Establishing Relationships

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/

Object-Oriented Programming (OOP)

Identifying classes

- Represents business entities
- Defines properties (data)
- Defines methods (actions/behavior)

Separating responsibilities

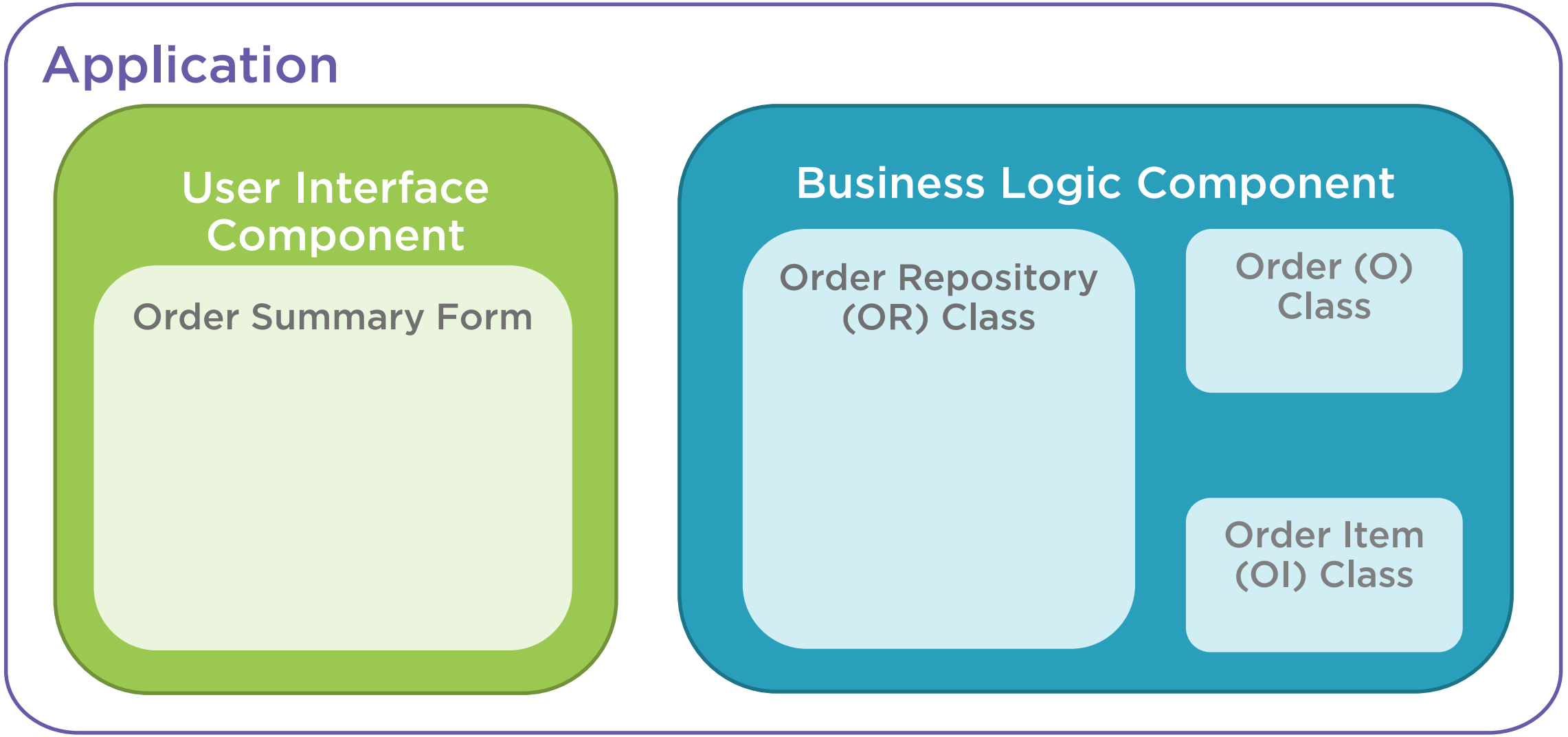
- Minimizes coupling
- Maximizes cohesion
- Simplifies maintenance
- Improves testability

