Unit 7 Mid-module Assignment

Design Rationale

Document Contents
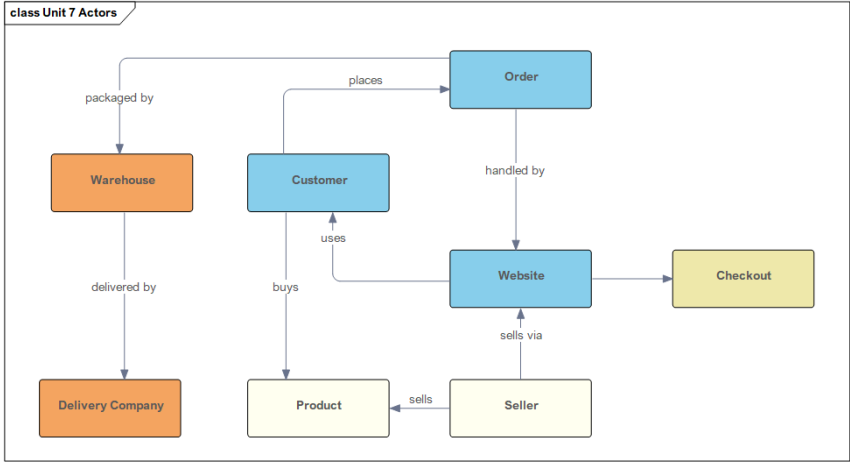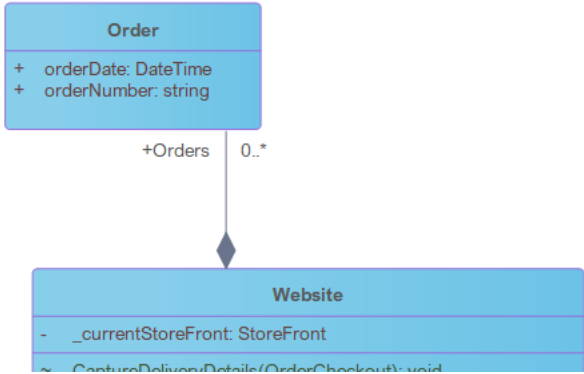
Methodology

Method	Description
Scenario Context	To understand the scenario correctly, I read the scenario text and split each statement into a different requirement (shown in Table 2). For each statement the associated nouns (classes) and verbs (operations) were identified.
Domain knowledge	I used real-world examples such as Amazon to guide the choice of relationships between classes and their multiplicities. I looked at the system as a customer and system implementor—reasoning about which class *owns* what and whether *association, aggregation, composition,* or *dependency* relationships are suitable.
Learned knowledge	Based on learnings from previous units, two points stood out: (1) associations are *structural,* and (2) aggregations (while not recommended due to similarity with the *association*) usually represent *collections*. During the development of the diagrams, I generated relevant source code from the models and tested each requirement against the code model. I modelled classes that adhered to a single responsibility pattern, attempting as far as possible to keep similar functionality isolated within each category. Lastly, the activity diagram utilised Interruptible Activity Region, which I researched to assist in modelling the dynamic behaviour of sequential activity.

Method	Description
Colour Scheme	I used the following colour scheme to group classes by function. This helped to ensure each class was focused on its purpose to meet one or more requirements.
Association Role Names	Where an association end requires a specific name, this is shown on the class diagram as a specific role name (with UML visibility). For instance: Here "Orders" is the name used by Website when associating to an Order class. If no role name is specified, the association's role name is taken from the class name at the end of the association. For instance,
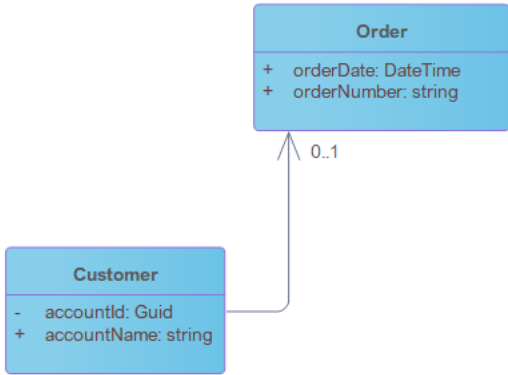
Method	Description
	The role name for Customer referencing an Order will be "Order".

Table 2 Requirements referred to when generating the mid-module UML diagrams

#	System Requirement	Analysis of Operation(s)	Analysis of Class(es)
1.	The system will allow customers to search for products sold at the website and add them to their shopping basket.	SearchForProducts	Customer Product ShoppingBasket
2.	The order is "In Progress" when customers search for products and add them to their shopping basket.	AddProductsToBasket	Order Customer Product ShoppingBasket
3.	Products are sold both by the website and several third-party sellers as part of the website's marketplace.	SellProducts	Product Seller Marketplace
4.	Third-party sellers can be individuals or organisations.	RegisterSeller	Seller
5.	Once third-party sellers have registered as a seller, they maintain their product catalogue, pricing, and storefront.	RegisterSeller CreateCatalog CreatePriceList CreateStoreFront	ProductCatalog PriceList StoreFront
6.	If a product is temporarily out of stock, then then it will be marked as unavailable.	CheckProductAvailability	Product StockLevel

| #	System Requirement	Analysis of Operation(s)	Analysis of Class(es)
7.	Once a customer has finished their shopping, the order is "processing" and they are taken through the checkout process.	CheckOut	Customer Order
8.	The checkout process allows a customer to specify delivery details.	AddDeliveryDetails	Customer DeliveryDetails
9.	Customers have the choice of several delivery options, including standard delivery using the postal service and the company's own in-house courier service.	SelectDeliveryOption	Customer DeliveryOption
10.	Products ordered from third-party sellers at the marketplace do not have access to the in-house courier system.	FilterDeliveryOptions	Product Seller Marketplace DeliveryOption
11.	After delivery options are chosen, the customer is presented with the payment options.	AddPaymentDetails	DeliveryOption Customer PaymentOption
12.	Payment options support a credit or debit card, an online payment service (e.g., PayPal) or a gift voucher.		PaymentOption
13.	An additional payment option is a promotional code.	AddPromoCode	PromoCode
14.	Promotional codes are periodically issued by the website.	CreatePromoCode	PromoCode Website

#	System Requirement	Analysis of Operation(s)	Analysis of Class(es)
15.	Customers can store their payment details.	StorePaymentDetails	Customer PaymentDetails
16.	Customers' payment details can be reused for each purchase.	RetrievePaymentDetails	Customer PaymentDetails
17.	Before the payment details are submitted, the order is "pending payment".		PaymentDetails Order
18.	After the customer submits their payment information, the validity of the account used (whether it is PayPal, a credit/debit card or gift voucher) will be verified by the appropriate service.	ValidatePaymentDetails	Customer PaymentDetails PaymentAccount PaymentService
19.	When the payment account is verified successfully, the order is "awaiting picking".	SubmitOrderForPicking	PaymentAccount Order
20.	When a customer completes a transaction, the order is passed to the warehouse staff.	SubmitOrderForPicking	Customer Transaction Order WarehouseStaff
21.	The warehouse staff look up the (order) items on the stock database.	LookupOrderStockItems	WarehouseStaff StockDatabase OrderItem
22.	The Stock database determines the location of the item in the warehouse.	LocateStockLocation	StockDatabase ItemLocation

#	System Requirement	Analysis of Operation(s)	Analysis of Class(es)
23.	Once the items are successfully collected and packaged, the order will be considered "ready for delivery".	PackageStockItem	OrderItem Order
24.	When the order is "ready for delivery", it is distributed to the appropriate delivery company, depending on the customer's preferences.	SendPackageToDelivery Company	Order DeliveryCompany DeliveryOption
25.	When a member of the warehouse staff picks an item, the item is scanned with a handheld terminal.	ScanStockItem	WarehouseStaff OrderItem HandheldTerminal
26.	The handheld terminal automatically updates the product stock levels (reflected on the website).	UpdateStockLevels	HandheldTerminal StockLevel Website
27.	Finally, the order is collected by the appropriate delivery company and marked as "shipped".	DeliverPackage	Order DeliveryCompany

