Application Engineering and Development INFO 5100

A case study in Order Booking

Learning Objectives

- **how to go from a problem to code**
- **Understand the difference between important and critical data**

- **What is business intelligence**
- **How local knowledge become global**
  - Aggregating small pieces of data to big ideas for running a business
- **How a well-designed object model is the brain for running a business**
- **How to use data to motivate people do certain things right**

Xerox

Xerox sales model is for their sales people to go visit with Institutional clients

Xerox was facing tough competition from Japanese companies

Xerox's sales teams were very slow and inflexible. Negotiations Required that they go back to higher manager for approval of deals. Decisions were made counter to Xerox's interests. This left their sales people frustrated.

Xerox new sales strategy

Build an app to empower the sales teams to make pricing decisions in the field based on the customer circumstances.

Xerox came up with range pricing

Each product will have 3 prices: high, low, and target

The sales person is supposed to hit the target on each item on the deal but allowed to vary on some items as long as the total is greater than the sum of the target.

But there is a problem, a big one

Screen representation of user tasks

Login Screen represents Login Task

Xerox Sales Console

User Login

User:

Password:

Role:

login
>>

User Screens (contd.)

Xerox Sales Console

John smith

Activity: Manage Customers

Customer name

Serve customer >>

Review sales order history >>

Review sales commission >>

logout

User Screens (contd.)

Xerox Sales Console

John smith

Customer Information summary	Person Contact Information

Activity: Serve Customer

View customer history

Book customer order >>

Check order status >>

Browse product catalog >>

Xerox Sales Console

John smith

Customer Information summary

Person Contact Information

Activity: Book Customer Order

My commission

Supplier

Product Id	Product Name	Floor Price	Ceiling Price	Target Price	Add
					>>
					>>

Order Items

Product Id	Product Name	Actual Price

Cancel Order >>

Submit Order >>

Xerox Sales Console

Sales person's name: John smith

Customer Information summary

Person Contact Information

Activity: Browse Product Catalog

Product Description

Find >>

Printing Catalog

Product Id	Product Name	Target Price	Description
			View Detail >>
			View Detail >>

Done >>

Xerox Sales Console

Sales person's name: John smith

Customer Information summary

Person Contact Information

Activity: Check Order Status

Time frame

Begin time | End time | Find >>

Order List

Order Id	Date started	Date completed	Status	Action
				View
				View

Done >>

Sales Process Use Case

1. Manage Customers

1.a Review sales order history

1.b Review sales commission

:

1.c Serve customer

1.c.a Browse product catalog

1.c.b Check order status

1.c.c Book customer order

- 1.c.c.a Submit customer order
- 1.c.c.b Cancel customer order
- 1.c.c.c Save customer order

Sales Process Use Case

Sales Person

1. Manage Customers

1.a Review sales order history

1.b Review sales commission

:

Customer name

1.c Serve customer

Product Catalog

1.c.a Browse product catalog

1.c.b Check order status

Customer

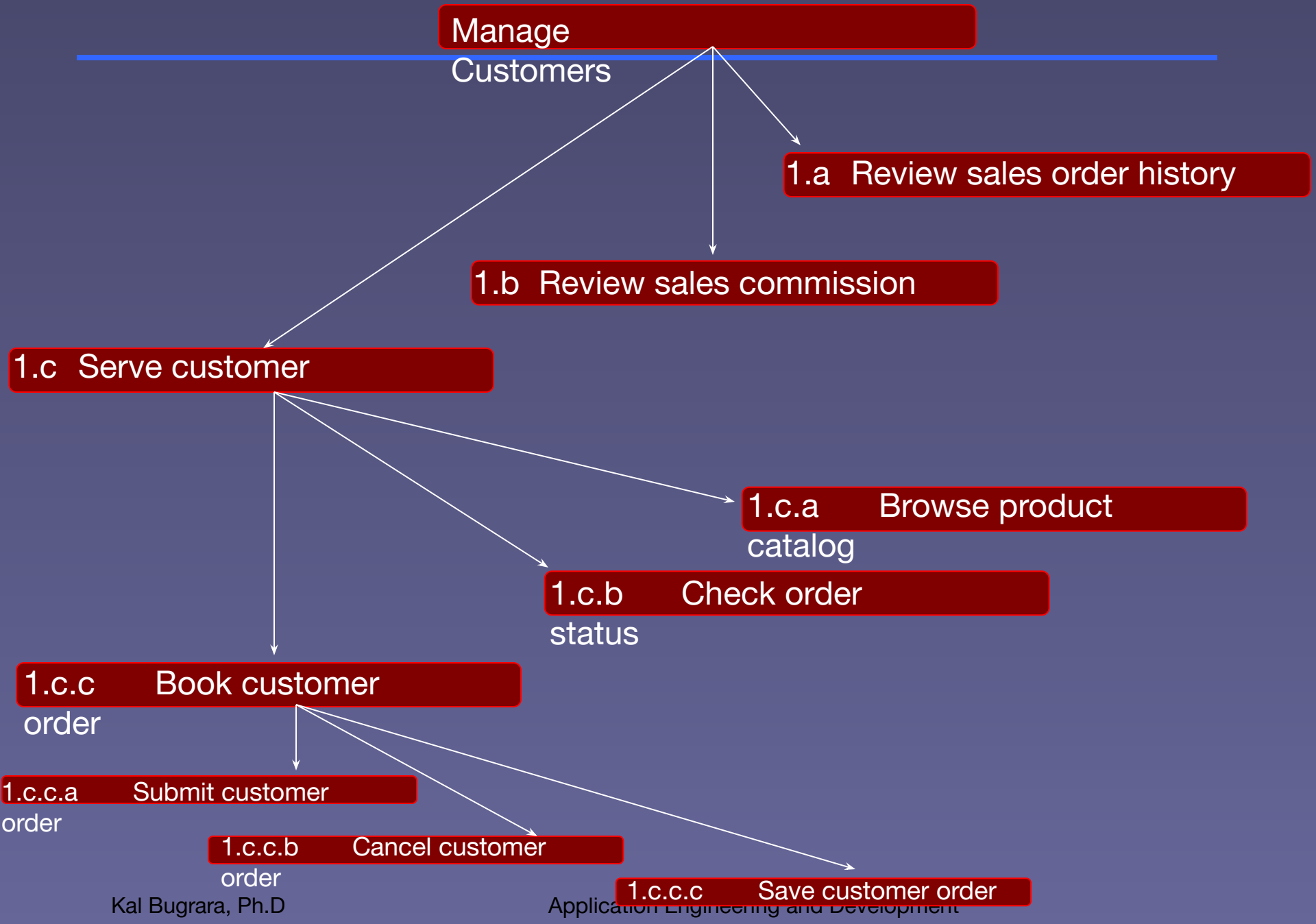
1.c.c Book customer order

1.c.c.a Submit customer order

1.c.c.b Cancel customer order

1.c.c.c Save customer order

Navigation Sequence



Implicit in the use case above is the fact that there is an authentication step that

