Encapsulation and Abstraction

Maurice Naftalin
Java Champion, JavaOne Rock Star
Author: *Mastering Lambdas, Java Generics and Collections*
@mauricenaftalin

Module Overview

Encapsulation: Isolating System Changes

Assigning Responsibilities

Creating an `Order`

The Single Responsibility Principle

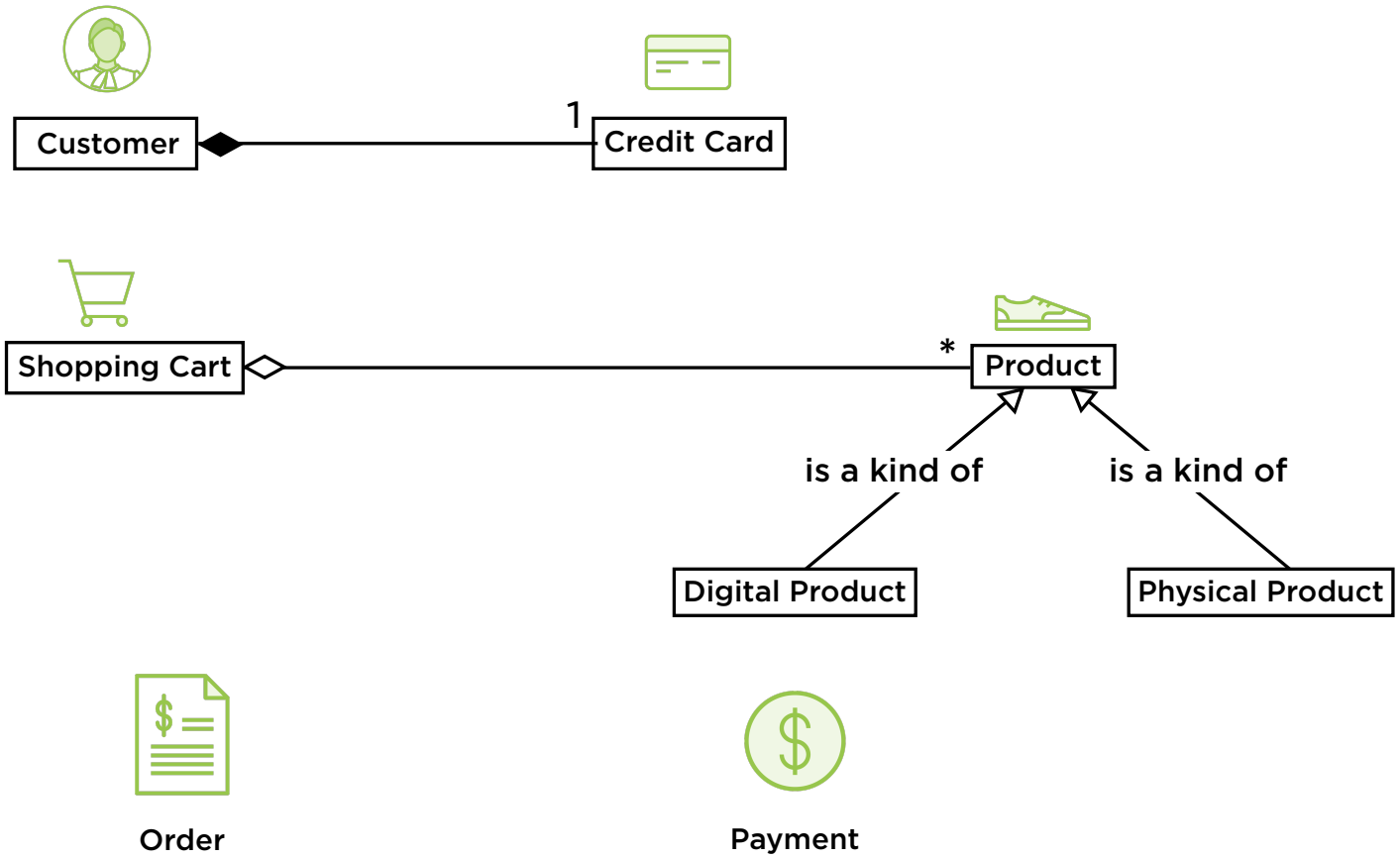
Defensive Copying

Abstraction and Encapsulation

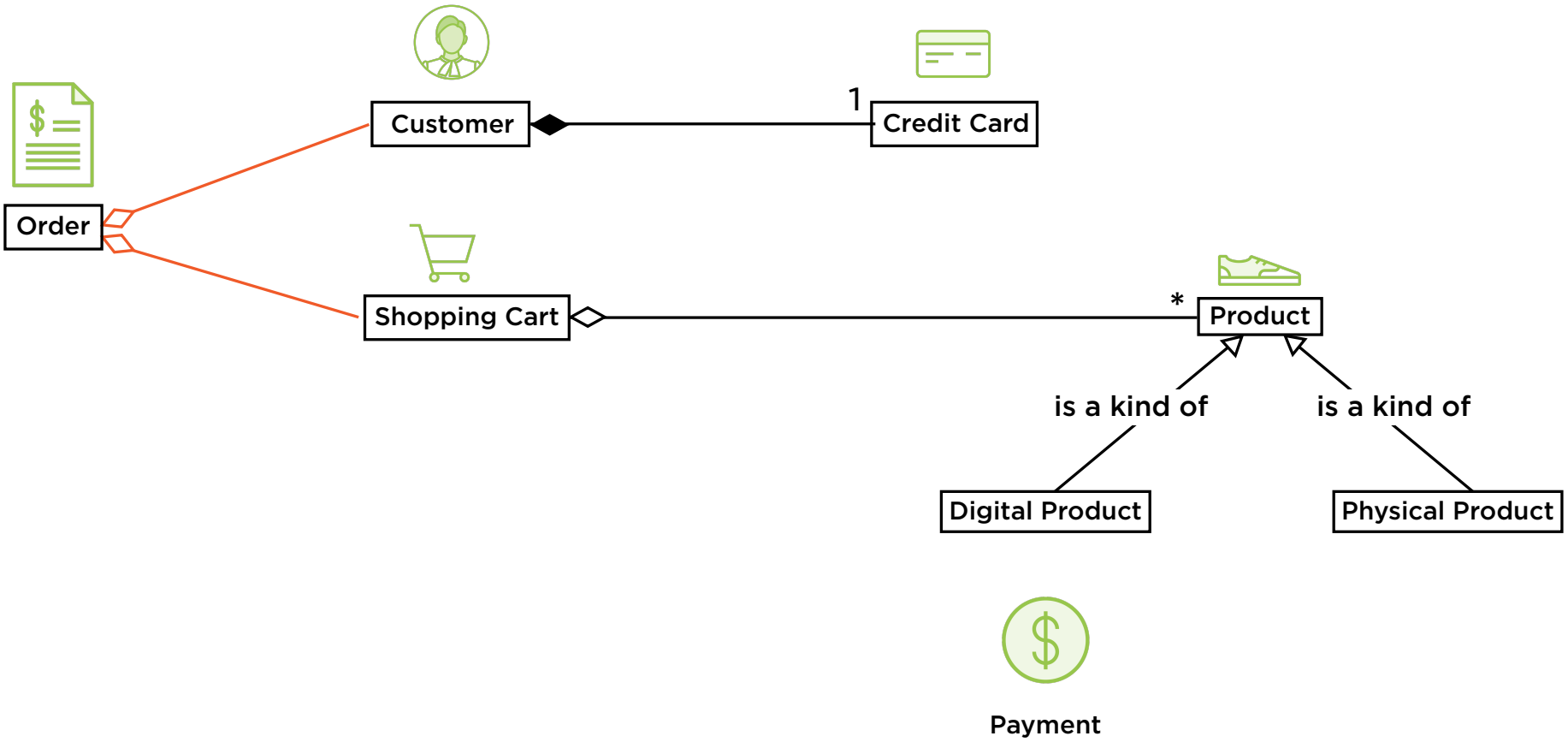
Use Case 1: Create Order and Check Out

"A customer creates an order by first adding various products (digital or physical) to a shopping cart, then checks out, making a payment using a credit card."