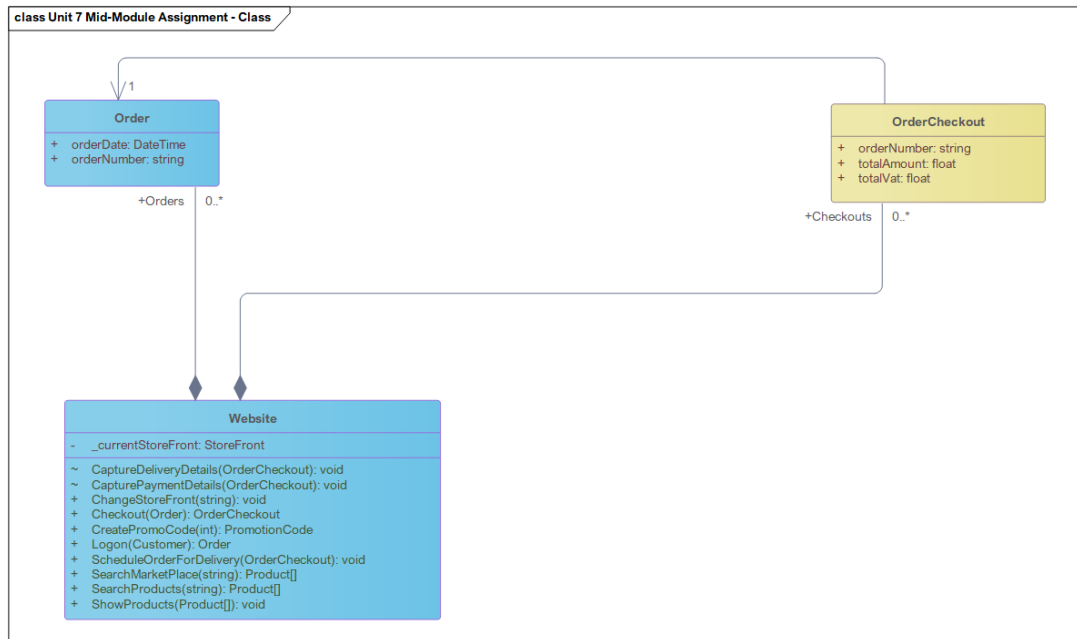# UML Diagrams

Class Diagram

*Figure 1 Mid-module class diagram overview*
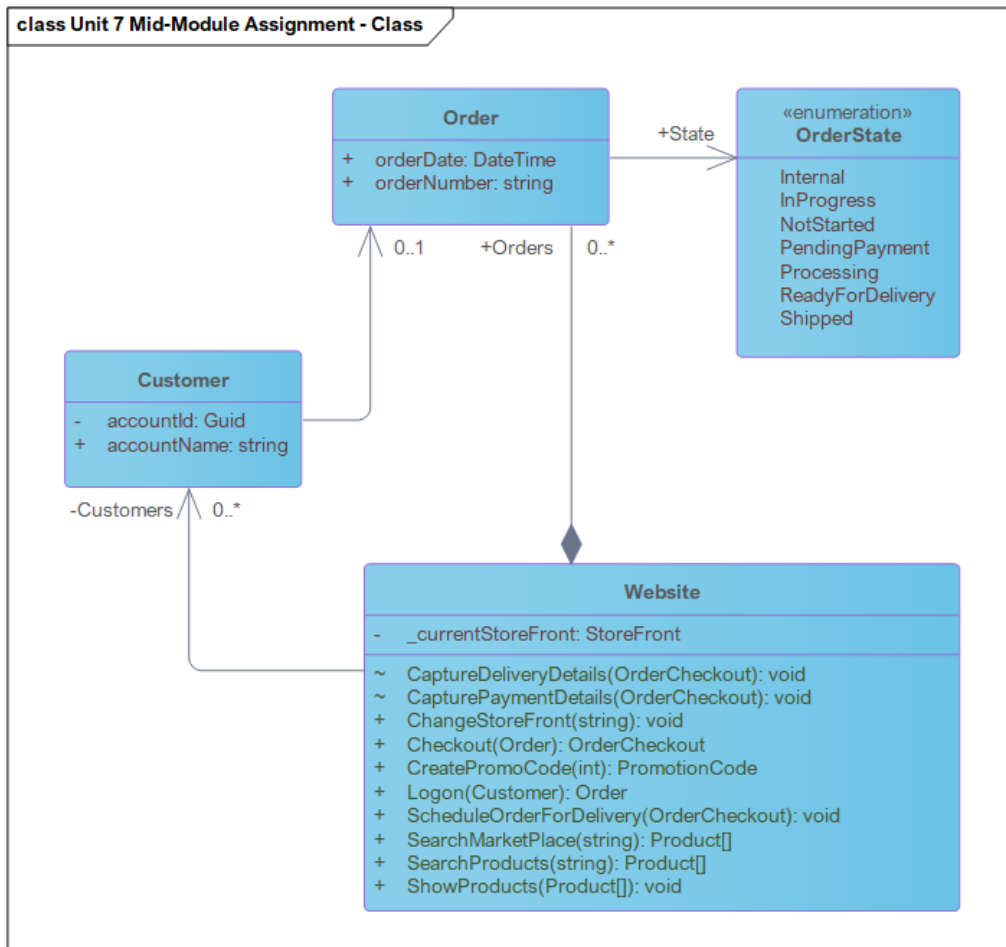
- A website **creates** zero or more orders.
- Orders start when a customer logs onto the website. After logon, the website passes the customer a **reference to** an order.

```
class Unit 7 Mid-Module Assignment - Class
```

Order
+ orderDate: DateTime
+ orderNumber: string

OrderCheckout
+ orderNumber: string
+ totalAmount: float
+ totalVat: float

+Orders   0..*

+Checkouts   0..*

Website
- _currentStoreFront: StoreFront
~ CaptureDeliveryDetails(OrderCheckout): void
~ CapturePaymentDetails(OrderCheckout): void
+ ChangeStoreFront(string): void
+ Checkout(Order): OrderCheckout
+ CreatePromoCode(int): PromotionCode
+ Logon(Customer): Order
+ ScheduleOrderForDelivery(OrderCheckout): void
+ SearchMarketPlace(string): Product[]
+ SearchProducts(string): Product[]
+ ShowProducts(Product[]): void