Establishing relationships

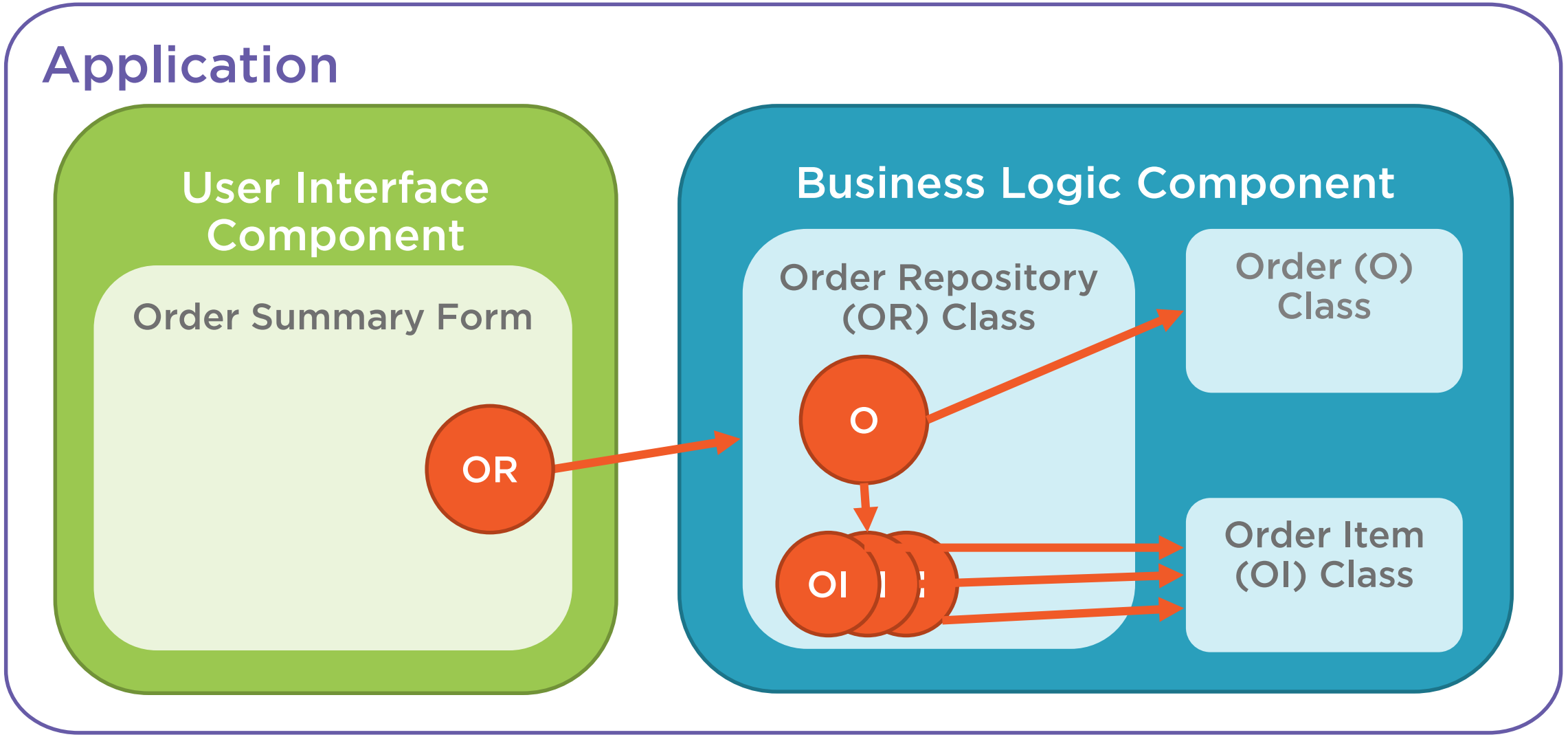
- Defines how objects work together to perform the operations of the application

Leveraging reuse

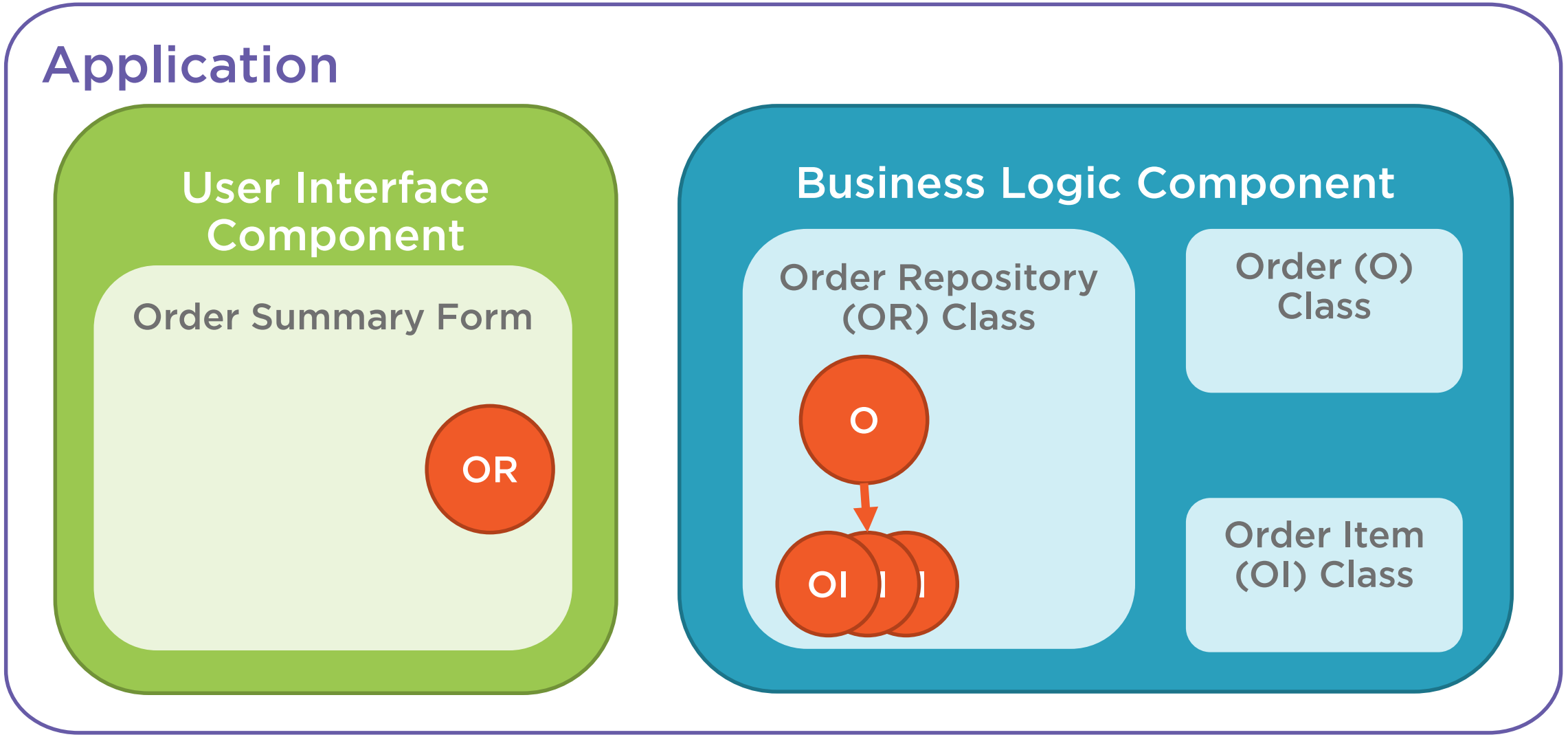
Working Together



Working Together



Working Together



Module Outline

- Defining relationships
- Types of relationships
- Collaboration
- Composition
- Composition: references
- Composition: IDs
- Inheritance