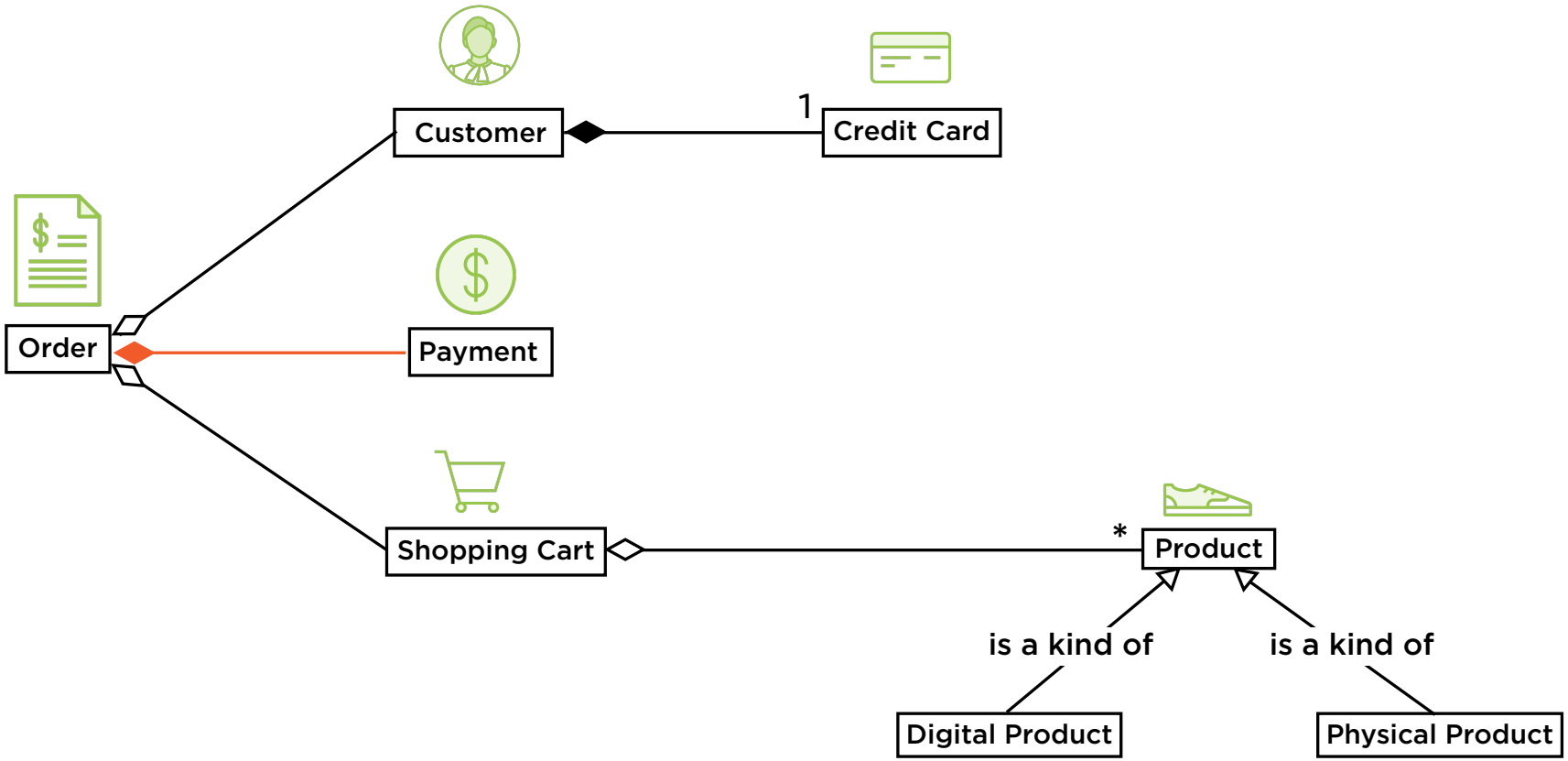
The Design so Far



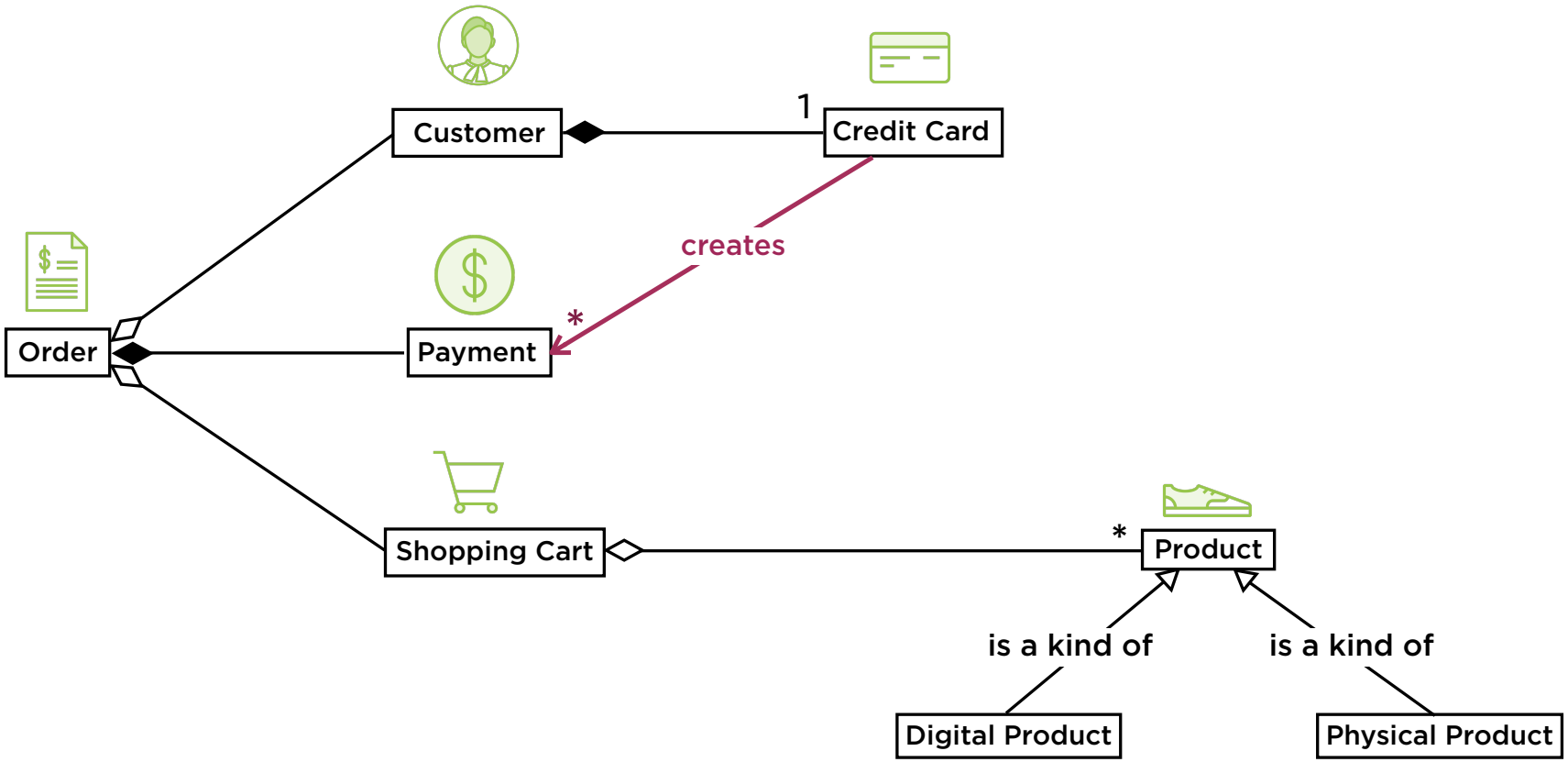
More Component Relationships



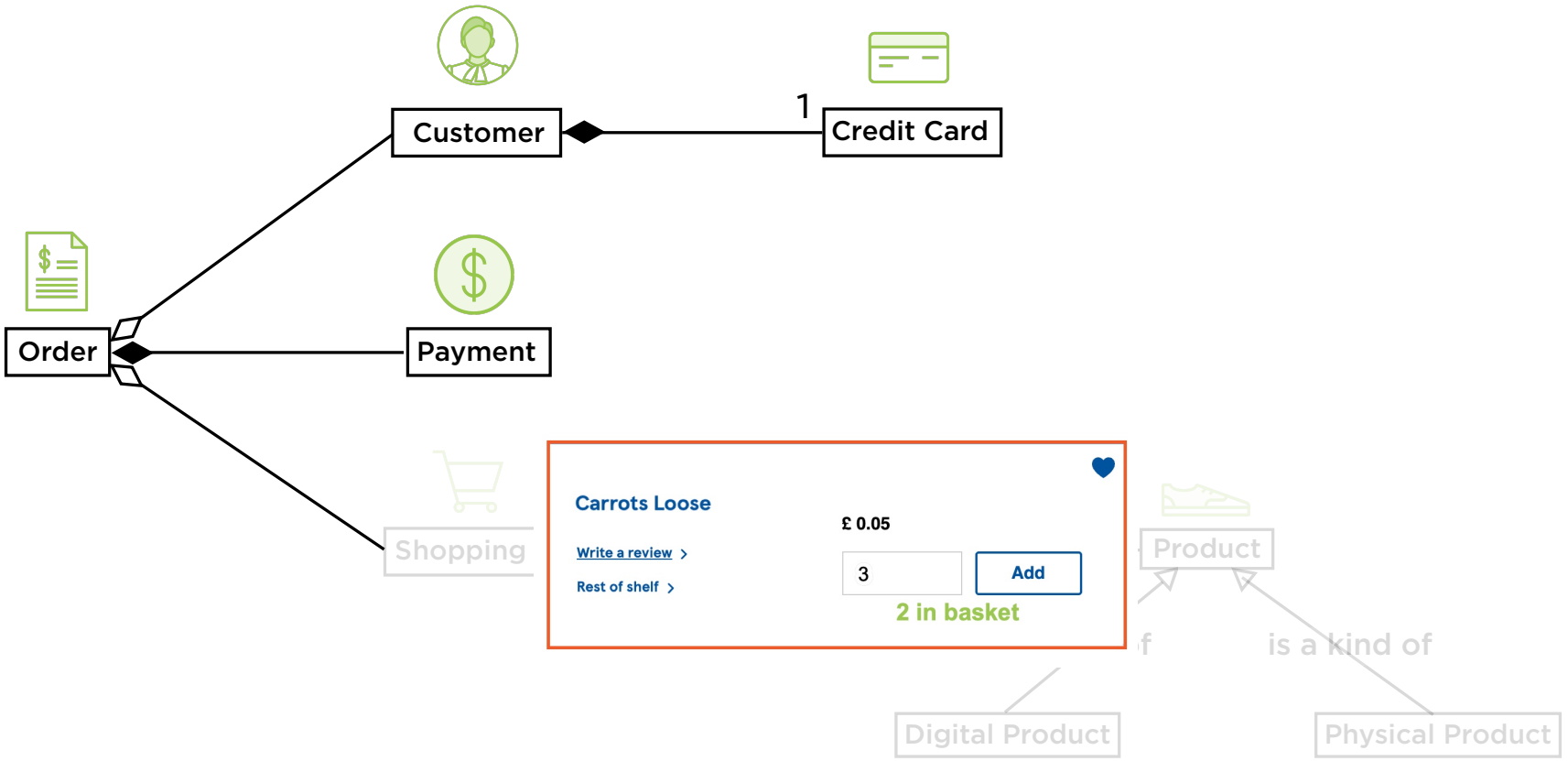
More Component Relationships



More Component Relationships



Finding Missing Classes

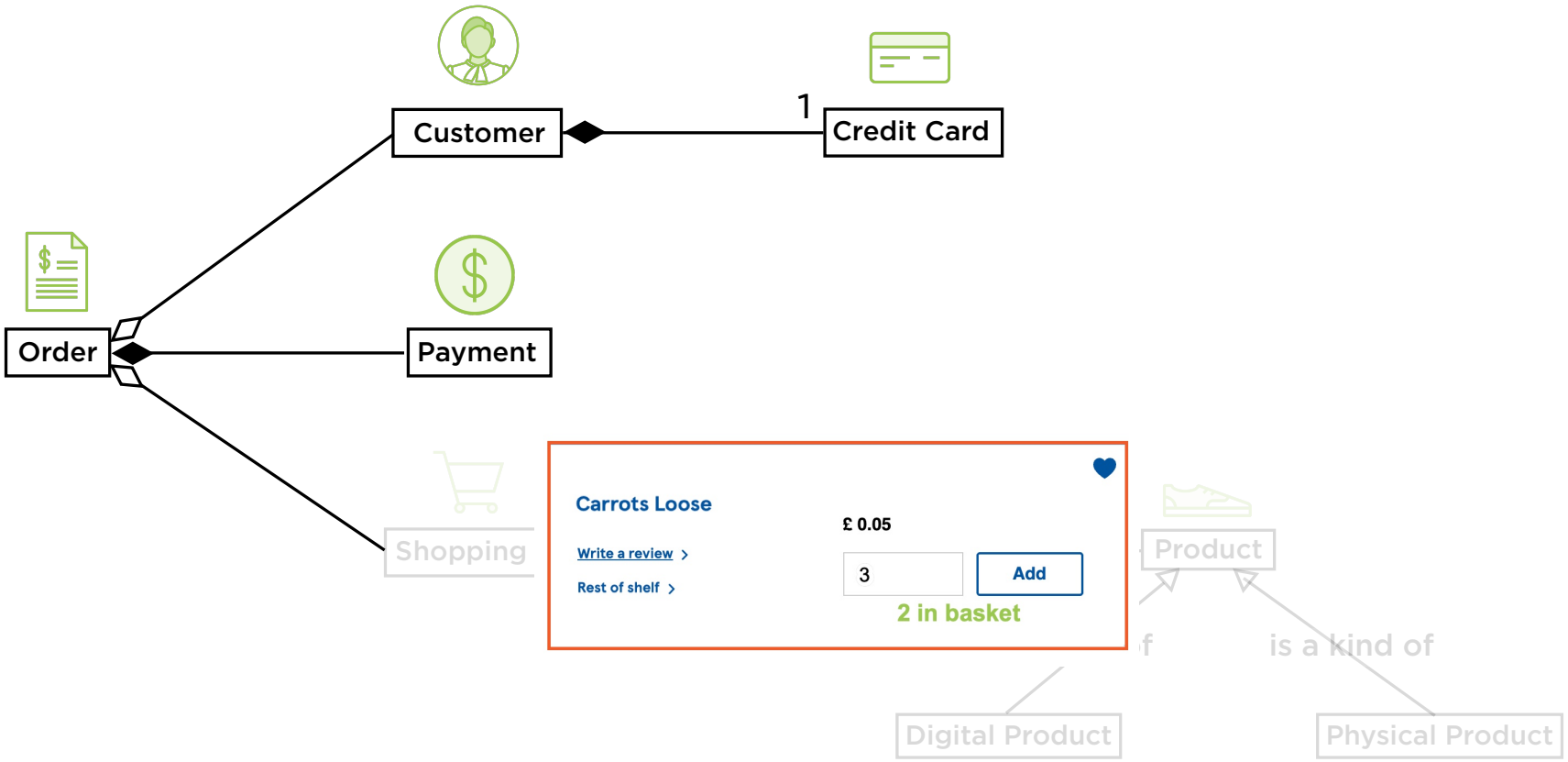