- Checkout functionality provided via the website; therefore, website **creates** and gathers checkout data and provides a **reference to** the order being checked out.
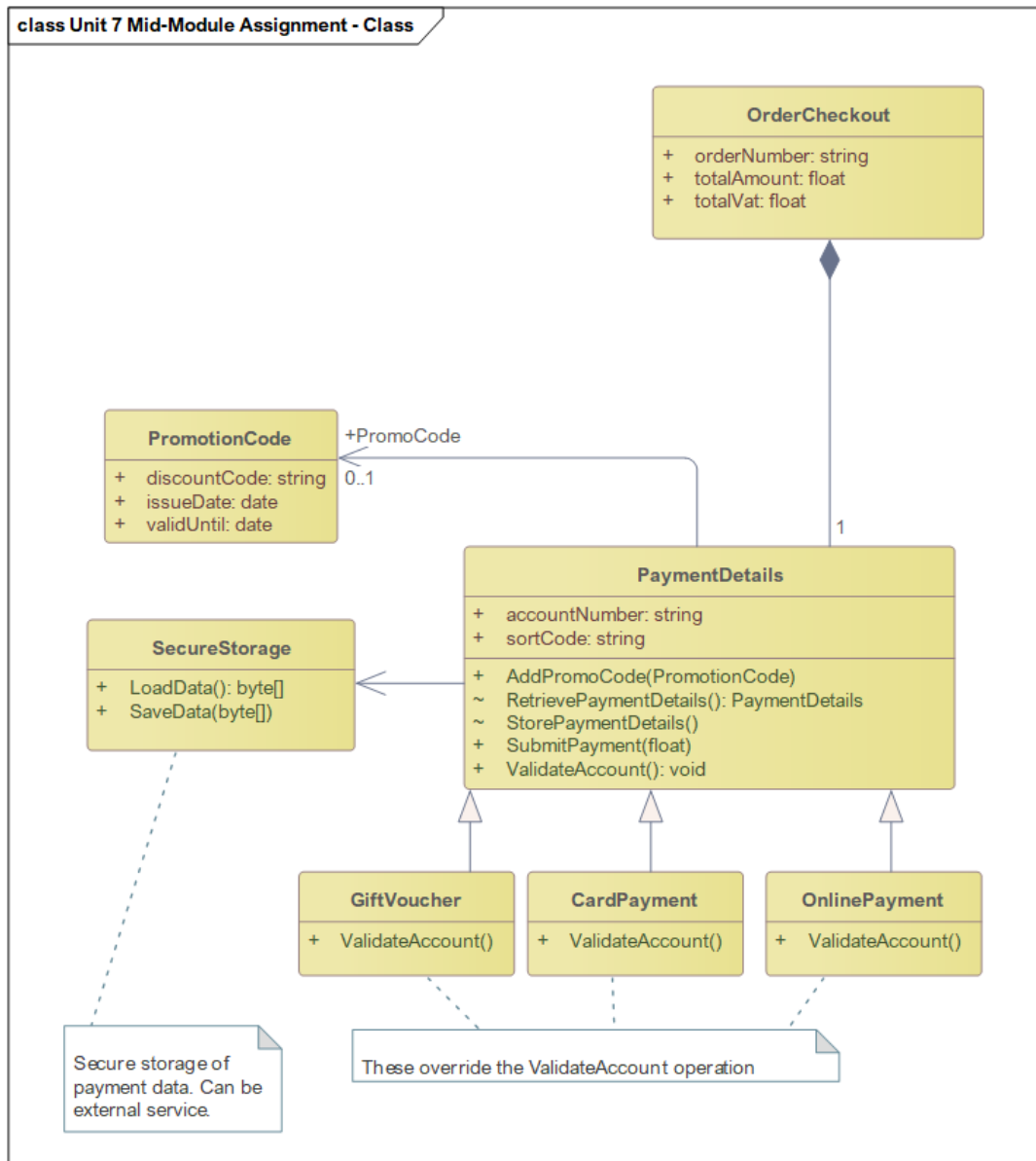
- Payment details supports securely storing details and **uses** a storage component—this knowledge is not duplicated across child types.
- Promo code provides payment discounts. Therefore, payment details **refer to** zero or one promo code.
- Every order checkout **has** a single payment record.
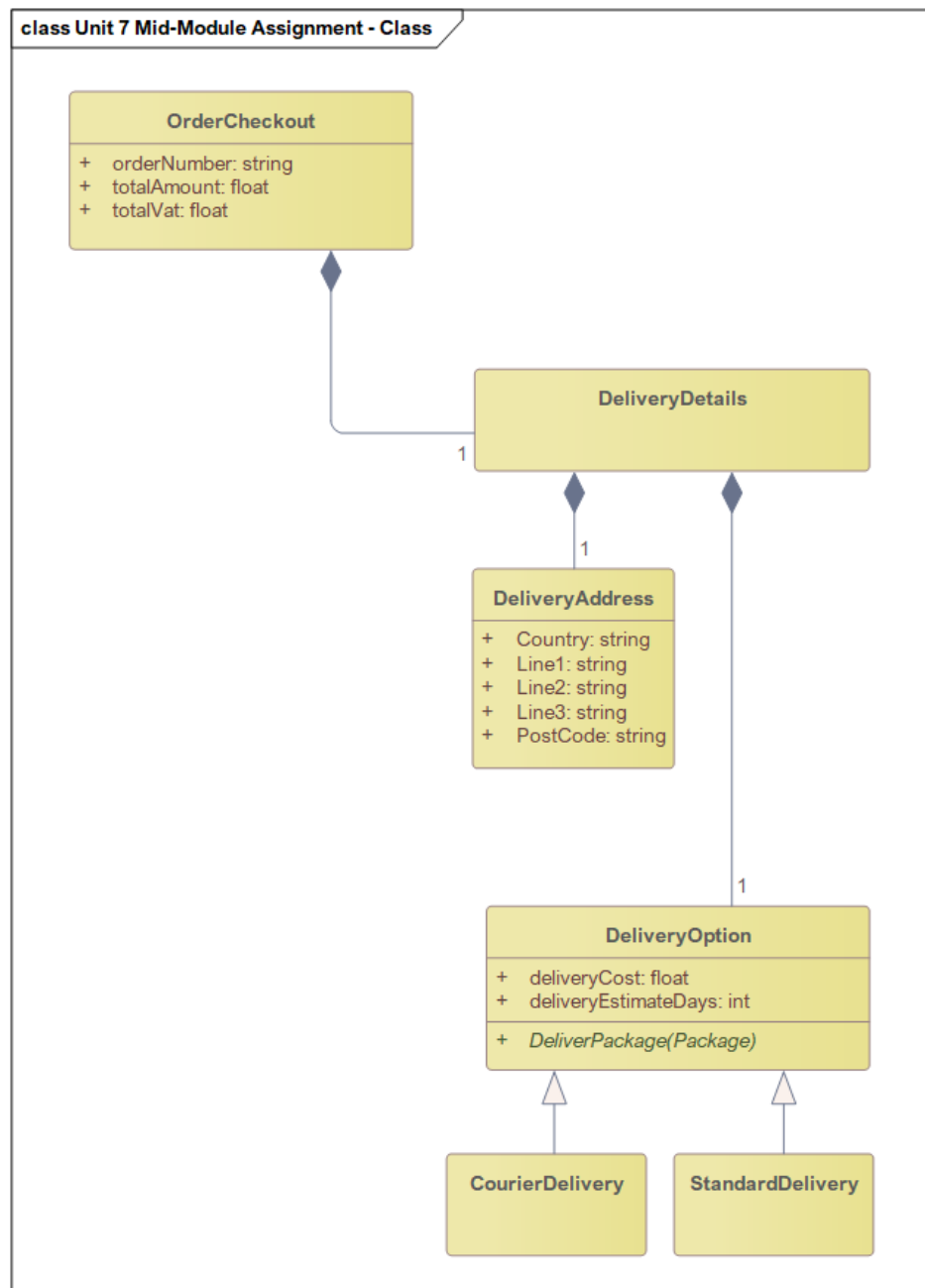
- Delivery details **contains** one address and one **generalisation** of a delivery option. Sellers hold the actual instance of DeliveryOption used for delivery process.

Rationale for Website-to-ShoppingBasket


```
class Unit 7 Mid-Module Assignment - Class

┌─────────────────────────────┐              ┌──────────────────────────────┐
│        ShoppingBasket       │              │            Order             │
├─────────────────────────────┤              ├──────────────────────────────┤
│ +   AddProducts(Product[])  │      1  ◆    │ +   orderDate: DateTime      │
│                             │              │ +   orderNumber: string      │
└─────────────────────────────┘              └──────────────────────────────┘
                                                0..1      +Orders    0..*
```

ShoppingBasket

+ AddProducts(Product[])   1

Order

+ orderDate: DateTime
+ orderNumber: string

0..1   +Orders   0..*

Customer

- accountId: Guid
+ accountName: string

-Customers   0..*

Website

- _currentStoreFront: StoreFront

~ CaptureDeliveryDetails(OrderCheckout): void
~ CapturePaymentDetails(OrderCheckout): void
+ ChangeStoreFront(string): void
+ Checkout(Order): OrderCheckout
+ CreatePromoCode(int): PromotionCode
+ Logon(Customer): Order
+ ScheduleOrderForDelivery(OrderCheckout): void
+ SearchMarketPlace(string): Product[]
+ SearchProducts(string): Product[]
+ ShowProducts(Product[]): void

- Order (financial transaction) **owns** a single shopping basket that contains the transaction's line items. Line items are removed if the transaction is removed.

