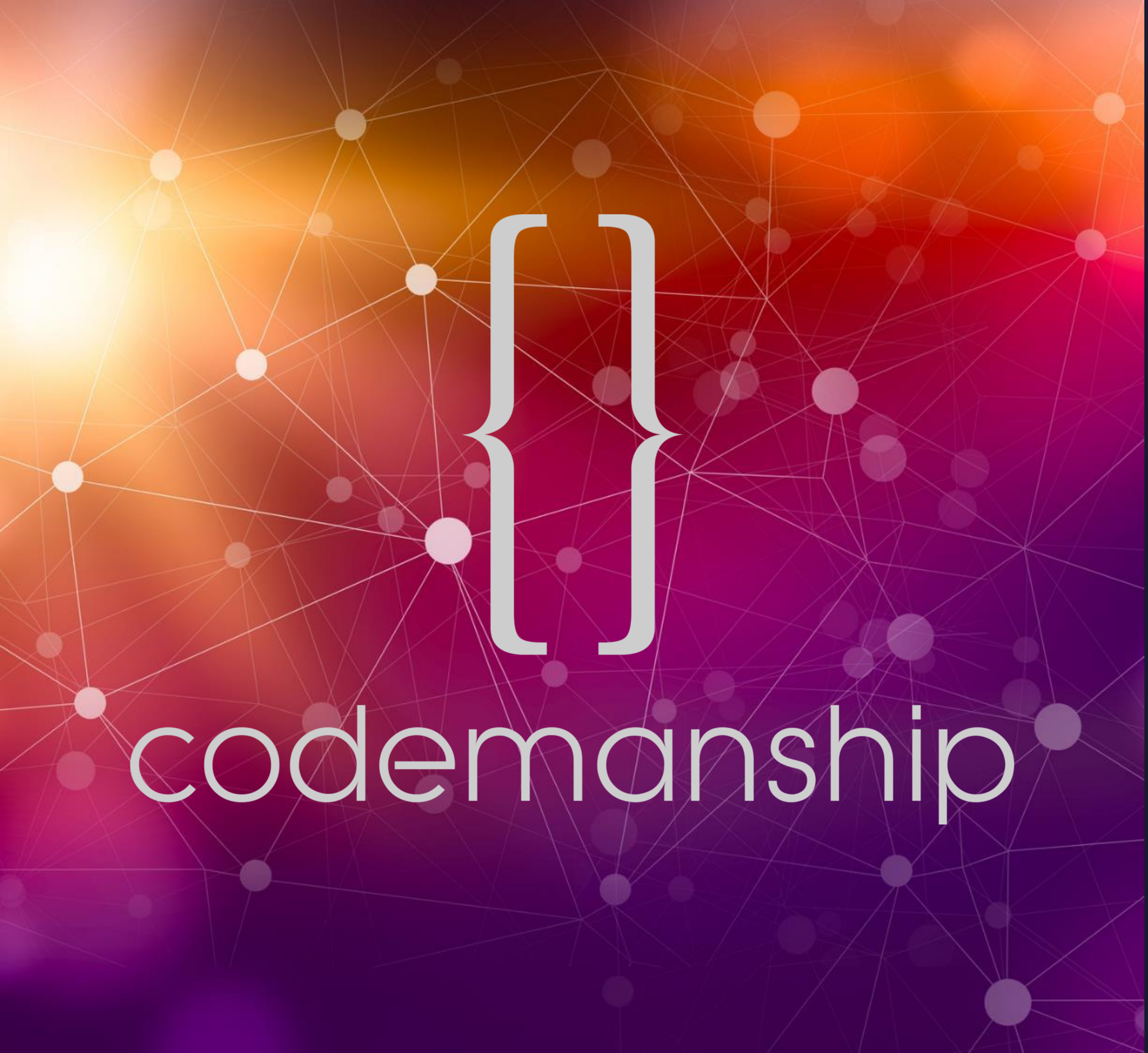
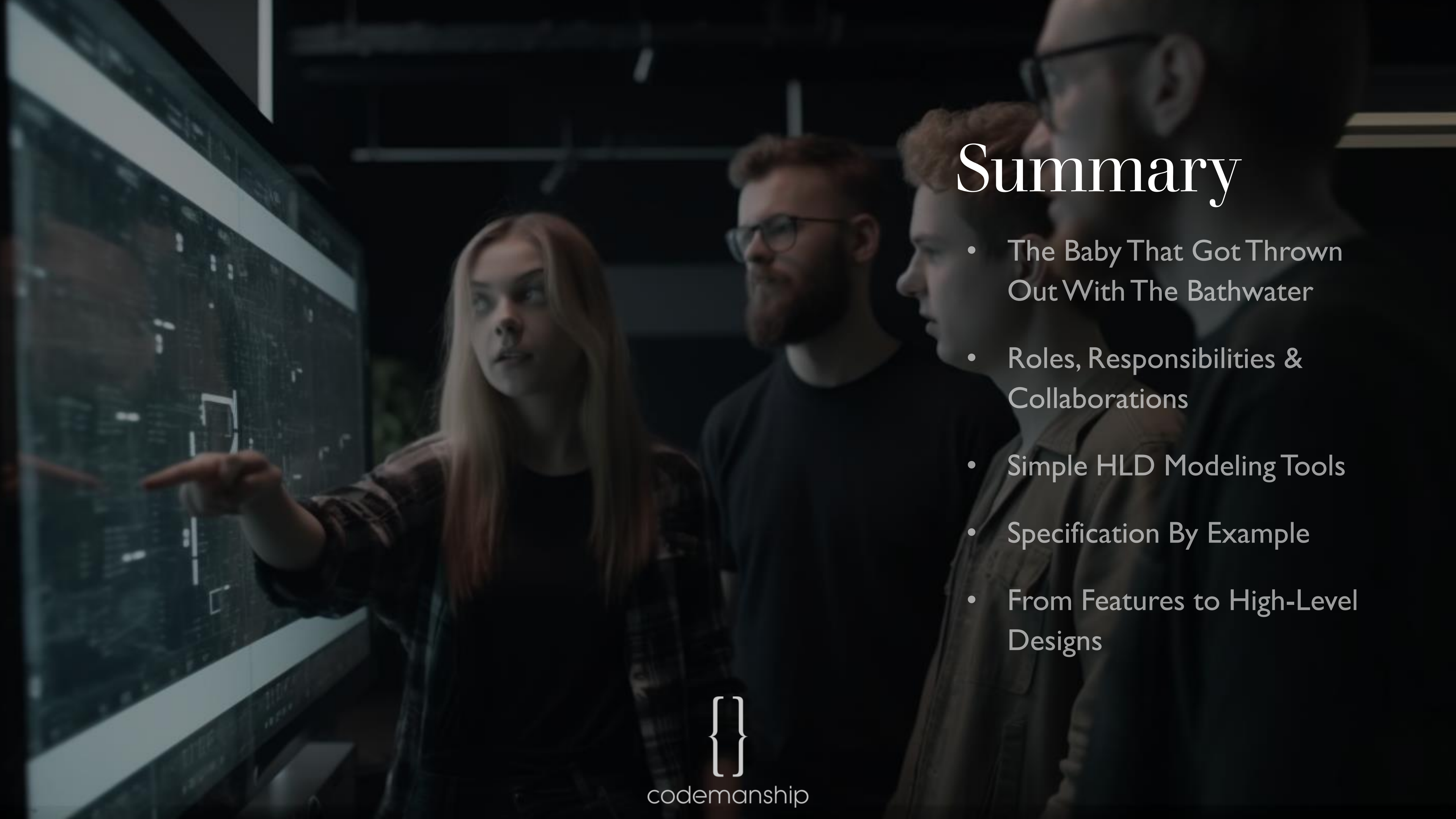


High-Level Design

A Test-Driven Approach



Summary

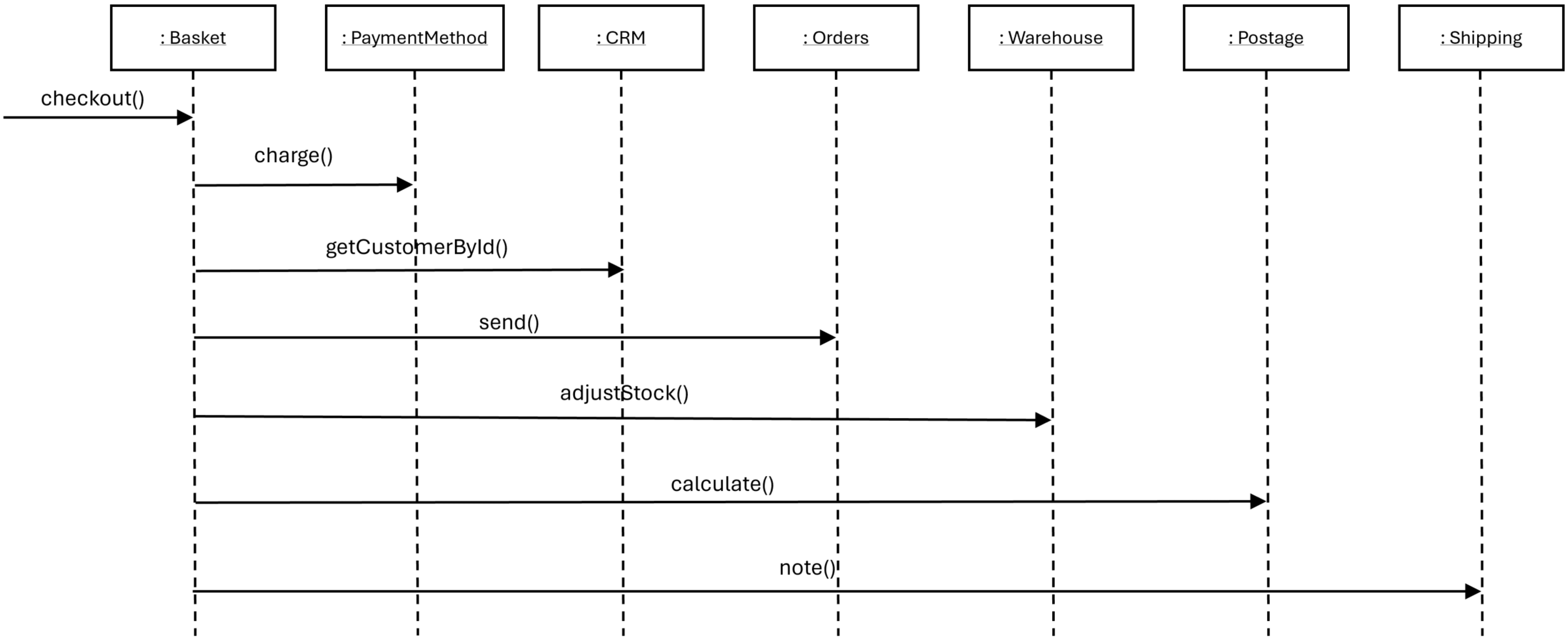
- The Baby That Got Thrown Out With The Bathwater
- Roles, Responsibilities & Collaborations
- Simple HLD Modeling Tools
- Specification By Example
- From Features to High-Level Designs

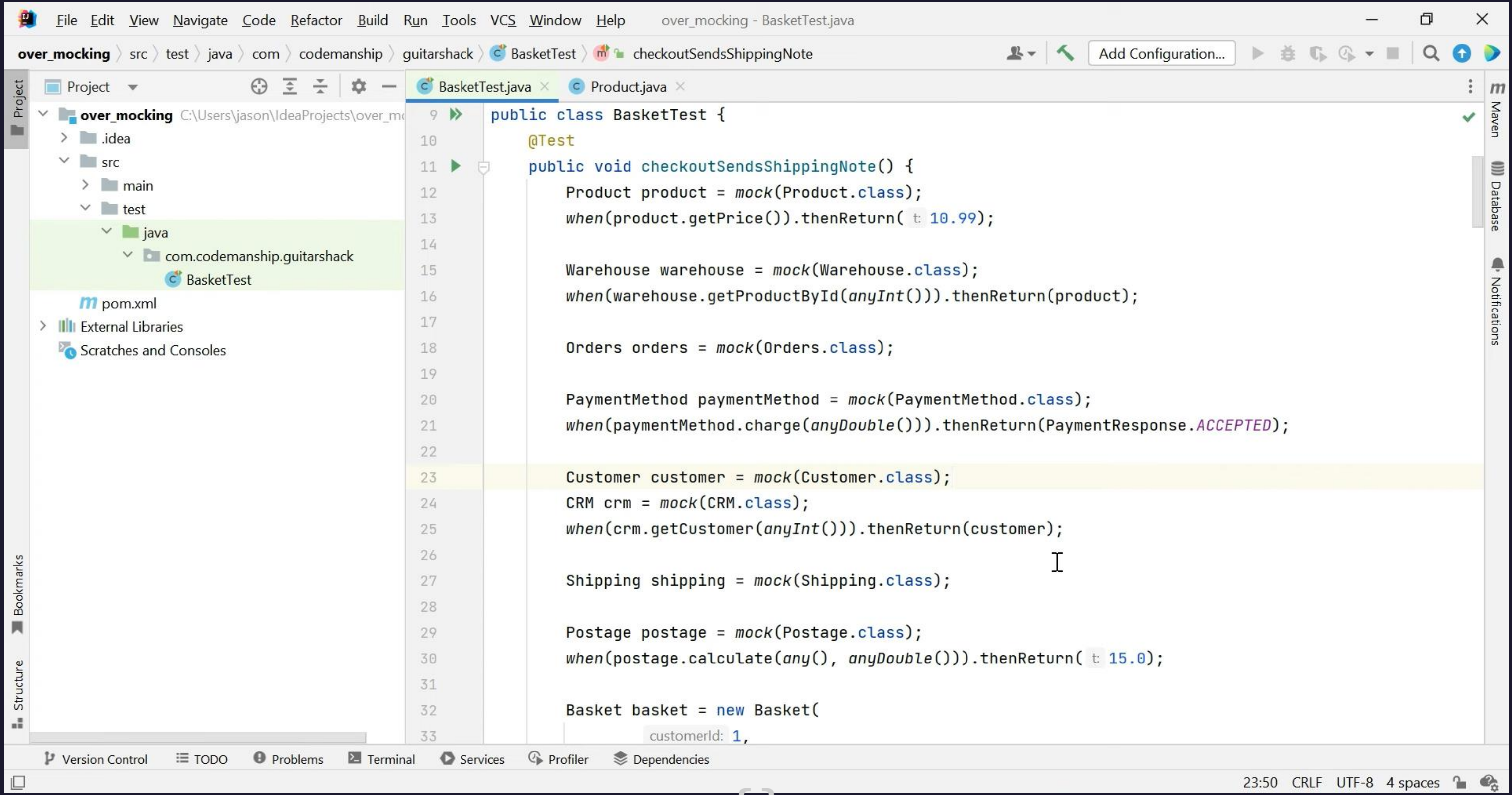
codemanship

The Baby That Got Thrown Out With The Bathwater

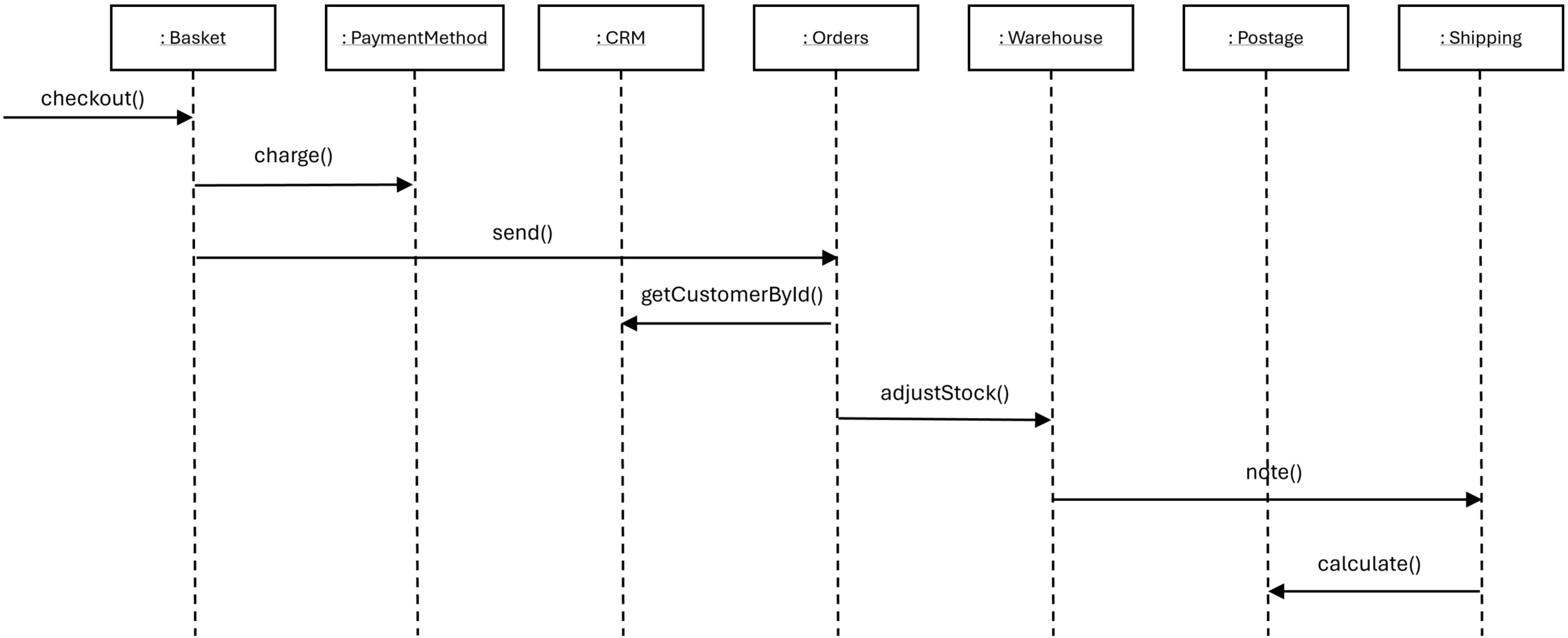
How Jumping Straight Into Code Created an Epidemic of God Modules

codemanship


: Basket
: PaymentMethod
: CRM
: Orders
: Warehouse
: Postage
: Shipping
checkout()
charge()
getCustomerById()
send()
adjustStock()
calculate()
note()

