.setPreferredSize(new Dimension(52, 25));
                expirationTextField.setText("10/2015");
                formC.anchor = GridBagConstraints.LINE_END;
                formC.gridwidth = 1;
                formC.gridx = 1;
                formC.gridy = 6;
                paymentForm.add(expirationTextField, formC);
                JTextField codeTextField = new JTextField();
                codeTextField.setPreferredSize(new Dimension(26, 25));
                codeTextField.setText("123");
                formC.gridy = 7;
                paymentForm.add(codeTextField, formC);
                panelC.insets = new Insets(0,10,10,10);
                panelC.gridy = 4;
                panel.add(paymentForm, panelC);
                JButton cancelButton = new JButton("Cancel");
                cancelButton.addActionListener(new
                                ActionListener() {
                                                publicvoid actionPerformed(ActionEvent e) {
                                                                ui.displayCustomerScreen();
                                                }
                                        }
                                );
                panelC.anchor = GridBagConstraints.LINE_START;
        panelC.insets = new Insets(0,40,0,5);
                panelC.gridy = 5;
                panel.add(cancelButton, panelC);
                JButton payButton = new JButton("Pay");
                payButton.addActionListener(new
                                ActionListener() {
                                                publicvoid actionPerformed(ActionEvent e) {
                                                                if (Cart.getInstance().getTotal().compareTo(BigDecimal.ZERO) > 0) {
                                                                                boolean result =
ui.getCartSystem().pay(cardNumberTextField.getText(), Cart.getInstance().getTotal());
                                                                                if (result) {
                                                                                JOptionPane.showMessageDialog(null, "Payment
successful.");
```

```
                                                Cart.getInstance().clear();
                                                ui.displayCustomerScreen();
                                                        } else {
                                                                JOptionPane.showMessageDialog(null,
"Payment failed.");
                                                        }
                                                } else {
                                                JOptionPane.showMessageDialog(null, "Cart is empty.");
                                                }
                                        }
                                }
                        );
                panelC.anchor = GridBagConstraints.LINE_END;
        panelC.insets = new Insets(0,5,0,40);
                panel.add(payButton, panelC);
                Cart.getInstance().addListener(new
                                ChangeListener() {
                                        @Override
                                        publicvoid stateChanged(ChangeEvent arg0) {
                                                sidePanel.repaint();
                                        }

                });
                sidePanel.add(panel);
                this.add(sidePanel, BorderLayout.EAST);
        }

/**
   * Displays a Product's details.
   *
   * @param product          the Product whose details are to be displayed.
   */
publicvoid displayProductForm(Product product) {
        JPanel productForm = new JPanel();
        productForm.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;

        JLabel label;
        label = new JLabel("ID:");
        c.insets = new Insets(10,20,10,0);
        c.weightx = 0.5;
        c.gridx = 0;
        c.gridy = 0;
        productForm.add(label, c);
        label = new JLabel("Name:");
        c.gridy = 1;
        productForm.add(label, c);
        label = new JLabel("Description: ");
        c.gridy = 2;
        productForm.add(label, c);
        label = new JLabel("Price:");
        c.gridy = 3;
        productForm.add(label, c);
        label = new JLabel("Quantity:");
        c.gridy = 4;
        productForm.add(label, c);

        label = new JLabel(String.valueOf(product.getID()));
```

```
            c.insets = new Insets(10,20,10,20);
            c.weightx = 0.5;
            c.gridx = 2;
            c.gridy = 0;
            productForm.add(label, c);
            label = new JLabel(product.getName());
            c.gridy = 1;
            productForm.add(label, c);
            label = new JLabel(product.getDescription());
            c.gridy = 2;
            productForm.add(label, c);
            label = new JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(product.getSellPrice())));
            c.gridy = 3;
            productForm.add(label, c);
            label = new JLabel(String.valueOf(product.getQuantity()));
            c.gridy = 4;
            productForm.add(label, c);
            JOptionPane.showMessageDialog(ui, productForm, "Product Detail", JOptionPane.INFORMATION_MESSAGE);
    }

/**
     * Assembles and returns a JPanel for the supplied Product,
     * with product summary and increment/decrement buttons.
     *
            * @param product                The Product represented by the JPanel.
            * @return                       a JPanel filled with appropriate labels, buttons, etc.
            * @precondition          product is a valid reference
            */
@Override
public JPanel addLine(final Product product) {
            GridBagLayout grid = new GridBagLayout();
            GridBagConstraints headerC = new GridBagConstraints();
            if (browsePanel.getComponentCount() == 0) {
                        JPanel headerLine = new JPanel();
            headerLine.setLayout(grid);
                        JLabel nameHeaderLabel = new JLabel("Name");
            nameHeaderLabel.setPreferredSize(new Dimension(120, 25));
            headerC.fill = GridBagConstraints.HORIZONTAL;
            headerC.anchor = GridBagConstraints.CENTER;
            headerC.gridwidth = 4;
            headerC.insets = new Insets(10,15,0,5);
            headerC.weightx = 0.4;
            headerC.gridx = 0;
            headerC.gridy = 0;
            headerLine.add(nameHeaderLabel, headerC);
                        JLabel priceHeaderLabel = new JLabel("Price");
            priceHeaderLabel.setPreferredSize(new Dimension(90, 25));
            priceHeaderLabel.setHorizontalAlignment(JLabel.RIGHT);
            headerC.anchor = GridBagConstraints.LINE_END;
            headerC.fill = GridBagConstraints.NONE;
            headerC.gridwidth = 1;
            headerC.weightx = 0.2;
            headerC.gridx = 4;
            headerLine.add(priceHeaderLabel, headerC);
            headerC.gridx = 5;
                        JLabel quantityHeaderLabel = new JLabel("Quantity");
                        quantityHeaderLabel.setPreferredSize(new Dimension(50, 25));
            headerLine.add(quantityHeaderLabel, headerC);
            headerC.anchor = GridBagConstraints.CENTER;
```