Defining Relationships

Customer	Product	Order	Order Item
•Name •Email address •Home address •Work address •Validate()	•Product name •Description •Current price •Validate()	•Customer •Order date •Shipping addr. •Order items •Validate()	•Product •Quantity •Purchase price •Validate()

Customer Repository	Product Repository	Order Repository	Address
•Retrieve() •Save()	•Retrieve() •Save()	•Retrieve() •Save()	•Street line 1 + 2 •City •State/Province •Postal Code •Country •Address type •Validate()

Defining Relationships

Product Repository

Product

Order Repository

Order

Order Item

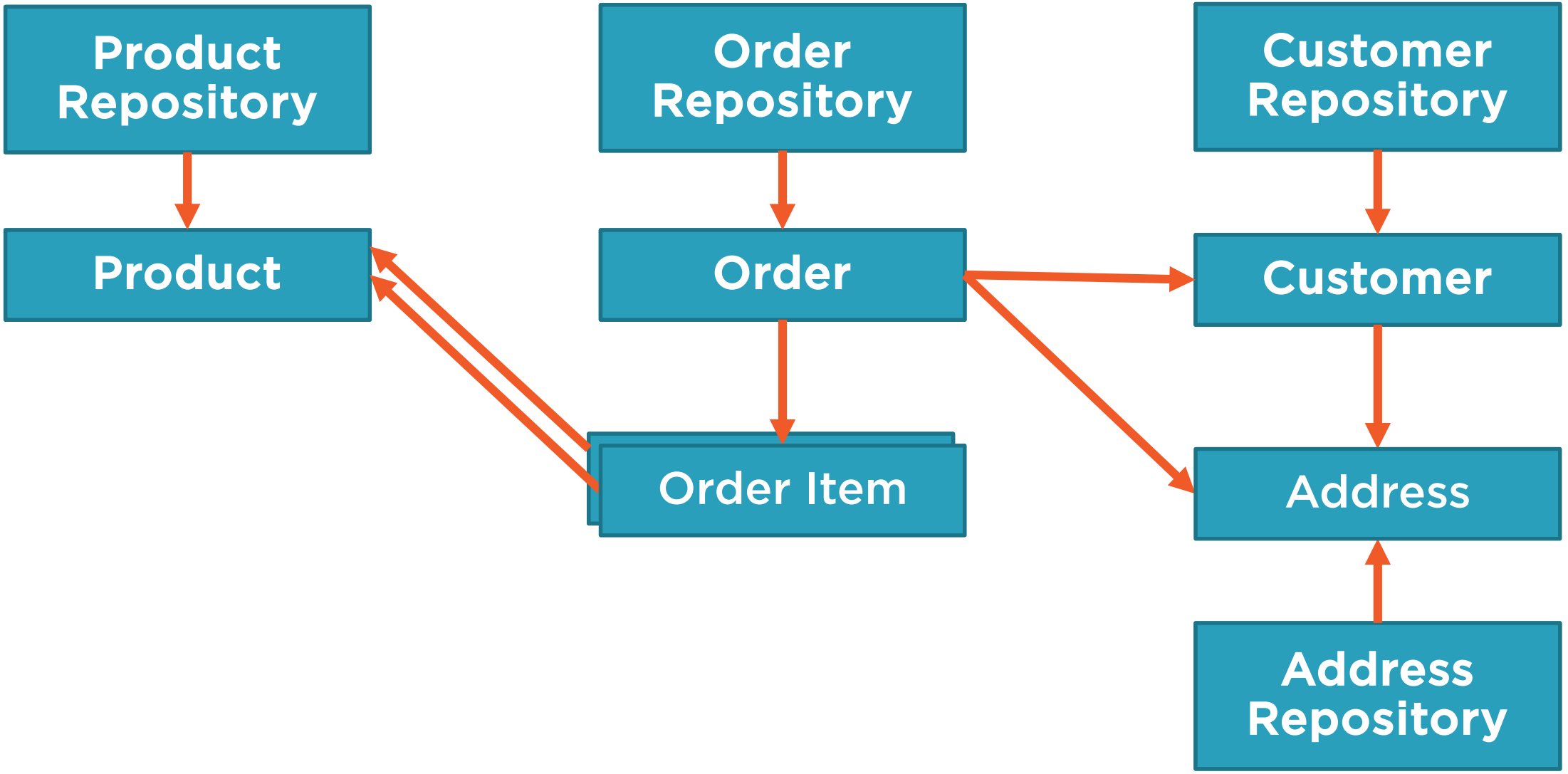
Customer Repository

Customer

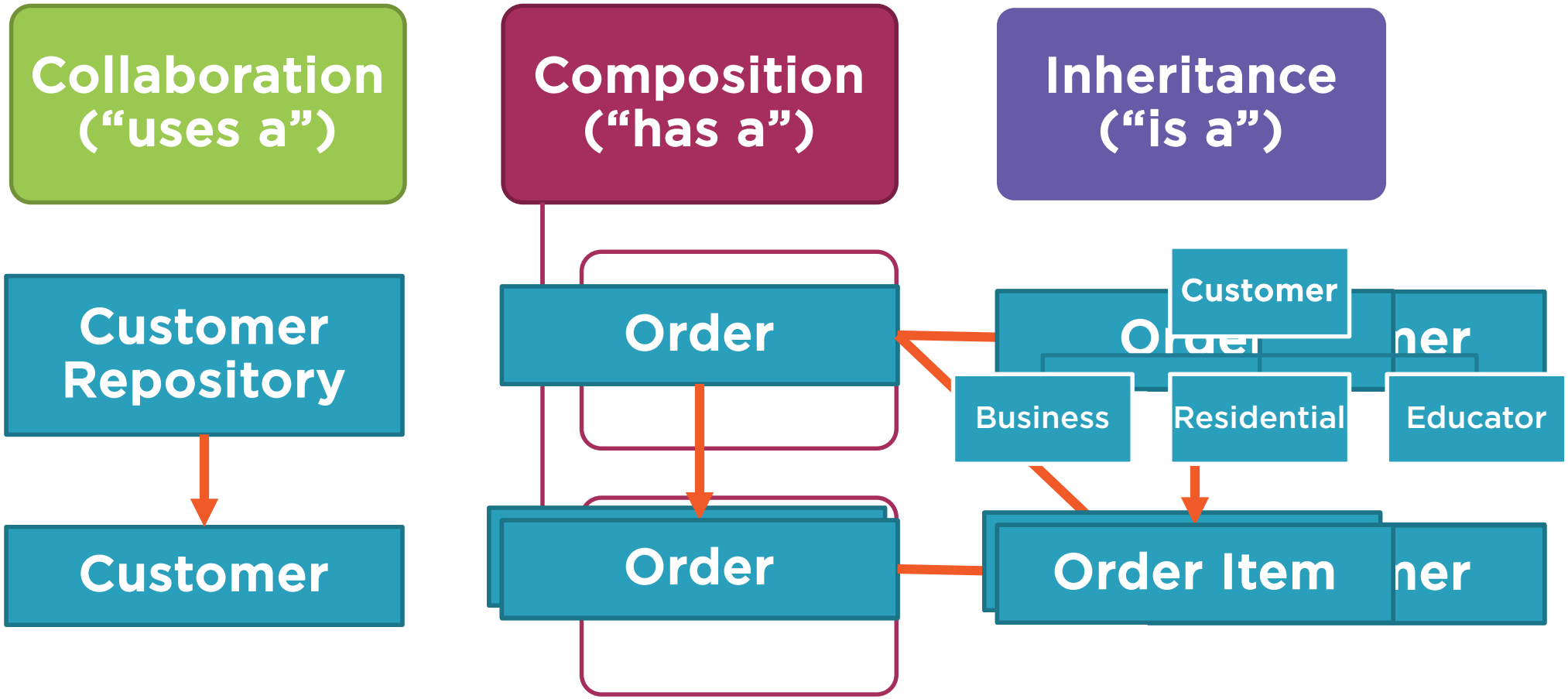
Address

Address Repository

Defining Relationships

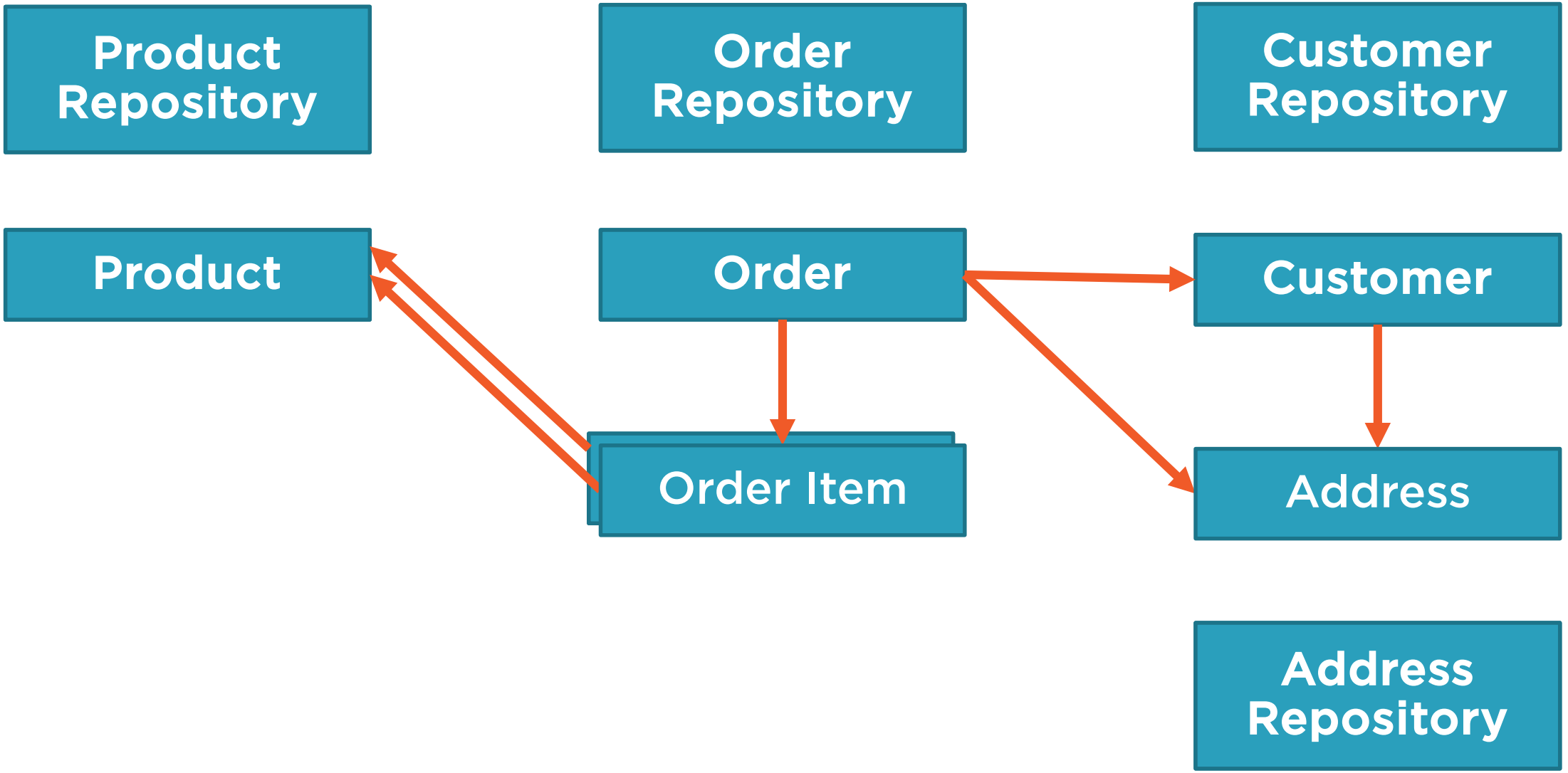


Types of Relationships



Collaboration ("uses a")

Composition ("has a")



Composition ("has a")

Product

Order

- Customer
- Order date
- Shipping addr.
- Order items
- Validate()

Order Item

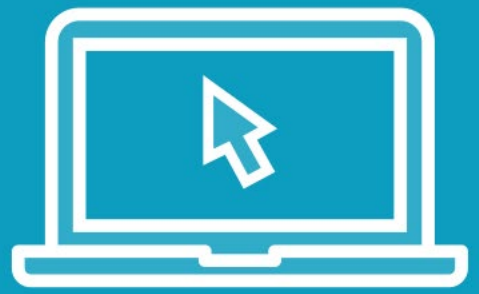
- Product
- Quantity
- Purchase price
- Validate()

Customer

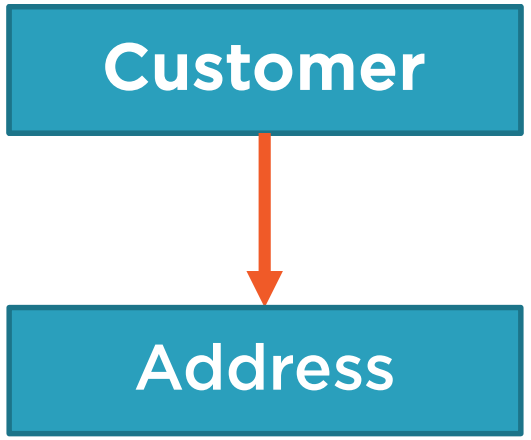
- Name
- Email address
- Home address
- Work address
- Validate()