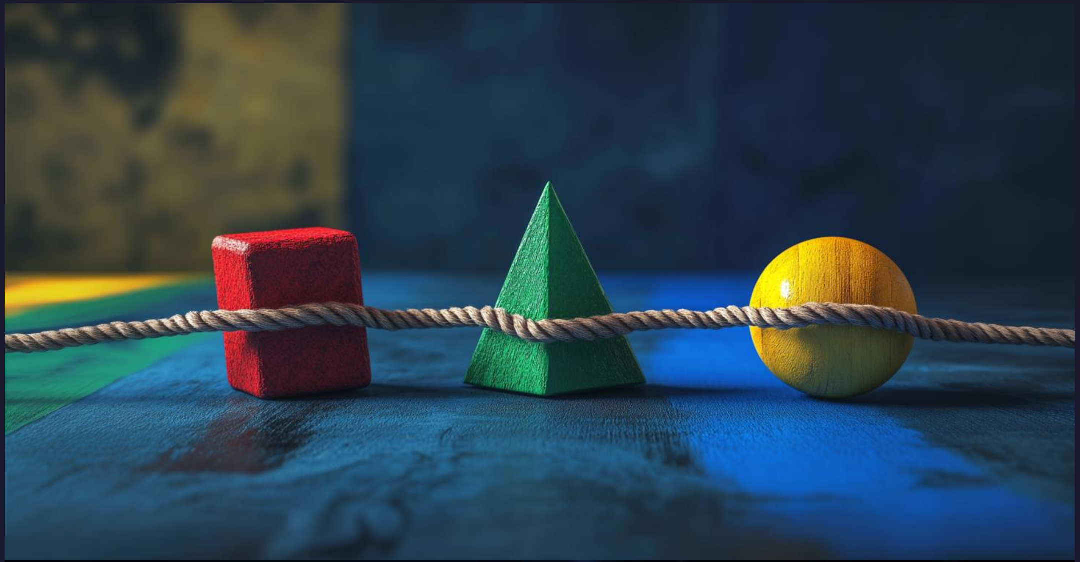

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
over_mocking - BasketTest.java
over_mocking 〉 src 〉 test 〉 java 〉 com 〉 codemanship 〉 guitarshack 〉 BasketTest 〉 checkoutSendsShippingNote
Add Configuration...
Project
over_mocking C:\Users\jason\IdeaProjects\over_mo
.idea
src
main
test
java
com.codemanship.guitarshack
BasketTest
pom.xml
External Libraries
Scratches and Consoles
BasketTest.java
Product.java
Maven
Database
Notifications
Bookmarks
Structure
Version Control
TODO
Problems
Terminal
Services
Profiler
Dependencies
23:50 CRLF UTF-8 4 spaces

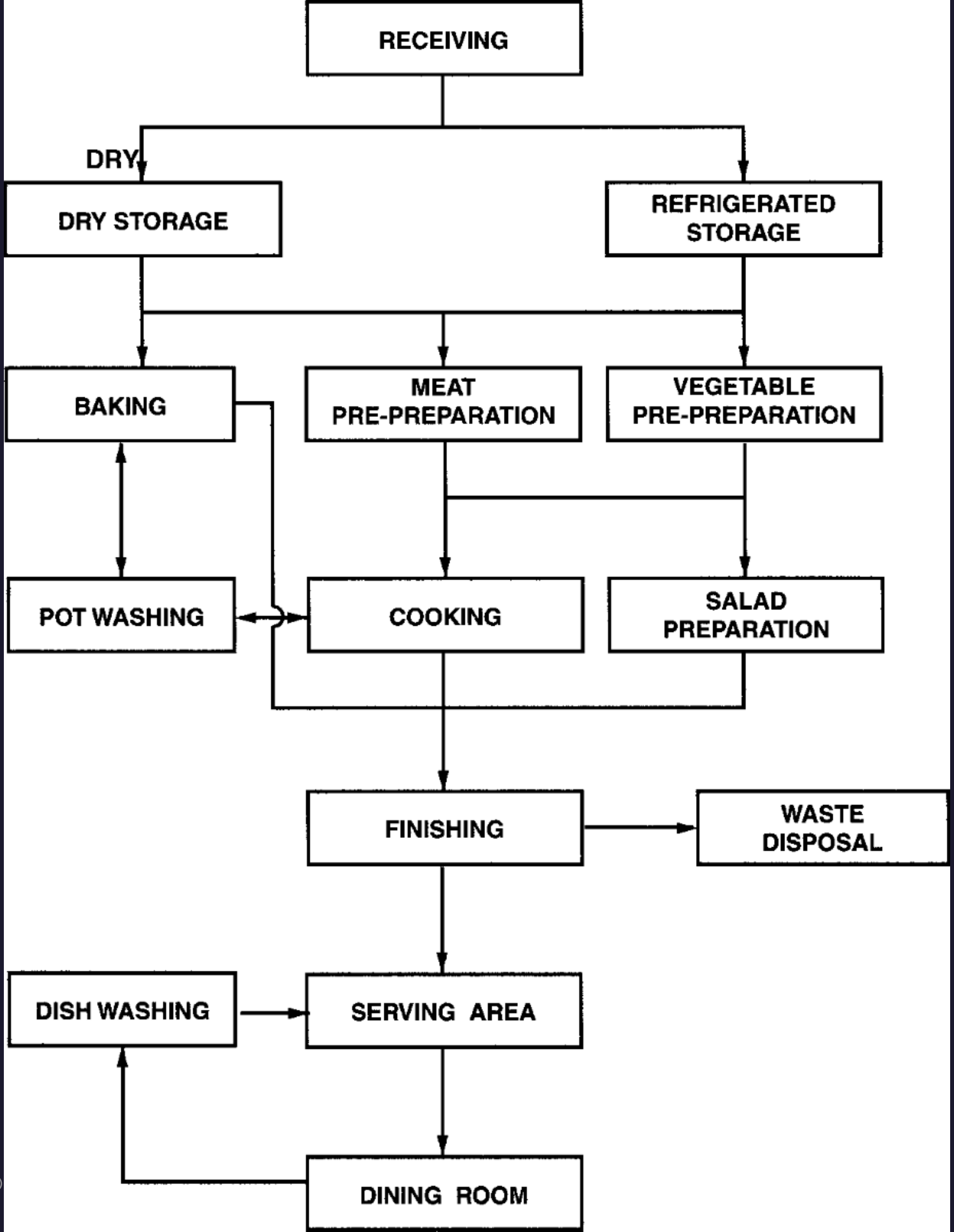
```java
public class BasketTest {
    @Test
    public void checkoutSendsShippingNote() {
        Product product = mock(Product.class);
        when(product.getPrice()).thenReturn( t: 10.99);

        Warehouse warehouse = mock(Warehouse.class);
        when(warehouse.getProductById(anyInt())).thenReturn(product);

        Orders orders = mock(Orders.class);

        PaymentMethod paymentMethod = mock(PaymentMethod.class);
        when(paymentMethod.charge(anyDouble())).thenReturn(PaymentResponse.ACCEPTED);

        Customer customer = mock(Customer.class);
        CRM crm = mock(CRM.class);
        when(crm.getCustomer(anyInt())).thenReturn(customer);

        Shipping shipping = mock(Shipping.class);

        Postage postage = mock(Postage.class);
        when(postage.calculate(any(), anyDouble())).thenReturn( t: 15.0);

        Basket basket = new Basket(
                customerId: 1,
```


: Basket
: PaymentMethod
: CRM
: Orders
: Warehouse
: Postage
: Shipping
checkout()
charge()
send()
getCustomerById()
adjustStock()
note()
calculate()