```
headerC.gridx = 6;
            JLabel buttonHeaderLabel = new JLabel();
            buttonHeaderLabel.setPreferredSize(new Dimension(30, 25));
headerLine.add(buttonHeaderLabel, headerC);
            browsePanel.add(headerLine);
}
final JButton incrementButton = new JButton("+");
final JButton decrementButton = new JButton("-");
final JLabel quantityLabel = new JLabel(String.valueOf(product.getQuantity()));
final JPanel line = new
                        JPanel() {
                                    publicvoid repaint() {
                                                quantityLabel.setText(String.valueOf(product.getQuantity()));
                                                if (product.getQuantity() < 1) {
                                                            decrementButton.setEnabled(false);
                                                }
                                                else {
                                                            decrementButton.setEnabled(true);
                                                }
                                                if (Inventory.getInstance().getMatchingProduct(product).getQuantity() < 1) {
                                                            incrementButton.setEnabled(false);
                                                }
                                                else {
                                                            incrementButton.setEnabled(true);
                                                }
                                                super.repaint();
                                    }
                        };
line.setLayout(grid);

JLabel nameLabel = new JLabel(product.getName());
nameLabel.setPreferredSize(new Dimension(120, 35));
nameLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
nameLabel.addMouseListener(new
                        MouseAdapter(){
                                    publicvoid mouseClicked(MouseEvent e) {
                                                displayProductForm(product);
                                    };
                        }
);
GridBagConstraints lineC = new GridBagConstraints();
lineC.fill = GridBagConstraints.HORIZONTAL;
lineC.gridwidth = 4;
lineC.insets = new Insets(10,15,0,5);
lineC.weightx = 0.4;
lineC.gridx = 0;
lineC.gridy = 0;
line.add(nameLabel, lineC);

JLabel priceLabel = new JLabel(String.valueOf(NumberFormat.getCurrencyInstance().format(product.getSellPrice())));
priceLabel.setPreferredSize(new Dimension(70, 35));
priceLabel.setHorizontalAlignment(JLabel.RIGHT);
lineC.anchor = GridBagConstraints.LINE_END;
lineC.fill = GridBagConstraints.NONE;
lineC.gridwidth = 1;
lineC.weightx = 0.2;
lineC.gridx = 4;
line.add(priceLabel, lineC);
```

```
		lineC.anchor = GridBagConstraints.CENTER;
		lineC.gridx = 5;
		quantityLabel.setPreferredSize(new Dimension(50, 43));
		quantityLabel.setHorizontalAlignment(JLabel.RIGHT);
		line.add(quantityLabel, lineC);

		incrementButton.addActionListener(new
				ActionListener(){
					publicvoid actionPerformed(ActionEvent arg0) {
						Inventory.getInstance().decrement(product);
						Cart.getInstance().increment(product);
					};
				}
		);
		incrementButton.setMargin(new Insets(0,4,0,4));
		incrementButton.setSize(new Dimension(10, 2));
		lineC.anchor = GridBagConstraints.NORTH;
		lineC.gridx = 6;
		line.add(incrementButton, lineC);

		decrementButton.addActionListener(new
				ActionListener(){
					publicvoid actionPerformed(ActionEvent arg0) {
						Inventory.getInstance().increment(product);
						Cart.getInstance().decrement(product);
					};
				}
		);
		decrementButton.setMargin(new Insets(0,5,0,6));
		decrementButton.setSize(new Dimension(10, 2));
		lineC.anchor = GridBagConstraints.SOUTH;
		line.add(decrementButton, lineC);

		product.addListener(new ChangeListener() {
				@Override
				publicvoid stateChanged(ChangeEvent e) {
					line.repaint();
				}
			});
		return line;
	}

}
```

AllTests.java:

```
package shoppingCart.unitTests;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;
```

```
@RunWith(Suite.class)
@SuiteClasses({ CartSystemTest.class, CartTest.class, DBManagerTest.class,
                InventoryTest.class, PaymentValidatorTest.class, ProductTest.class,
                PrunningIteratorTest.class, UserListTest.class, UserTest.class })
public class AllTests {

}
```

CartSystemTest.java:

```
/**
 *
 */
package shoppingCart.unitTests;

import static org.junit.Assert.*;

import java.math.BigDecimal;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import shoppingCart.system.CartSystem;

/**
 *  @author Newman Souza
 *  @author Seth Moore
 */
public class CartSystemTest {

        /**
         * @throws java.lang.Exception
         */
        @BeforeClass
        public static void setUpBeforeClass() throws Exception {
        }

        /**
         * @throws java.lang.Exception
         */
        @AfterClass
        public static void tearDownAfterClass() throws Exception {
        }

        /**
         * @throws java.lang.Exception
         */
        @Before
        public void setUp() throws Exception {
        }
```

```
	/**
	 * @throws java.lang.Exception
	 */
	@After
	public void tearDown() throws Exception {
	}

	/**
	 * Test method for {@link shoppingCart.system.CartSystem#CartSystem()}.
	 */
	@Test
	public void testCartSystem() {
		CartSystem cs = new CartSystem();
		assertFalse(cs == null);
	}

	/**
	 * Test method for {@link shoppingCart.system.CartSystem#login(java.lang.String, java.lang.String)}.
	 */
	@Test
	public void testLogin() {
		CartSystem cs = new CartSystem();
		assertEquals("Seller", cs.login("Newman", "newman"));
	}

	/**
	 * Test method for {@link shoppingCart.system.CartSystem#pay(java.lang.String, java.math.BigDecimal)}.
	 */
	@Test
	public void testPay() {
		CartSystem cs = new CartSystem();
		assertTrue(cs.pay("anything", new BigDecimal("99.99")));
	}