Use Case 1: Create Order and Check Out

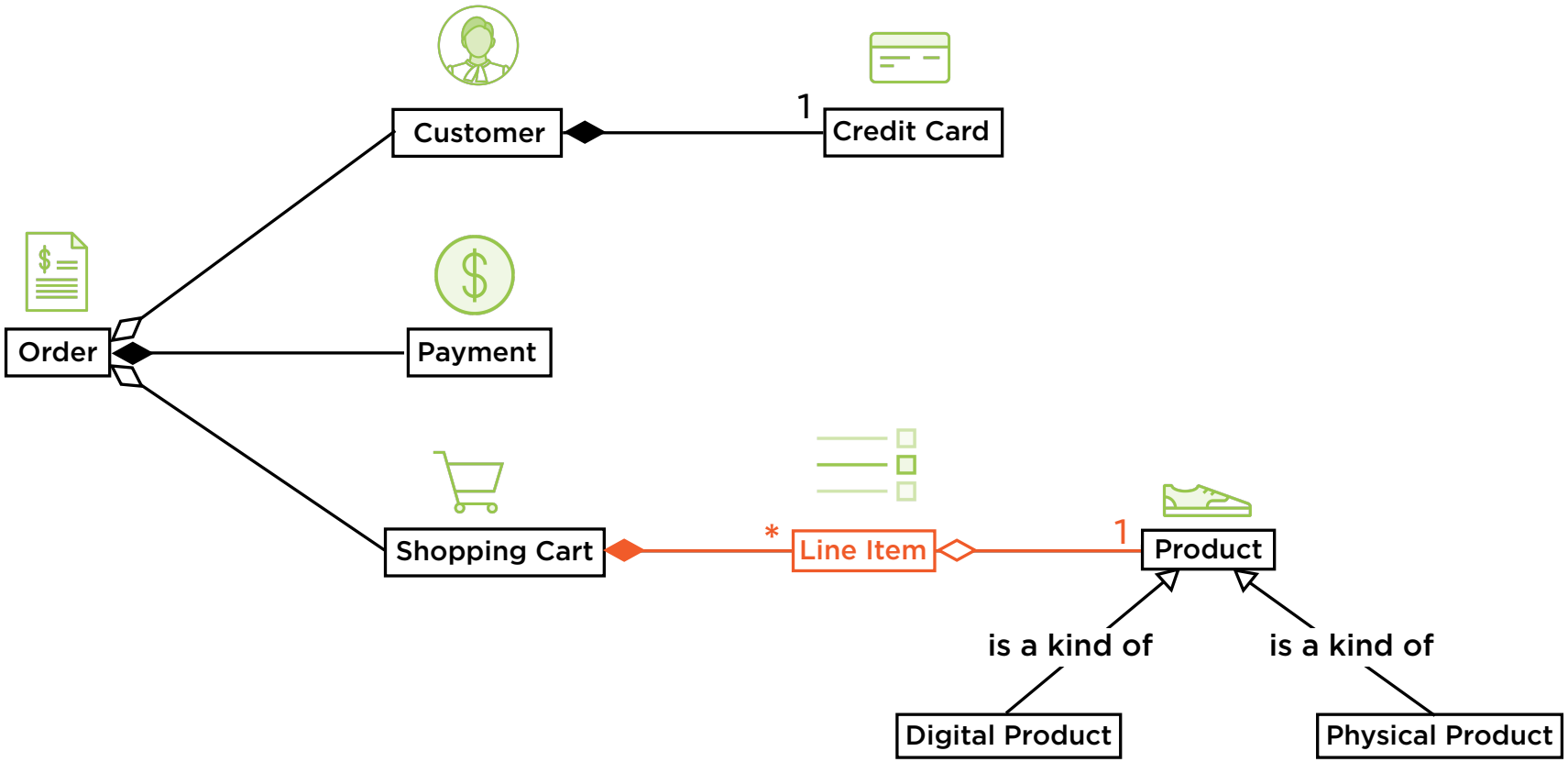
different quantities of

"A customer creates an order by first adding various products (digital or physical) to a shopping cart, then checks out, making a payment using a credit card."

Finding Missing Classes



Finding Missing Classes



Encapsulation: What?