Roles, Responsibilities & Collaborations

The Essence of High-Level Design





RECEIVING

DRY

DRY STORAGE

REFRIGERATED STORAGE

BAKING

MEAT PRE-PREPARATION

VEGETABLE PRE-PREPARATION

POT WASHING

COOKING

SALAD PREPARATION

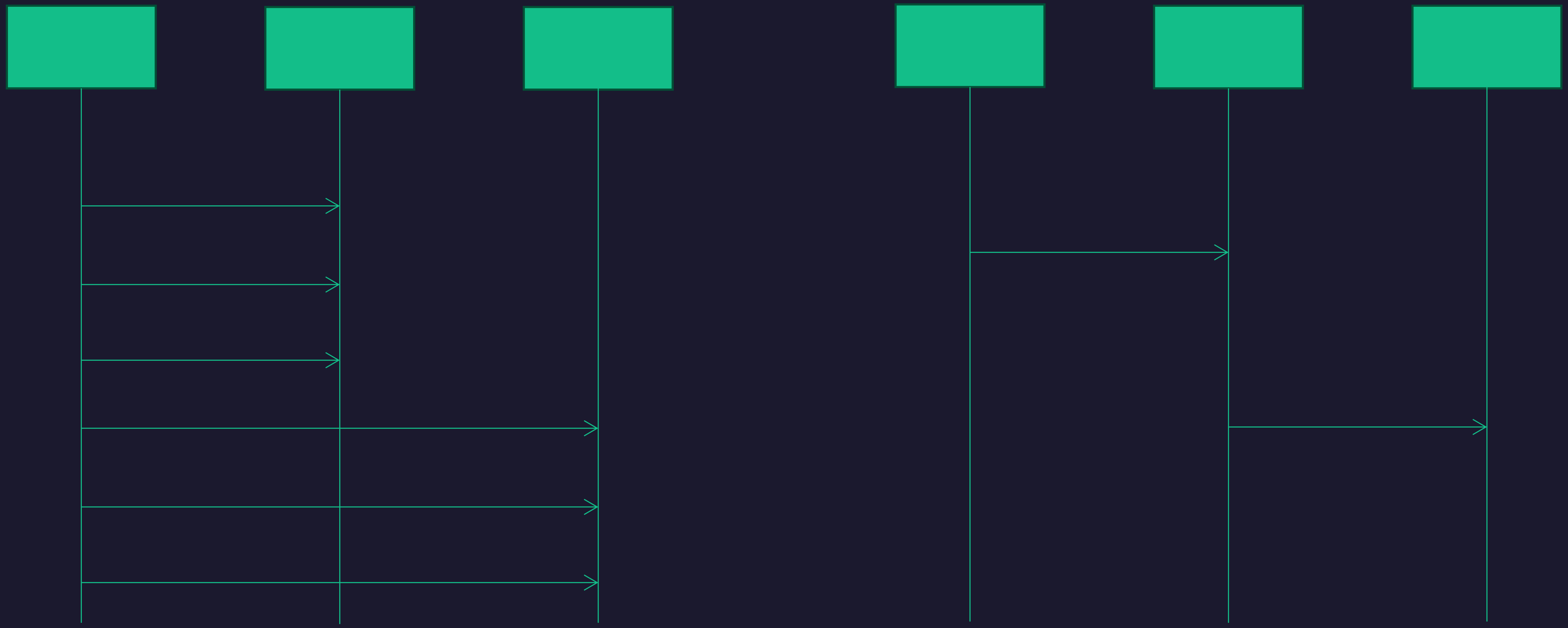
FINISHING

WASTE DISPOSAL

DISH WASHING

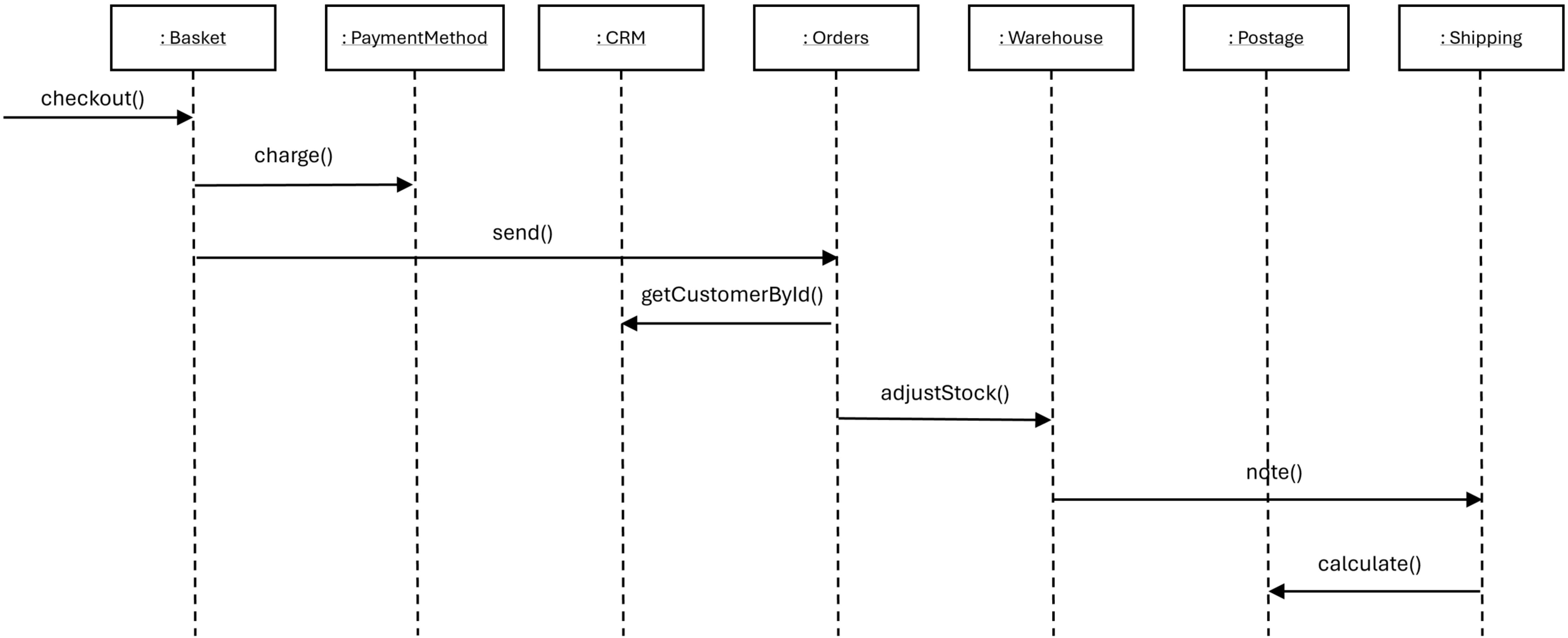
SERVING AREA

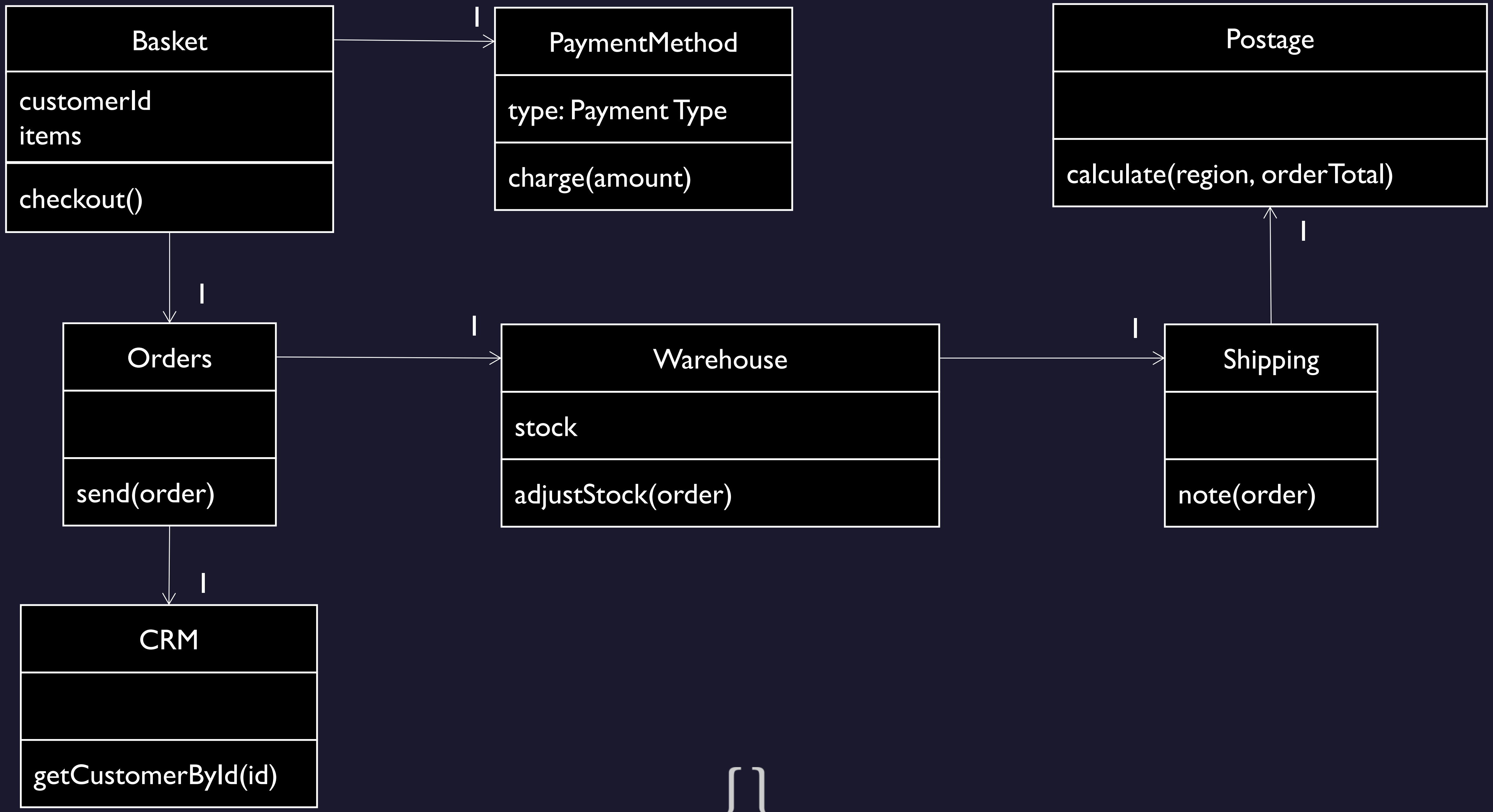
DINING ROOM

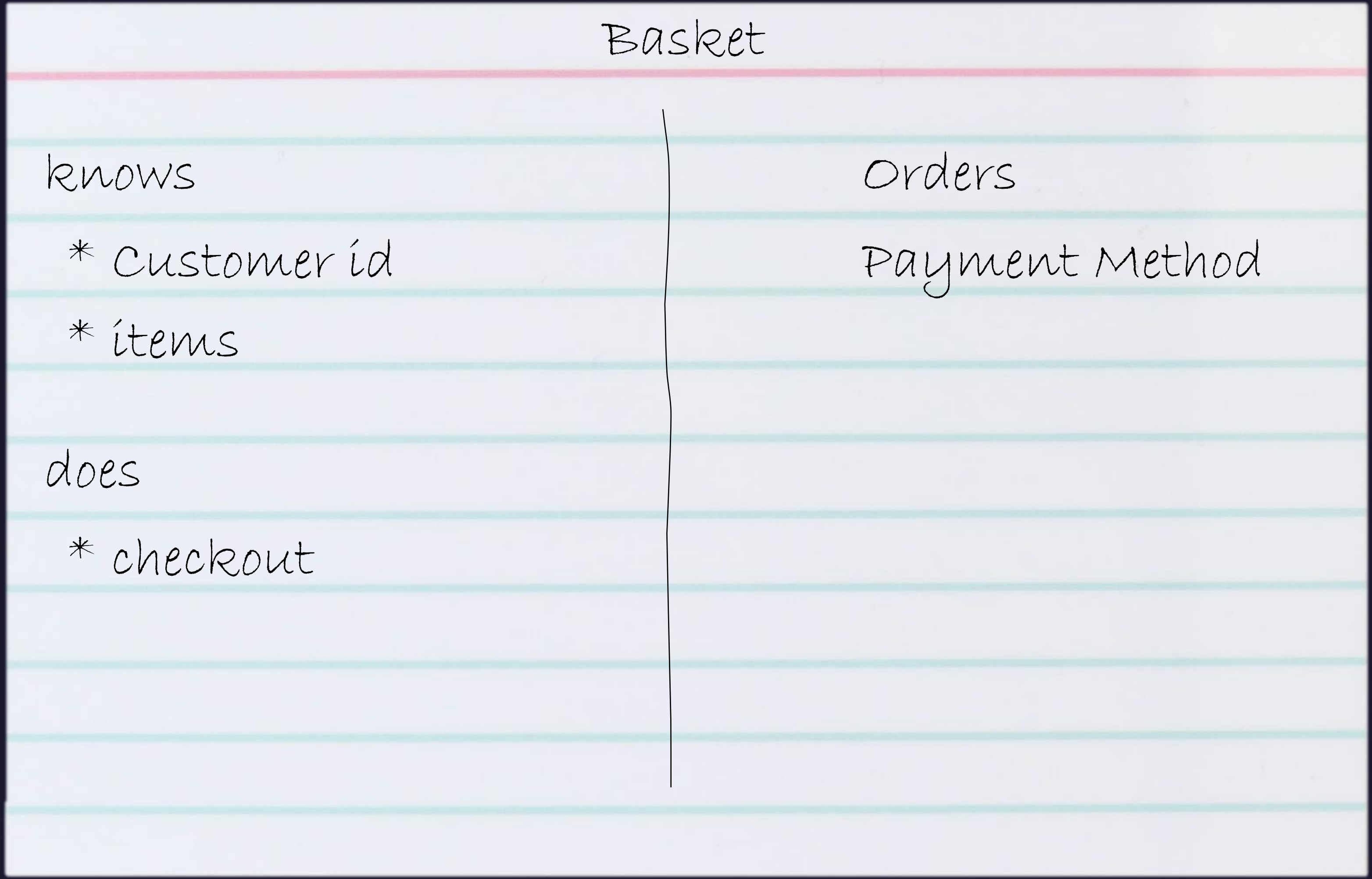


Simple High-Level Design Tools

Visualising Roles, Responsibilities & Collaborations


: Basket
: PaymentMethod
: CRM
: Orders
: Warehouse
: Postage
: Shipping
checkout()
charge()
send()
getCustomerById()
adjustStock()
note()
calculate()


Basket
customerId
items
checkout()
1
PaymentMethod
type: Payment Type
charge(amount)
Postage
calculate(region, orderTotal)
1
1
Orders
send(order)
1
Warehouse
stock
adjustStock(order)
1
Shipping
note(order)
1
CRM
getCustomerById(id)



Basket

knows

* Customer id

* items

does

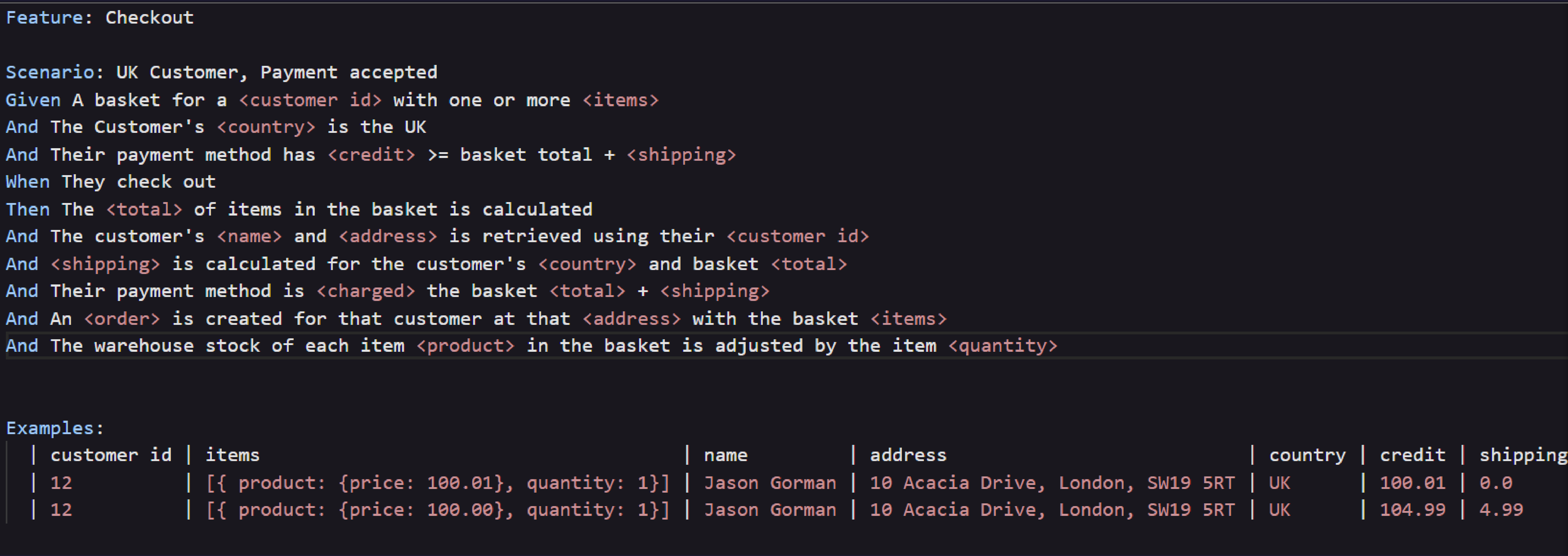
* checkout

Orders

Payment Method

Specification By Example

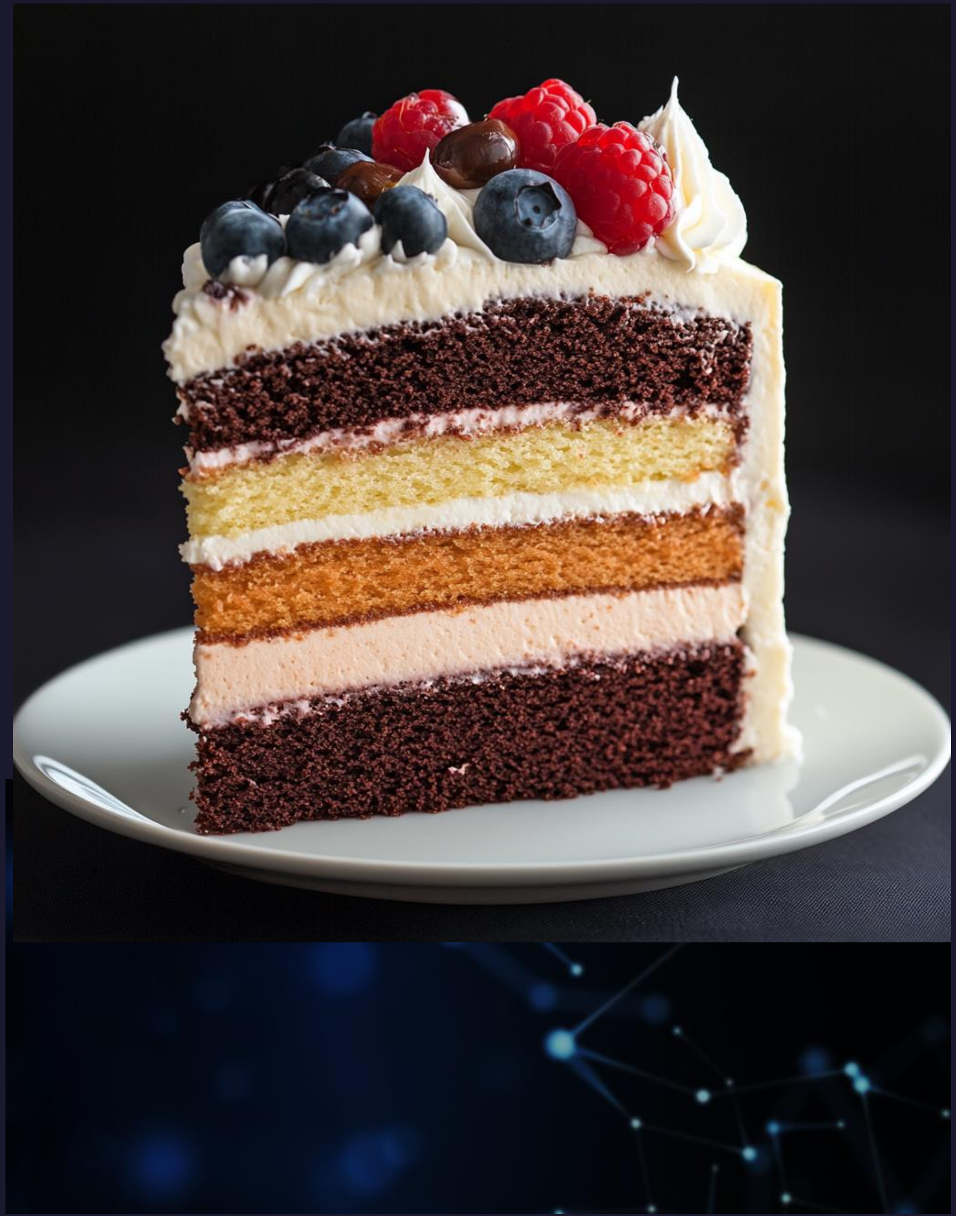
Pinning Down Requirements With Test Cases



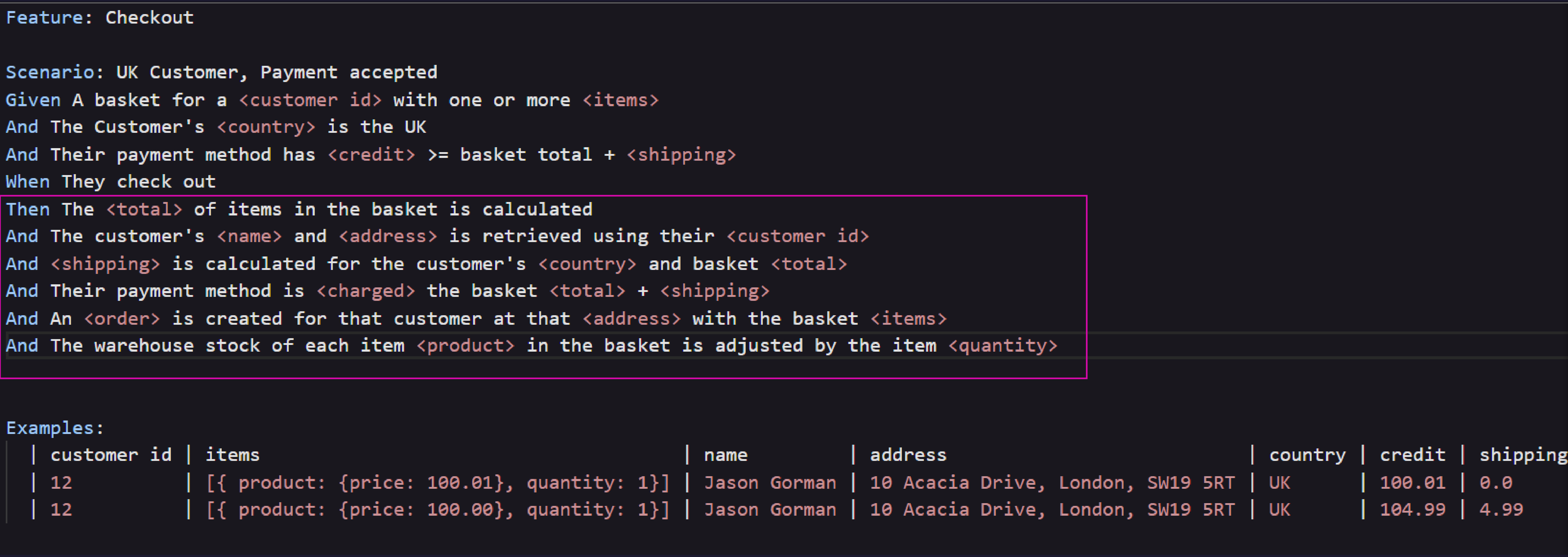
```gherkin
Feature: Checkout

Scenario: UK Customer, Payment accepted
Given A basket for a <customer id> with one or more <items>
And The Customer's <country> is the UK
And Their payment method has <credit> >= basket total + <shipping>
When They check out
Then The <total> of items in the basket is calculated
And The customer's <name> and <address> is retrieved using their <customer id>
And <shipping> is calculated for the customer's <country> and basket <total>
And Their payment method is <charged> the basket <total> + <shipping>
And An <order> is created for that customer at that <address> with the basket <items>
And The warehouse stock of each item <product> in the basket is adjusted by the item <quantity>


Examples:
  | customer id | items                                         | name         | address                              | country  | credit  | shipping
  | 12          | [{ product: {price: 100.01}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK       | 100.01 | 0.0
  | 12          | [{ product: {price: 100.00}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK       | 104.99 | 4.99
```

From Features To High-Level Designs

Mapping Roles, Responsibilities & Collaborations One Slice At A Time



Going Responsibilities First....



```
Feature: Checkout

Scenario: UK Customer, Payment accepted
Given A basket for a <customer id> with one or more <items>
And The Customer's <country> is the UK
And Their payment method has <credit> >= basket total + <shipping>
When They check out
Then The <total> of items in the basket is calculated
And The customer's <name> and <address> is retrieved using their <customer id>
And <shipping> is calculated for the customer's <country> and basket <total>
And Their payment method is <charged> the basket <total> + <shipping>
And An <order> is created for that customer at that <address> with the basket <items>
And The warehouse stock of each item <product> in the basket is adjusted by the item <quantity>


Examples:
  | customer id | items                                       | name         | address                             | country | credit | shipping
  | 12          | [{ product: {price: 100.01}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 100.01 | 0.0
  | 12          | [{ product: {price: 100.00}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 104.99 | 4.99
```