Rationale for Website-to-StoreFront
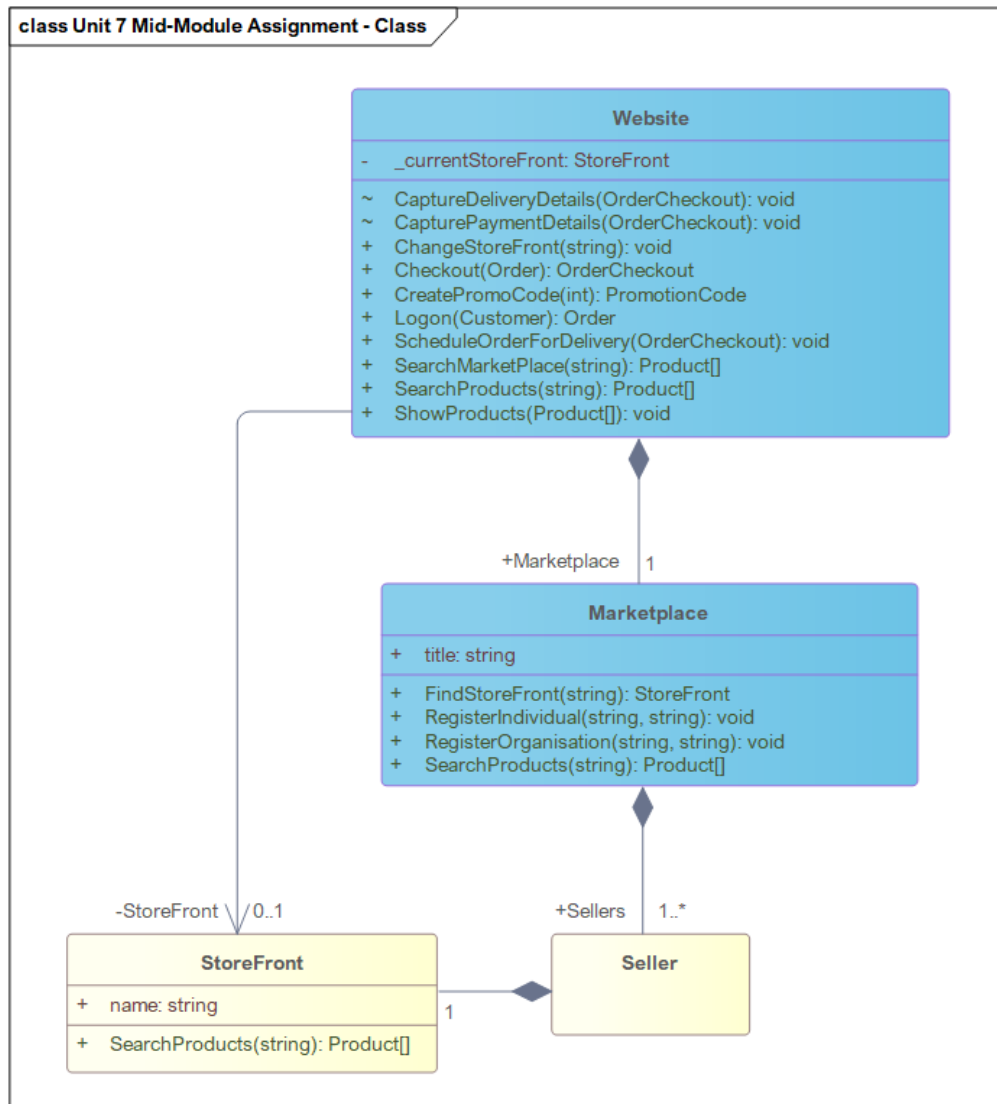
- Website **has** a single storefront **owned** by a seller.
- Customers search for products from a single storefront (`SearchProducts()` operation) or the entire marketplace (`SearchMarketplace()` operation).
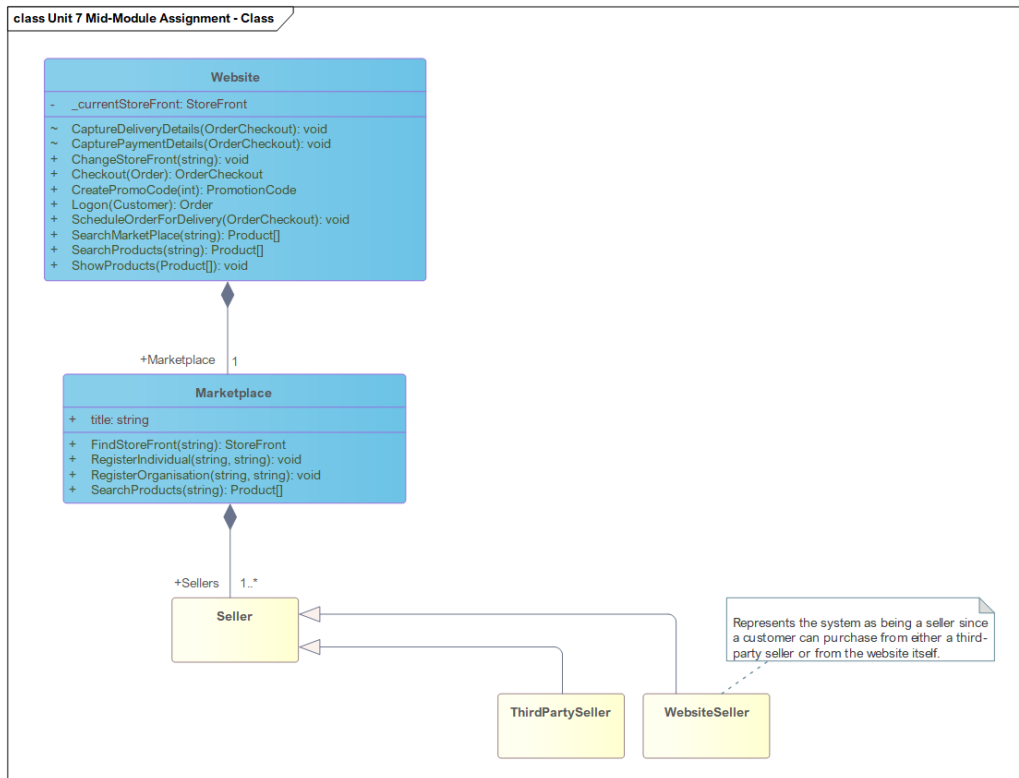
- Website **has** a single marketplace that registers sellers.
- At least one seller (WebsiteSeller) is always present (the marketplace constructor creates an instance of WebsiteSeller)
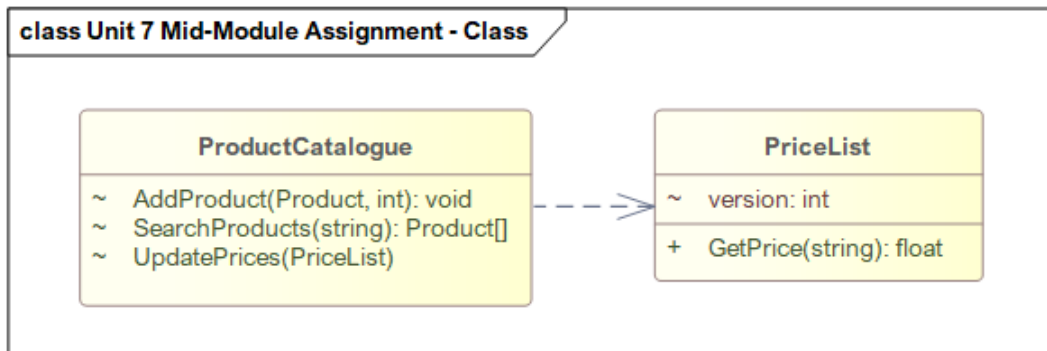
Rationale for Seller-to-DeliveryOptions
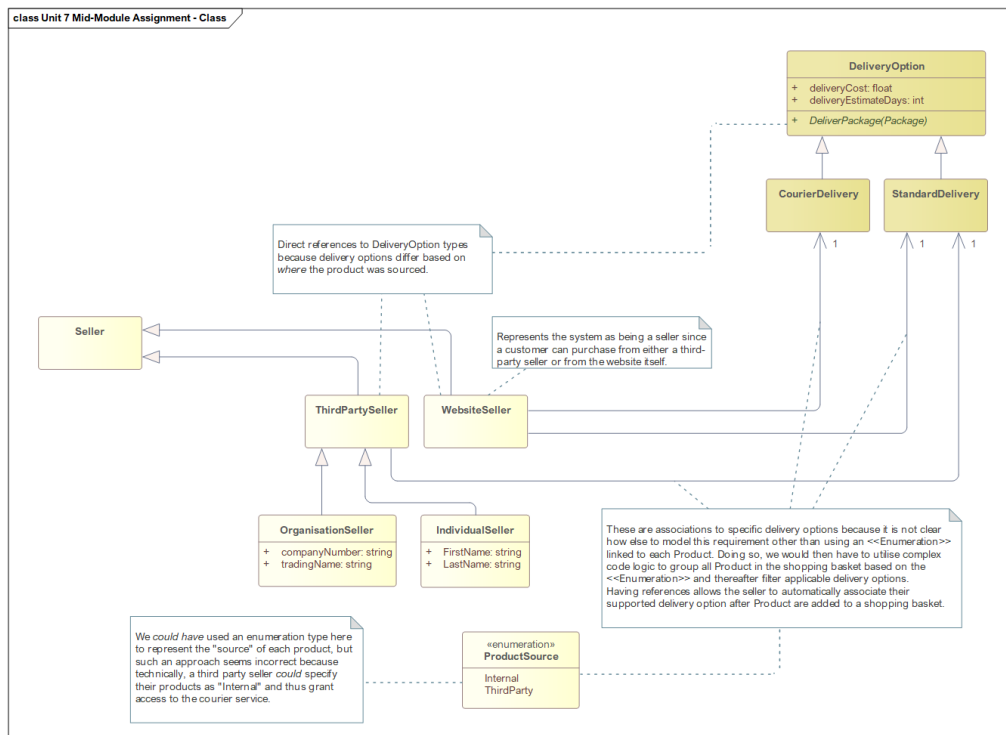
- ThirdPartySeller and WebsiteSeller *reference* delivery options because it is easier to model this association than use complex specialisations of product or attaching enumerations to a product. *Please see attached notes.*
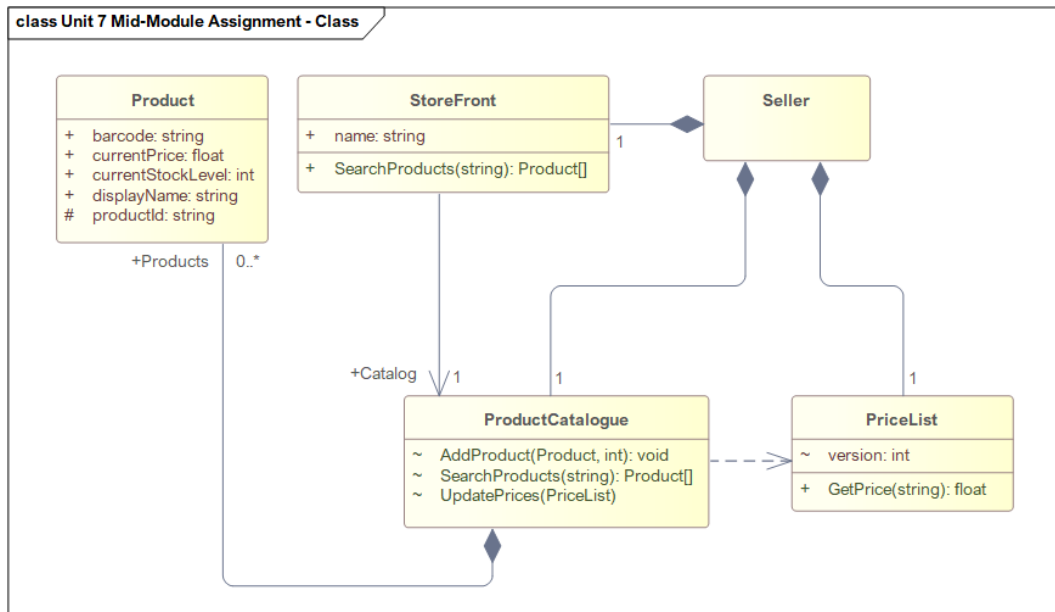
- Seller ***owns*** a single storefront, catalogue, and price list.