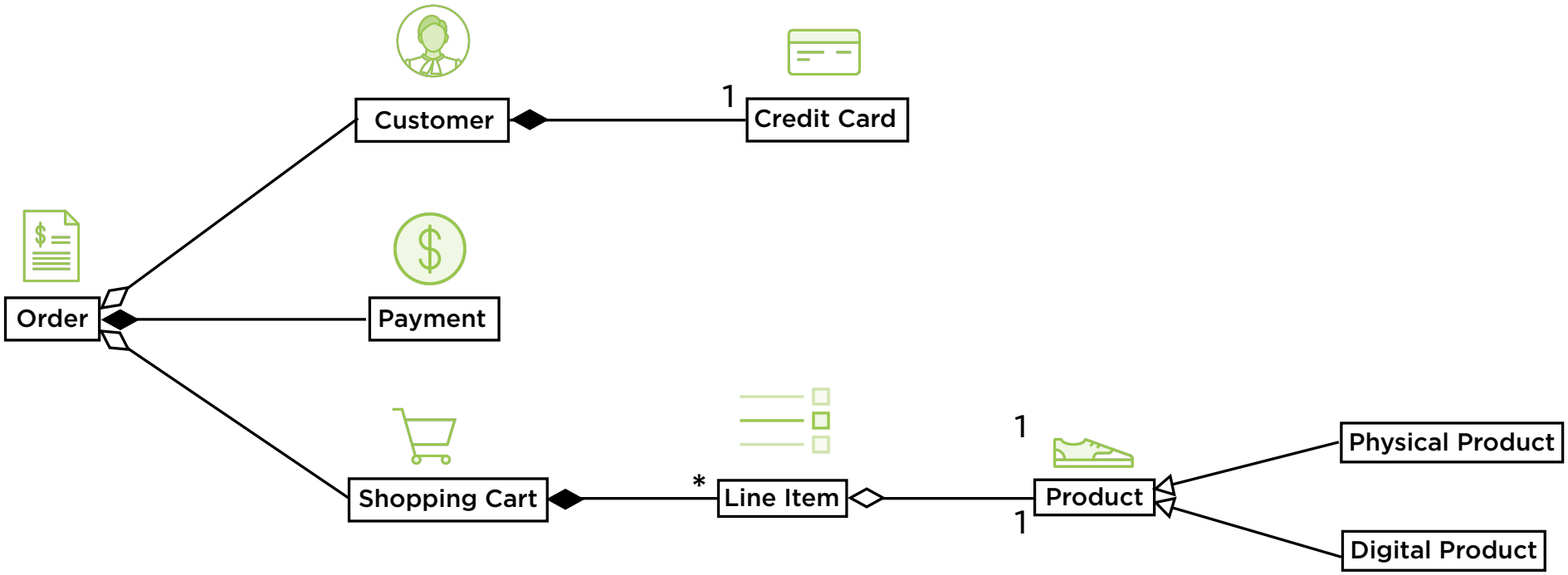
Hiding implementation detail

Especially detail that might impact on the rest of the system

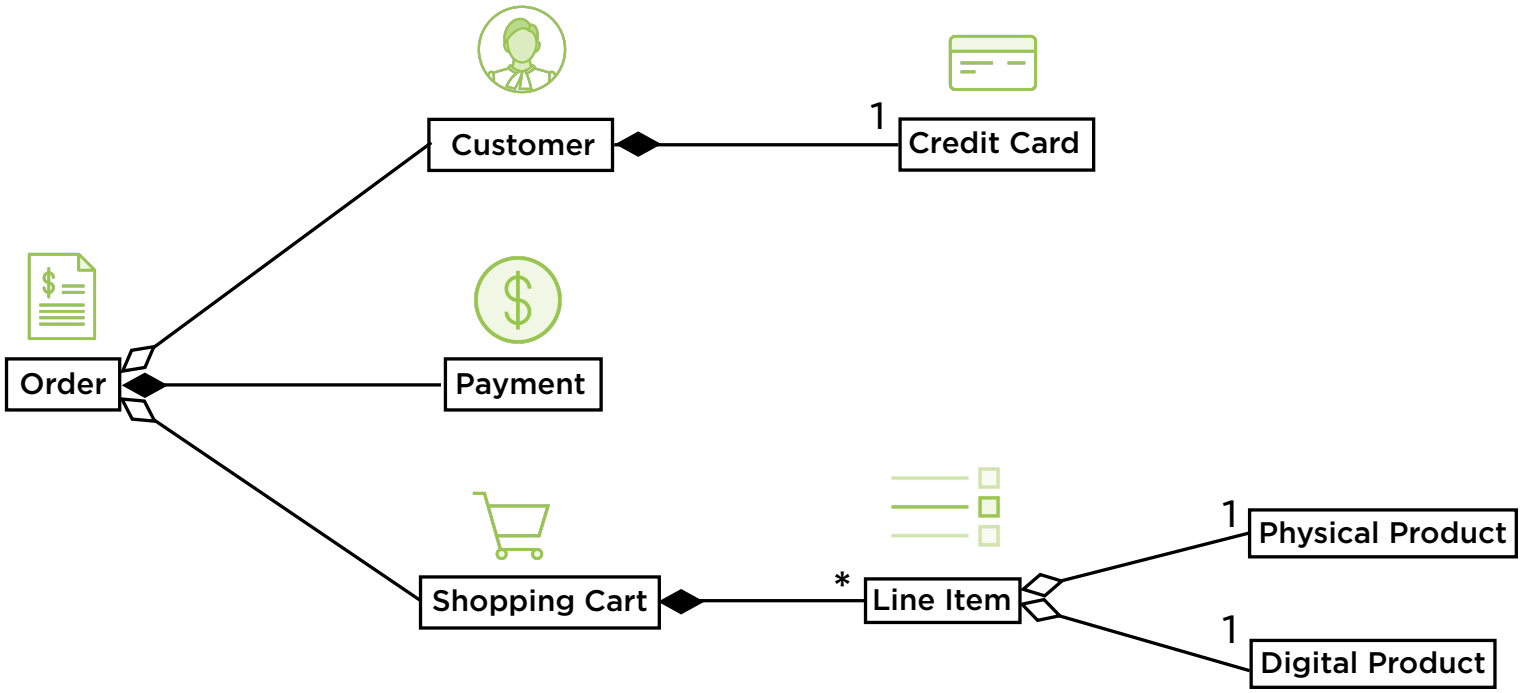
Mechanisms for hiding detail

Access modifiers; packages; modules. Also inheritance

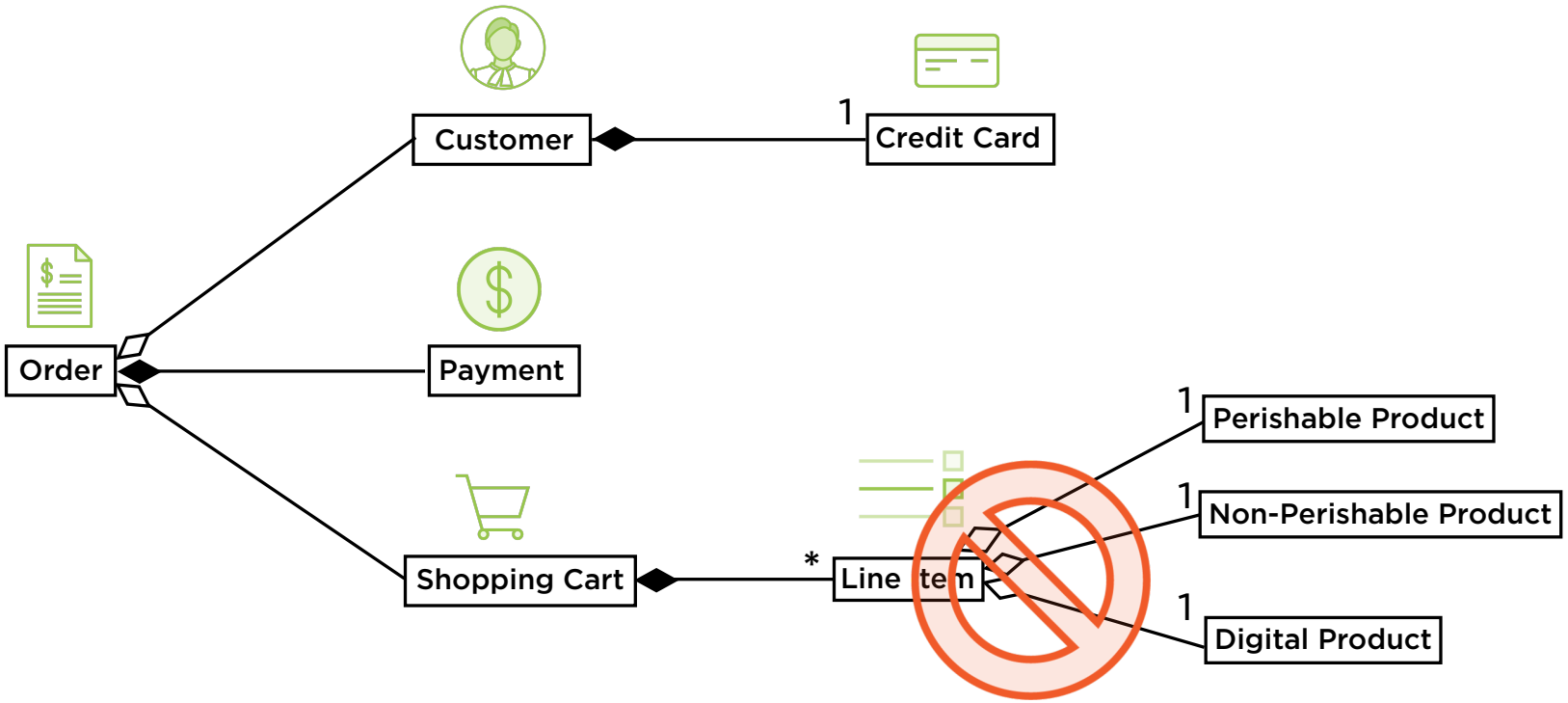
Encapsulation: Why?



Encapsulation: Why?

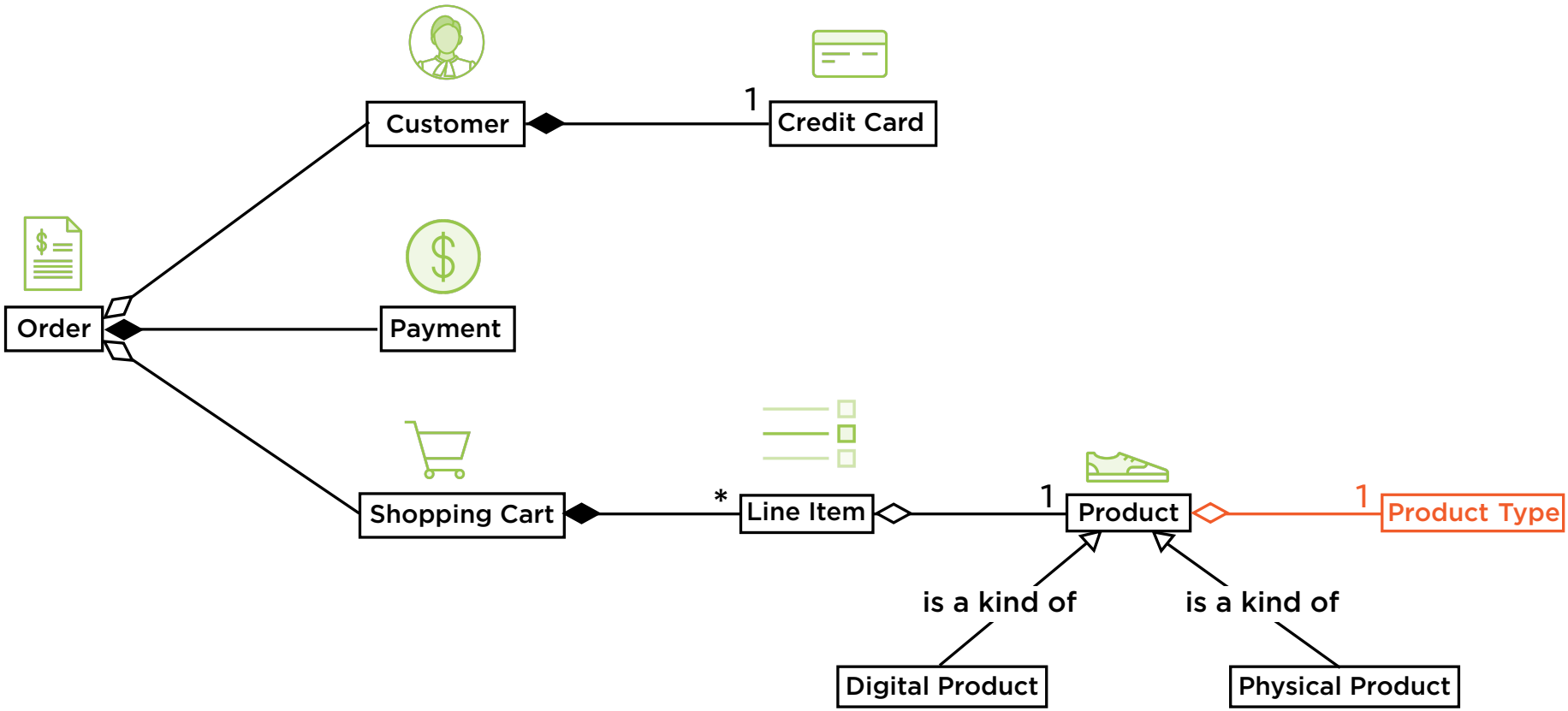


Encapsulation: Why?



Principle:
Encapsulate What Varies

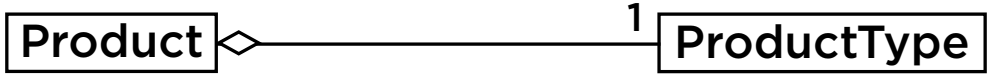
Encapsulation: Inheritance or Composition?