}
```

DBManagerTest.java:

```
/**
 *
 */
package shoppingCart.unitTests;

import static org.junit.Assert.*;

import java.io.File;
import java.math.BigDecimal;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import shoppingCart.model.Inventory;
import shoppingCart.model.Product;
```

```
import shoppingCart.model.UserList;
import shoppingCart.system.DBManager;

/**
 *  @author Newman Souza
 *  @author Seth Moore
 */
public class DBManagerTest {

	private String invSaveFile = ".\\inv.dat";
	private String userSaveFile = ".\\user.dat";
	/**
	 * @throws java.lang.Exception
	 */
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@AfterClass
	public static void tearDownAfterClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@Before
	public void setUp() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@After
	public void tearDown() throws Exception {
		Inventory.getInstance().clear();
	}

	/**
	 * Test method for {@link shoppingCart.system.DBManager#loadInventory()}.
	 */
	@Test
	public void testLoadInventory() {
		BigDecimal bd = new BigDecimal("20.00");
		Inventory inventory = Inventory.getInstance();
		Product p = new Product(0, "name1", "description", bd, bd, 10);
		inventory.add(p);
		inventory.add(new Product(1, "name", "description", bd, bd, 10));
		inventory.add(new Product(2, "name", "description", bd, bd, 10));
		inventory.add(new Product(3, "name", "description", bd, bd, 10));
		DBManager db = new DBManager();
		db.saveInventory(inventory, invSaveFile);
		inventory.clear();
		inventory = db.loadInventory(invSaveFile);
		assertFalse(inventory.getMatchingProduct(p) == null);
		assertEquals(inventory.getMatchingProduct(p).getName(), p.getName());
		assertFalse(inventory.getMatchingProduct(p) == p);
```

```
		assertEquals(inventory.getCosts(), bd.multiply(BigDecimal.valueOf(40)));
		File file = new File(invSaveFile);
		file.delete();
	}

	/**
	 * Test method for {@link shoppingCart.system.DBManager#saveInventory(shoppingCart.model.Inventory)}.
	 */
	@Test
	public void testSaveInventory() {
		BigDecimal bd = new BigDecimal("20.00");
		Inventory inventory = Inventory.getInstance();
		Product p = new Product(0, "name1", "description", bd, bd, 10);
		inventory.add(p);
		inventory.add(new Product(1, "name", "description", bd, bd, 10));
		inventory.add(new Product(2, "name", "description", bd, bd, 10));
		inventory.add(new Product(3, "name", "description", bd, bd, 10));
		DBManager db = new DBManager();
		db.saveInventory(inventory, invSaveFile);
		inventory.clear();
		inventory = db.loadInventory(invSaveFile);
		assertFalse(inventory.getMatchingProduct(p) == null);
		assertEquals(inventory.getMatchingProduct(p).getName(), p.getName());
		assertFalse(inventory.getMatchingProduct(p) == p);
		assertEquals(inventory.getCosts(), bd.multiply(BigDecimal.valueOf(40)));
		File file = new File(invSaveFile);
		file.delete();
	}

	/**
	 * Test method for {@link shoppingCart.system.DBManager#loadUserList()}.
	 */
	@Test
	public void testLoadUserList() {
		UserList userList = new UserList();
		userList.addUser("username", "password", "type");
		userList.addUser("user", "pass", "ty");
		userList.addUser("name", "word", "pe");
		DBManager db = new DBManager();
		db.saveUserList(userList, userSaveFile);
		userList = new UserList();
		userList = db.loadUserList(userSaveFile);
		assertEquals(userList.validate("user", "pass"), "ty");

		File file = new File(userSaveFile);
		file.delete();
	}

	/**
	 * Test method for {@link shoppingCart.system.DBManager#saveUserList(shoppingCart.model.UserList)}.
	 */
	@Test
	public void testSaveUserList() {
		UserList userList = new UserList();
		userList.addUser("username", "password", "type");
		userList.addUser("user", "pass", "ty");
		userList.addUser("name", "word", "pe");
		DBManager db = new DBManager();
		db.saveUserList(userList, userSaveFile);
```

```
                userList = new UserList();
                userList = db.loadUserList(userSaveFile);
                assertEquals(userList.validate("user", "pass"), "ty");

                File file = new File(userSaveFile);
                file.delete();
        }

}
```

PaymentValidatorTest.java:

```
/**
 *
 */
package shoppingCart.unitTests;

import static org.junit.Assert.*;

import java.math.BigDecimal;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import shoppingCart.system.PaymentValidator;

/**
 *  @author Newman Souza
 *  @author Seth Moore
 */
public class PaymentValidatorTest {

        /**
         * @throws java.lang.Exception
         */
        @BeforeClass
        public static void setUpBeforeClass() throws Exception {
        }

        /**
         * @throws java.lang.Exception
         */
        @AfterClass
        public static void tearDownAfterClass() throws Exception {
        }

        /**
         * @throws java.lang.Exception
         */
        @Before
        public void setUp() throws Exception {
        }
```

```
	/**
	 * @throws java.lang.Exception
	 */
	@After
	public void tearDown() throws Exception {
	}

	/**
	 * Test method for {@link shoppingCart.system.PaymentValidator#validate(java.lang.String, java.math.BigDecimal)}.
	 */
	@Test
	public void testValidate() {
		PaymentValidator pv = new PaymentValidator();
		assertTrue(pv.validate("anything", new BigDecimal("500.00")));
	}

}
```

InventoryTest.java:

```
/**
 *
 */
package shoppingCart.unitTests;

import static org.junit.Assert.*;

import java.math.BigDecimal;
import java.util.Iterator;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import shoppingCart.model.Inventory;
import shoppingCart.model.Product;