does

* Calculate total of items

does

* Retrieve customer's name & address

does

* Calculate shipping

does

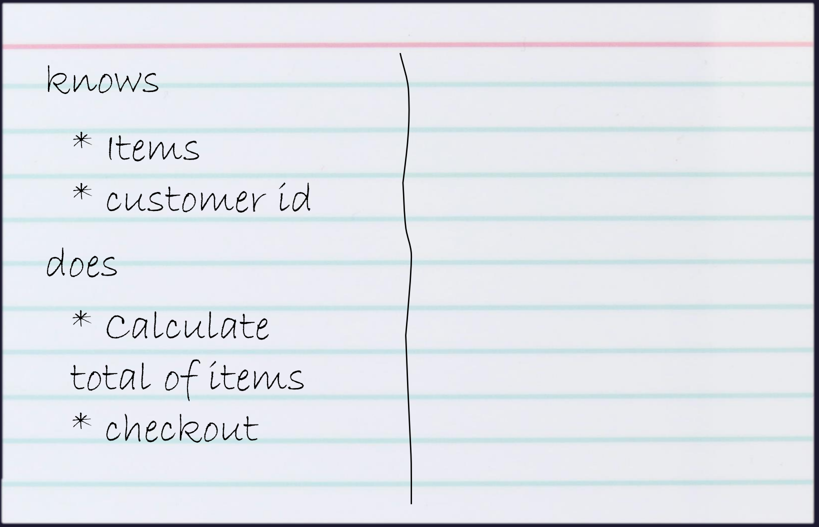
* Charge basket total + shipping

does

* Create order

does

* Adjust product stock

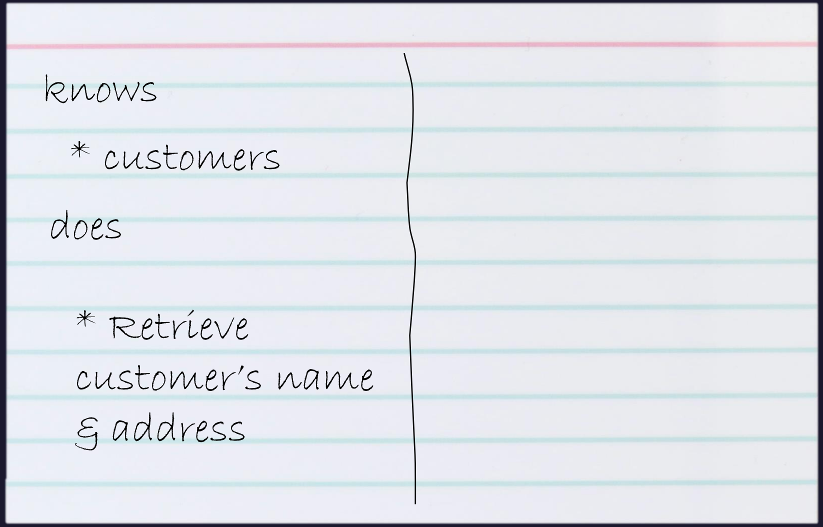


knows

- * Items
- * customer id

does

- * Calculate total of items
- * checkout

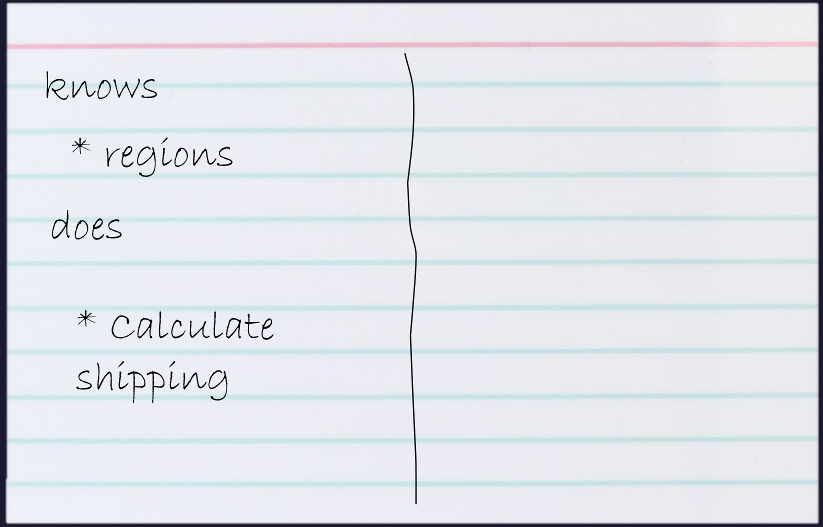


knows

- * customers

does

- * Retrieve customer's name & address

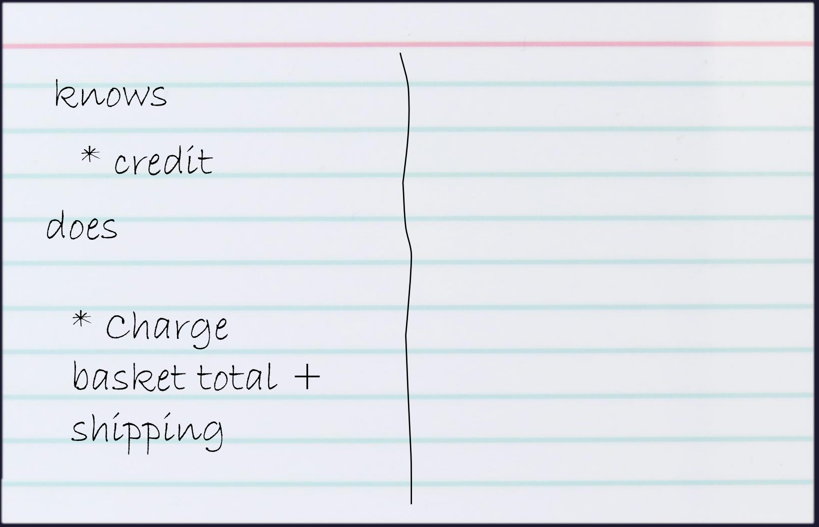


knows

- * regions

does

- * Calculate shipping

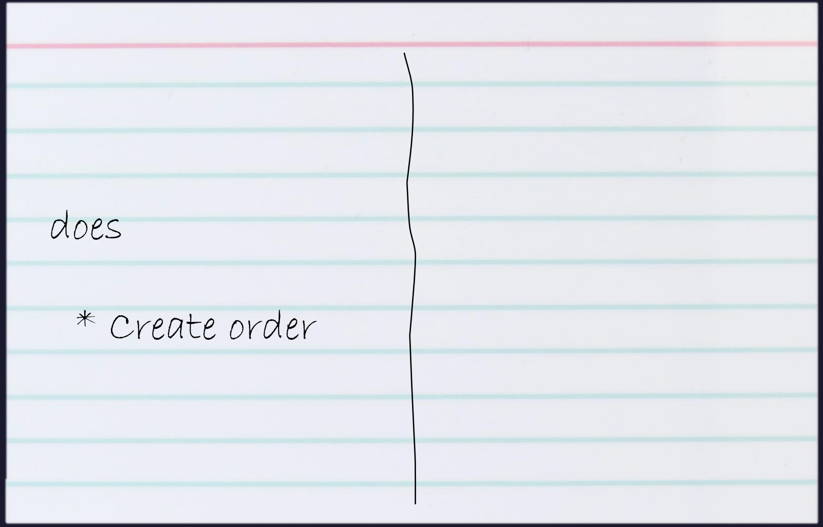


knows

- * credit

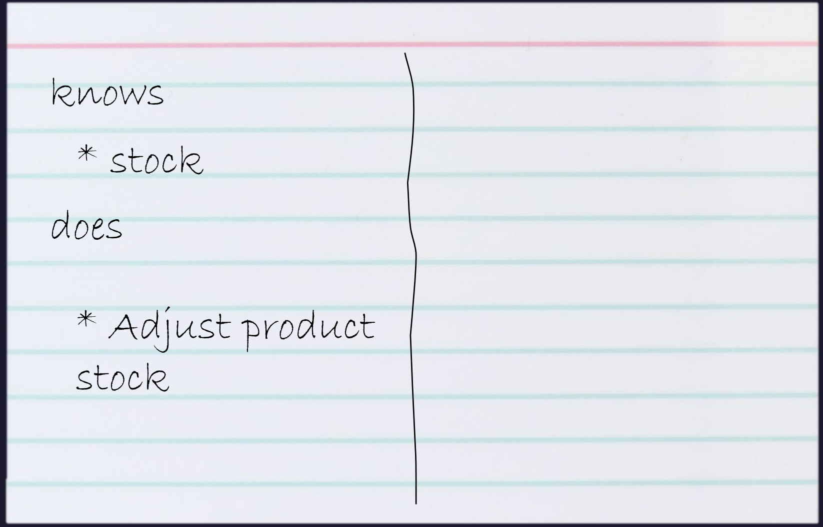
does

- * Charge basket total + shipping



does

- * Create order

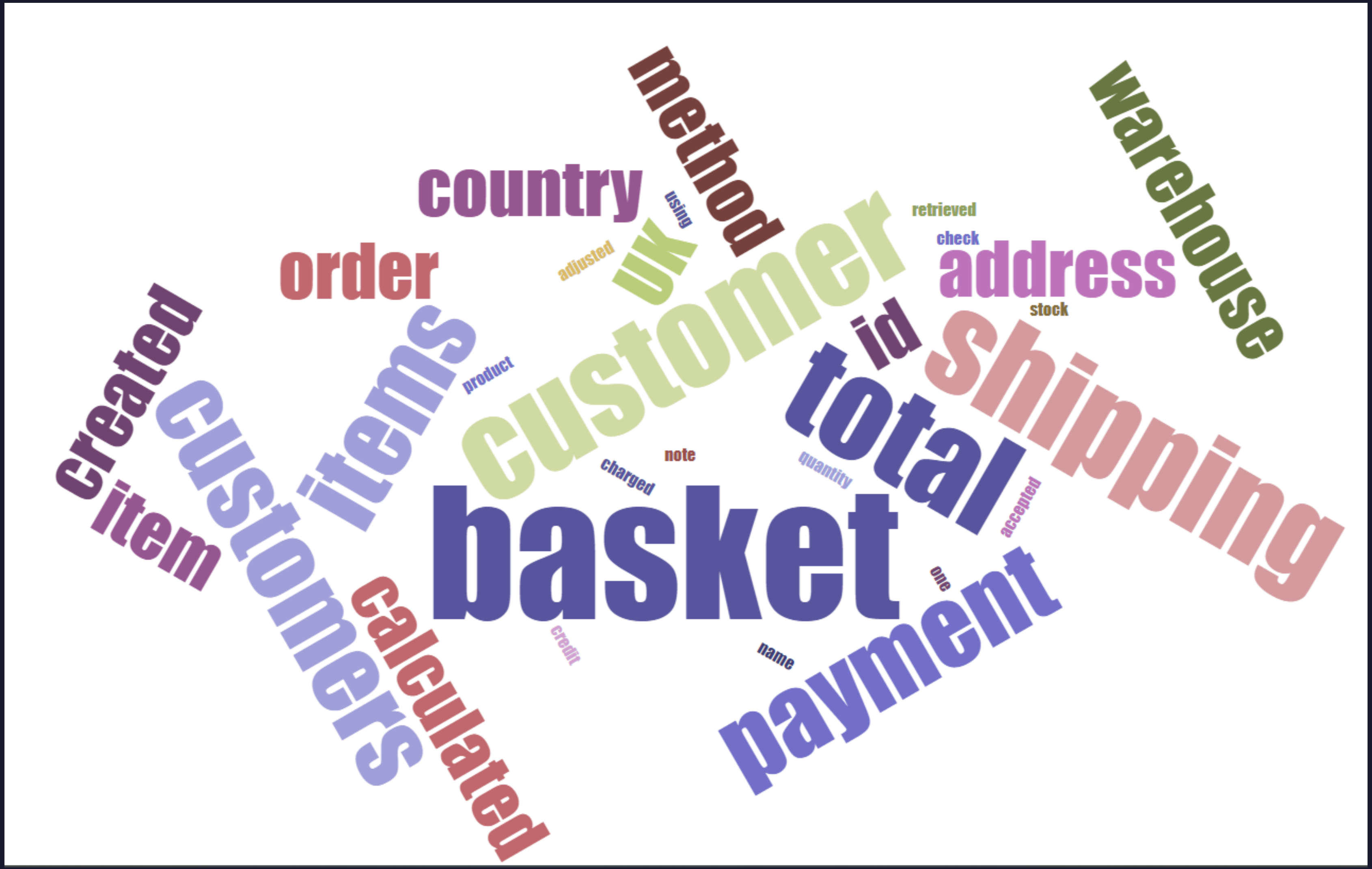


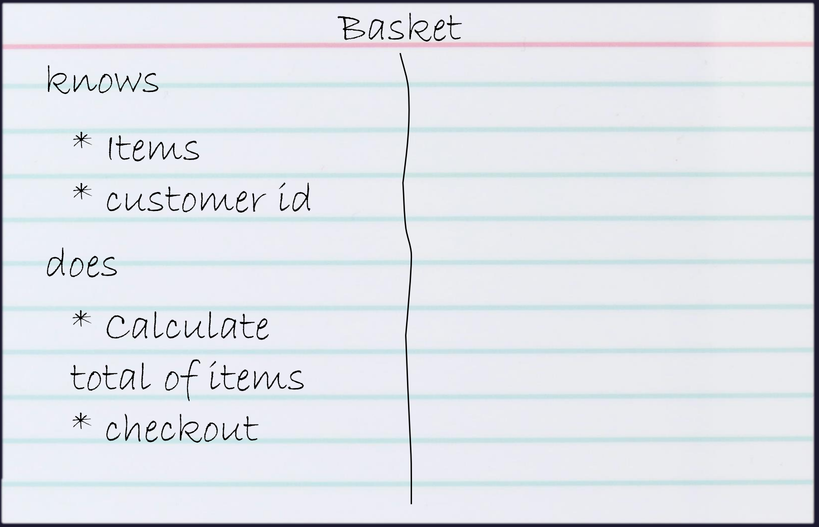
knows

- * stock

does

- * Adjust product stock


method
warehouse
country
using
retrieved
check
address
UK
adjusted
order
customer
id
stock
shipping
created
items
product
total
customers
note
charged
quantity
accepted
item
basket
one
calculated
credit
name
payment