- A catalogue ***consists of*** many products.

- The storefront ***references*** the seller's catalogue (shown to the customer).

class Unit 7 Mid-Module Assignment - Class

ProductCatalogue

~   AddProduct(Product, int): void
~   SearchProducts(string): Product[]
~   UpdatePrices(PriceList)

PriceList

~   version: int

+   GetPrice(string): float

- The catalogue **uses** a price list which is *dynamically* provided by a seller.

Rationale for *Website-to-Warehouse*

class Unit 7 Mid-Module Assignment - Class

Warehouse

~   AddOrderForDelivery(OrderCheckout)
+   GetAvailableTerminal(): HandheldTerminal
+   PackageOrder(OrderCheckout): Package
+   RetrieveNextOrder(): OrderCheckout
+   SendPackageForDelivery(Package)

-Warehouse

1

Website

-   _currentStoreFront: StoreFront

~   CaptureDeliveryDetails(OrderCheckout): void
~   CapturePaymentDetails(OrderCheckout): void
+   ChangeStoreFront(string): void
+   Checkout(Order): OrderCheckout
+   CreatePromoCode(int): PromotionCode
+   Logon(Customer): Order
+   ScheduleOrderForDelivery(OrderCheckout): void
+   SearchMarketPlace(string): Product[]
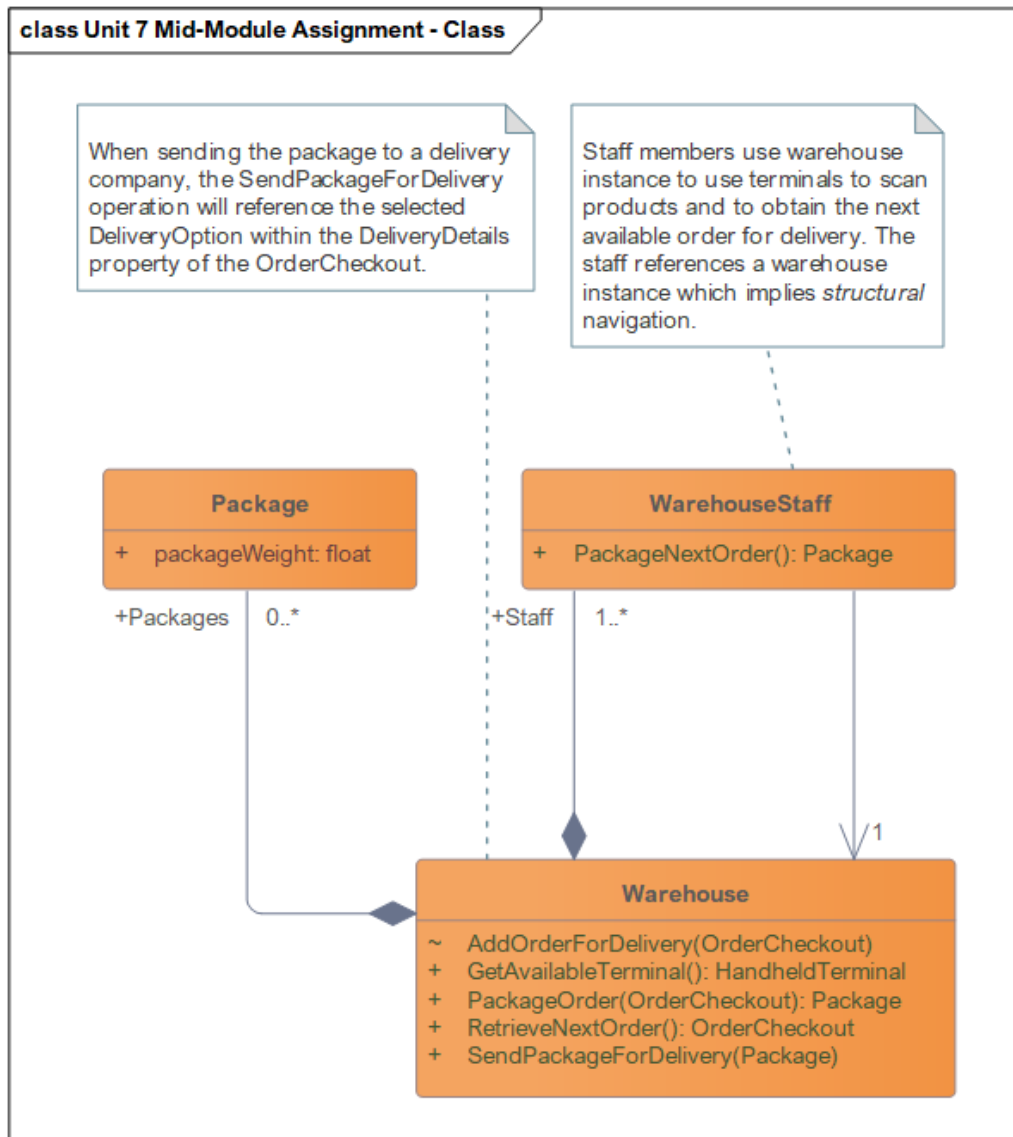+   SearchProducts(string): Product[]
+   ShowProducts(Product[]): void

