6.1P

Task 2.

addCartItem

First to check if the cart contains an item I overrode the *equals* and *hashCode* methods of CartItem so that the arraylist method *indexOf* will return the items index of a cartItem with the same id if it exists, otherwise it will return -1.

```
@Override
public boolean equals(Object obj) {
    if (obj.getClass() == this.getClass()) {
        CartItem cartItem = (CartItem) obj;
        return (cartItem.itemId.equals(this.itemId));
    }
    return false;
}
```

```
@Override
public int hashCode() {
    int hash = 3;
    hash = 37 * hash + Objects.hashCode(this.itemId);
    return hash;
}
```

Here our method checks if the item exists and has an index in the cart. If it does, it retrieves the object and increments its quantity. Otherwise it attempts to add an item to the card. The method returns false only on an exception.

```
@Override
public Boolean addCartItem(CartItem cartItem) {
    try {
        Integer index = cart.indexOf(cartItem);
        // If cart contains the itemm
        if (index != -1) {
            // Increment the quantity
            CartItem item = cart.get(index);
            item.setQuantity(item.getQuantity() + 1);
        } else {
            // Add new item to cartd
            if (!add(cartItem)) {
                return false;
            }
        }
        return true;
    } catch (Exception ex) {
        return false;
    }
}
```

deleteCartItem

The arraylist already has this method inbuilt. Arraylist.remove(object) will find the first occurrence of an object & if it finds one it will delete it and return true, otherwise it will return false.
Because I've overridden equals to return true if two cartItems have the same id I can just create a new cartItem with that id to pass as a parameter.

```
@Override
public Boolean deleteCartItem(String itemId) {
    return cart.remove(new CartItem(itemId, "", 0, 0));
}
```

updateCartItem

This checks if the item exists using the index, then uses the arraylist set method to replace the item.

```
    @Override
    public Boolean updateCartItem(CartItem cartItem) {
        Integer index = cart.indexOf(cartItem);
        if (index != -1) {
            cart.set(index, cartItem);
            return true;
        }
        return false;
    }
```

Task 4. Testing

For testing purposes I changed the quantities of the cart items to 1 to be less confusing.

```
CartItem item1 = new CartItem("000001", "Intel Core i7 CPU", 349.99, 1);
CartItem item2 = new CartItem("000002", "Intel SSD 512GB", 299.99, 1);
```

addToCart

This test adds 5 items to the cart. It then checks if the item exists by finding its index. It then uses the index to retrieve the item and check that its quantity is equal to 5.

First we test that it returns true on a successful add:

```
ArrayList cart = shopCart.getCart();

System.err.println("RETURNS TRUE AFTER ADD:");

Boolean result = true;
for (int i = 0; i < 5; i++) {
    Boolean res = shopCart.addCartItem(item1);
    if (!res) {
        result = false;
    }
}

if (result) {
    System.err.println("SUCCESS: addToCart returns true!");
} else {
    System.err.println("FAILURE: addToCart returns false!");
}
```

Then we check that the item exists in the cart and its quantity has been updated appropriately.

```
ArrayList cart = shopCart.getCart();
Integer index = cart.indexOf(item1);

System.out.print("ITEM EXISTS AFTER INSERT:");

if (index != -1) {
    System.out.println("SUCCESS: Item exists in cart after add!");
    CartItem updatedItem = (CartItem) cart.get(index);
    if (updatedItem.getQuantity() == 5) {
        System.out.println("SUCCESS: Quantity updated after insert!");
    } else {
        System.out.println("FAILURE: Quantity not updated correctly!");
    }
} else {
    System.out.println("Failure: Item does not exist after add!");
}
```

This is the console output from the test

```
Start OF addToCart TESTS

RETURNS TRUE AFTER ADD:
SUCCESS: addToCart returns true!
ITEM EXISTS AFTER INSERT:
SUCCESS: Item exists in cart after add!
SUCCESS: Quantity updated after insert!

END OF addToCart TESTS
```

deleteFromCart

This is the test:

```
74          System.out.println("ITEM DOES NOT EXIST AFTER DELETE");
75          // Remove the item
76          result = shopCart.deleteCartItem(item1.getItemId());
77
78          cart = shopCart.getCart();
79          index = cart.indexOf(item1);
80
            if (index == -1 && result) {
82              System.out.println("SUCCESS: Item does not exist after delete!");
83          } else {
84              System.out.println("FAILURE: Item exists after delete!");
85          }
86
87          System.out.println("\nEND OF deleteFromCart TESTS\n");
```

It calls the delete method and checks that it returns true, and then checks that indexOf returns -1 indicating the item does not exist.