Address

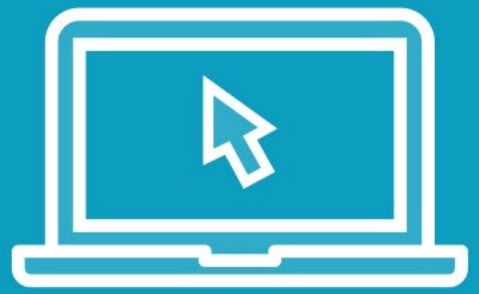
Demo



Implementing a composition relationship

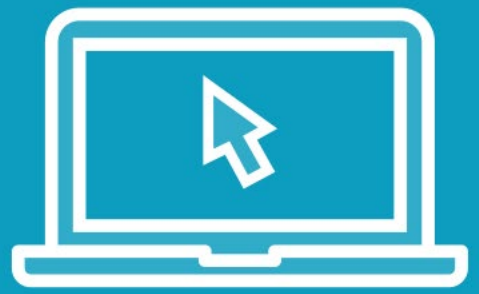


Demo



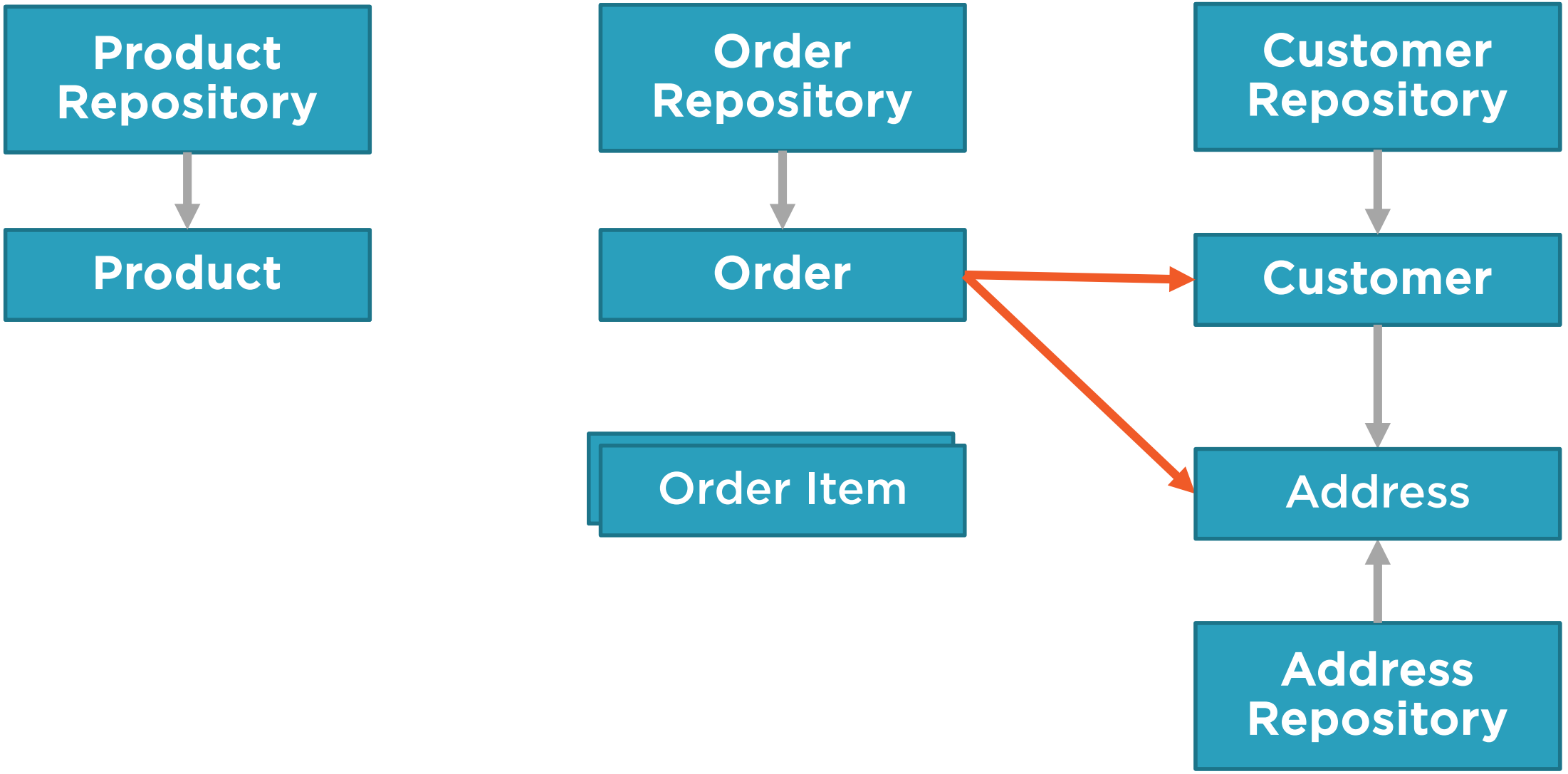
Populating the referenced object

Demo



Testing a composition relationship

Relationships