/**
 *  @author Newman Souza
 *  @author Seth Moore
 */
public class InventoryTest {

	private Inventory inventory;
	private BigDecimal initialRevenues;
	private BigDecimal initialCosts;
	private Product firstProduct;
	private Product lastProduct;
```

```
/**
 * @throws java.lang.Exception
 */
@BeforeClass
public static void setUpBeforeClass() throws Exception {
}

/**
 * @throws java.lang.Exception
 */
@AfterClass
public static void tearDownAfterClass() throws Exception {
}

/**
 * @throws java.lang.Exception
 */
@Before
public void setUp() throws Exception {
   inventory = Inventory.getInstance();
   inventory.add(new Product(0, "name0", "description0", new BigDecimal("1.00"), new BigDecimal("2.00"), 10));
   inventory.add(new Product(1, "name1", "description1", new BigDecimal("1.00"), new BigDecimal("2.00"), 10));
   inventory.add(new Product(2, "name2", "description2", new BigDecimal("1.00"), new BigDecimal("2.00"), 10));
   inventory.add(new Product(3, "name3", "description3", new BigDecimal("1.00"), new BigDecimal("2.00"), 10));
   initialCosts = inventory.getCosts();
   initialRevenues = inventory.getRevenues();
   Iterator<Product> iter = inventory.iterator();
   firstProduct = iter.next();
   while (iter.hasNext()) lastProduct = iter.next();
}

/**
 * @throws java.lang.Exception
 */
@After
public void tearDown() throws Exception {
            inventory.clear();
}

@Test
public void testGetInstance(){
            Inventory inv = Inventory.getInstance();
            assertTrue(inv == inventory);
}

/**
 * Test method for {@link shoppingCart.model.Inventory#clear()}.
 */
@Test
public void testClear() {
            inventory.add(new Product(99, "name", "des", new BigDecimal("0.00"), new BigDecimal("0.00"), 5));
            inventory.clear();
            assertTrue(inventory.iterator().hasNext() == false);
            assertEquals(new BigDecimal("0.00"), inventory.getProfits());
}

/**
 * Test method for {@link shoppingCart.model.Inventory#increment(shoppingCart.model.Product)}.
 */
```

```
@Test
public void testIncrement() {
		BigDecimal prevRevenues = inventory.getRevenues();
		BigDecimal sellPrice = lastProduct.getSellPrice();
		int quantity = lastProduct.getQuantity();
		inventory.increment(lastProduct);
		assertEquals(prevRevenues.subtract(sellPrice), inventory.getRevenues());
		assertEquals(quantity + 1, lastProduct.getQuantity());
}

/**
 * Test method for {@link shoppingCart.model.Inventory#decrement(shoppingCart.model.Product)}.
 */
@Test
public void testDecrement() {
		BigDecimal prevRevenues = inventory.getRevenues();
		BigDecimal sellPrice = firstProduct.getSellPrice();
		int quantity = firstProduct.getQuantity();
		inventory.decrement(firstProduct);
		assertEquals(prevRevenues.add(sellPrice), inventory.getRevenues());
		assertEquals(quantity - 1, firstProduct.getQuantity());
}

/**
 * Test method for {@link shoppingCart.model.Inventory#getMatchingProduct(shoppingCart.model.Product)}.
 */
@Test
public void testGetMatchingProduct() {
		Product p = (Product)firstProduct.clone();
		assertFalse(p == firstProduct);
		assertTrue(inventory.getMatchingProduct(p) == firstProduct);
		p = (Product)lastProduct.clone();
		p.update(p.getID(), "not", "a", new BigDecimal("99.33"), new BigDecimal("99.33"), 934);
		assertFalse(p == lastProduct);
		assertTrue(inventory.getMatchingProduct(p) == lastProduct);
		p.update(222, "not", "a", new BigDecimal("99.33"), new BigDecimal("99.33"), 934);
		assertTrue(inventory.getMatchingProduct(p) == null);
}

/**
 * Test method for {@link shoppingCart.model.Inventory#add(shoppingCart.model.Product)}.
 */
@Test
public void testAdd() {
   BigDecimal previousCosts = inventory.getCosts();
   Product p = new Product(99, "name", "description", new BigDecimal("10.00"), new BigDecimal("20.00"), 10);
   BigDecimal additionCost = new BigDecimal("100.00");
   inventory.add(p);
   assertEquals(previousCosts.add(additionCost), inventory.getCosts());
   assertFalse(p == inventory.getMatchingProduct(p));
   assertEquals(p, inventory.getMatchingProduct(p));
   assertEquals(p.getQuantity(), inventory.getMatchingProduct(p).getQuantity());
}

/**
 * Test method for {@link shoppingCart.model.Inventory#remove(shoppingCart.model.Product)}.
 */
```

```
@Test
public void testRemove() {
	inventory.remove(firstProduct);
	assertTrue(inventory.getMatchingProduct(firstProduct) == null);
}

/**
 * Test method for {@link shoppingCart.model.Inventory#iterator()}.
 */
@Test
public void testIterator() {
	inventory.clear();
	assertFalse(inventory.iterator().hasNext());
	Product p1 = new Product(1, "name", "desc", new BigDecimal("2.00"), new BigDecimal("2.00"), 10);
	Product p2 = new Product(2, "name", "desc", new BigDecimal("2.00"), new BigDecimal("2.00"), 10);
	Product p3 = new Product(3, "name", "desc", new BigDecimal("2.00"), new BigDecimal("2.00"), 10);
	inventory.add(p1);
	inventory.add(p2);
	inventory.add(p3);
	Iterator<Product> iter = inventory.iterator();
	assertTrue(iter.hasNext());
	assertEquals(p1, iter.next());
	while (iter.hasNext()) p1 = iter.next();
	assertEquals(p3, p1);
}