- Website **uses** a *private* instance of warehouse to start the delivery process. We do not expose this field to the outside world.
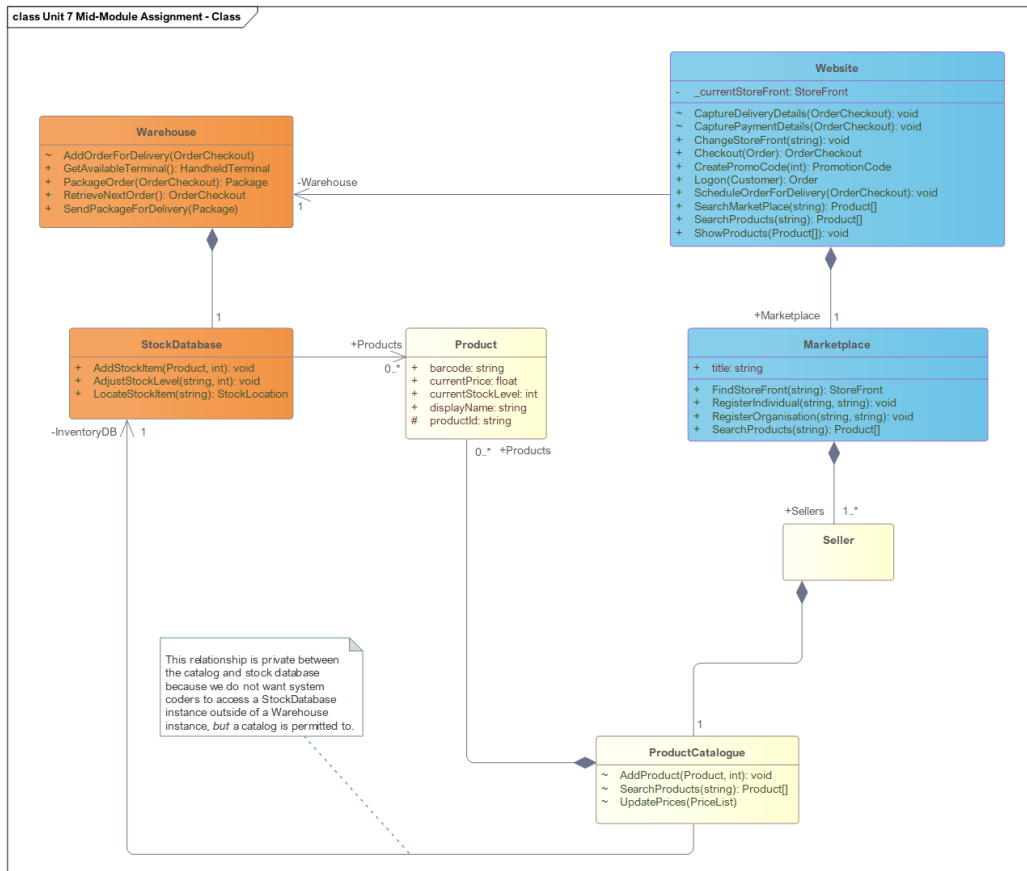
- Catalogue **references** a stock database passed from website to marketplace and from marketplace to every seller, who then provide it into their catalogue.

class Unit 7 Mid-Module Assignment - Class

When sending the package to a delivery company, the SendPackageForDelivery operation will reference the selected DeliveryOption within the DeliveryDetails property of the OrderCheckout.

Staff members use warehouse instance to use terminals to scan products and to obtain the next available order for delivery. The staff references a warehouse instance which implies *structural* navigation.

Package

+ packageWeight: float

+Packages   0..*

WarehouseStaff

+ PackageNextOrder(): Package

+Staff   1..*

1

Warehouse

~ AddOrderForDelivery(OrderCheckout)
+ GetAvailableTerminal(): HandheldTerminal
+ PackageOrder(OrderCheckout): Package
+ RetrieveNextOrder(): OrderCheckout
+ SendPackageForDelivery(Package)

- Warehouse ***owns/stores*** zero or more packages created by staff. If the warehouse disappears, so too do the packages.
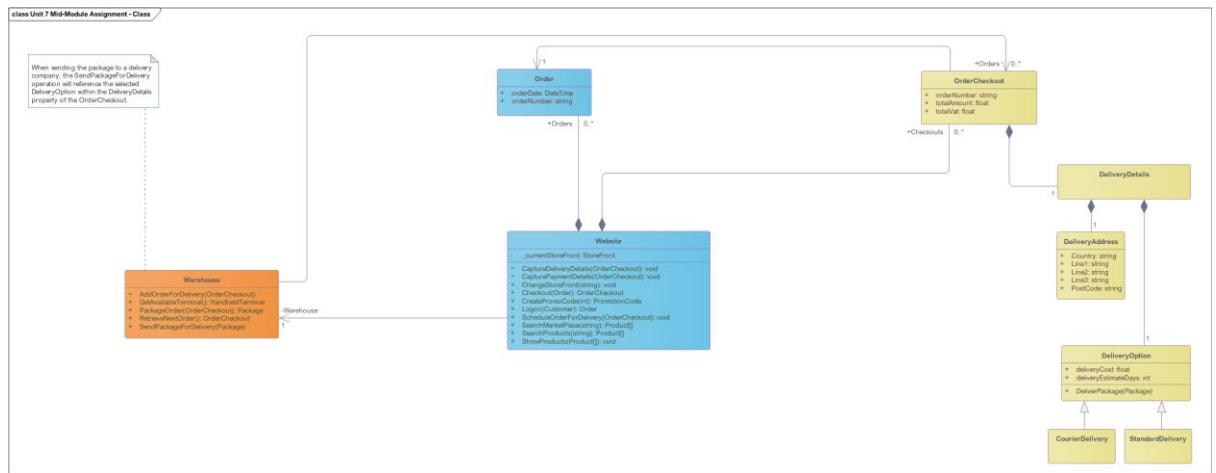
- Warehouse *references* zero or more checkouts (created by website) for orders that are being packaged. The selected delivery company is obtained from the delivery options of a checkout.
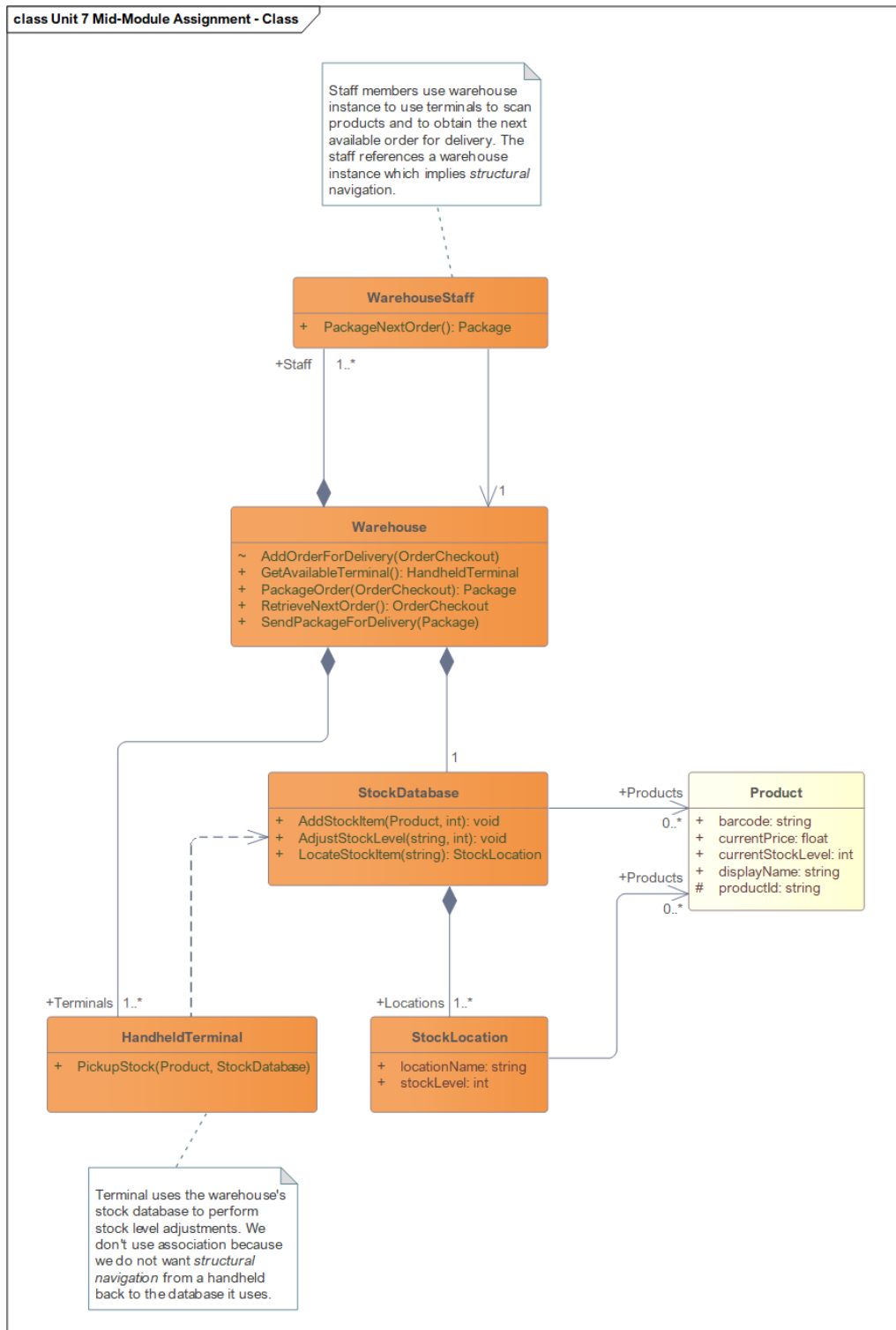
- Staff *uses* one warehouse instance to retrieve a single terminal.