Advantages of Using IDs

Reduces coupling

Increases efficiency

Object Relationships

Collaboration
("uses a")

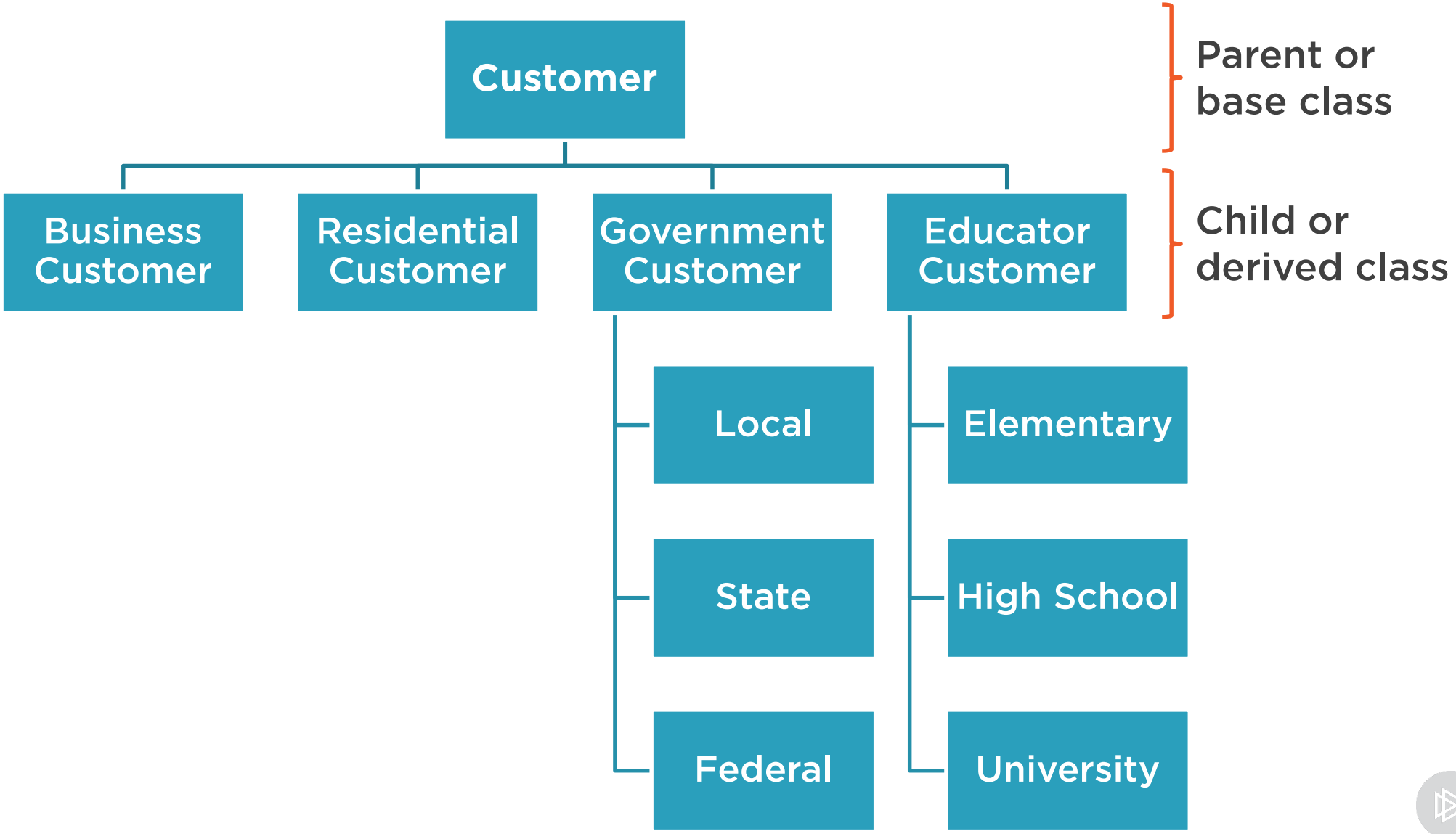
Composition
("has a")

Inheritance
("is a")

“The new system must manage business, residential, government, and educator types of customers.”