/**
 * Test method for {@link shoppingCart.model.Inventory#getCosts()}.
 */
@Test
public void testGetCosts() {
	assertEquals(new BigDecimal("40.00") ,inventory.getCosts());
}

/**
 * Test method for {@link shoppingCart.model.Inventory#getRevenues()}.
 */
@Test
public void testGetRevenues() {
	assertEquals(new BigDecimal("0.00") ,inventory.getRevenues());
}

/**
 * Test method for {@link shoppingCart.model.Inventory#getProfits()}.
 */
@Test
public void testGetProfits() {
	assertEquals(initialRevenues.subtract(initialCosts), inventory.getProfits());
}

/**
 * Test method for {@link shoppingCart.model.Inventory#getNewID()}.
 */
```

```
	@Test
	public void testGetNewID() {
		assertTrue(inventory.getNewID() == lastProduct.getID() + 1);
		inventory.clear();
		assertTrue(inventory.getNewID() == 1);
	}

	/**
	 * Test method for {@link shoppingCart.model.Inventory#update(shoppingCart.model.Product)}.
	 */
	@Test
	public void testUpdate() {
		Product p1 = (Product)firstProduct.clone();
		BigDecimal newInvoicePrice = new BigDecimal("100");
		p1.update(p1.getID(), "dk", "ff", newInvoicePrice, p1.getSellPrice(), firstProduct.getQuantity() + 1);
		inventory.update(p1);
		BigDecimal expectedCosts = initialCosts.add(newInvoicePrice);
		assertEquals(expectedCosts, inventory.getCosts());

		Product p2 = (Product)firstProduct.clone();
		newInvoicePrice = new BigDecimal("100");
		p2.update(p2.getID(), "dk", "ff", newInvoicePrice, p2.getSellPrice(), firstProduct.getQuantity() - 1);
		inventory.update(p2);
		assertEquals(expectedCosts, inventory.getCosts());
	}

	/**
	 * Test method for {@link shoppingCart.model.Cart#addListener(javax.swing.event.ChangeListener)}.
	 */
	@Test
	public void testAddListener() {
		final StringBuffer sBuff = new StringBuffer();
		inventory.addListener(new
				ChangeListener(){

					@Override
					public void stateChanged(ChangeEvent arg0) {
						sBuff.append('1');
					}
		});
		assertEquals("", sBuff.substring(0));
		inventory.increment(firstProduct);
		assertEquals("1", sBuff.substring(0));
		inventory.decrement(lastProduct);
		assertEquals("11", sBuff.substring(0));
		Product p = (Product)firstProduct.clone();
		p.update(firstProduct.getID(), "", "", new BigDecimal("0.00"), new BigDecimal("0.00"), 55);
		inventory.update(p);
		assertEquals("111", sBuff.substring(0));
		inventory.add(new Product(66, "3", "3", new BigDecimal("0.00"), new BigDecimal("0.00"), 55));
		assertEquals("1111", sBuff.substring(0));
		inventory.remove(lastProduct);
		assertEquals("11111", sBuff.substring(0));
		inventory.clear();
		assertEquals("111111", sBuff.substring(0));
	}
}
```

CartTest.java:

```
/**
 *
 */
package shoppingCart.unitTests;

import static org.junit.Assert.*;

import java.math.BigDecimal;
import java.util.Iterator;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import shoppingCart.model.Cart;
import shoppingCart.model.Product;

/**
 *  @author Newman Souza
 *  @author Seth Moore
 */
public class CartTest {

	Cart cart;
	Product firstProduct;
	Product lastProduct;
	BigDecimal initialTotal;
	int initialQuantity;


	/**
	 * @throws java.lang.Exception
	 */
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@AfterClass
	public static void tearDownAfterClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@Before
	public void setUp() throws Exception {
		cart = Cart.getInstance();
		cart.add(new Product(0, "name0", "description0", new BigDecimal("1.00"), new BigDecimal("2.00"), 10));
		cart.add(new Product(1, "name1", "description1", new BigDecimal("1.00"), new BigDecimal("2.00"), 10));
```

```
            cart.add(new Product(2, "name2", "description2", new BigDecimal("1.00"), new BigDecimal("2.00"), 10));
            cart.add(new Product(3, "name3", "description3", new BigDecimal("1.00"), new BigDecimal("2.00"), 10));

            initialTotal = new BigDecimal("80.00");
            initialQuantity = 40;

            Iterator<Product> iter = cart.iterator();
            firstProduct = iter.next();
            while (iter.hasNext()) lastProduct = iter.next();
    }

    /**
     * @throws java.lang.Exception
     */
    @After
    public void tearDown() throws Exception {
            cart.clear();
    }

    @Test
    public void testGetInstance(){
            Cart cart2 = Cart.getInstance();
            assertTrue(cart2 == cart);
    }

    /**
     * Test method for {@link shoppingCart.model.Cart#clear()}.
     */
    @Test
    public void testClear() {
            assertTrue(cart.iterator().hasNext());
            assertTrue(cart.getQuantity() > 0);
            cart.clear();
            assertFalse(cart.iterator().hasNext());
            assertTrue(cart.getQuantity() == 0);
            assertEquals(new BigDecimal("0.00"), cart.getTotal());
    }

    /**
     * Test method for {@link shoppingCart.model.Cart#increment(shoppingCart.model.Product)}.
     */
    @Test
    public void testIncrement() {
            int quantity = lastProduct.getQuantity();
            cart.increment(lastProduct);
            assertEquals(quantity + 1, lastProduct.getQuantity());
    }

    /**
     * Test method for {@link shoppingCart.model.Cart#decrement(shoppingCart.model.Product)}.
     */
    @Test
    public void testDecrement() {
            int quantity = lastProduct.getQuantity();
            cart.decrement(lastProduct);
            assertEquals(quantity - 1, lastProduct.getQuantity());
    }
```