- Terminal ***depends on*** a single stock database because terminal requires functionality of the database that holds information about multiple stock items, their location and stock levels.

Activity Diagram

Figure 2 used requirements in Table 2 to model the sequential flow of actions that support a customer placing an online order.
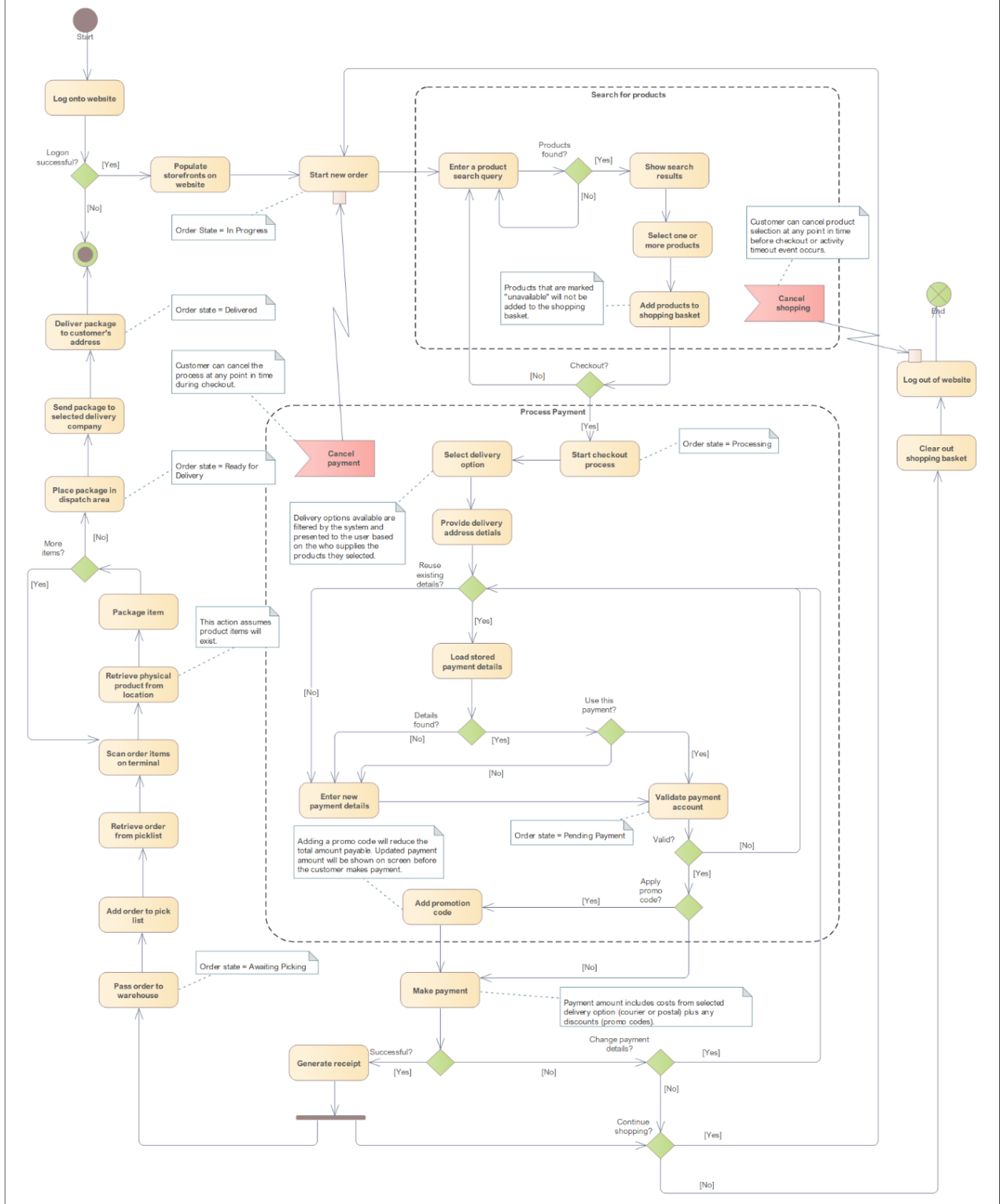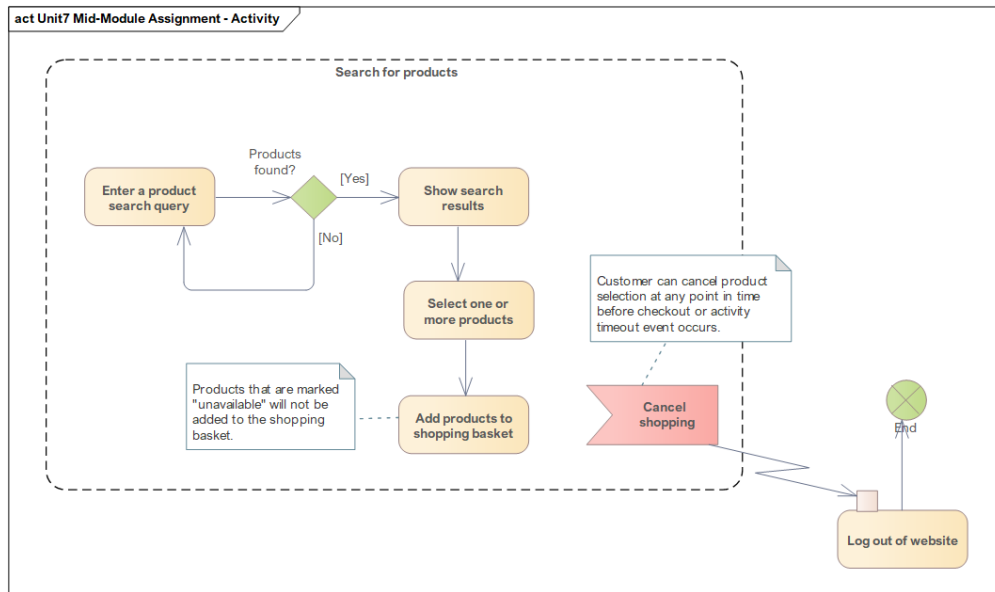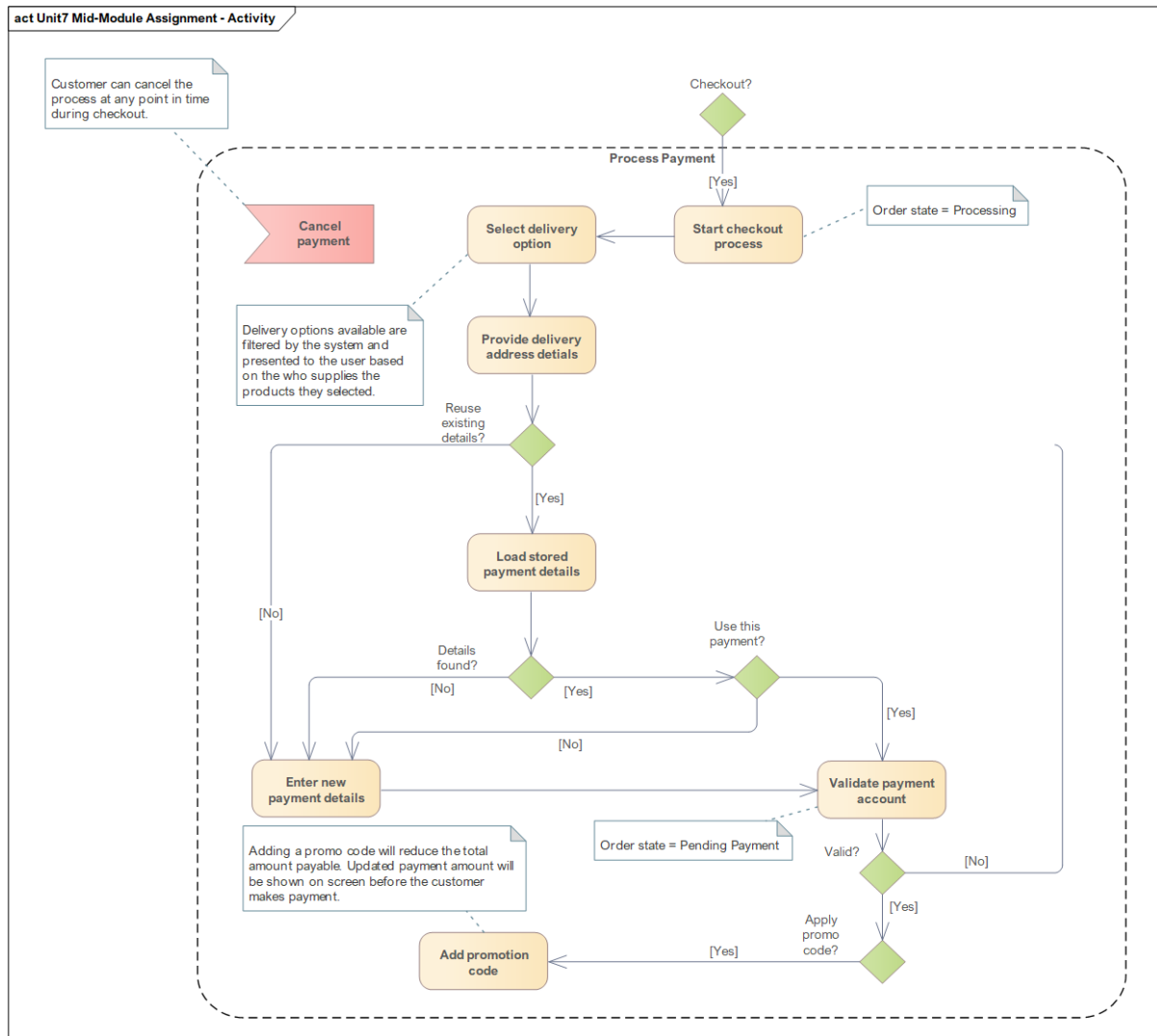
*Figure 2 Mid-module activity diagram overview*