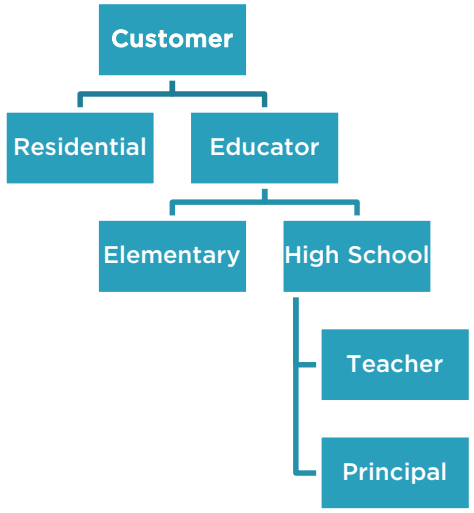
From the requirements

Inheritance ("is a")



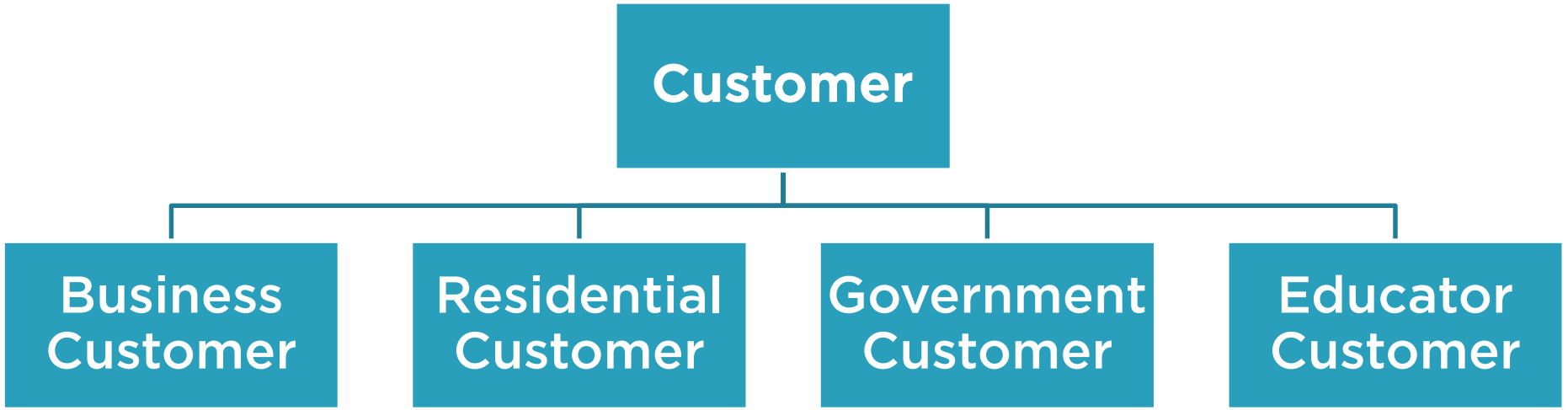
Inheritance in C#

A class can only have one parent class

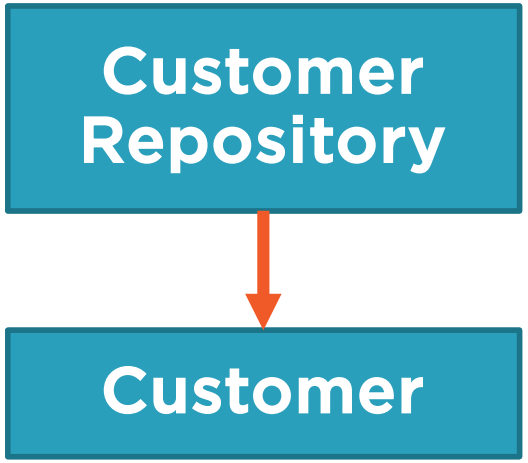


There can be any number of inheritance levels

Inheritance



Collaboration ("uses a")

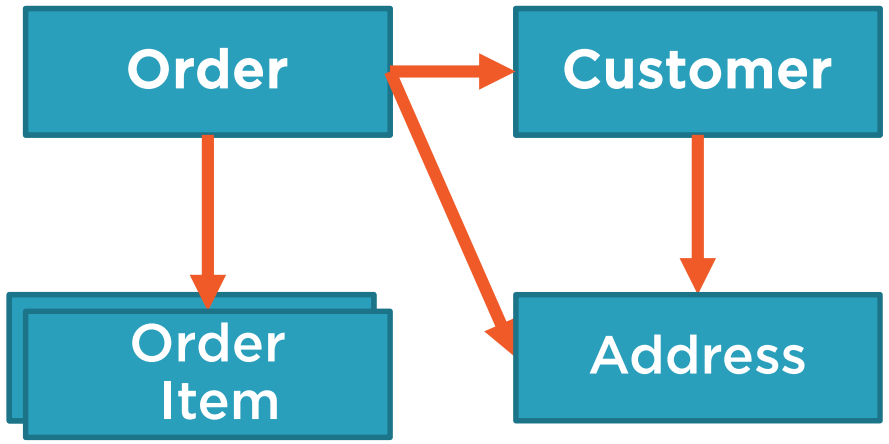


A class uses another class that is otherwise unrelated

Customer Repository "uses a" Customer instance to populate data

```
Customer customer = new Customer(customerId)
{
  EmailAddress = "fbaggins@hobbiton.me",
  FirstName = "Frodo",
  LastName = "Baggins"
};
```

Composition ("has a")