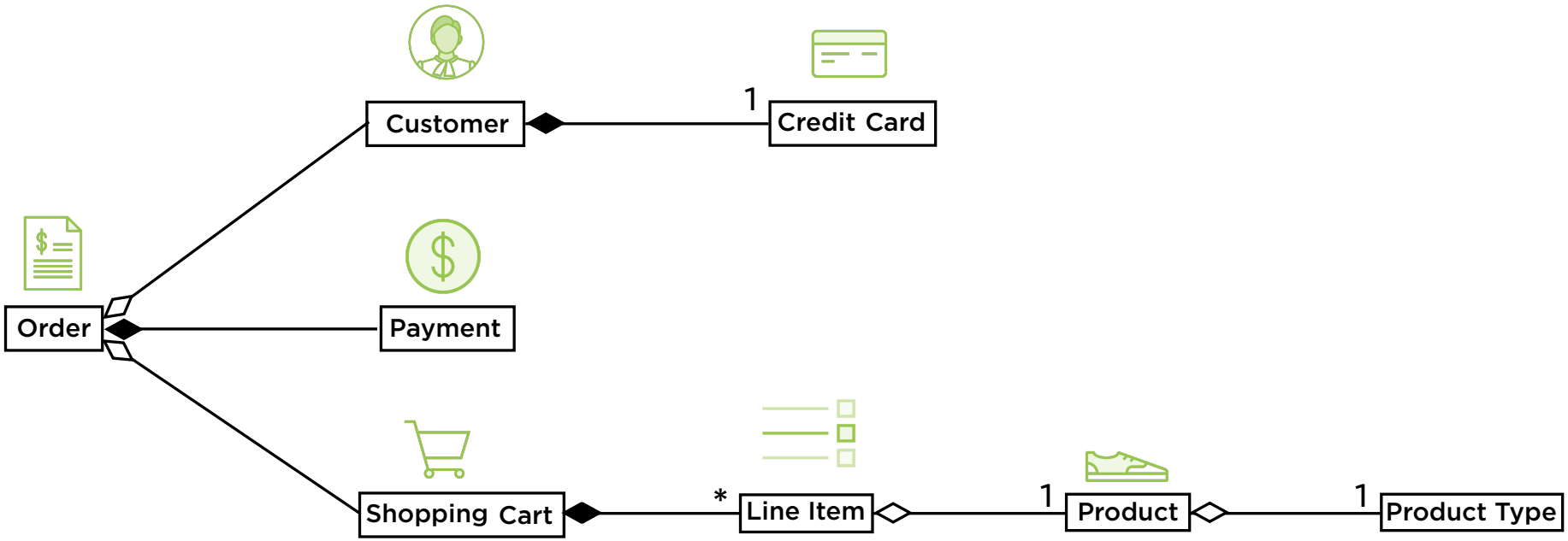
```java
public class Product {
    ...
    private ProductType type;
    ...
}
```

```java
public enum ProductType {
    DIGITAL, PHYSICAL;
}
```