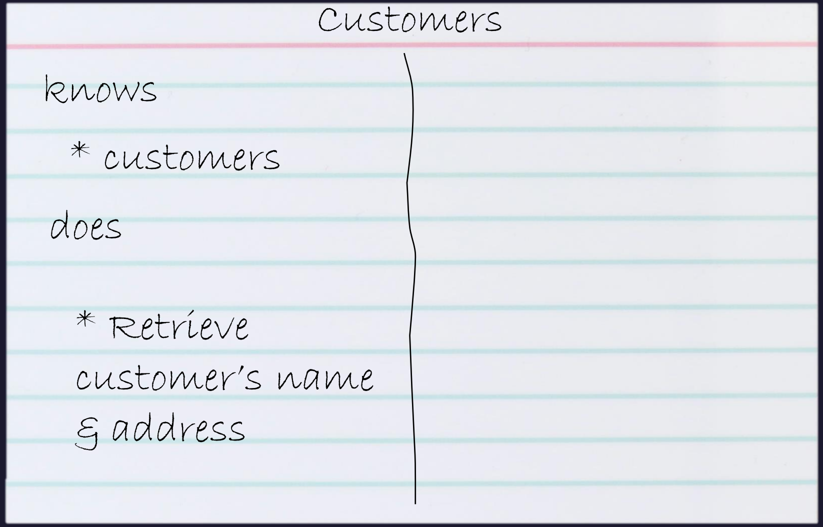
Basket

knows

* Items
* customer id

does

* Calculate total of items
* checkout



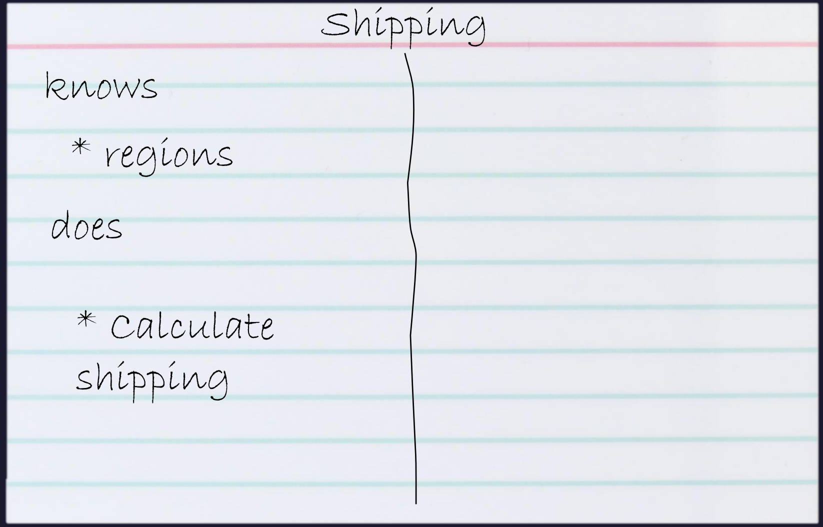
Customers

knows

* customers

does

* Retrieve customer's name & address



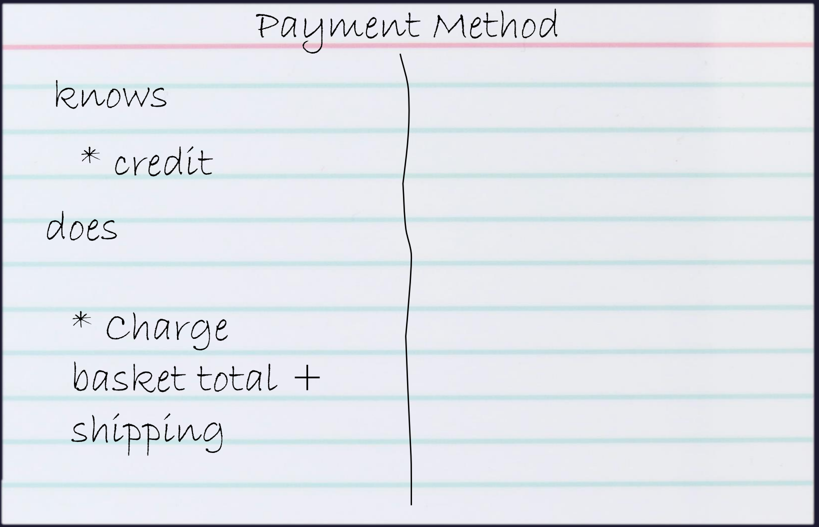
Shipping

knows

* regions

does

* Calculate shipping



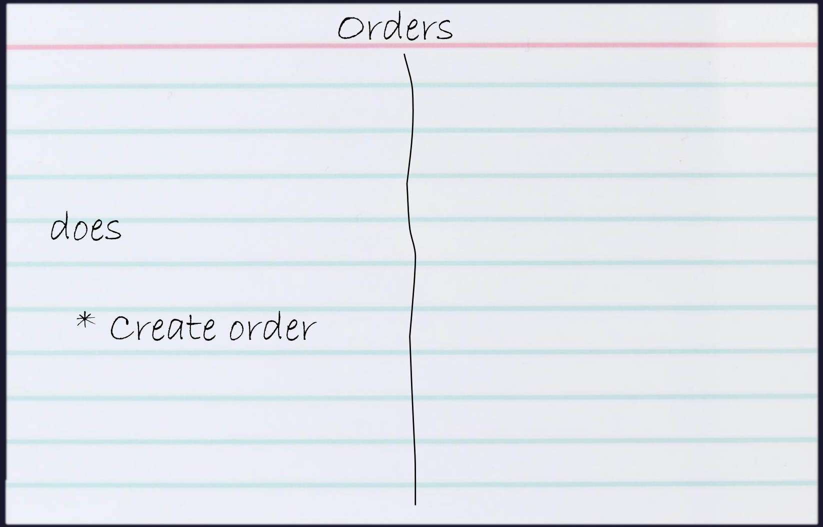
Payment Method

knows

* credit

does

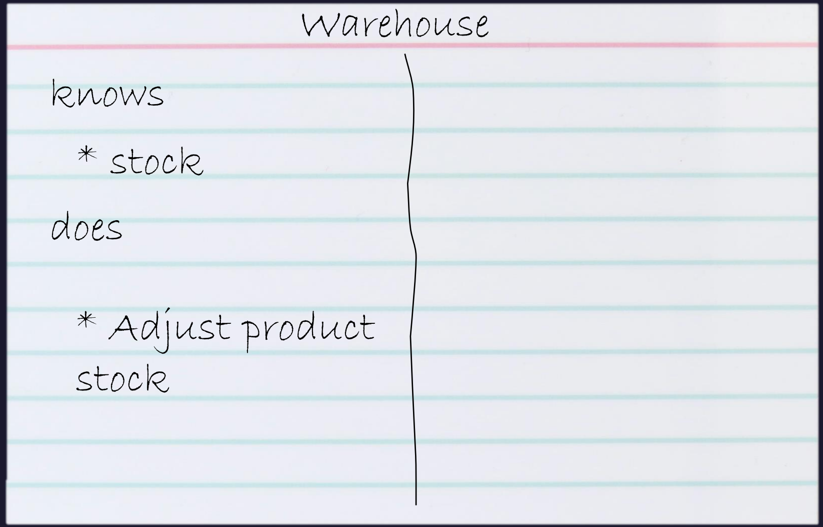
* Charge basket total + shipping



Orders

does

* Create order



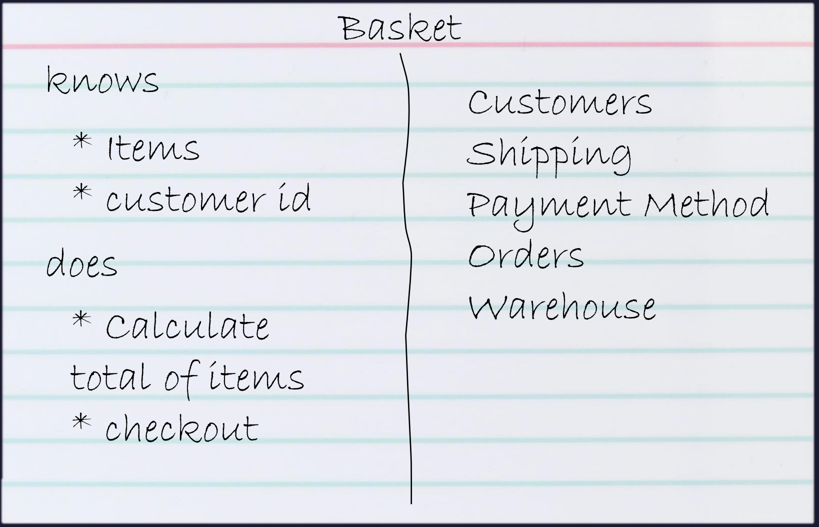
Warehouse

knows

* stock

does

* Adjust product stock



Basket

knows

* Items

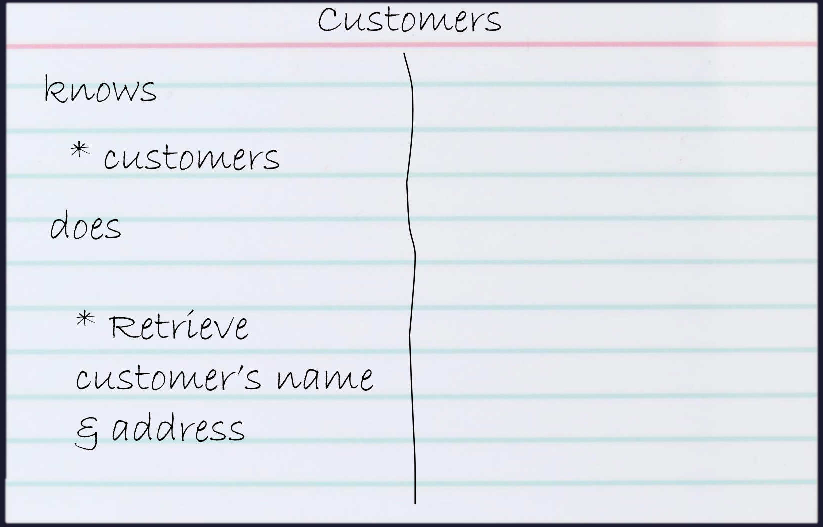
* customer id

does

* Calculate total of items

* checkout

Customers
Shipping
Payment Method
Orders
Warehouse



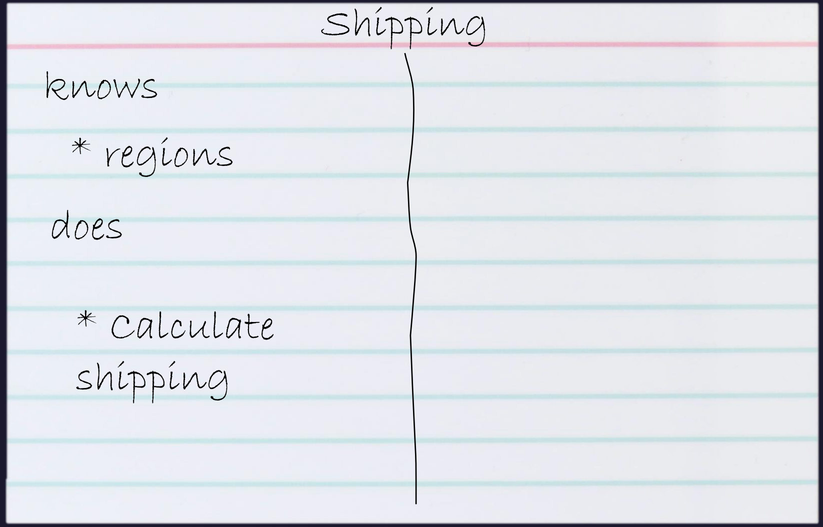
Customers

knows

* customers

does

* Retrieve customer's name & address



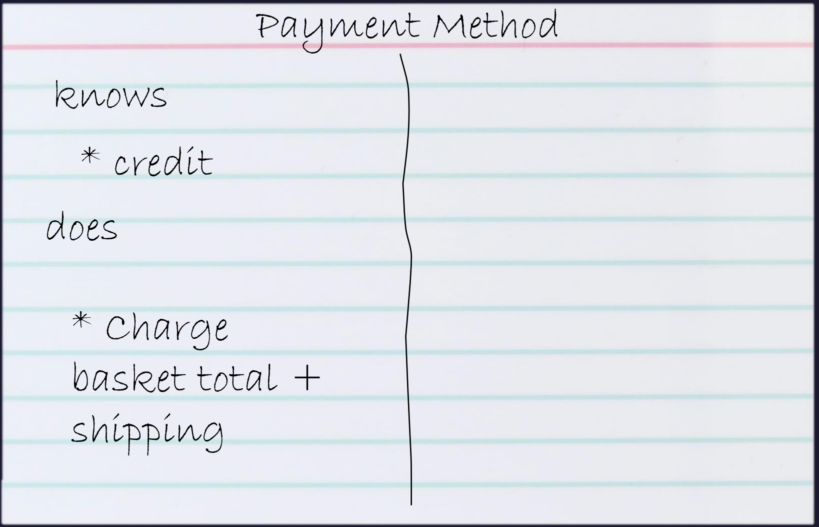
Shipping

knows

* regions

does

* Calculate shipping



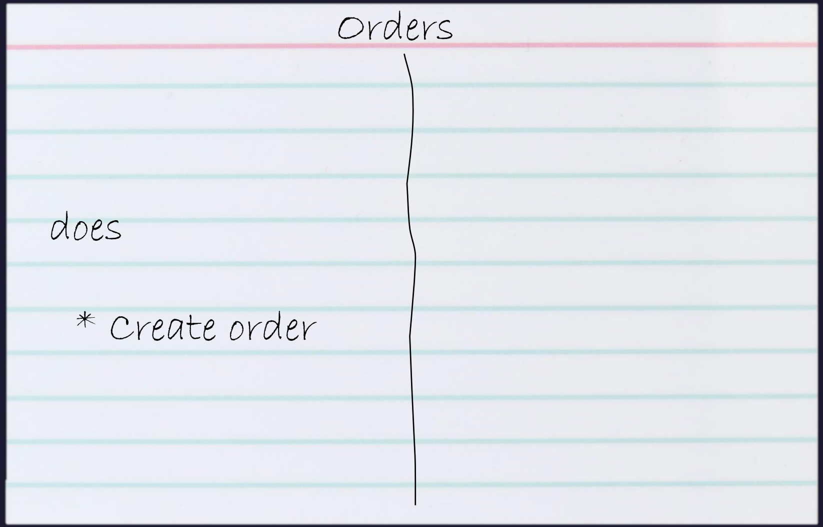
Payment Method

knows

* credit

does

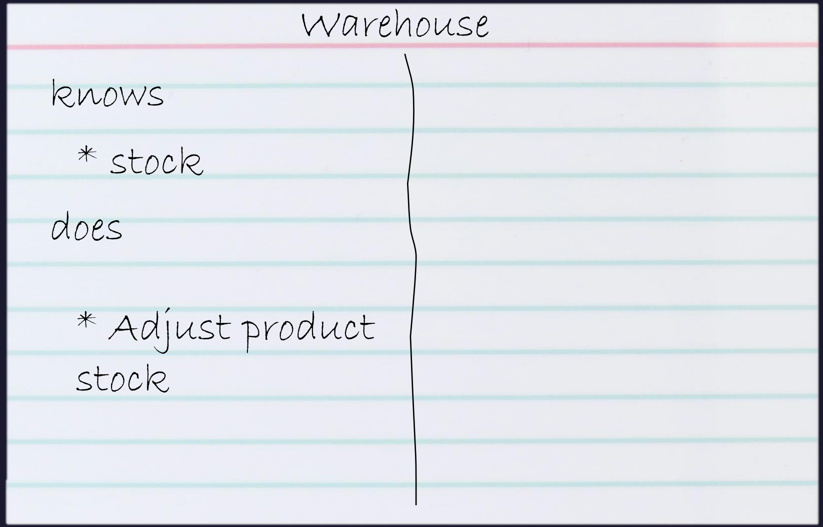
* Charge basket total + shipping



Orders

does

* Create order



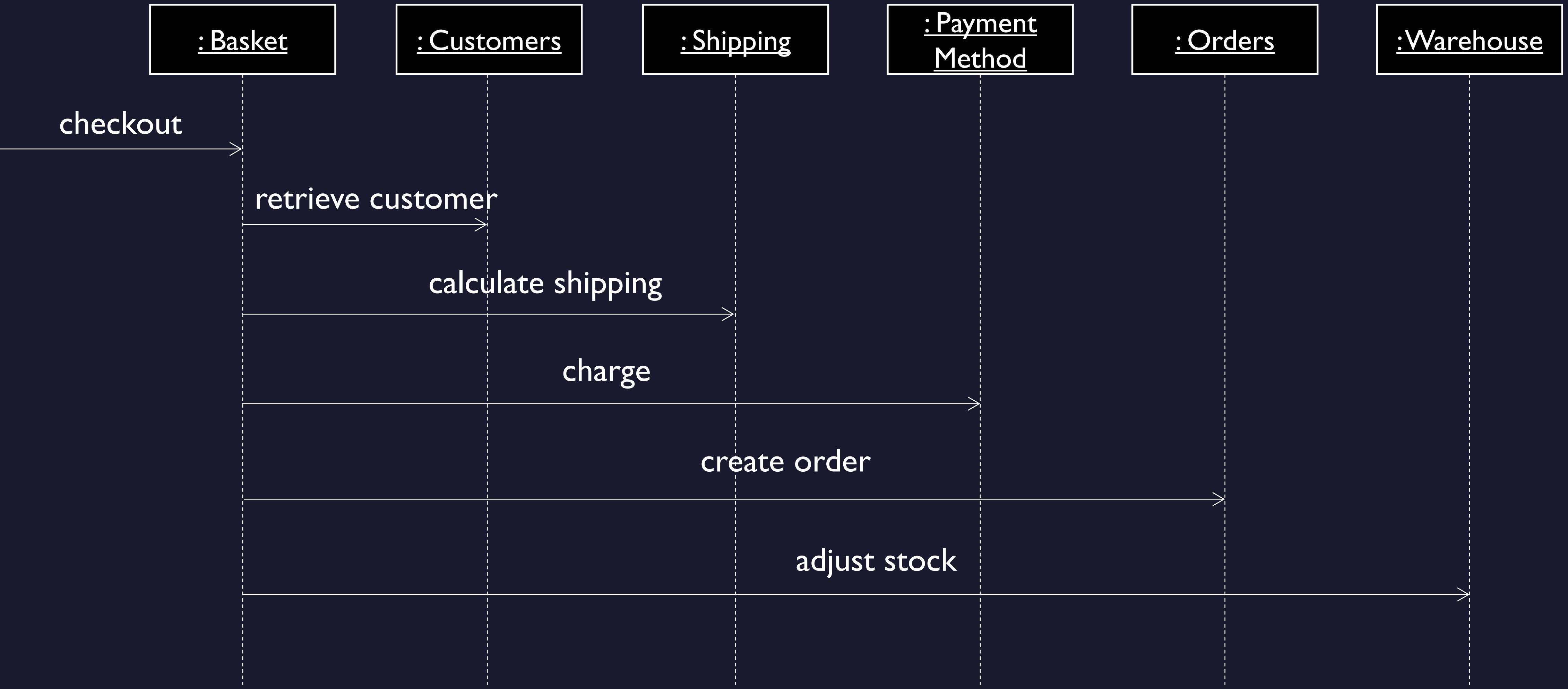
Warehouse

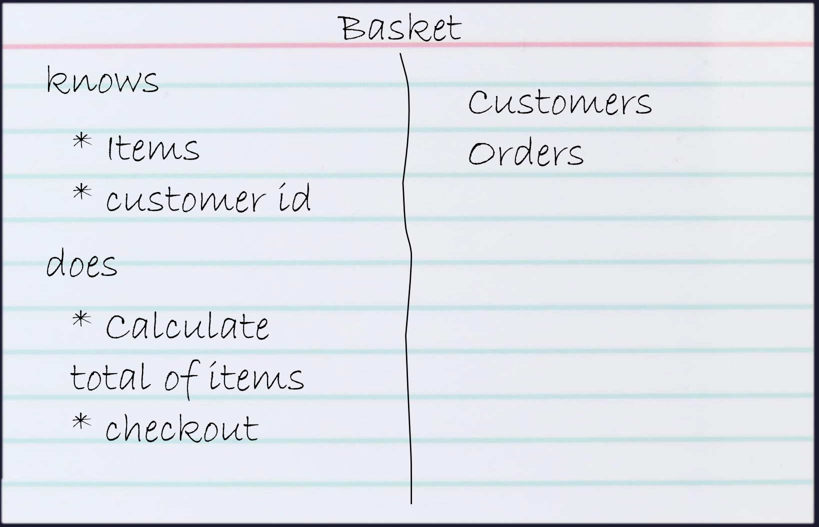
knows

* stock

does

* Adjust product stock


: Basket
: Customers
: Shipping
: Payment Method
: Orders
: Warehouse
checkout
retrieve customer
calculate shipping
charge
create order
adjust stock