The second test checks that it returns false if the item does not exist.

```
System.err.println("RETURNS FALSE IF ITEM DOES NOT EXIST:");
result = shopCart.deleteCartItem(item1.getItemId());

if (!result) {
    System.out.println("SUCCESS: delete returns false if item does not exist!");
} else {
    System.out.println("FAILURE: delete returns true if item does not exist!");
}
```

Console output:

```
Start OF deleteFromCart TESTS

ITEM DOES NOT EXIST AFTER DELETE
SUCCESS: Item does not exist after delete!
RETURNS FALSE IF ITEM DOES NOT EXIST:
SUCCESS: delete returns false if item does not exist!

END OF deleteFromCart TESTS
```

Update item

In this test I create a new item with the same id as item 2. If the update returns true it checks if the item is updated by checking its fields.

```
100             shopCart.addCartItem(item2);
101             CartItem updatedItem = new CartItem(item2.getItemId(), "updated",9.9, 4);
102
103             result = shopCart.updateCartItem(updatedItem);
104
105             if (result) {
106                 System.out.println("SUCCESS: Returned true on successful update!");
107                 cart = shopCart.getCart();
108                 index = cart.indexOf(item2);
109                 CartItem updateResult = (CartItem) cart.get(index);
110
111                 if (updateResult.getDescription() == updatedItem.getDescription()
112                         && updateResult.getQuantity() == updatedItem.getQuantity()
113                         && updateResult.getUnitPrice() == updatedItem.getUnitPrice()) {
114                     System.out.print("SUCCESS: Updated cart item successfully!");
115                 }
116                 else {
117                     System.out.print("FAILURE: Unable to update cart item!");
118                 }
119             } else {
120                 System.out.println("FAILURE: Returned false on update!");
121             }
```

The second test attempts to update an item not in the cart and checks if it returns false.

```
System.out.println("RETURNS FALSE UPDATING NONEXISTENT ITEM: w");
result = shopCart.updateCartItem(item1);

if (!result) {
    System.out.println("SUCCESS: Returned false updating item that does not exist");
} else {
    System.out.println("FAILURE: Returned true updating item that does not exist");
}
```

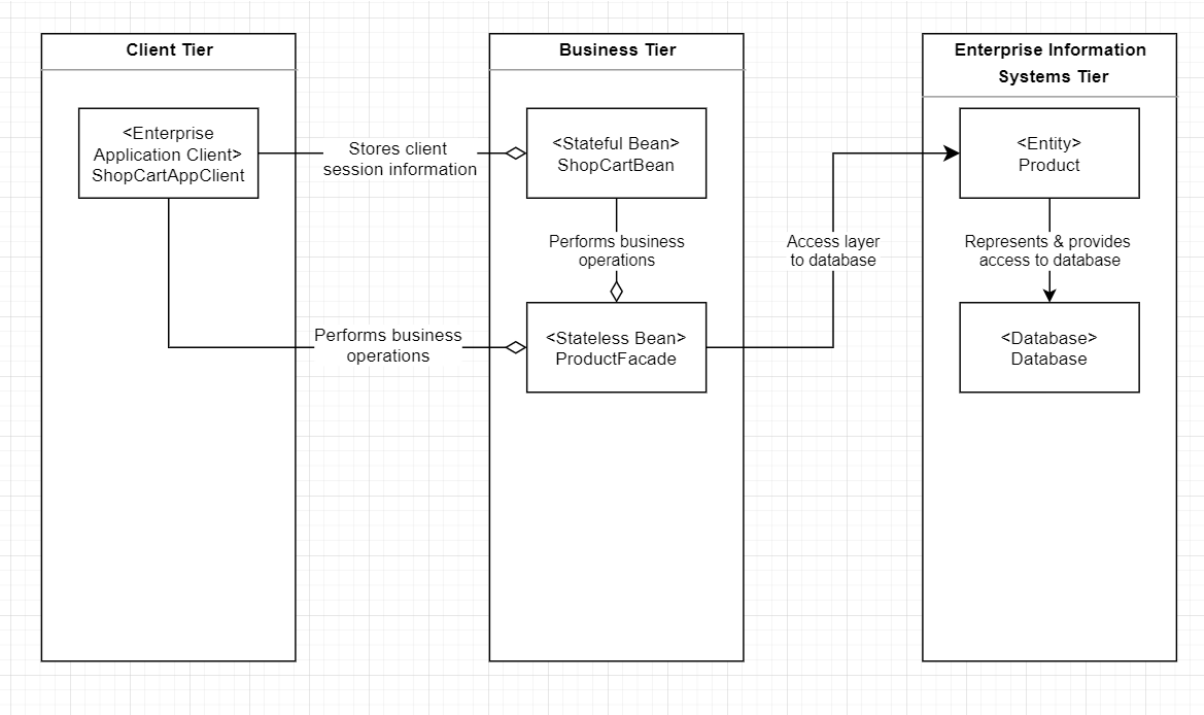
```
START OF updateCartItem TESTS

UPDATE ITEM DETAILS:
SUCCESS: Returned true on successful update!
SUCCESS: Updated cart item successfully!RETURNS FALSE UPDATING NONEXISTENT ITEM: w
SUCCESS: Returned false updating item that does not exist

END OF updateCartItem TESTS
```

Task 5

5.1 Design



5.2

ShopCartClient

This is in the client tier as this software will exist on the client's hardware, rather than on a server instance. Its role is providing an interface for either the user or their system to interact with the enterprise server & database. It does this by making calls to the servers' remote interfaces over the network.

ShopCartBean

Its role is to store the client's session information over the duration of the client's interaction with the system, and to persist this information if necessary.

ProductFacade

This bean would provide business method calls either directly to client, or to other parts of the enterprise system.

Product

The product entity here represents all entities in the system. These are used to define & access the database via objects. These will be used directly by the beans to edit the data in the database, & indirectly represented by DTO's to provide read-only access to the database's information.

Database

The information is stored here and is only access via the entities. This is to simplify accessing data as it is directly represented as objects.