must be completed before the user is allowed to use the system

0. Login/validate the user

0.1 Ask the security service if user has the right to use the system

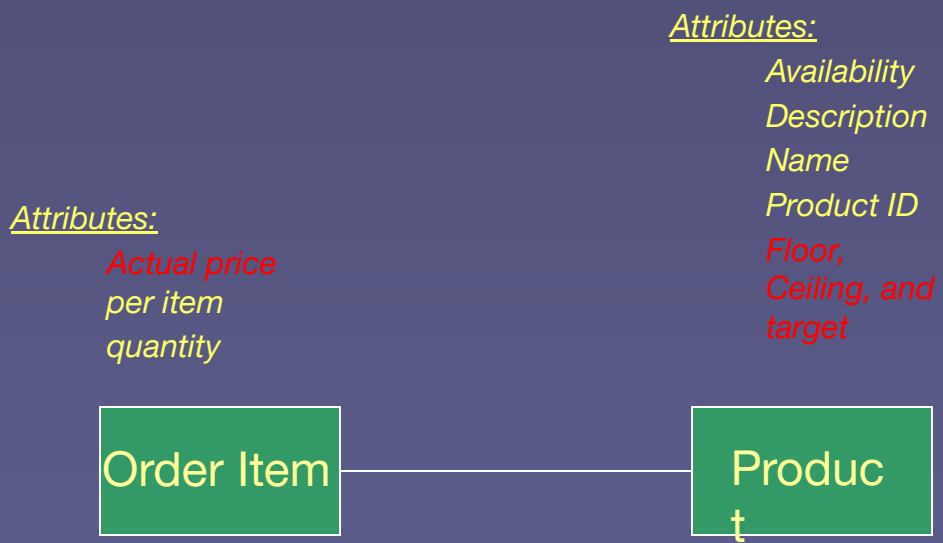
0.2 If user is valid then continue with next step