Basket

knows

* Items

* customer id

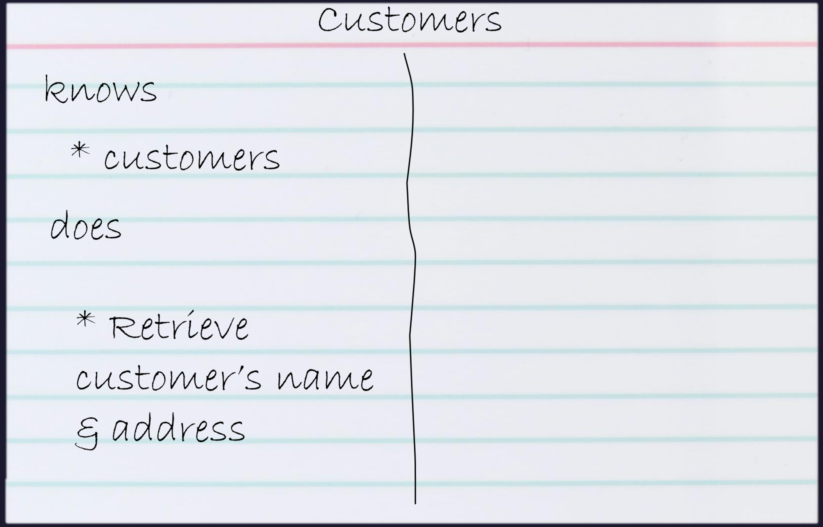
does

* Calculate total of items

* checkout

Customers

Orders



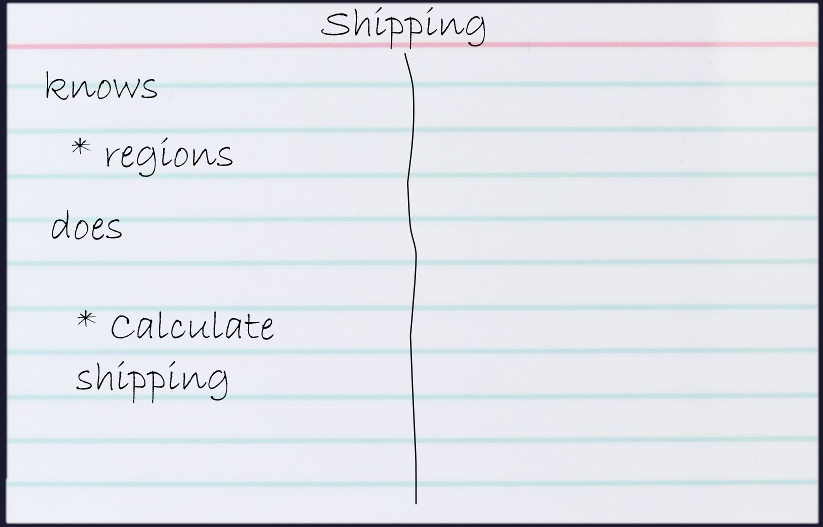
Customers

knows

* customers

does

* Retrieve customer's name & address



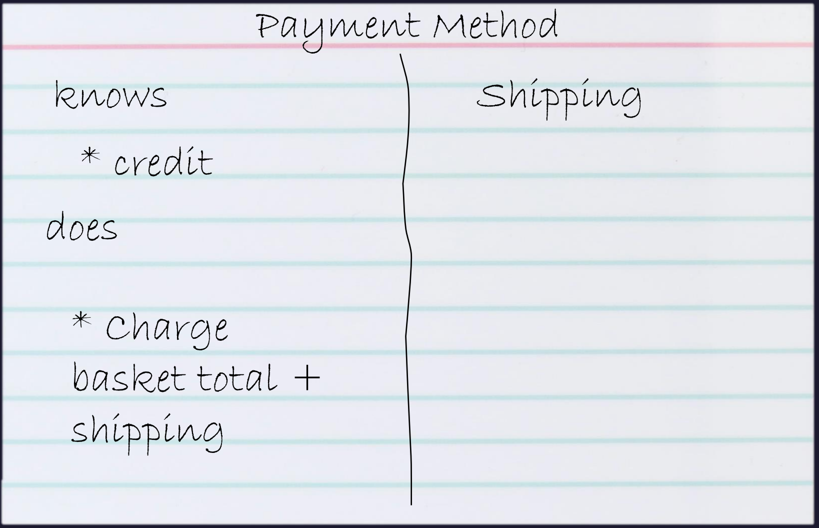
Shipping

knows

* regions

does

* Calculate shipping



Payment Method

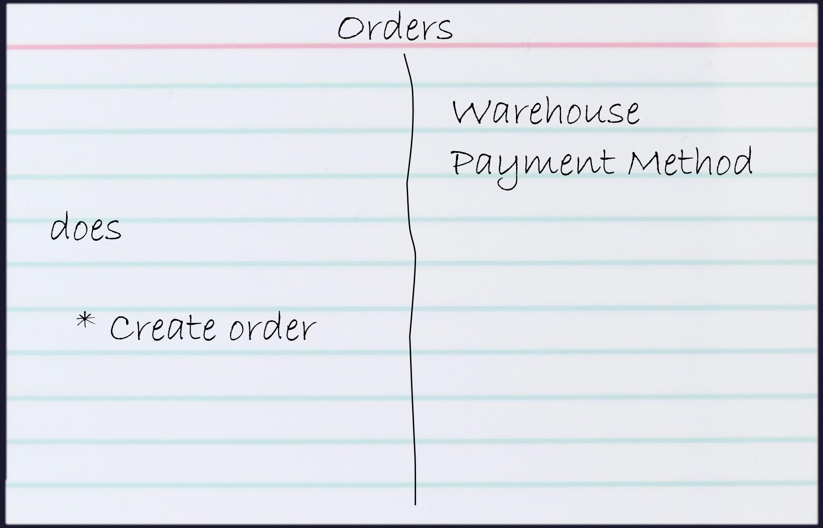
knows

* credit

does

* Charge basket total + shipping

Shipping



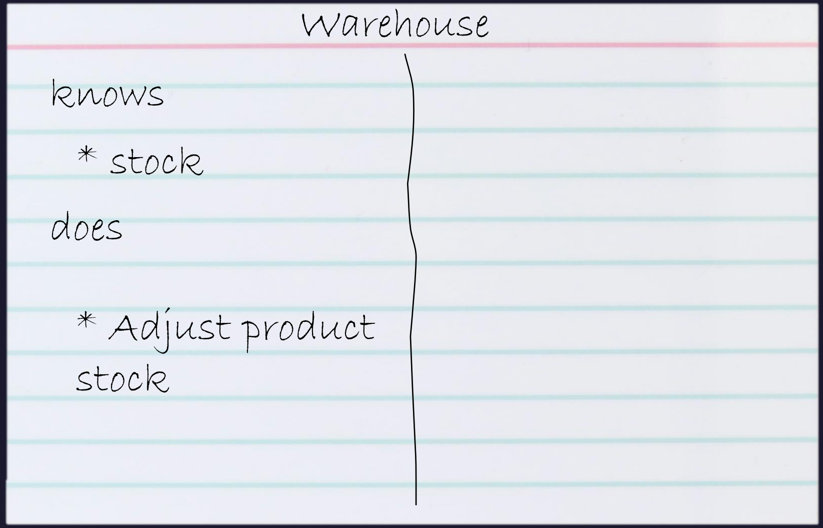
Orders

does

* Create order

Warehouse

Payment Method



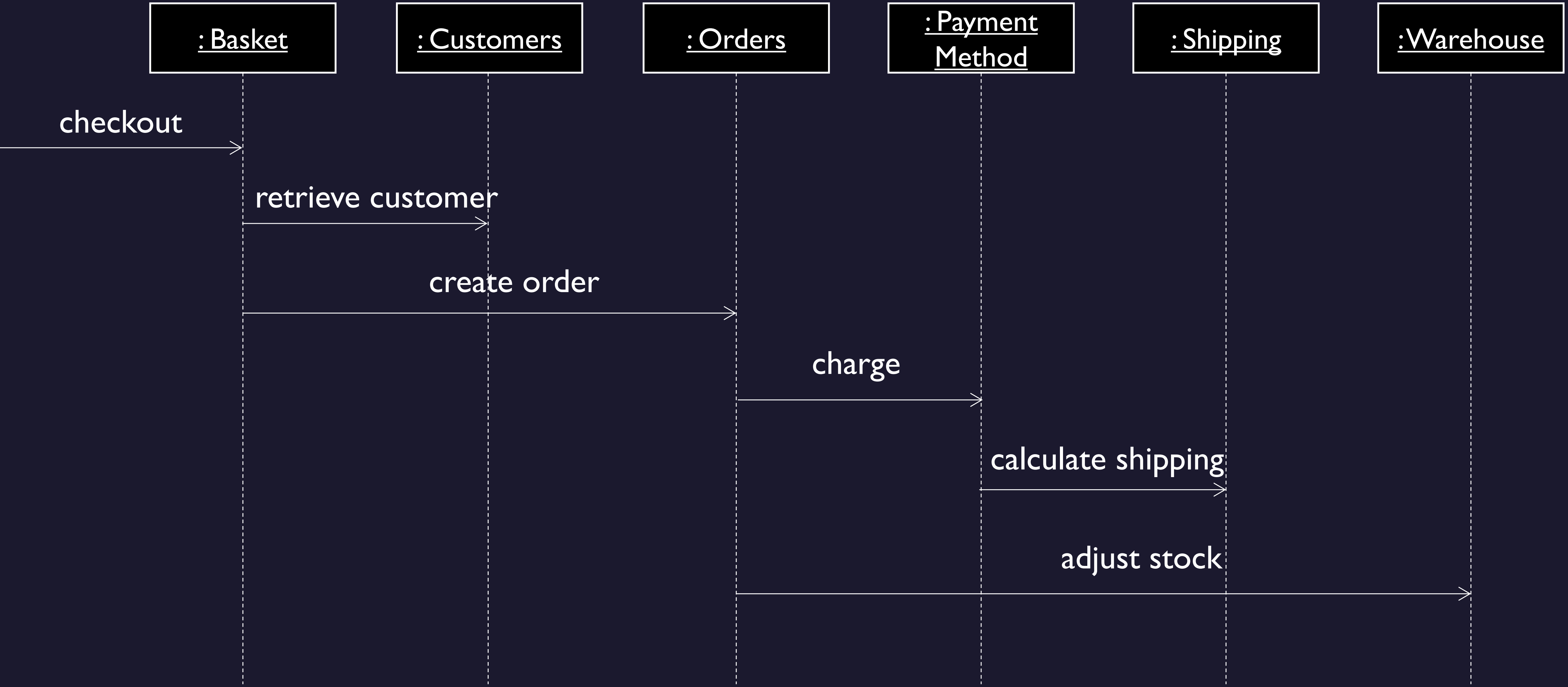
Warehouse

knows

* stock

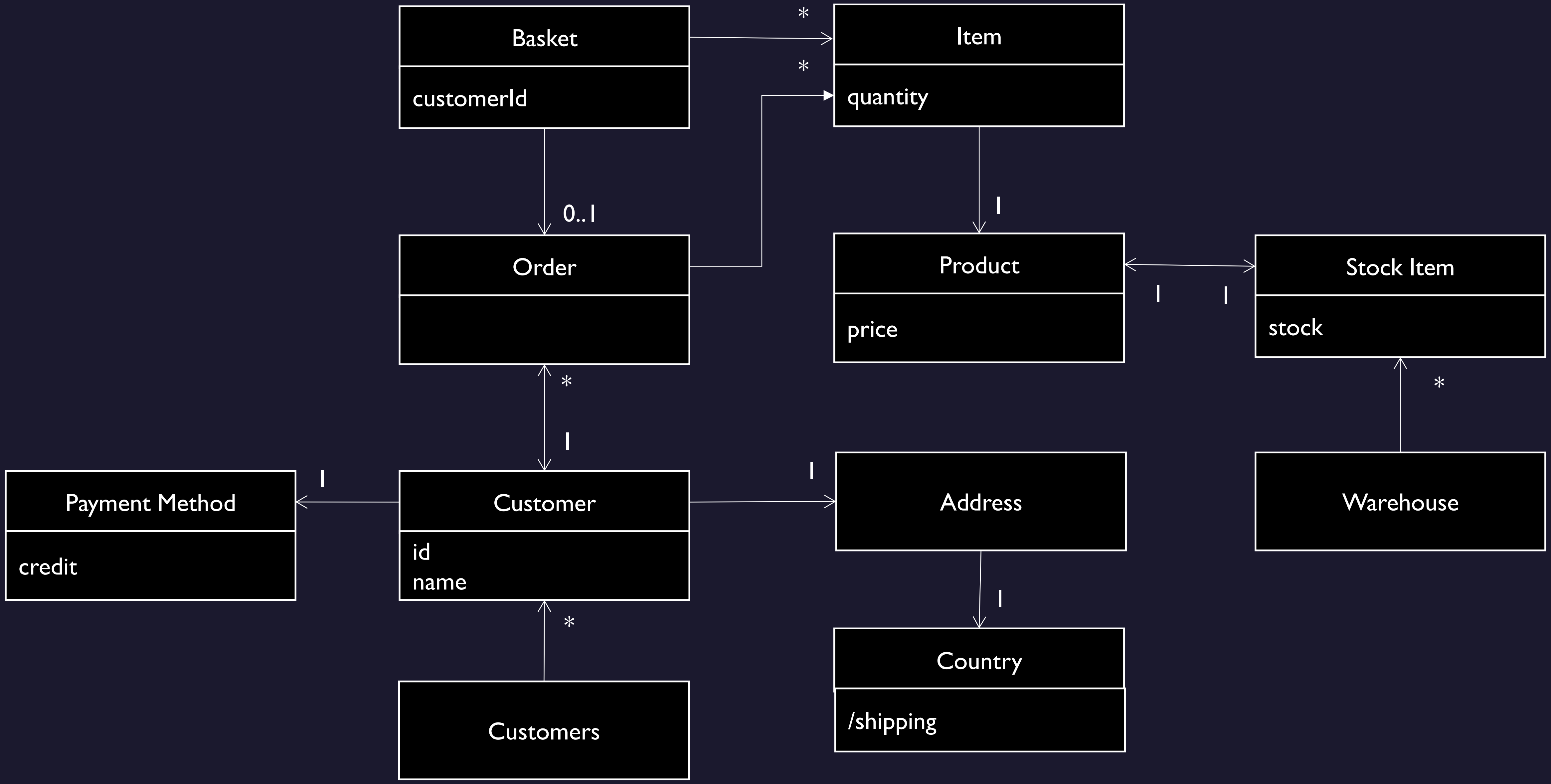
does

* Adjust product stock

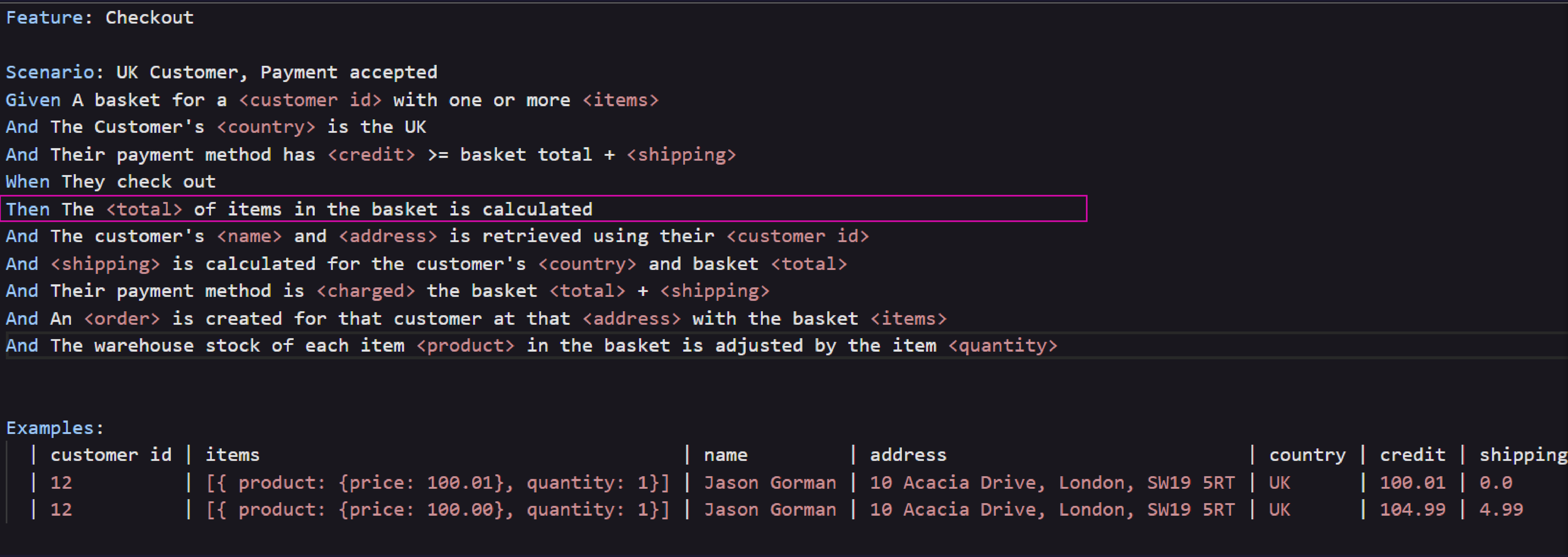


codemanship

Going Data First….

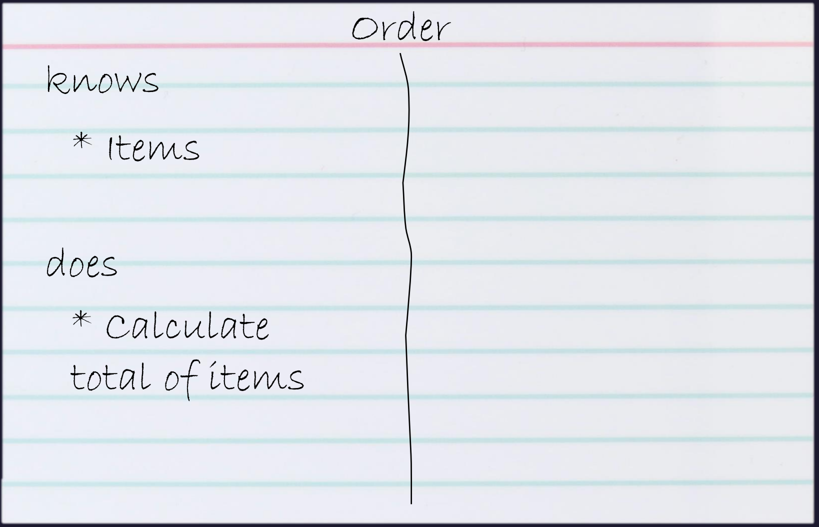


codemanship



```
Feature: Checkout

Scenario: UK Customer, Payment accepted
Given A basket for a <customer id> with one or more <items>
And The Customer's <country> is the UK
And Their payment method has <credit> >= basket total + <shipping>
When They check out
Then The <total> of items in the basket is calculated
And The customer's <name> and <address> is retrieved using their <customer id>
And <shipping> is calculated for the customer's <country> and basket <total>
And Their payment method is <charged> the basket <total> + <shipping>
And An <order> is created for that customer at that <address> with the basket <items>
And The warehouse stock of each item <product> in the basket is adjusted by the item <quantity>


Examples:
  | customer id | items                                       | name         | address                             | country | credit | shipping
  | 12          | [{ product: {price: 100.01}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 100.01 | 0.0
  | 12          | [{ product: {price: 100.00}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 104.99 | 4.99
```

codemanship



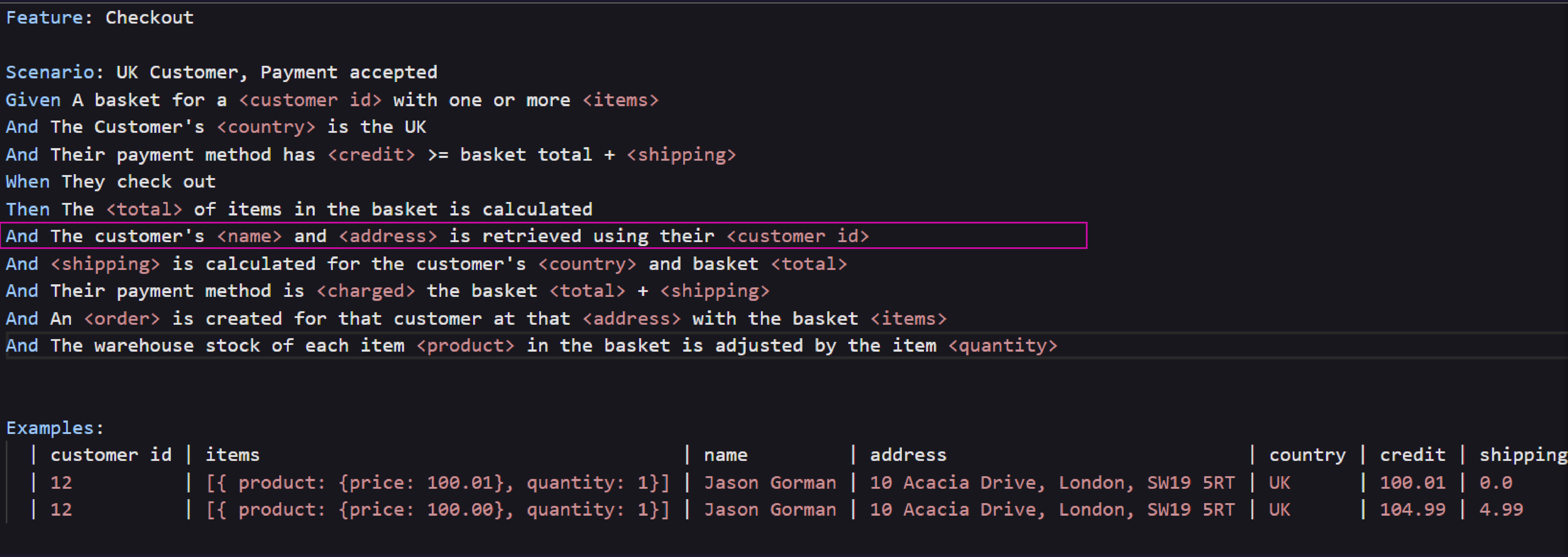
Order

knows

* Items

does

* Calculate total of items



```
Feature: Checkout

Scenario: UK Customer, Payment accepted
Given A basket for a <customer id> with one or more <items>
And The Customer's <country> is the UK
And Their payment method has <credit> >= basket total + <shipping>
When They check out
Then The <total> of items in the basket is calculated
And The customer's <name> and <address> is retrieved using their <customer id>
And <shipping> is calculated for the customer's <country> and basket <total>
And Their payment method is <charged> the basket <total> + <shipping>
And An <order> is created for that customer at that <address> with the basket <items>
And The warehouse stock of each item <product> in the basket is adjusted by the item <quantity>


Examples:
  | customer id | items                                       | name         | address                           | country | credit | shipping
  | 12          | [{ product: {price: 100.01}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 100.01 | 0.0
  | 12          | [{ product: {price: 100.00}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 104.99 | 4.99
```