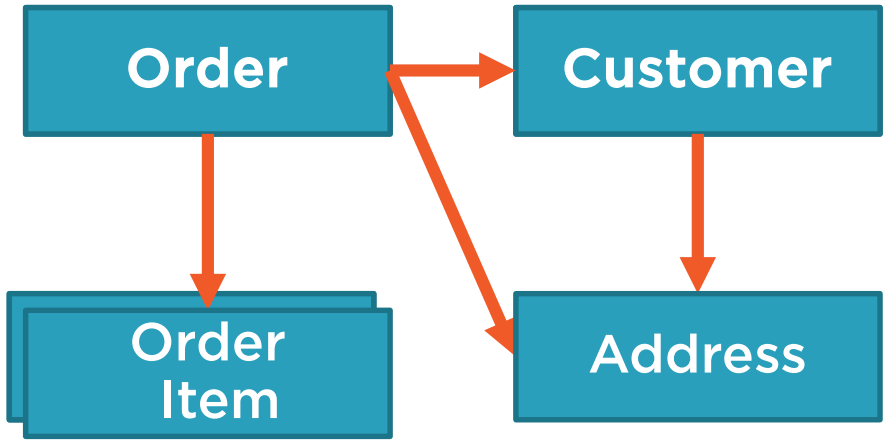
A class is made up of parts from other classes

Order "has a" customer

Order "has a" shipping address

Order "has a" set of order items

Composition ("has a")



Implement as a property reference

```
public List<OrderItem> orderItems { get; set; }
```

Or as an Id

```
public int CustomerId { get; set; }
```

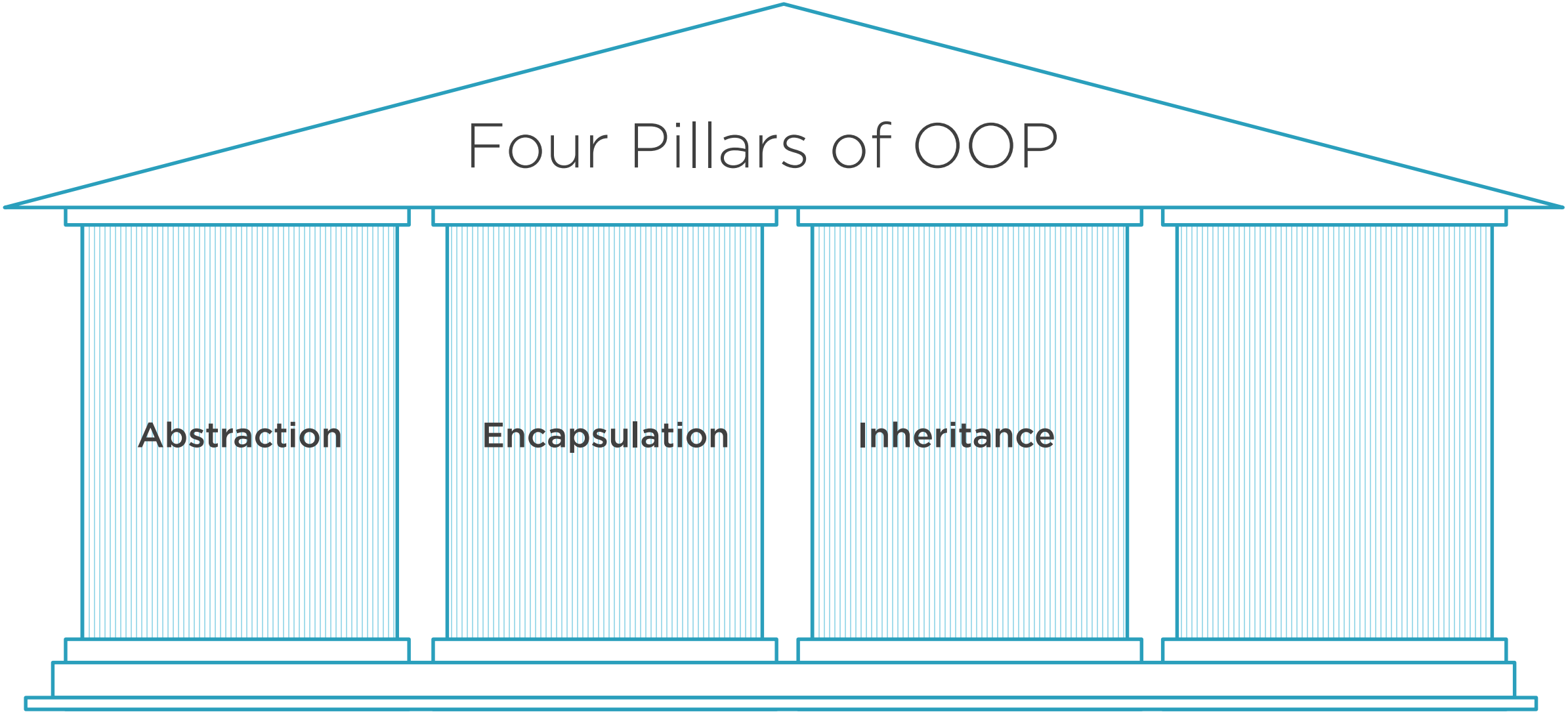
Inheritance ("is a")

Define classes that are a more specialized version of another class

Business Customer, Residential Customer, Educator Customer

Only implement an inheritance relationship if the specific class type adds unique code

Four Pillars of OOP



Object-Oriented Programming (OOP)

Identifying classes

- Represents business entities
- Defines properties (data)
- Defines methods (actions/behavior)

Separating responsibilities

- Minimizes coupling
- Maximizes cohesion
- Simplifies maintenance
- Improves testability

Establishing relationships

- Defines how objects work together to perform the operations of the application

Leveraging reuse