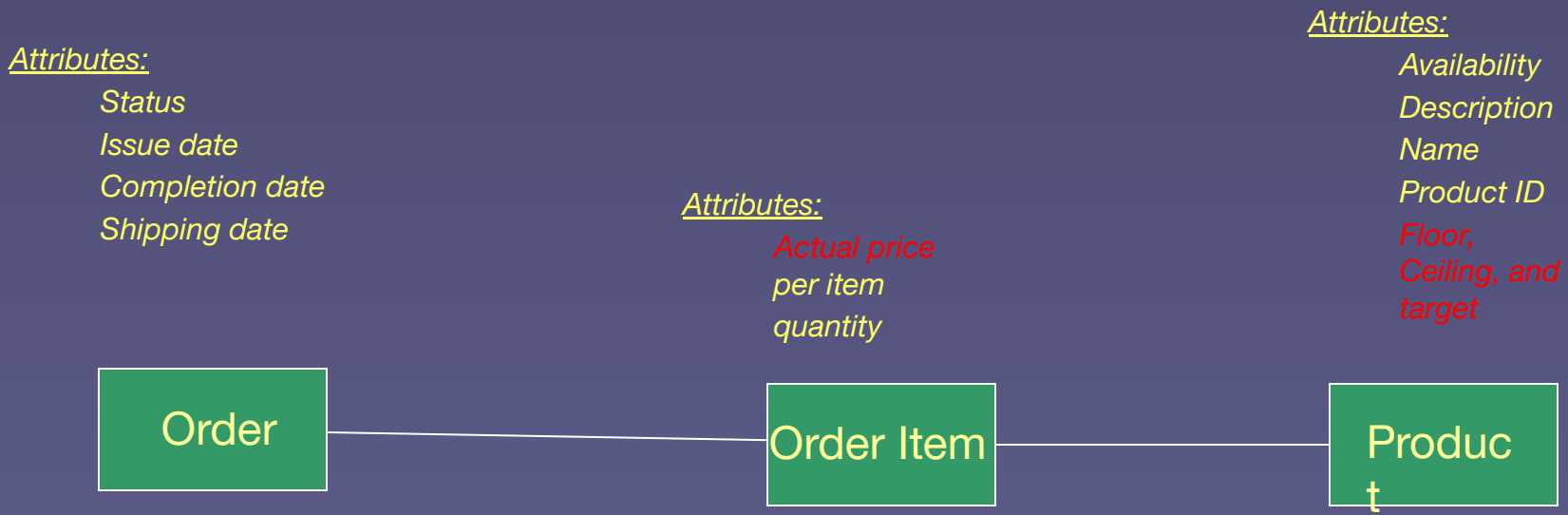
1. Manage Customers

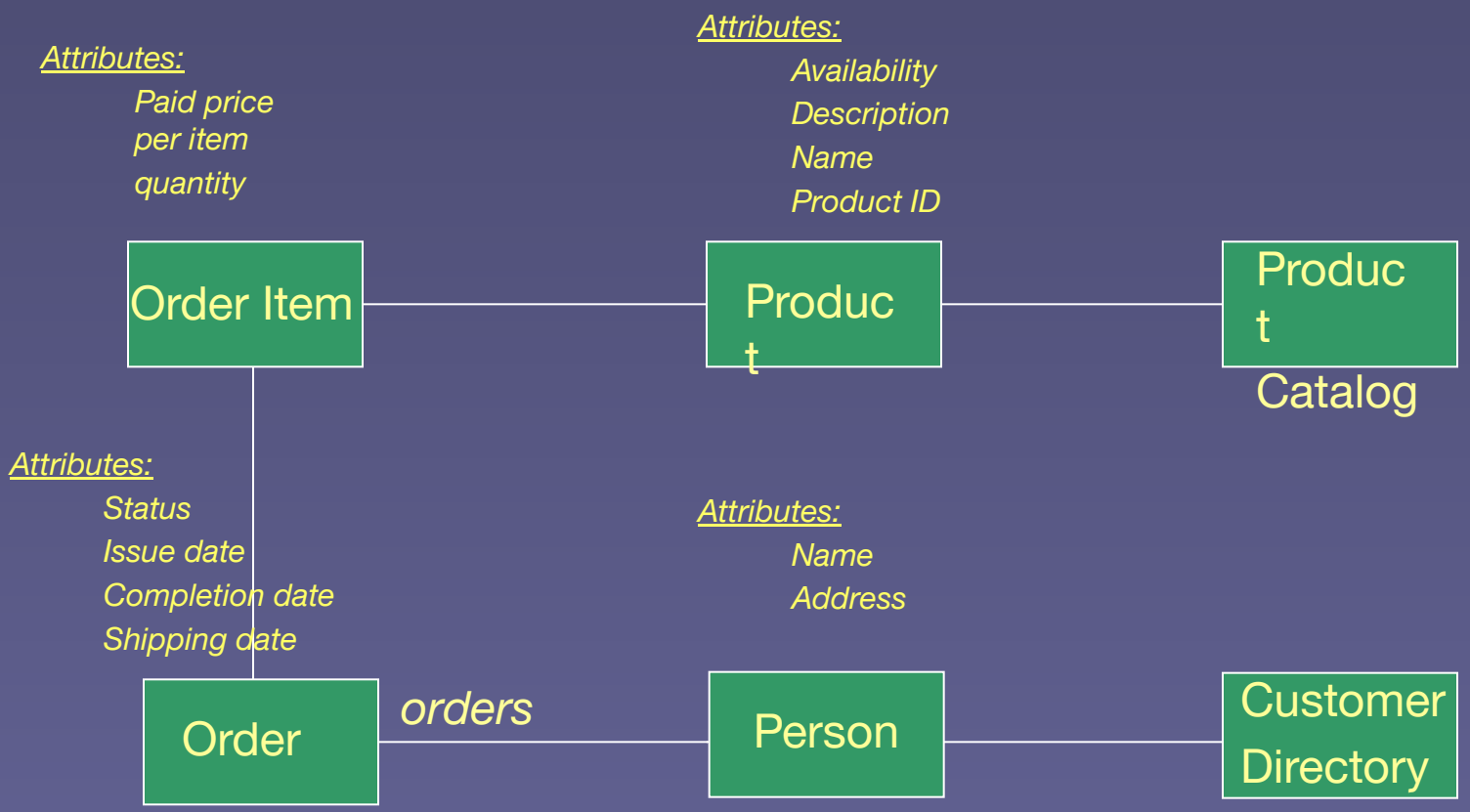
1.a Review sales order history

1.b Review sales commission

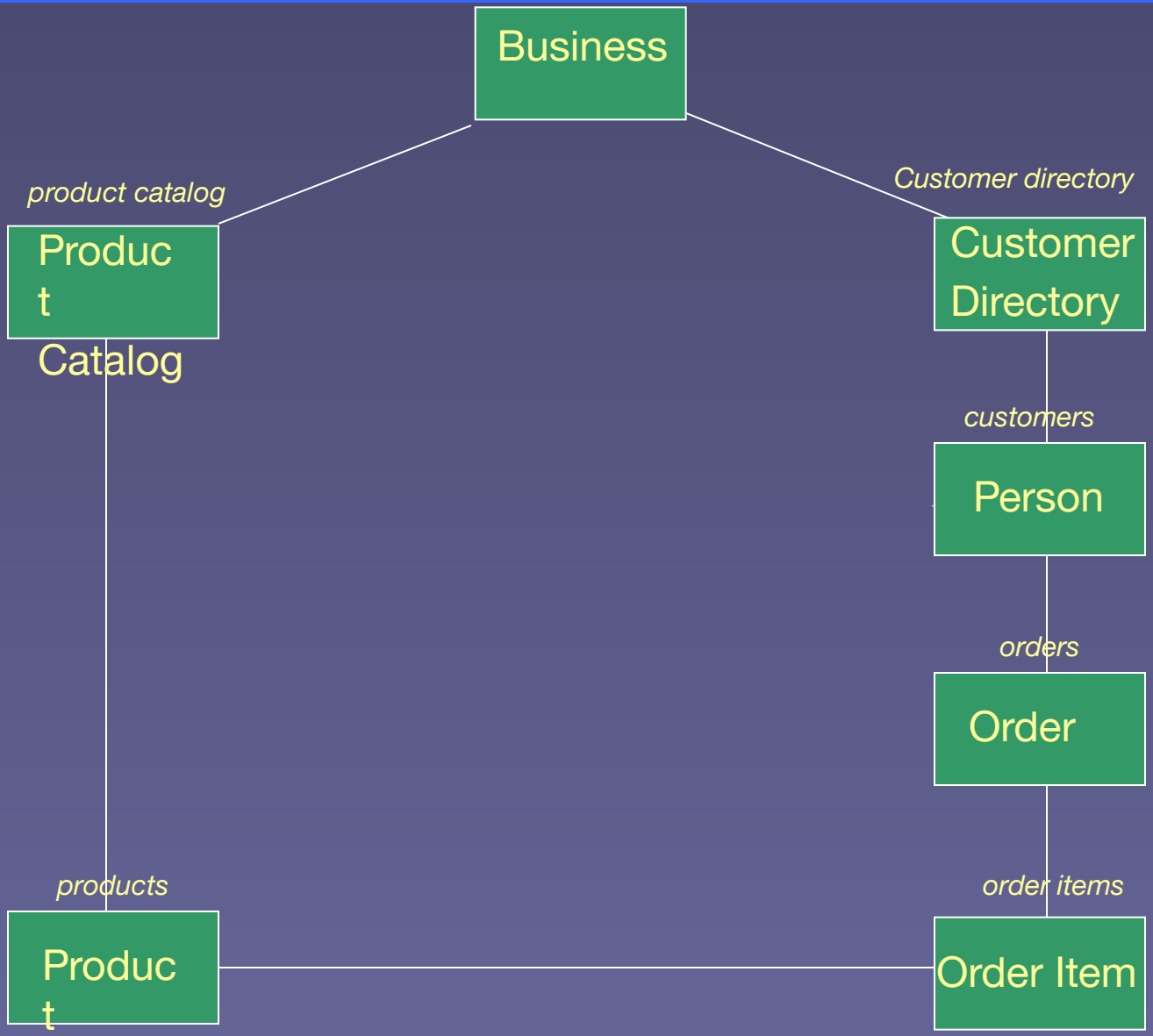
:


Attributes:
Actual price
per item
quantity
Attributes:
Availability
Description
Name
Product ID
Floor,
Ceiling, and
target
Order Item
Produc
t


Attributes:
Status
Issue date
Completion date
Shipping date
Attributes:
Actual price per item
quantity
Attributes:
Availability
Description
Name
Product ID
Floor, Ceiling, and target
Order
Order Item
Produc t

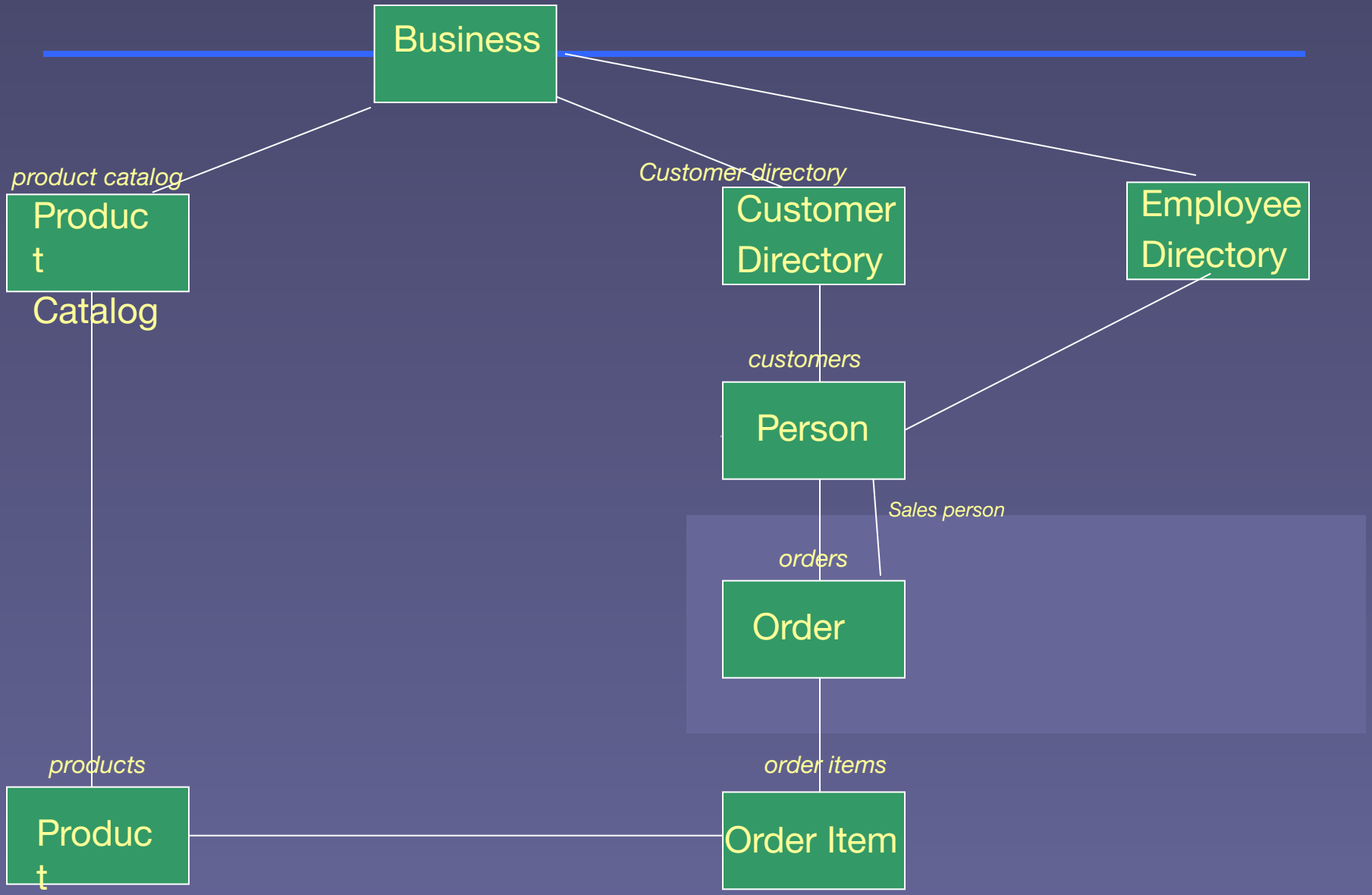

Attributes:
Paid price per item
quantity
Attributes:
Availability
Description
Name
Product ID
Order Item
Produc t
Produc t Catalog
Attributes:
Status
Issue date
Completion date
Shipping date
Attributes:
Name
Address
Order
orders
Person
Customer Directory