```
/**
 * Test method for {@link shoppingCart.model.Cart#getMatchingProduct(shoppingCart.model.Product)}.
 */
@Test
public void testGetMatchingProduct() {
		Product p = (Product)firstProduct.clone();
		assertFalse(p == firstProduct);
		assertTrue(cart.getMatchingProduct(p) == firstProduct);
		p = (Product)lastProduct.clone();
		p.update(p.getID(), "not", "a", new BigDecimal("99.33"), new BigDecimal("99.33"), 934);
		assertFalse(p == lastProduct);
		assertTrue(cart.getMatchingProduct(p) == lastProduct);
		p.update(222, "not", "a", new BigDecimal("99.33"), new BigDecimal("99.33"), 934);
		assertTrue(cart.getMatchingProduct(p) == null);
}

/**
 * Test method for {@link shoppingCart.model.Cart#add(shoppingCart.model.Product)}.
 */
@Test
public void testAdd() {
   Product p = new Product(99, "name", "description", new BigDecimal("10.00"), new BigDecimal("20.00"), 10);
   cart.add(p);
   assertFalse(p == cart.getMatchingProduct(p));
   assertEquals(p, cart.getMatchingProduct(p));
   assertEquals(p.getQuantity(), cart.getMatchingProduct(p).getQuantity());
}

/**
 * Test method for {@link shoppingCart.model.Cart#remove(shoppingCart.model.Product)}.
 */
@Test
public void testRemove() {
		cart.remove(firstProduct);
		assertTrue(cart.getMatchingProduct(firstProduct) == null);
}

/**
 * Test method for {@link shoppingCart.model.Cart#iterator()}.
 */
@Test
public void testIterator() {
		cart.clear();
		assertFalse(cart.iterator().hasNext());
		Product p1 = new Product(1, "name", "desc", new BigDecimal("2.00"), new BigDecimal("2.00"), 10);
		Product p2 = new Product(2, "name", "desc", new BigDecimal("2.00"), new BigDecimal("2.00"), 10);
		Product p3 = new Product(3, "name", "desc", new BigDecimal("2.00"), new BigDecimal("2.00"), 10);
		cart.add(p1);
		cart.add(p2);
		cart.add(p3);
		Iterator<Product> iter = cart.iterator();
		assertTrue(iter.hasNext());
		assertEquals(p1, iter.next());
		while (iter.hasNext()) p1 = iter.next();
		assertEquals(p3, p1);
}
```

```
	/**
	 * Test method for {@link shoppingCart.model.Cart#getTotal()}.
	 */
	@Test
	public void testGetTotal() {
		assertEquals(initialTotal, cart.getTotal());
	}

	/**
	 * Test method for {@link shoppingCart.model.Cart#getQuantity()}.
	 */
	@Test
	public void testGetQuantity() {
		assertEquals(initialQuantity, cart.getQuantity());
	}

	/**
	 * Test method for {@link shoppingCart.model.Cart#addListener(javax.swing.event.ChangeListener)}.
	 */
	@Test
	public void testAddListener() {final StringBuffer sBuff = new StringBuffer();
	cart.addListener(new
			ChangeListener(){

				@Override
				public void stateChanged(ChangeEvent arg0) {
					sBuff.append('1');
				}
	});
	assertEquals("", sBuff.substring(0));
	cart.increment(firstProduct);
	assertEquals("1", sBuff.substring(0));
	cart.decrement(lastProduct);
	assertEquals("11", sBuff.substring(0));
	Product p = (Product)firstProduct.clone();
	p.update(firstProduct.getID(), "", "", new BigDecimal("0.00"), new BigDecimal("0.00"), 55);
	cart.add(new Product(66, "3", "3", new BigDecimal("0.00"), new BigDecimal("0.00"), 55));
	assertEquals("111", sBuff.substring(0));
	cart.remove(lastProduct);
	assertEquals("1111", sBuff.substring(0));
	cart.clear();
	assertEquals("11111", sBuff.substring(0));
	}

}
```

ProductTest.java:

```
/**
 *
 */
package shoppingCart.unitTests;

import static org.junit.Assert.*;

import java.math.BigDecimal;
```

```
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import shoppingCart.model.Product;

/**
 *  @author Newman Souza
 *  @author Seth Moore
 */
public class ProductTest {

	final int ID = 1, quantity = 2;
	final String name = "apple", description = "crisp";
	final BigDecimal sellPrice = new BigDecimal("1.25"), invoicePrice = new BigDecimal("0.40");
	Product product;
	String changeListenerString;

	/**
	 * @throws java.lang.Exception
	 */
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@AfterClass
	public static void tearDownAfterClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@Before
	public void setUp() throws Exception {
		product = new Product(ID, name, description, invoicePrice, sellPrice, quantity);
		product.addListener(new
					ChangeListener(){

						@Override
						public void stateChanged(ChangeEvent arg0) {
							changeListenerString = ((Product)arg0.getSource()).getName();
						}
		});
		changeListenerString = "FAIL";
	}

	/**
	 * @throws java.lang.Exception
	 */
```

```
	@After
	public void tearDown() throws Exception {
	}

	/**
	 * Test method for {@link shoppingCart.model.Product#hashCode()}.
	 */
	@Test
	public void testHashCode() {
		Product p = new Product(ID, name, description, invoicePrice, sellPrice, quantity);
		assertTrue(p.hashCode() == product.hashCode());
		p.update(ID, name, description, invoicePrice, sellPrice, 20);
		assertTrue(p.hashCode() == product.hashCode());
		p.update(33, name, description, invoicePrice, sellPrice, 20);
		assertFalse(p.hashCode() == product.hashCode());
		p.update(ID, name, description, new BigDecimal("33.33"), sellPrice, 20);
		assertTrue(p.hashCode() == product.hashCode());
		p.update(999, name, description, invoicePrice, sellPrice, 20);
		assertFalse(p.hashCode() == product.hashCode());
	}