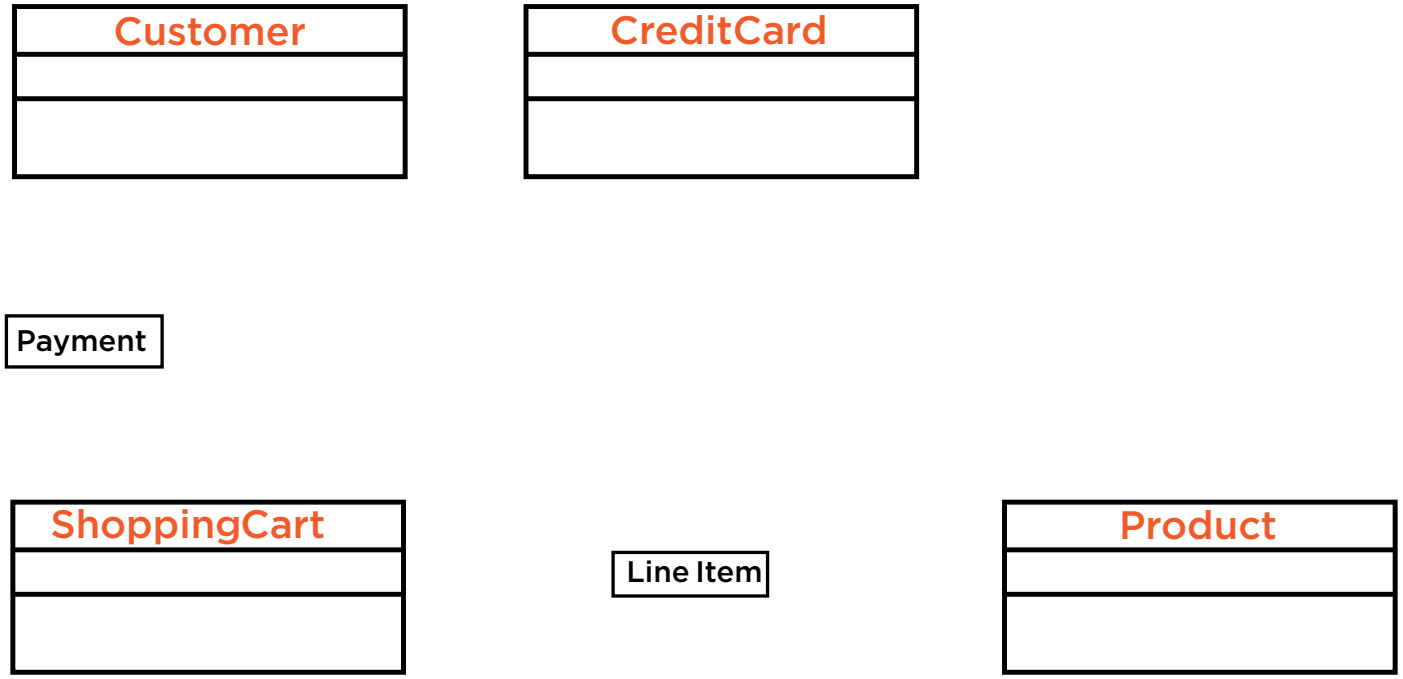
Encapsulating Product Type Variation



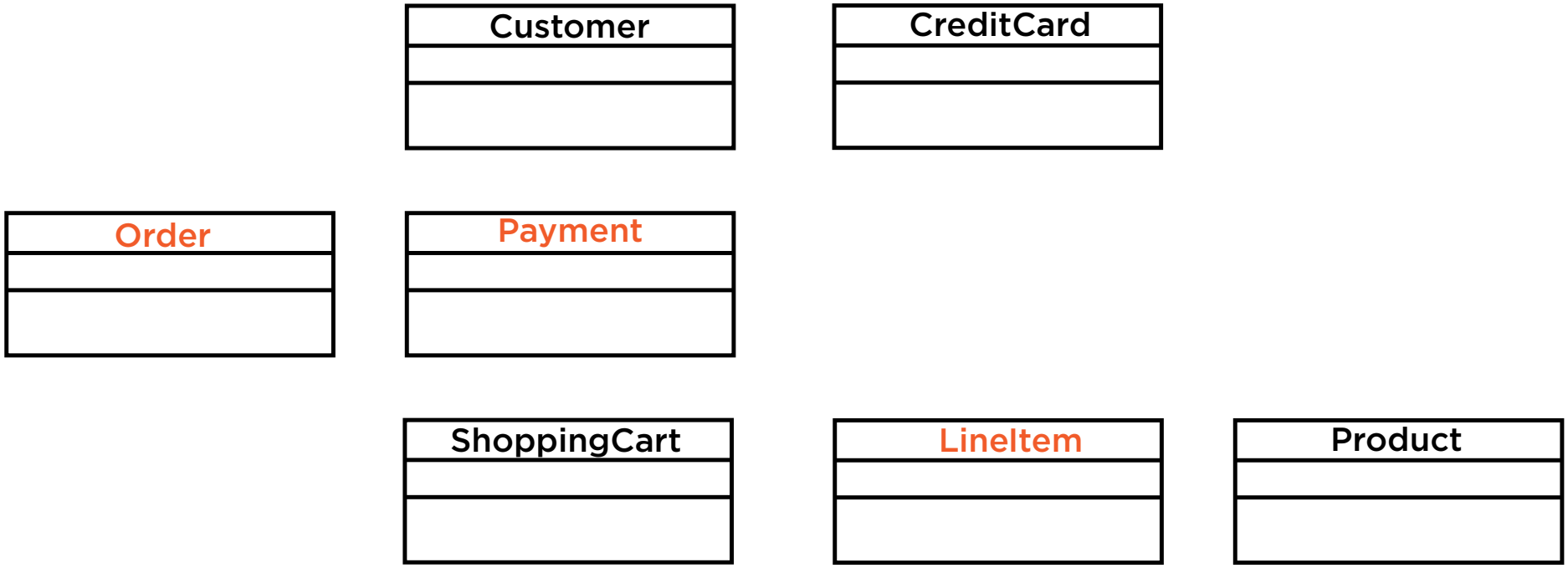
Deriving Software Classes



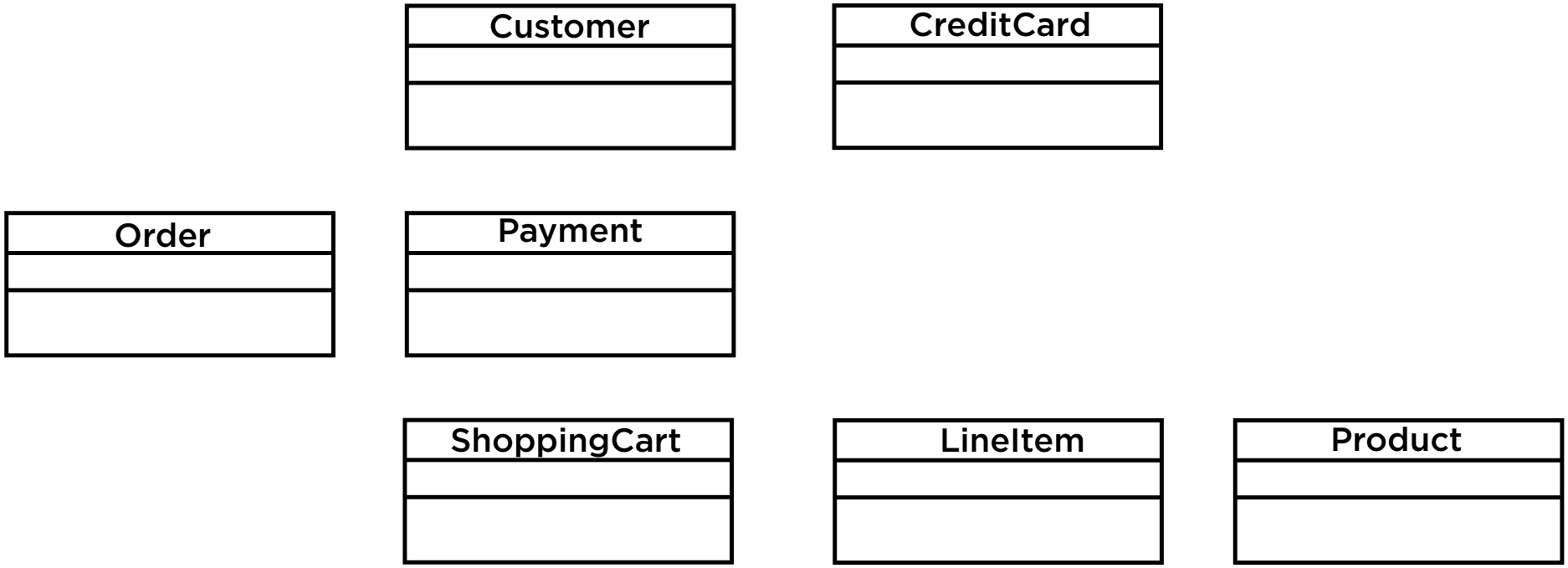
Deriving Software Classes



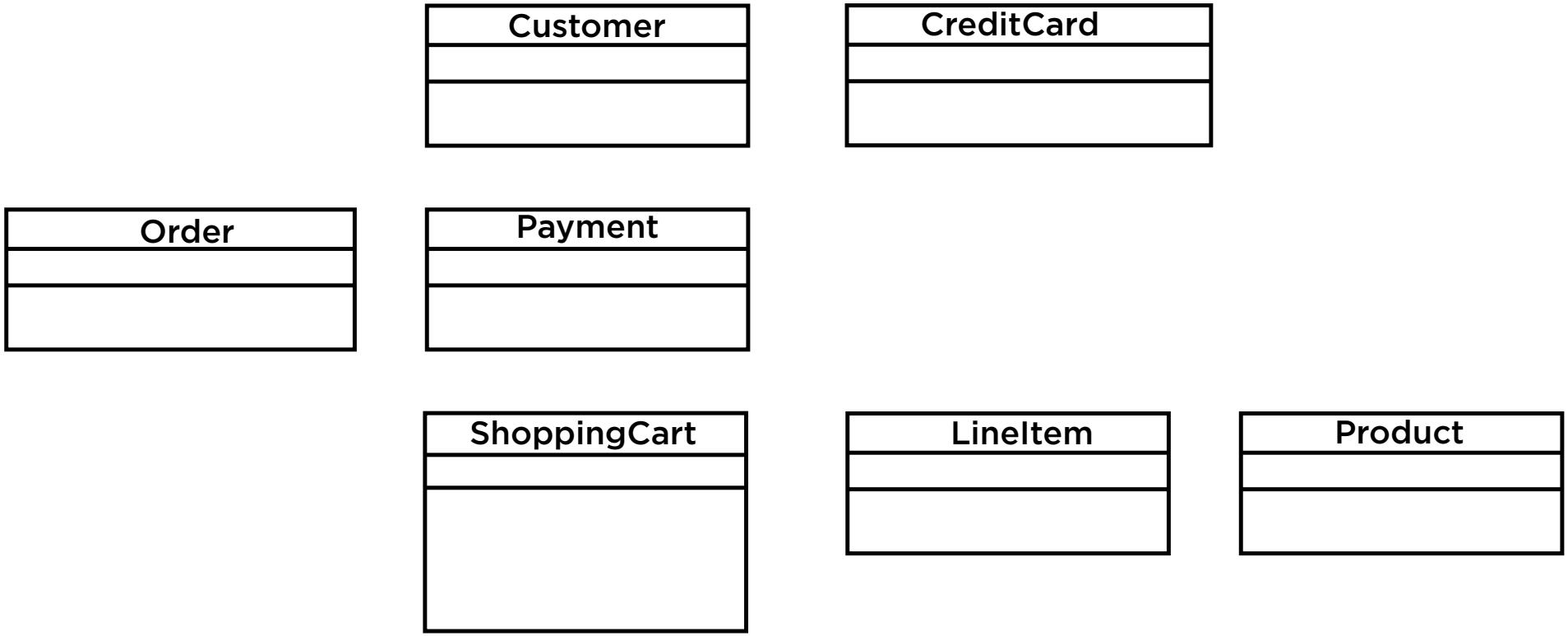
Deriving Software Classes



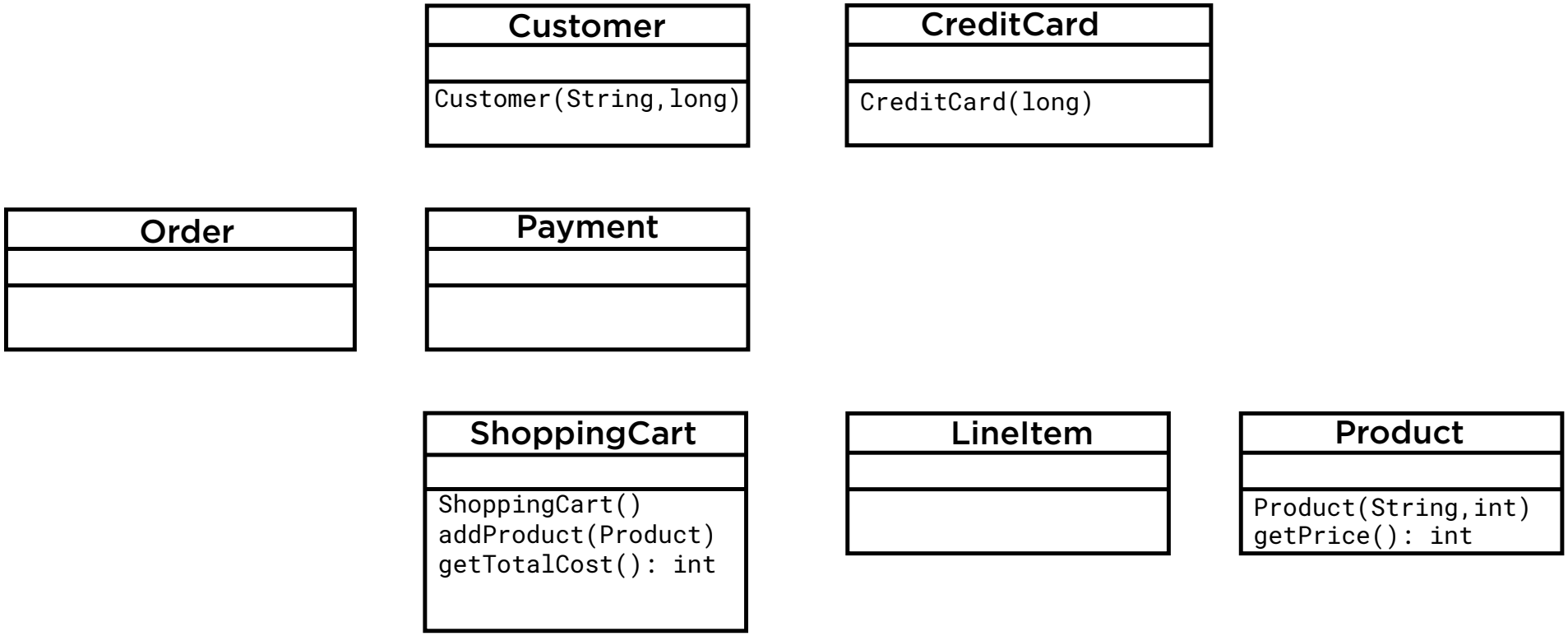
Assigning Responsibilities



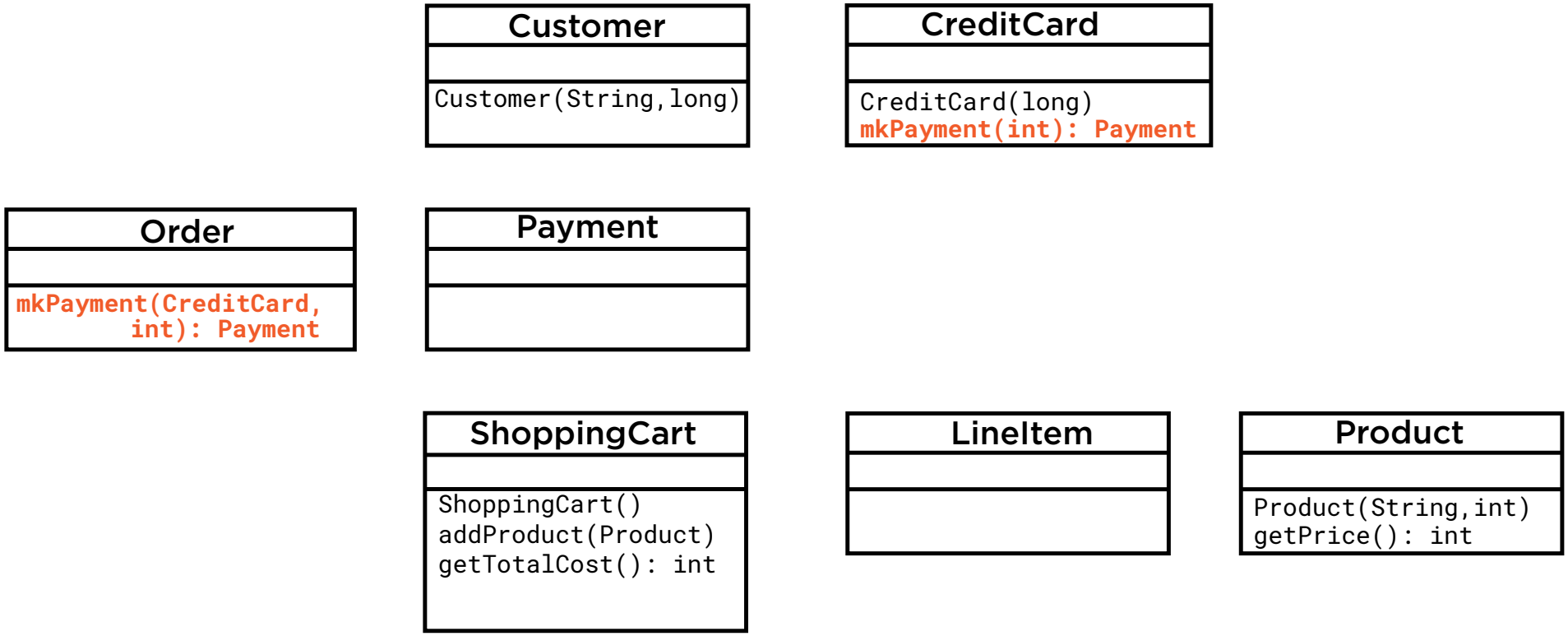
Assigning Responsibilities



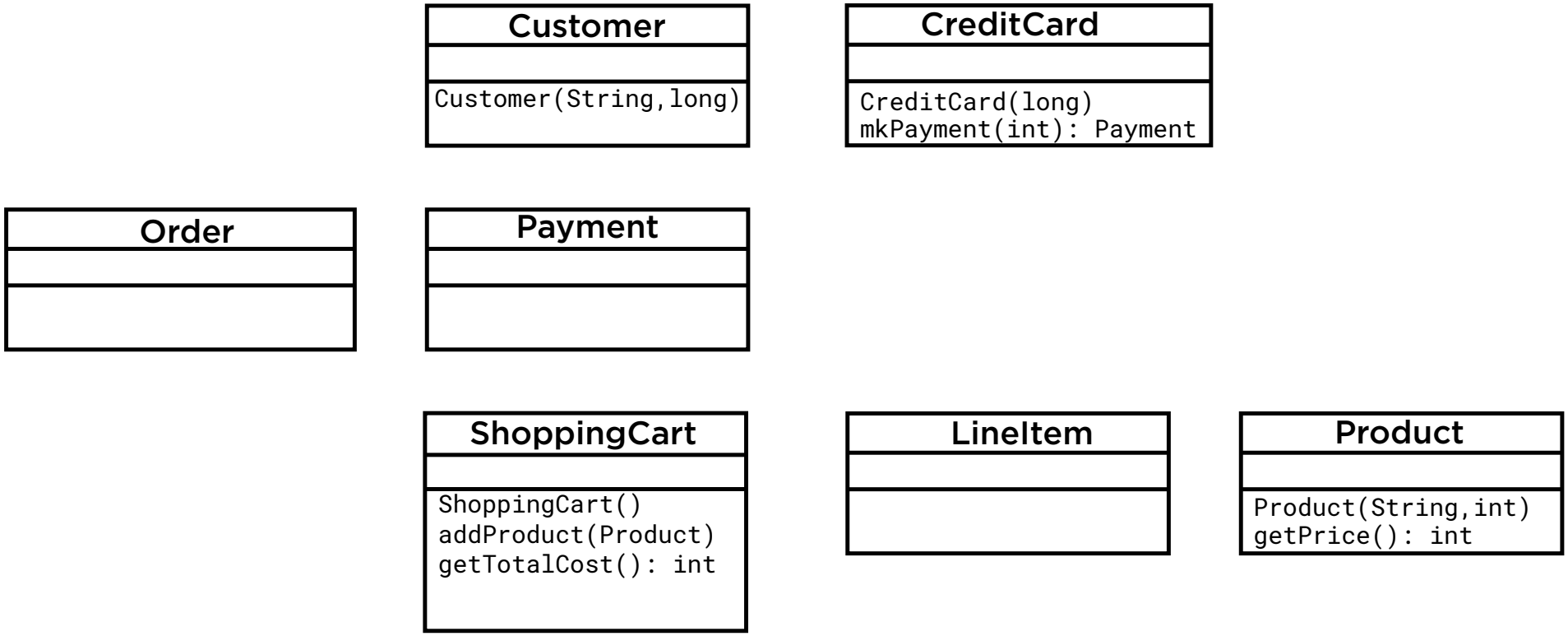
Assigning Responsibilities



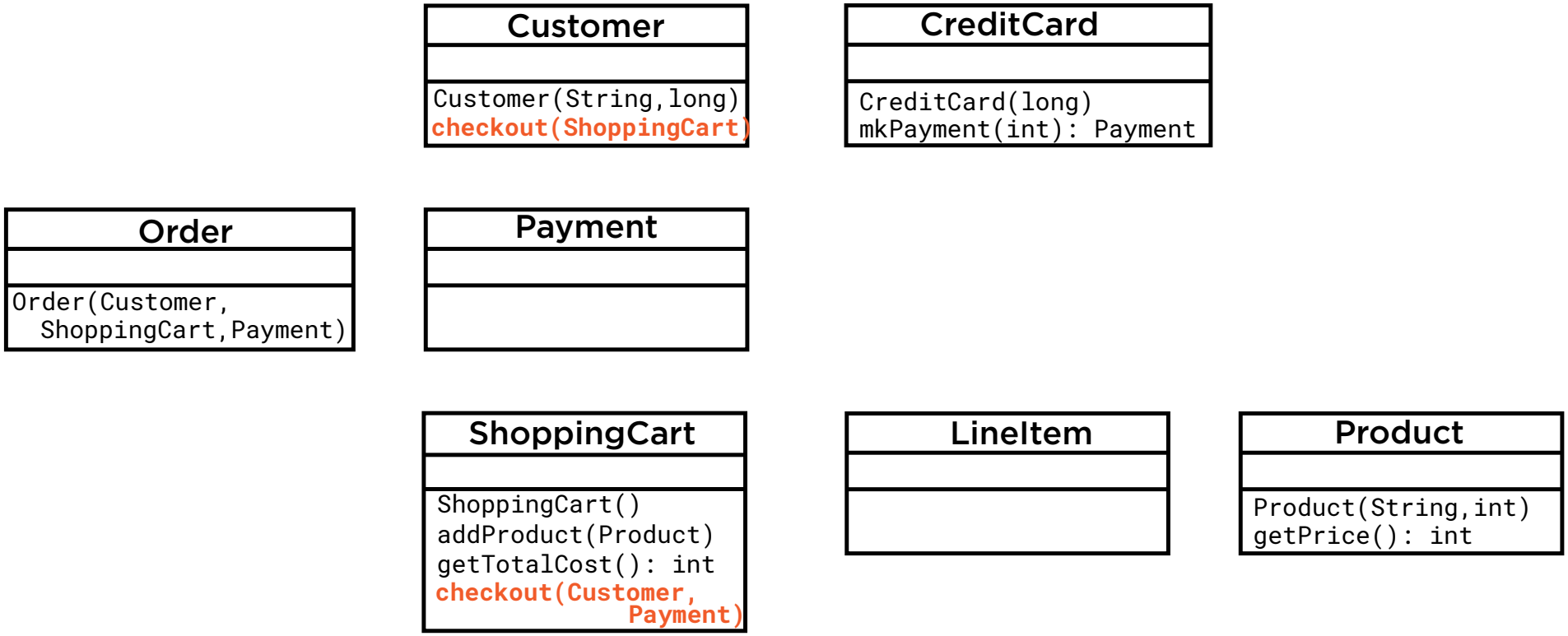
Assigning Responsibilities



Assigning Responsibilities



Assigning Responsibilities



```
public Order(Customer customer,
             ShoppingCart cart,
             Payment payment) {
    ...
}

public class Customer
   ...
   public Order checkout(ShoppingCart cart) {
       Payment p =
       creditCard.mkPayment(cart.getTotalCost());
       return new Order(this, cart, p);
   }
   ...
}

public class ShoppingCart
   ...
   public Order checkout(Customer cust) {
        Payment p =
        cust.getCard().mkPayment(getTotalCost());
        return new Order(cust, this, p);
    }
    ...
}
```

- `Order` **constructor needs all this information**
- `Customer` **object doesn't know about** `ShoppingCart`
- `ShoppingCart` **object doesn't know about** `Customer` ***or*** `CreditCart`

Demos

1. **Exercising the system**
2. **The Single Responsibility Principle**
3. **Defending encapsulation**
   - Defensive copying
   - Copy constructors
   - Immutability

The Single Responsibility Principle (SRP)