Order

knows

* Items

does

* Calculate
total of items

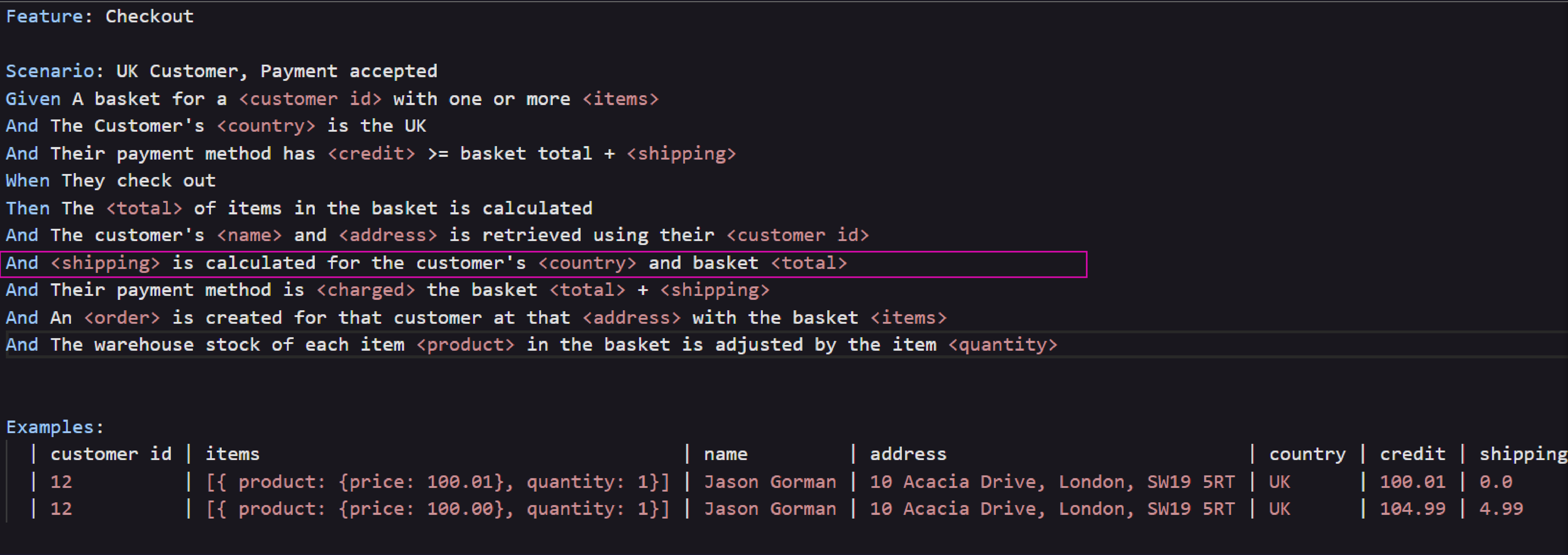
Customers

knows

* customers

does

* Retrieve
customer name
& address



```
Feature: Checkout

Scenario: UK Customer, Payment accepted
Given A basket for a <customer id> with one or more <items>
And The Customer's <country> is the UK
And Their payment method has <credit> >= basket total + <shipping>
When They check out
Then The <total> of items in the basket is calculated
And The customer's <name> and <address> is retrieved using their <customer id>
And <shipping> is calculated for the customer's <country> and basket <total>
And Their payment method is <charged> the basket <total> + <shipping>
And An <order> is created for that customer at that <address> with the basket <items>
And The warehouse stock of each item <product> in the basket is adjusted by the item <quantity>


Examples:
  | customer id | items                                        | name         | address                            | country | credit | shipping
  | 12          | [{ product: {price: 100.01}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 100.01 | 0.0
  | 12          | [{ product: {price: 100.00}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 104.99 | 4.99
```

Order

knows

* Items

does

* Calculate total of items

Customers

knows

* customers

does

* Retrieve customer name & address

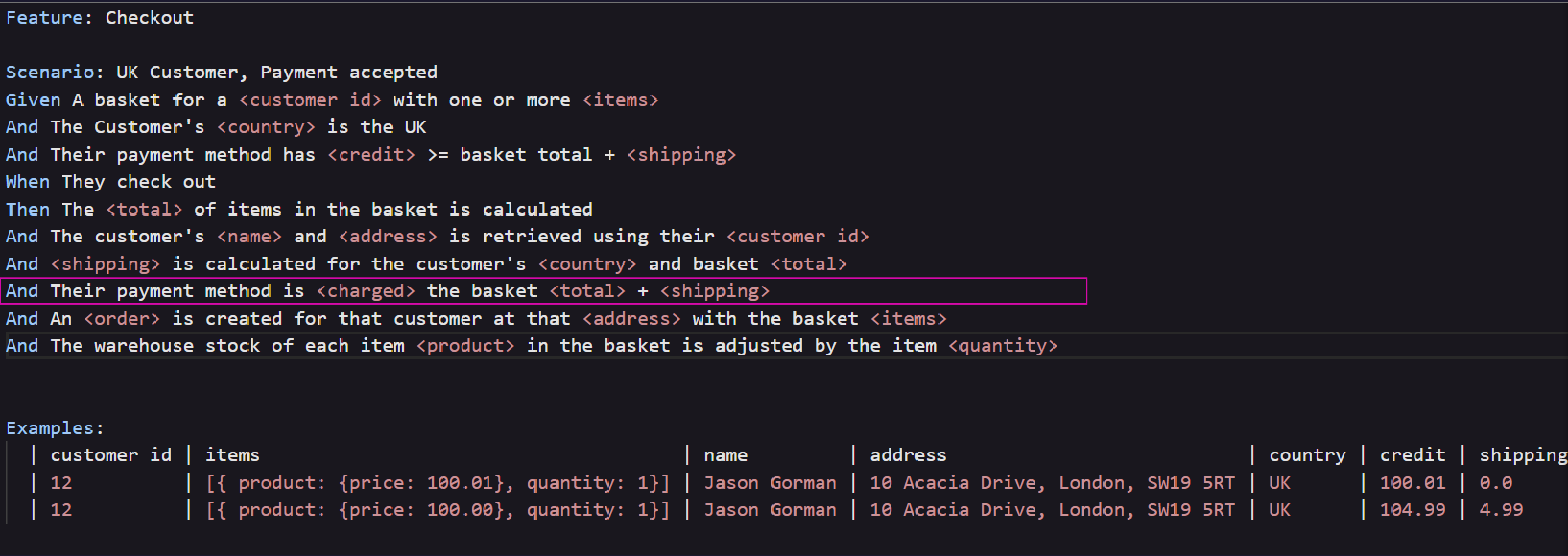
Country

knows

* ?

does

* Calculate shipping



```
Feature: Checkout

Scenario: UK Customer, Payment accepted
Given A basket for a <customer id> with one or more <items>
And The Customer's <country> is the UK
And Their payment method has <credit> >= basket total + <shipping>
When They check out
Then The <total> of items in the basket is calculated
And The customer's <name> and <address> is retrieved using their <customer id>
And <shipping> is calculated for the customer's <country> and basket <total>
And Their payment method is <charged> the basket <total> + <shipping>
And An <order> is created for that customer at that <address> with the basket <items>
And The warehouse stock of each item <product> in the basket is adjusted by the item <quantity>


Examples:
  | customer id | items                                   | name         | address                         | country | credit | shipping
  | 12          | [{ product: {price: 100.01}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 100.01 | 0.0
  | 12          | [{ product: {price: 100.00}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 104.99 | 4.99
```

Order

knows

* Items

does

* Calculate total of items

Customers

knows

* customers

does

* Retrieve customer name & address

Country

knows

* ?

does

* Calculate shipping

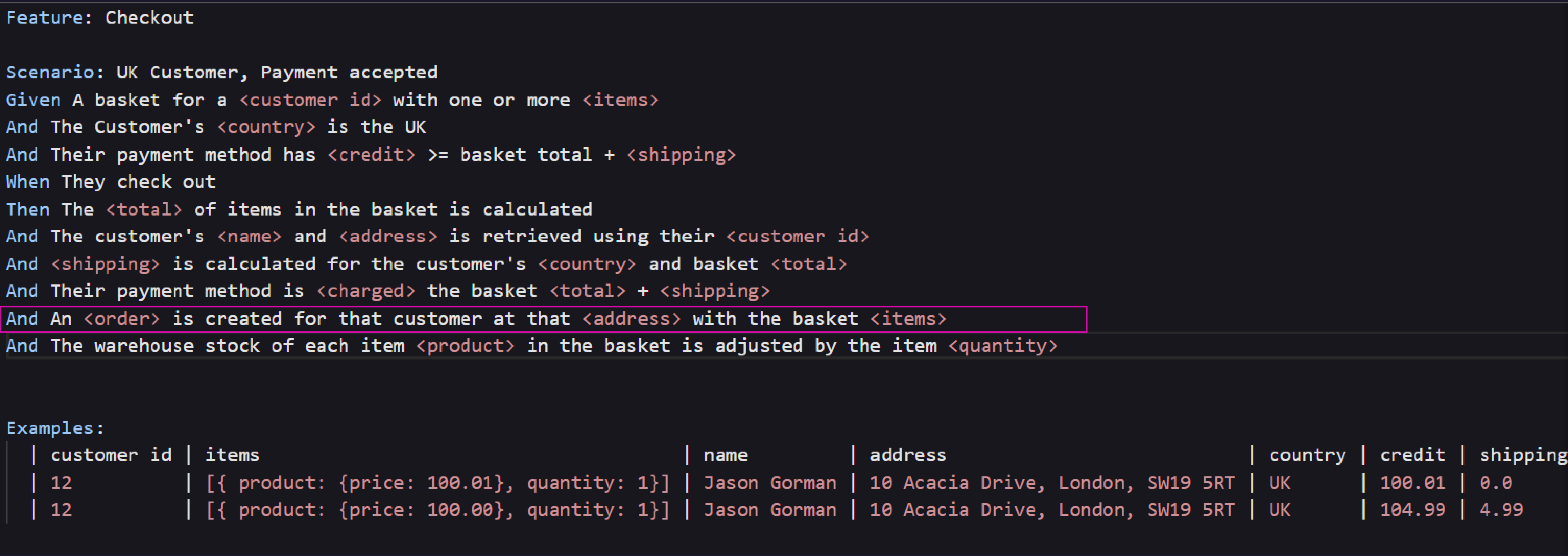
Payment Method

knows

* credit

does

* Charge total + shipping



```
Feature: Checkout

Scenario: UK Customer, Payment accepted
Given A basket for a <customer id> with one or more <items>
And The Customer's <country> is the UK
And Their payment method has <credit> >= basket total + <shipping>
When They check out
Then The <total> of items in the basket is calculated
And The customer's <name> and <address> is retrieved using their <customer id>
And <shipping> is calculated for the customer's <country> and basket <total>
And Their payment method is <charged> the basket <total> + <shipping>
And An <order> is created for that customer at that <address> with the basket <items>
And The warehouse stock of each item <product> in the basket is adjusted by the item <quantity>


Examples:
  | customer id | items                                  | name         | address                            | country | credit | shipping
  | 12          | [{ product: {price: 100.01}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 100.01 | 0.0
  | 12          | [{ product: {price: 100.00}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 104.99 | 4.99
```

Order

knows

* Items

does

* Calculate total of items

Customers

knows

* customers

does

* Retrieve customer name & address

Country

knows

* ?

does

* Calculate shipping

Payment Method

knows

* credit

does

* Charge total + shipping

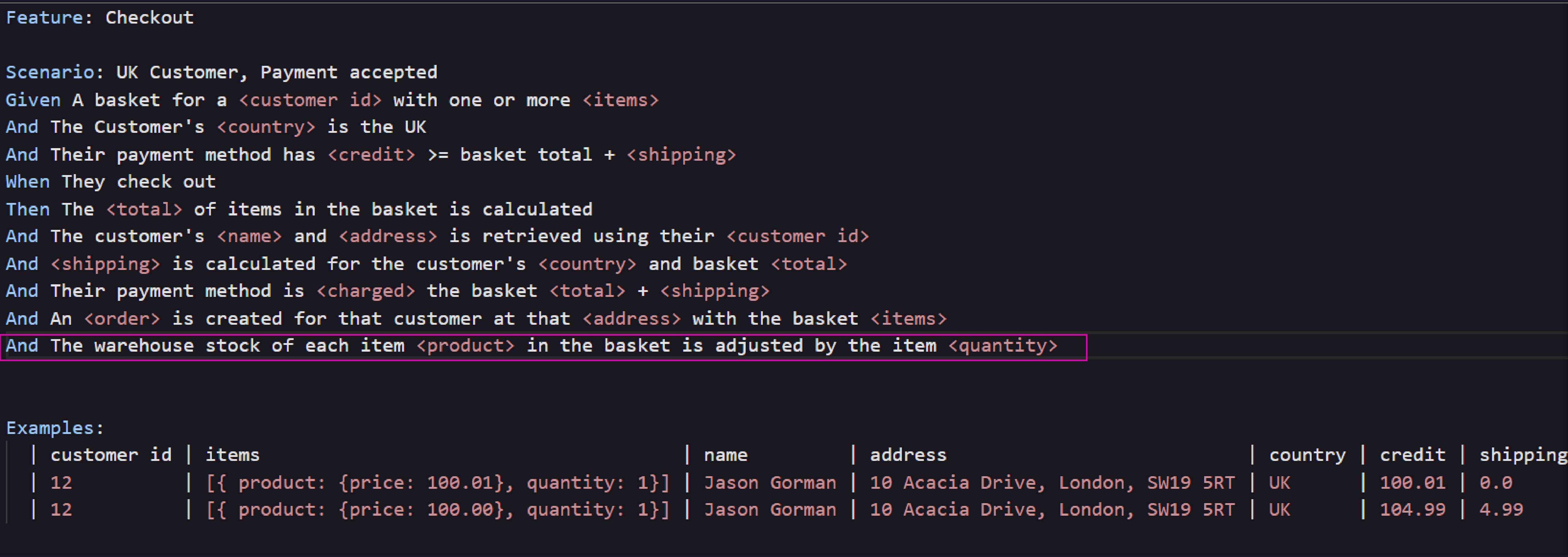
Basket

knows

- customer id
- items

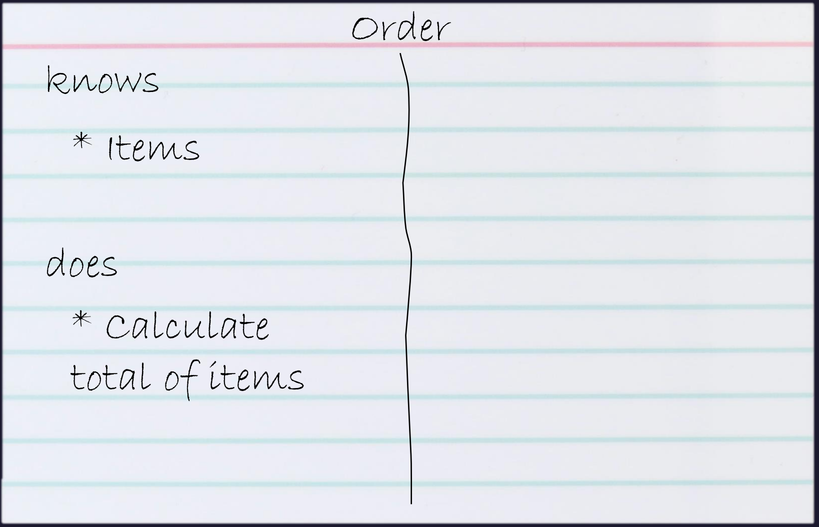
does

* Create order



```
Feature: Checkout

Scenario: UK Customer, Payment accepted
Given A basket for a <customer id> with one or more <items>
And The Customer's <country> is the UK
And Their payment method has <credit> >= basket total + <shipping>
When They check out
Then The <total> of items in the basket is calculated
And The customer's <name> and <address> is retrieved using their <customer id>
And <shipping> is calculated for the customer's <country> and basket <total>
And Their payment method is <charged> the basket <total> + <shipping>
And An <order> is created for that customer at that <address> with the basket <items>
And The warehouse stock of each item <product> in the basket is adjusted by the item <quantity>


Examples:
  | customer id | items                                         | name         | address                              | country | credit | shipping
  | 12          | [{ product: {price: 100.01}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 100.01 | 0.0
  | 12          | [{ product: {price: 100.00}, quantity: 1}] | Jason Gorman | 10 Acacia Drive, London, SW19 5RT | UK      | 104.99 | 4.99
```