	/**
	 * Test method for {@link shoppingCart.model.Product#Product(int, java.lang.String, java.lang.String,
java.math.BigDecimal, java.math.BigDecimal, int)}.
	 */
	@Test
	public void testProduct() {
		int ID = 1, quantity = 2;
		String name = "apple", description = "crisp";
		BigDecimal sellPrice = new BigDecimal("1.25"), invoicePrice = new BigDecimal("0.40");
		Product p = new Product(ID, name, description, invoicePrice, sellPrice, quantity);
		assertEquals(p.getID(), ID);
		assertEquals(p.getName(), name);
		assertEquals(p.getDescription(), description);
		assertEquals(p.getSellPrice(), sellPrice);
		assertEquals(p.getInvoicePrice(), invoicePrice);
		assertEquals(p.getQuantity(), quantity);
	}

	/**
	 * Test method for {@link shoppingCart.model.Product#update(int, java.lang.String, java.lang.String,
java.math.BigDecimal, java.math.BigDecimal, int)}.
	 */
	@Test
	public void testUpdate() {
		assertEquals(changeListenerString, "FAIL");
		int testInt = 5;
		String testString = "test";
		BigDecimal testDec = new BigDecimal("1.99");
		product.update(testInt, testString, testString, testDec, testDec, testInt);
		assertEquals(product.getID(), testInt);
		assertEquals(product.getName(), testString);
		assertEquals(product.getDescription(), testString);
		assertEquals(product.getSellPrice(), testDec);
		assertEquals(product.getInvoicePrice(), testDec);
		assertEquals(product.getQuantity(), testInt);
		assertEquals(changeListenerString, product.getName());
	}
```

```
/**
 * Test method for {@link shoppingCart.model.Product#increment()}.
 */
@Test
public void testIncrement() {
	assertEquals(changeListenerString, "FAIL");
	product.increment();
	assertTrue(product.getQuantity() == quantity + 1);
	assertEquals(changeListenerString, product.getName());
}

/**
 * Test method for {@link shoppingCart.model.Product#decrement()}.
 */
@Test
public void testDecrement() {
	assertEquals(changeListenerString, "FAIL");
	product.decrement();
	assertTrue(product.getQuantity() == quantity -1);
	assertEquals(changeListenerString, product.getName());
}

/**
 * Test method for {@link shoppingCart.model.Product#getID()}.
 */
@Test
public void testGetID() {
	assertTrue(product.getID() == ID);
}

/**
 * Test method for {@link shoppingCart.model.Product#getName()}.
 */
@Test
public void testGetName() {
	assertEquals(product.getName(), name);
}

/**
 * Test method for {@link shoppingCart.model.Product#getDescription()}.
 */
@Test
public void testGetDescription() {
	assertEquals(product.getDescription(), description);
}

/**
 * Test method for {@link shoppingCart.model.Product#getSellPrice()}.
 */
@Test
public void testGetSellPrice() {
	assertEquals(product.getSellPrice(), sellPrice);
}

/**
 * Test method for {@link shoppingCart.model.Product#getInvoicePrice()}.
 */
```

```
@Test
public void testGetInvoicePrice() {
	assertEquals(product.getInvoicePrice(), invoicePrice);
}

/**
 * Test method for {@link shoppingCart.model.Product#getQuantity()}.
 */
@Test
public void testGetQuantity() {
	assertTrue(product.getQuantity() == quantity);
}

/**
 * Test method for {@link shoppingCart.model.Product#clone()}.
 */
@Test
public void testClone() {
	Product p = (Product)product.clone();
	assertFalse(p == product);
	assertEquals(p, product);
	assertEquals(p.getQuantity(), 0);
}

/**
 * Test method for {@link shoppingCart.model.Product#addListener(javax.swing.event.ChangeListener)}.
 */
@Test
public void testAddListener() {
	product.addListener(new
				ChangeListener(){

					@Override
					public void stateChanged(ChangeEvent arg0) {
						changeListenerString += "anotherListener";
					}
	});
	product.increment();
	assertEquals(changeListenerString, product.getName()+"anotherListener");
}

/**
 * Test method for {@link shoppingCart.model.Product#removeListeners()}.
 */
@Test
public void testRemoveListeners() {
	product.addListener(new
				ChangeListener(){

					@Override
					public void stateChanged(ChangeEvent arg0) {
						changeListenerString += "anotherListener";
					}
	});
	product.removeListeners();
	product.increment();
	assertEquals(changeListenerString, "FAIL");

}
```

```
	/**
	 * Test method for {@link shoppingCart.model.Product#equals(java.lang.Object)}.
	 */
	@Test
	public void testEqualsObject() {
		Product p = new Product(ID, name, description, invoicePrice, sellPrice, quantity);
		assertTrue(p.equals(product));
		p = new Product(ID, name, description, invoicePrice, sellPrice, 0);
		assertTrue(p.equals(product));
		assertTrue(p.hashCode() == product.hashCode());
		p = new Product(ID, "not an apple", description, invoicePrice, sellPrice, 0);
		assertTrue(p.equals(product));
		p = new Product(999, name, description, invoicePrice, sellPrice, 0);
		assertFalse(p.equals(product));
	}

}
```

PrunningIteratorTest.java:

```
/**
 *
 */
package shoppingCart.unitTests;

import static org.junit.Assert.*;

import java.math.BigDecimal;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import shoppingCart.model.Inventory;
import shoppingCart.model.Product;
import shoppingCart.model.PrunningIterator;

/**
 *  @author Newman Souza
 *  @author Seth Moore
 */
public class PrunningIteratorTest {

	Inventory inventory;
	PrunningIterator iterator;
	Product p2;