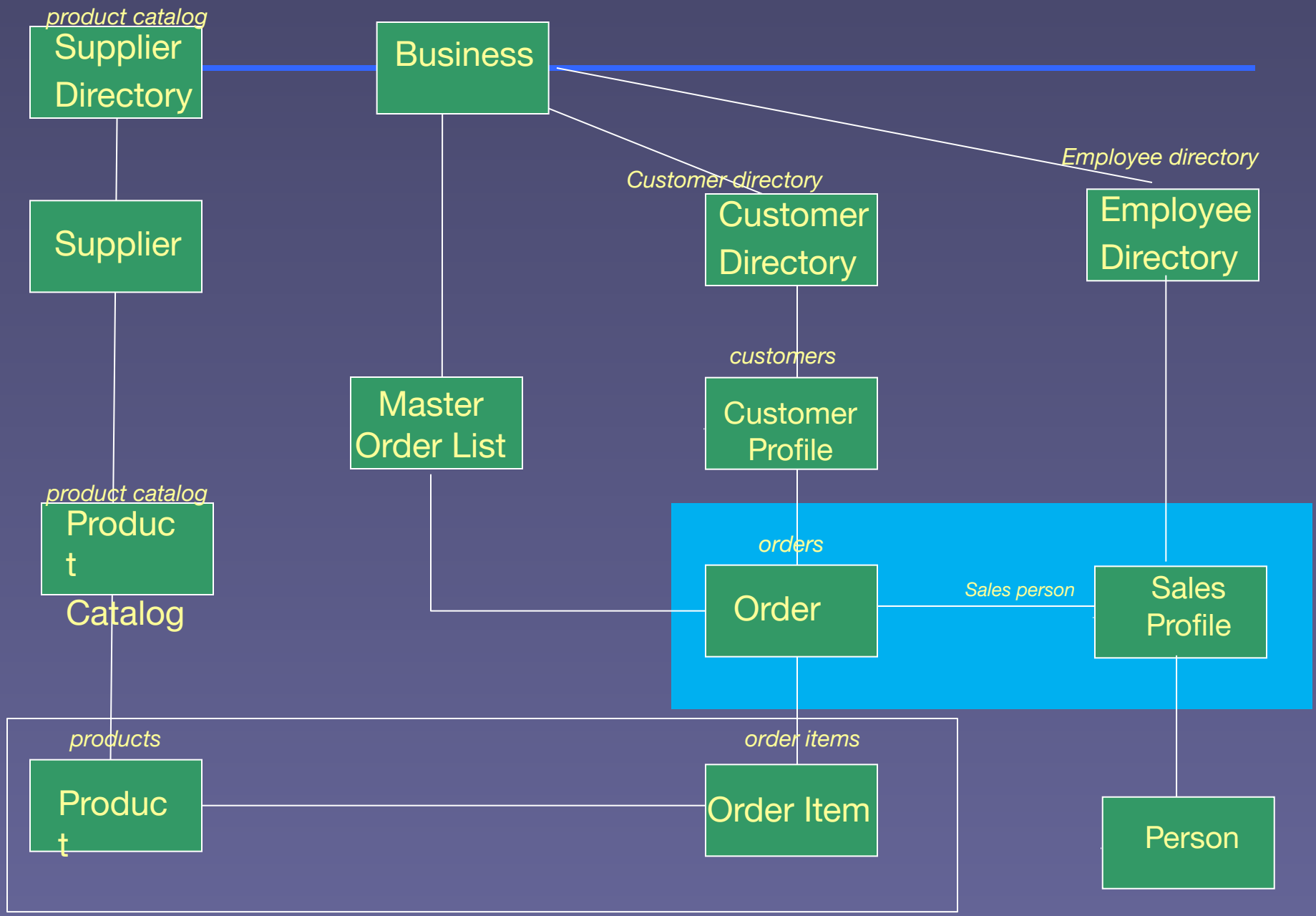
The complete business hierarchy



Sales Model

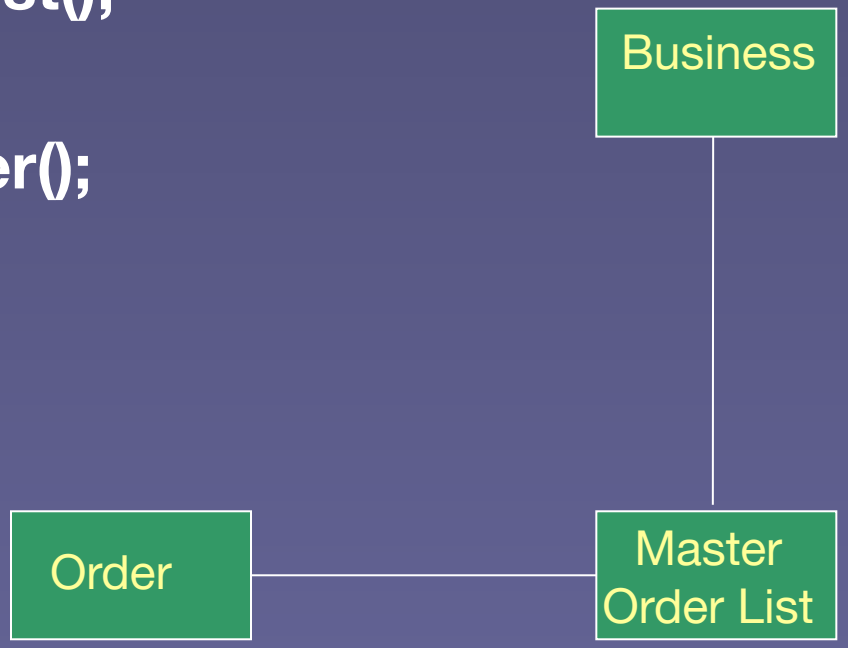
Sales person mediates the sale


Business
product catalog
Product Catalog
Customer directory
Customer Directory
Employee Directory
customers
Person
Sales person
orders
Order
products
Product
order items
Order Item


product catalog
Supplier Directory
Business
Employee directory
Customer directory
Supplier
Customer Directory
Employee Directory
customers
Master Order List
Customer Profile
product catalog
Produc t Catalog
orders
Order
Sales person
Sales Profile
products
order items
Produc t
Order Item
Person

How to create an order?

- **MasterOrderList class creates orders and keep them in an arraylist**
- **MasterOrderList mol = business.getMasterOrderList();**
- **Order order = mol.newOrder();**



How to create an order item

- **Scenario: The user wants to select a product and add it to the order:**

- **OrderItem oi = order.newOrderItem();**
- **oi.setProduct(selectedproduct);**
- **oi.setQuantity(q);**

- **Or**

- **OrderItem oi = order.newOrderItem(selectedproduct);**

How to list the order items

- ArrayList items = order.getOrderItems();
- For (Item I : items)
- {
- :
- i.getProduct() or i.getItemPrice() or i.getQuantity()
- :
- }

How to list all orders

- MasterOrderList mol = business.getMasterOrderList();

- For (Order selectedoder : mol.listoforders)
- {
- <show selectedorder info detail here>
- :
- }

How to calculate total order?

Define an operation called getOrderItemTotal() on the Order Item

Order Item

Define the getOrderTotal() operation on the class Order

Order

How to calculate order item total?

Order Item

Define the following operations on Order Item

getItemQuantity()

getPaidPrice()

Then

getOrderItem():

```
    return getItemQuantity()*getPaidPrice()
```

How to calculate order total?