- Searching for products can be **interrupted** by the "Cancel Shopping" event. *Interruptible Activity Region* is helpful to depict activities that can terminate abruptly.
- Activity pin on "Log out of website" represents an interaction point in the activity.

- The payment process can be **interrupted** by the "Cancel Payment" event.
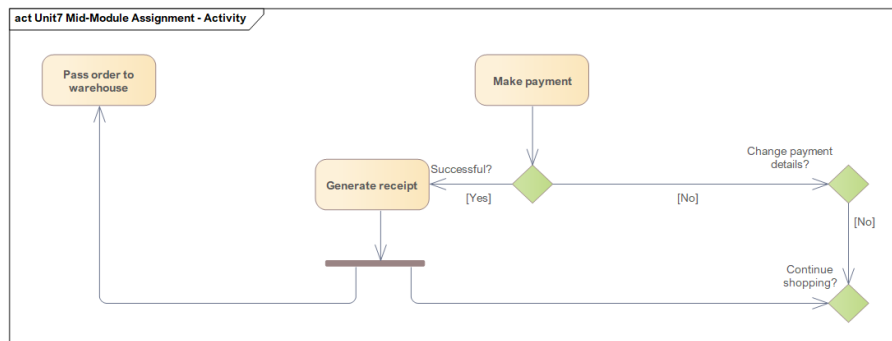
- The flow path **forks** into two tracks after Generate Receipt since submitting an order to the warehouse executes in parallel with allowing the user to continue shopping or not.

State Diagram

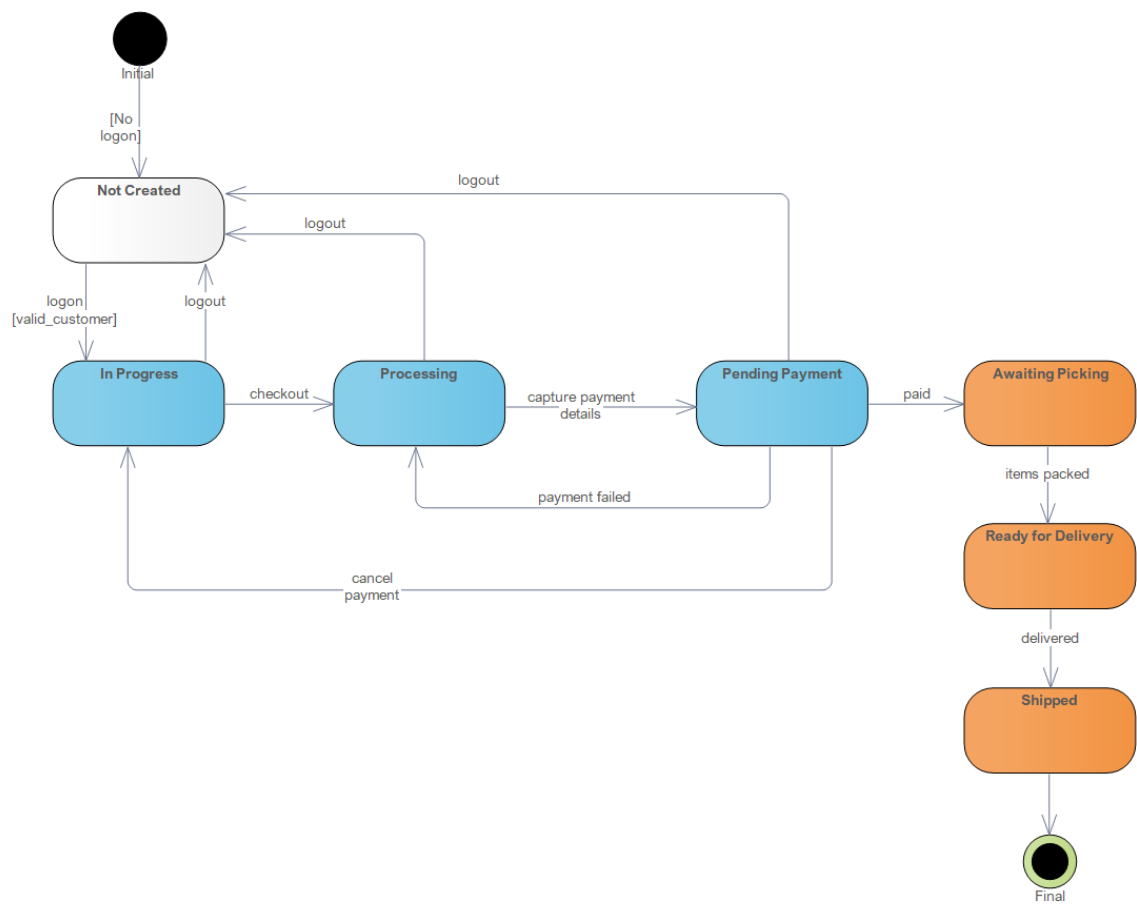Figure 3 shows state changes for an Order instance as it moves through the system. The state change is based on the activity diagram in Figure 2.

Figure 3 Order state changes