	/**
	 * @throws java.lang.Exception
	 */
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
	}
```

```
/**
 * @throws java.lang.Exception
 */
@AfterClass
public static void tearDownAfterClass() throws Exception {
}

/**
 * @throws java.lang.Exception
 */
@Before
public void setUp() throws Exception {
	inventory = Inventory.getInstance();
	p2 = new Product(23, "sam2", "ssld", new BigDecimal("1.00"), new BigDecimal("1.00"), 4);
	inventory.add(new Product(22, "sam1", "ssld", new BigDecimal("1.00"), new BigDecimal("1.00"), 0));
	inventory.add(p2);
	inventory.add(new Product(24, "sam3", "ssld", new BigDecimal("1.00"), new BigDecimal("1.00"), 2));
	inventory.add(new Product(25, "sam4", "ssld", new BigDecimal("1.00"), new BigDecimal("1.00"), 0));
	inventory.add(new Product(26, "sam5", "ssld", new BigDecimal("1.00"), new BigDecimal("1.00"), 8));
	inventory.add(new Product(27, "sam6", "ssld", new BigDecimal("1.00"), new BigDecimal("1.00"), 0));
	iterator = new PrunningIterator(inventory.iterator());
}

/**
 * @throws java.lang.Exception
 */
@After
public void tearDown() throws Exception {
	inventory.clear();
}

/**
 * Test method for {@link shoppingCart.model.PrunningIterator#PrunningIterator(java.util.Iterator)}.
 */
@Test
public void testPrunningIterator() {
	PrunningIterator iter = new PrunningIterator(inventory.iterator());
	assertTrue(iter.hasNext());
}

/**
 * Test method for {@link shoppingCart.model.PrunningIterator#hasNext()}.
 */
@Test
public void testHasNext() {
	int count = 0;
	while (iterator.hasNext()){
		@SuppressWarnings("unused")
		Product p = iterator.next();
		count++;
	}
	assertTrue(count == 3);
}

/**
 * Test method for {@link shoppingCart.model.PrunningIterator#next()}.
 */
```

```
	@Test
	public void testNext() {
		Product p = iterator.next();
		assertTrue(p.getQuantity() == p2.getQuantity());
		assertTrue(p.getID() == p2.getID());
		assertEquals(p2.getName(), p.getName());
	}

	/**
	 * Test method for {@link shoppingCart.model.PrunningIterator#remove()}.
	 */
	@Test(expected=UnsupportedOperationException.class)
	public void testRemove() {
		iterator.remove();
	}

}
```

UserListTest.java:

```
/**
 *
 */
package shoppingCart.unitTests;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import shoppingCart.model.UserList;

/**
 *  @author Newman Souza
 *  @author Seth Moore
 */
public class UserListTest {

	/**
	 * @throws java.lang.Exception
	 */
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@AfterClass
	public static void tearDownAfterClass() throws Exception {
	}
```

```
	/**
	 * @throws java.lang.Exception
	 */
	@Before
	public void setUp() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@After
	public void tearDown() throws Exception {
	}

	/**
	 * Test method for {@link shoppingCart.model.UserList#UserList()}.
	 */
	@Test
	public void testUserList() {
		UserList uList = new UserList();
		assertTrue(uList.validate("username", "password") == null);
	}

	/**
	 * Test method for {@link shoppingCart.model.UserList#addUser(java.lang.String, java.lang.String, java.lang.String)}.
	 */
	@Test
	public void testAddUser() {
		UserList uList = new UserList();
		uList.addUser("username", "password", "type");
		assertTrue(uList.validate("name", "word") == null);
		assertFalse(uList.validate("username", "password") == null);
	}

	/**
	 * Test method for {@link shoppingCart.model.UserList#validate(java.lang.String, java.lang.String)}.
	 */
	@Test
	public void testValidate() {UserList uList = new UserList();
	uList.addUser("username", "password", "type");
	uList.addUser("user", "pass", "pe");
	uList.addUser("name", "word", "ty");
	assertTrue(uList.validate("name", "pass") == null);
	assertEquals("ty", uList.validate("name", "word"));
	assertEquals("type", uList.validate("username", "password"));
	assertEquals("pe", uList.validate("user", "pass"));
	}

}
```

UserTest.java:

```
/**
 *
 */
package shoppingCart.unitTests;
```

```
import static org.junit.Assert.*;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import shoppingCart.model.User;

/**
 *  @author Newman Souza
 *  @author Seth Moore
 */
public class UserTest {

	User user;

	/**
	 * @throws java.lang.Exception
	 */
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@AfterClass
	public static void tearDownAfterClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@Before
	public void setUp() throws Exception {
		user = new User("username", "password", "type");
	}

	/**
	 * @throws java.lang.Exception
	 */
	@After
	public void tearDown() throws Exception {
	}

	/**
	 * Test method for {@link shoppingCart.model.User#User(java.lang.String, java.lang.String, java.lang.String)}.
	 */
	@Test
	public void testUser() {
		User u = new User("1", "2", "3");
		assertEquals("1", u.getUsername());
		assertEquals("3", u.getType());
		assertTrue(u.checkPassword("2"));
	}
```

```
	/**
	 * Test method for {@link shoppingCart.model.User#getUsername()}.
	 */
	@Test
	public void testGetUsername() {
		assertEquals("username", user.getUsername());
	}

	/**
	 * Test method for {@link shoppingCart.model.User#getType()}.
	 */
	@Test
	public void testGetType() {
		assertEquals("type", user.getType());
	}

	/**
	 * Test method for {@link shoppingCart.model.User#checkPassword(java.lang.String)}.
	 */
	@Test
	public void testCheckPassword() {
		assertFalse(user.checkPassword("not"));
		assertTrue(user.checkPassword("password"));
	}
}
```