Order

On the Order define

getOrderTotal():

Sum = 0

For each orderitem in the list of orderitems associated with the Order do the following step until done

Sum = sum + orderitem.getOrderItemTotal();

Return sum;

How to calculate total revenue by customer?

Customer

On the Customer class define

getTotalRevenues()

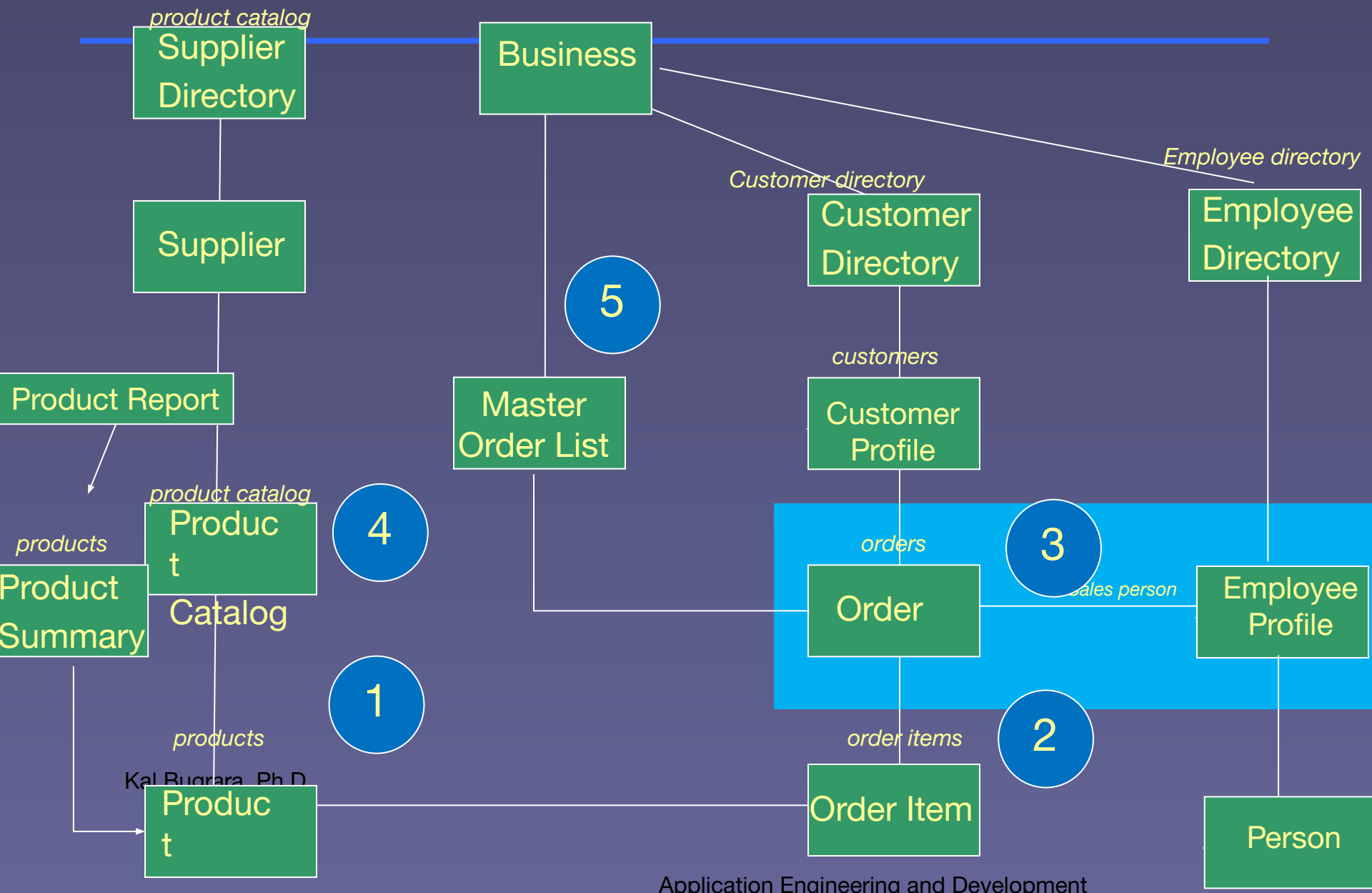
Sum = 0

For each order in the list of orders associated with the orderlist do the following step until done

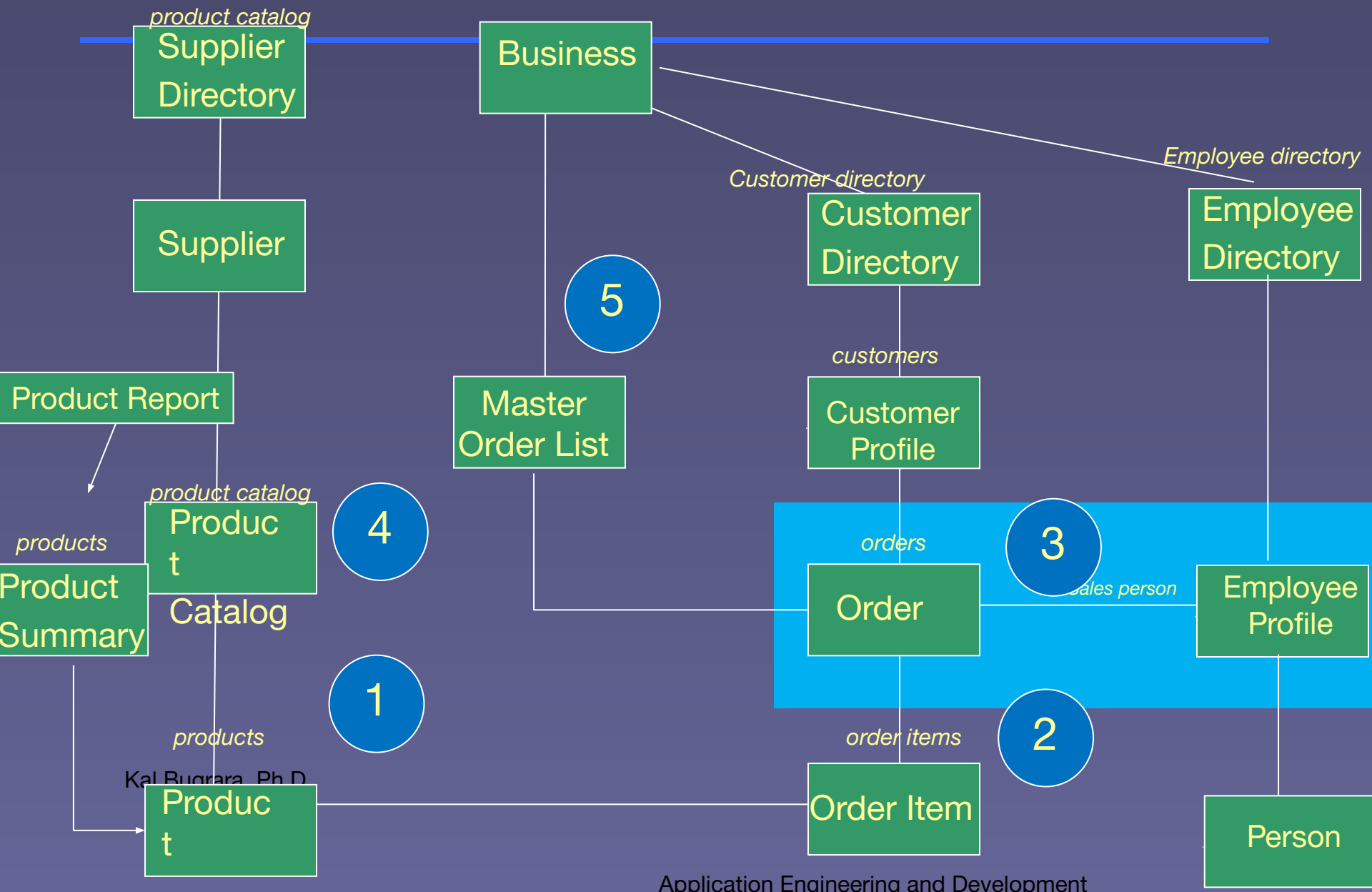
Sum = sum + order.getOrderTotal();