A class should have only *one* reason to change

Demo: Defending Encapsulation

How encapsulation fails

Ensuring encapsulation

- Defensive copying
- Copy constructors
- Immutability

The Need for Defensive Copying

Original
`ArrayList`

Defensive Copying

Original
`ArrayList`

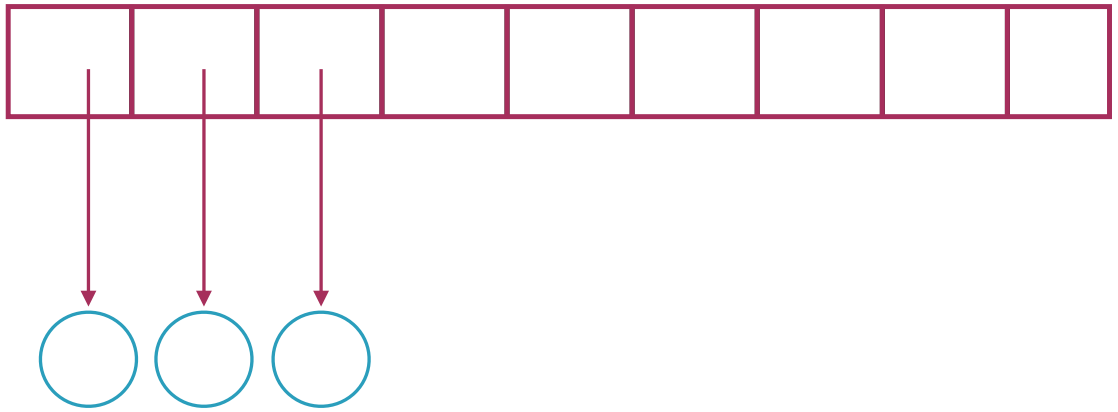
Client
access

Copy
`ArrayList`

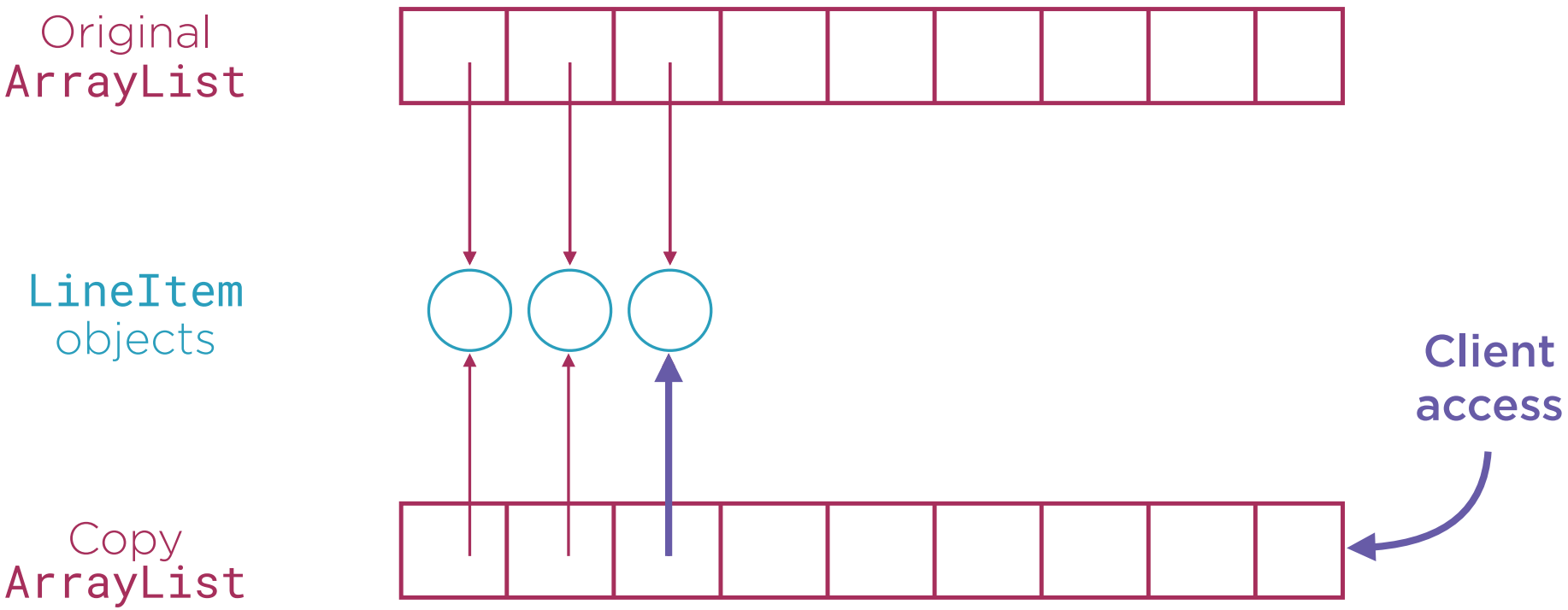
Defensive Copying

Original `ArrayList`

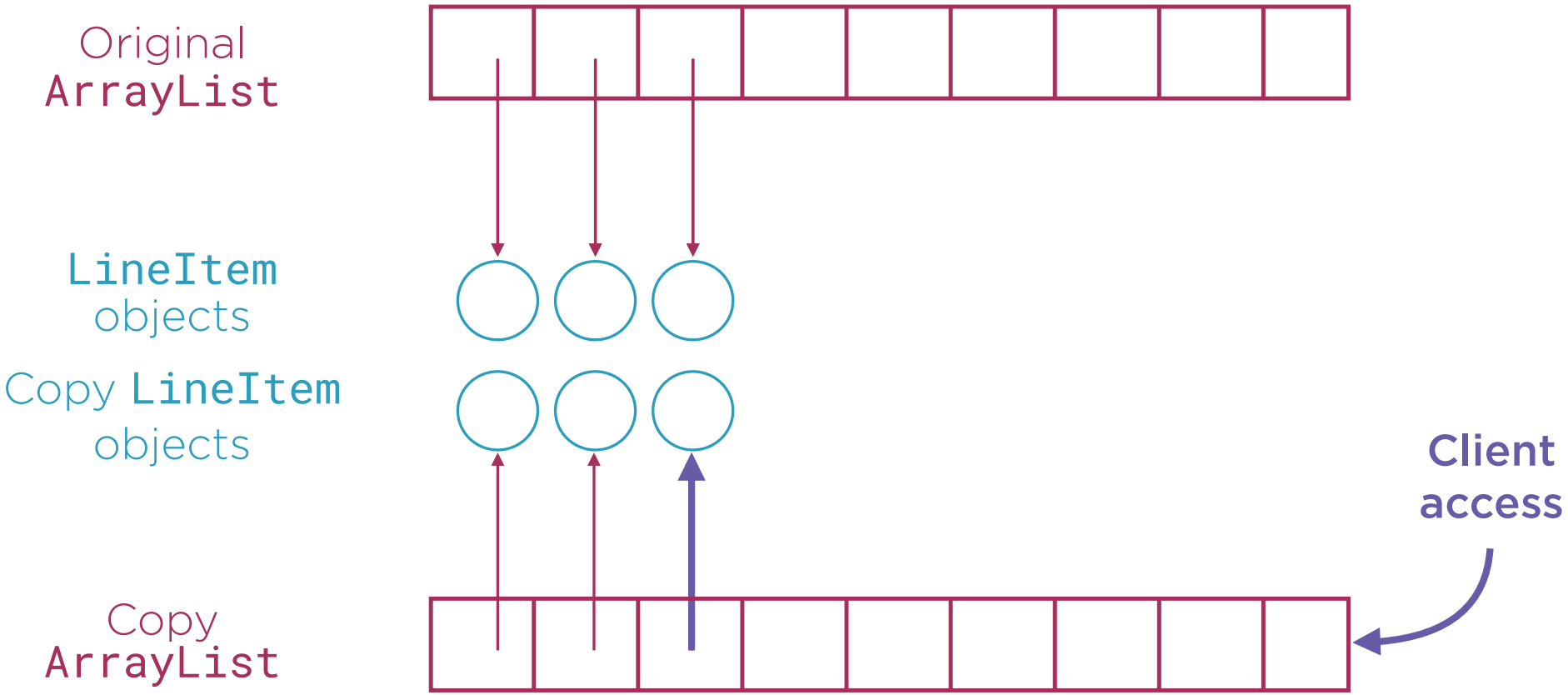
`LineItem` objects



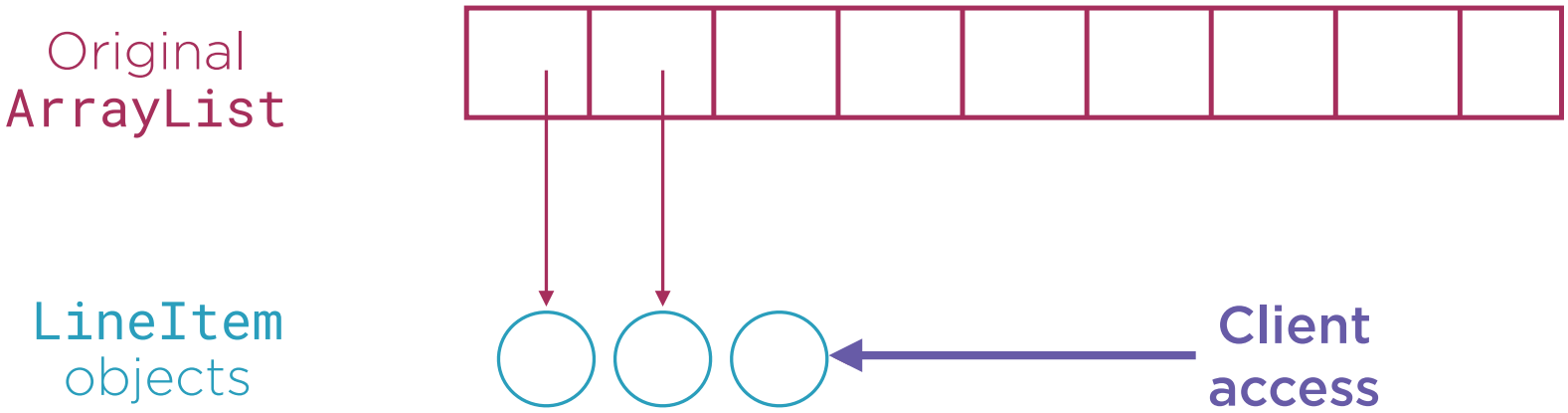
Defensive Copying — Fail!



Improved Defensive Copying



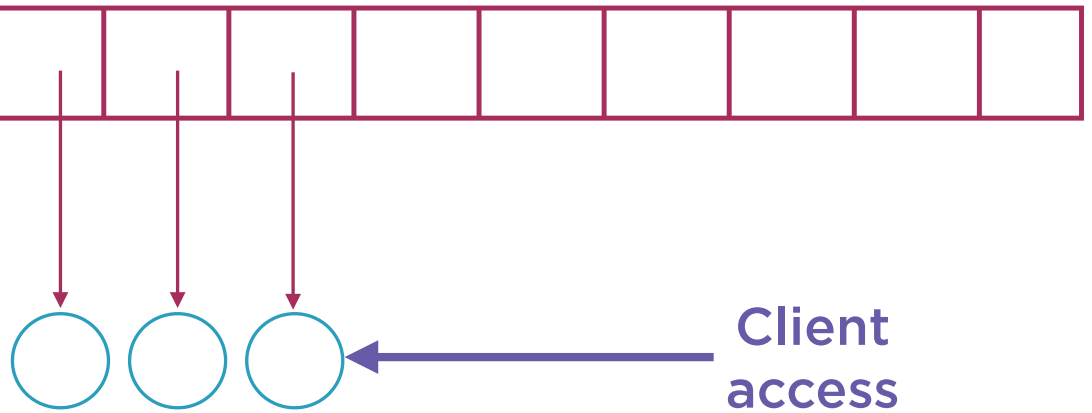
Improved Defensive Copying



Improved Defensive Copying — Fail!

Original `ArrayList`

`LineItem` objects



Abstraction and Encapsulation

Abstraction

- Hiding detail for a higher-level view
- Or defining things by their essential characteristics
- In software, can define abstract objects

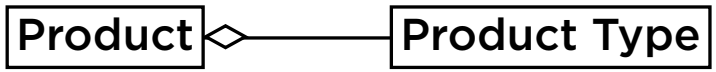
Encapsulation

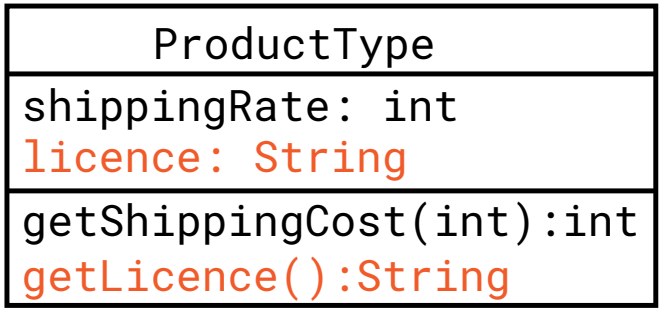
- An implementation of abstraction

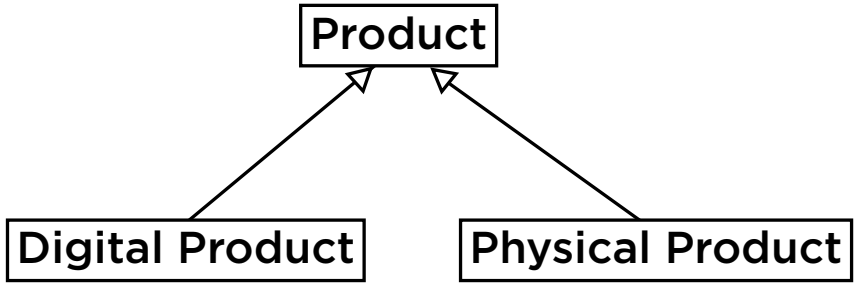
Encapsulation in `ProductType`

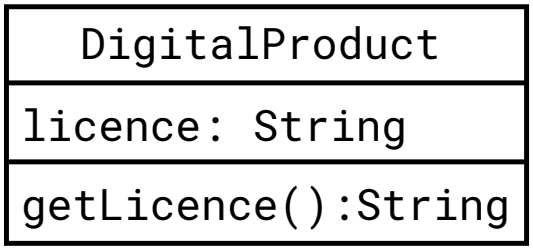
- What if other differences appear?

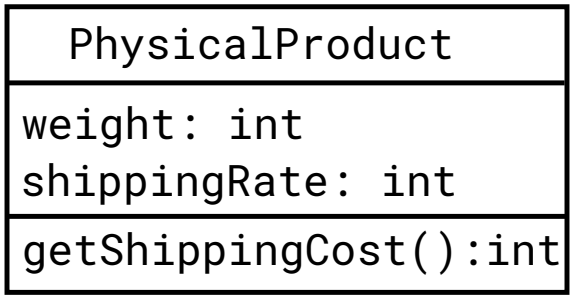
Aggregation or Inheritance?











Module Summary

Encapsulation: Isolating System Changes

The Single Responsibility Principle

Defensive Copying and Immutability

Abstraction and Encapsulation

Up Next:

Software Reuse with Inheritance