codemanship



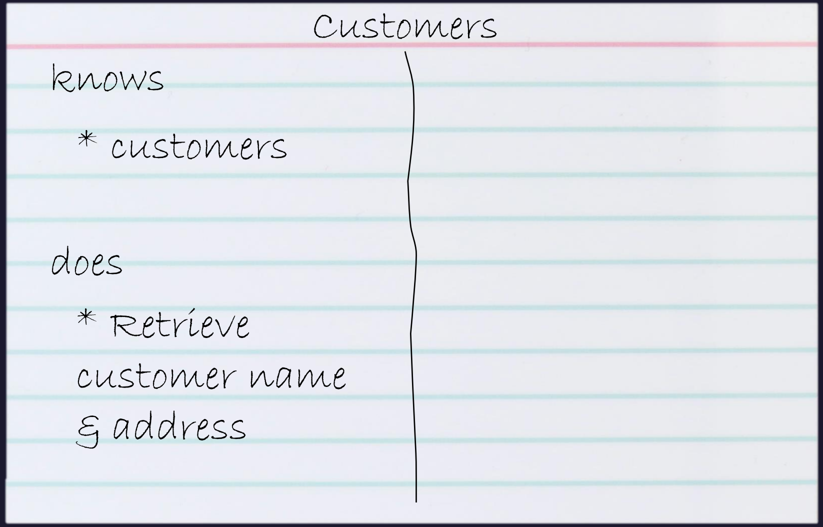
Order

knows

* Items

does

* Calculate total of items



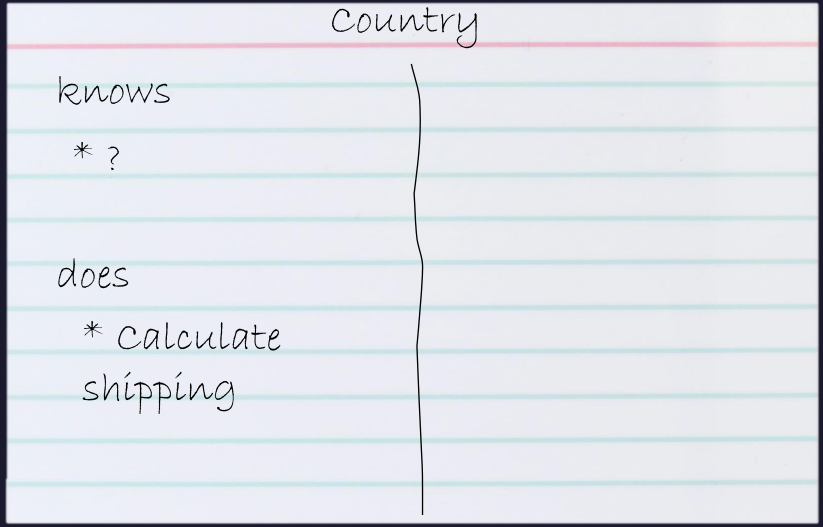
Customers

knows

* customers

does

* Retrieve customer name & address



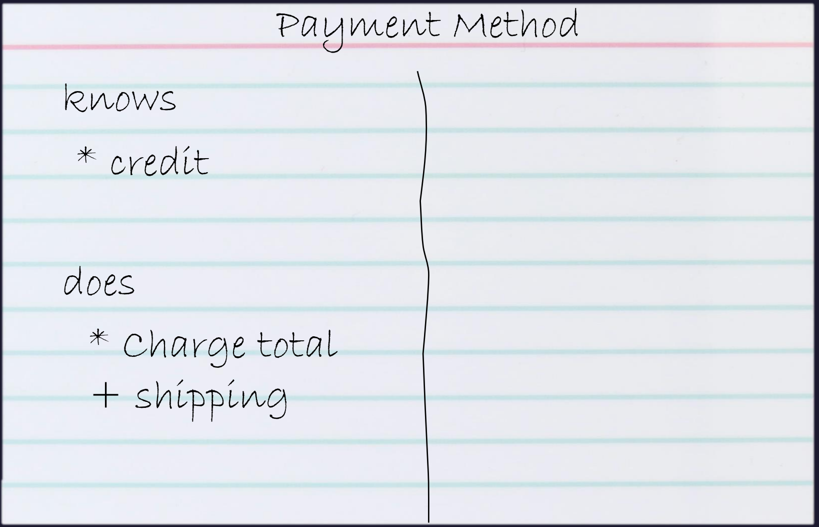
Country

knows

* ?

does

* Calculate shipping



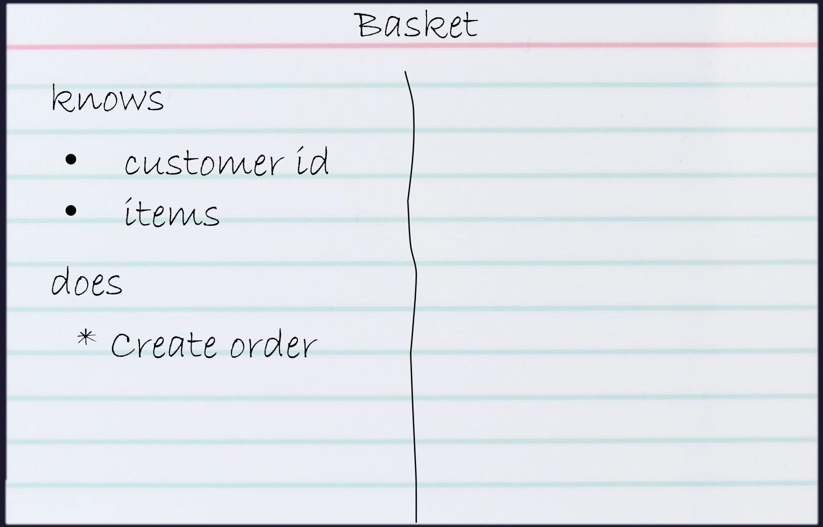
Payment Method

knows

* credit

does

* Charge total + shipping



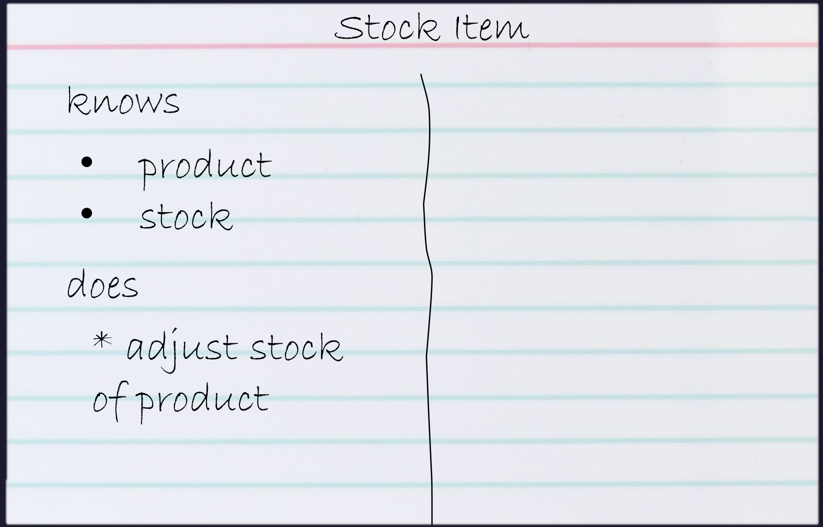
Basket

knows

- customer id
- items

does

* Create order



Stock Item

knows

- product
- stock

does

* adjust stock of product

Order

knows

* Items

does

* Calculate total of items

Country
Payment Method
Stock Item

Customers

knows

* customers

does

* Retrieve customer name & address

Country

knows

* ?

does

* Calculate shipping

Payment Method

knows

* credit

does

* Charge total + shipping

Basket

knows

- customer id
- items

does

* Create order

Customers
Order

Stock Item

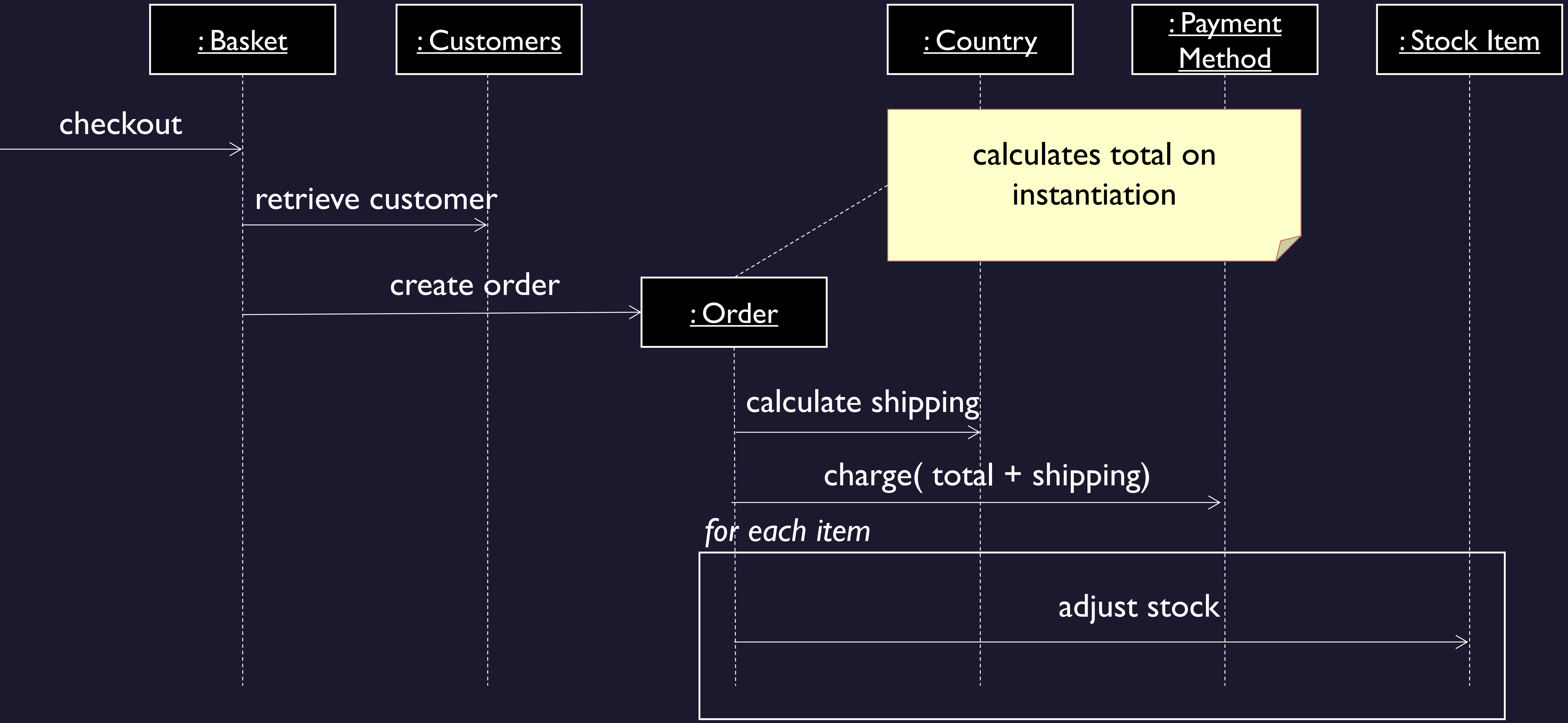
knows

- product
- stock

does

* adjust stock of product

codemanship


: Basket
: Customers
: Country
: Payment Method
: Stock Item
checkout
retrieve customer
create order
: Order
calculates total on instantiation
calculate shipping
charge(total + shipping)
for each item
adjust stock

Business Requirements

Guitar Shack

https://github.com/jasongorman/socratesuk

Guitar Shack Sales System

- You are tasked with developing a sales and stock control system for a guitar shop
- The initial system will have functionality in three areas:
    - Orders
    - Warehouse
    - Sales History
    - Shipping

Order Features

- **Add item** – add an item to an order. An order item has a product and a quantity. There must be sufficient stock of that product to fulfil the order
- **Total including shipping** – calculate the total amount payable for the order, including shipping to the address
- **Confirm** – when an order is confirmed, the stock levels of every product in the items are adjusted by the item quantity, and then the order is added to the sales history.

Warehouse Features

- **Check stock** – for a specified product, get the current stock level
- **Adjust stock** – deduct a quantity from a product's stock level
- **Receive stock** – add new stock to a product, and if the product is new, add it to the catalogue first
- **Restock Alerts** – when stock's adjusted, if that product has reached its restock threshold, where we'll need to order more to avoid running out before more stock arrives from the manufacturer, an alert is sent to the warehouse manager. Use the product's sales history and its restock lead time to calculate what the restock level should be.

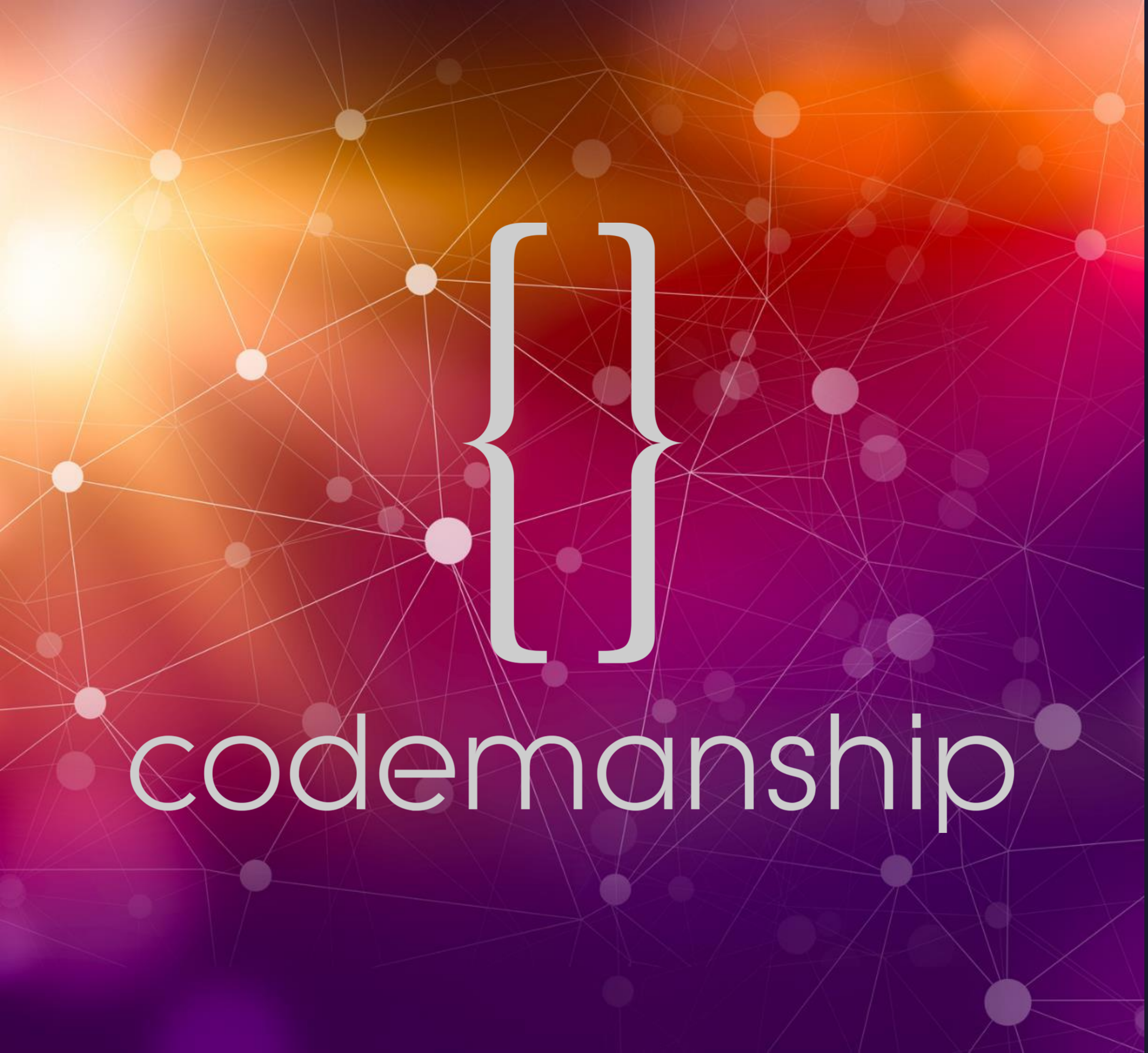
Sales History Features

- **List orders of a specific product** (optionally within a date range)
- **List orders shipped to an address**

Shipping

- **Shipping charges for orders are calculated as follows:**

Region	**Order total > £100**	**Order total <= £100**
UK	FREE	£4.99
EU	£4.99	£7.99
OTHER	£7.99	£12.99



Contact us

www.codemanship.com

Twitter: @codemanship