Return sum;

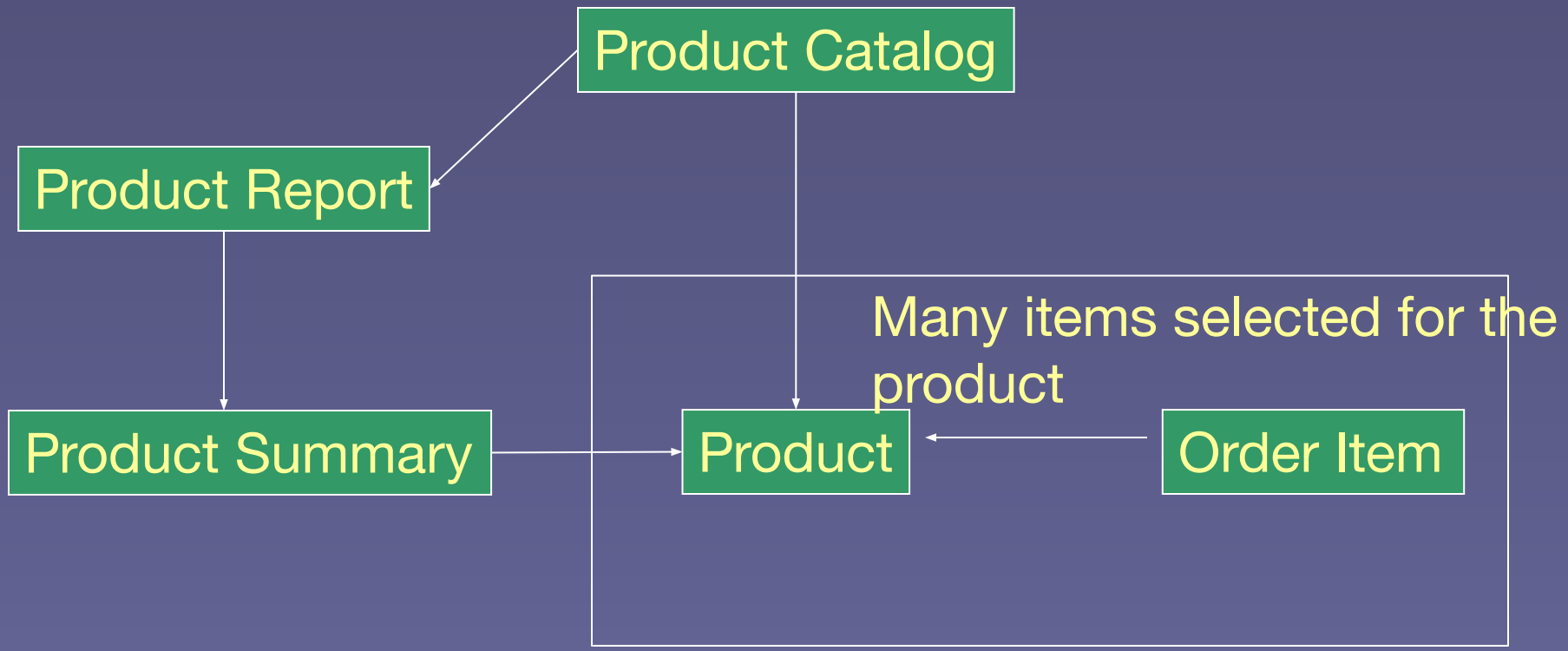
Coding Steps



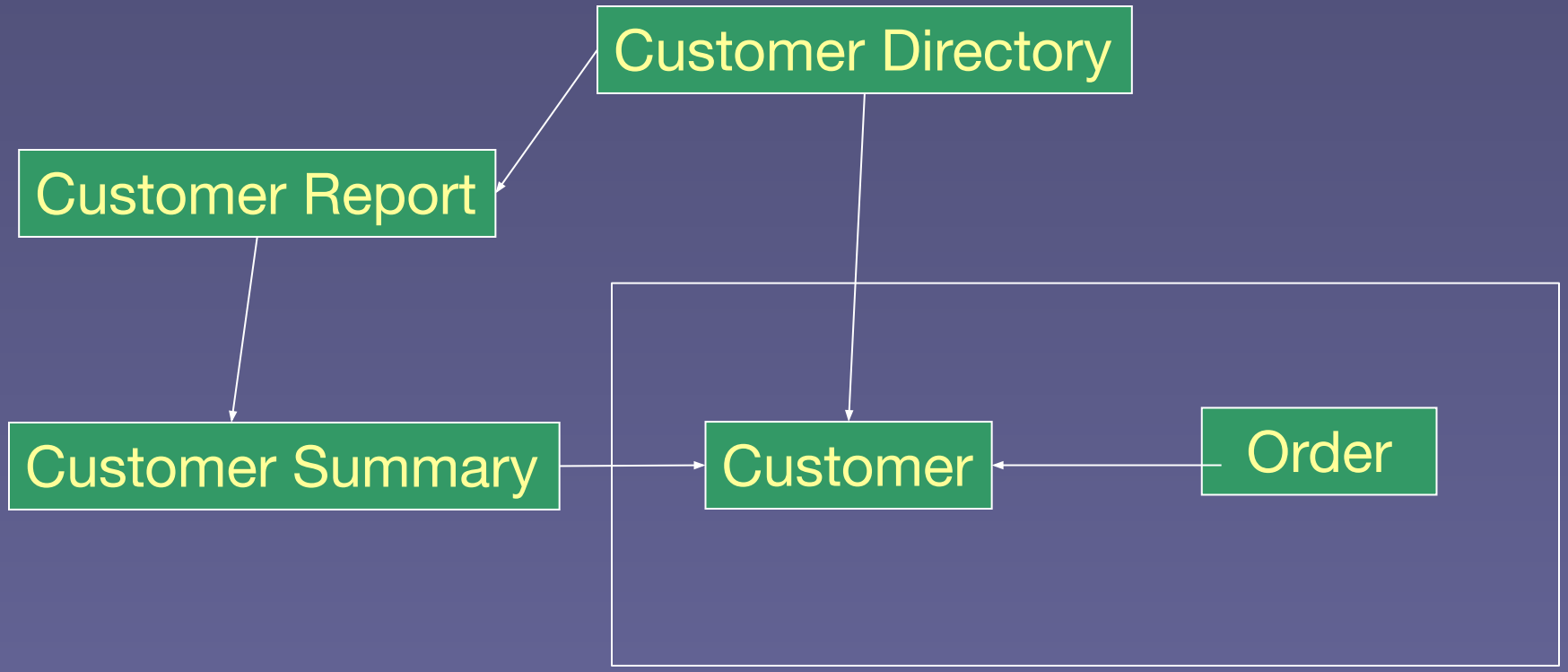
Coding Steps



Product Performance

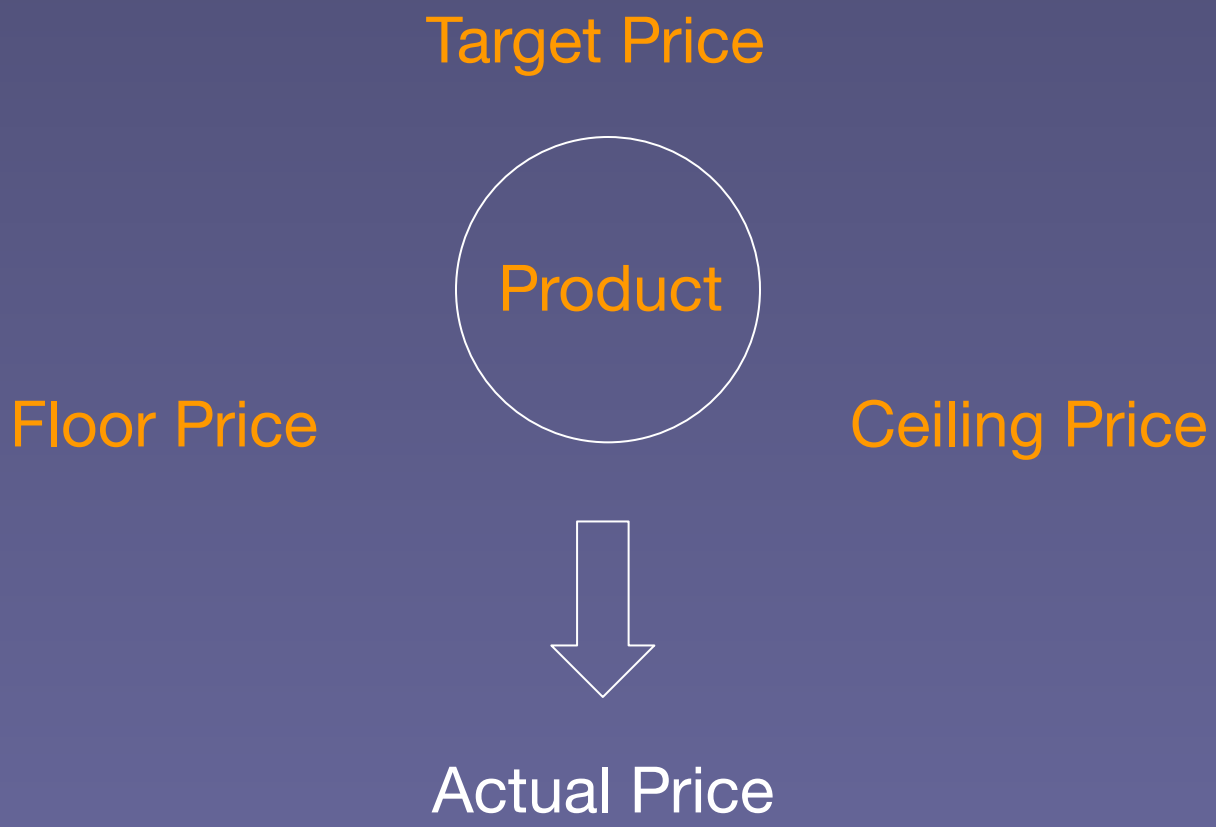


Customer Performance



Range Pricing: Product

What matters is the price for the total package:
Some products in the sales order are sold at lower price and others are at higher price.



Key business intelligence decisions

Product

Ceiling Price

Most Actual Prices

Are above target

Adjust target higher

Target Price

Floor Price

Key business intelligence decisions

Product

Ceiling Price

Target Price

Most Actual Prices

Are below target
Adjust target lower

Floor Price

Key business intelligence decisions

The top 10 profitable customers [can you sell them more]

The top 10 profitable products [get rid of unprofitable products]

The top 3 sales persons [Reward best sales people]

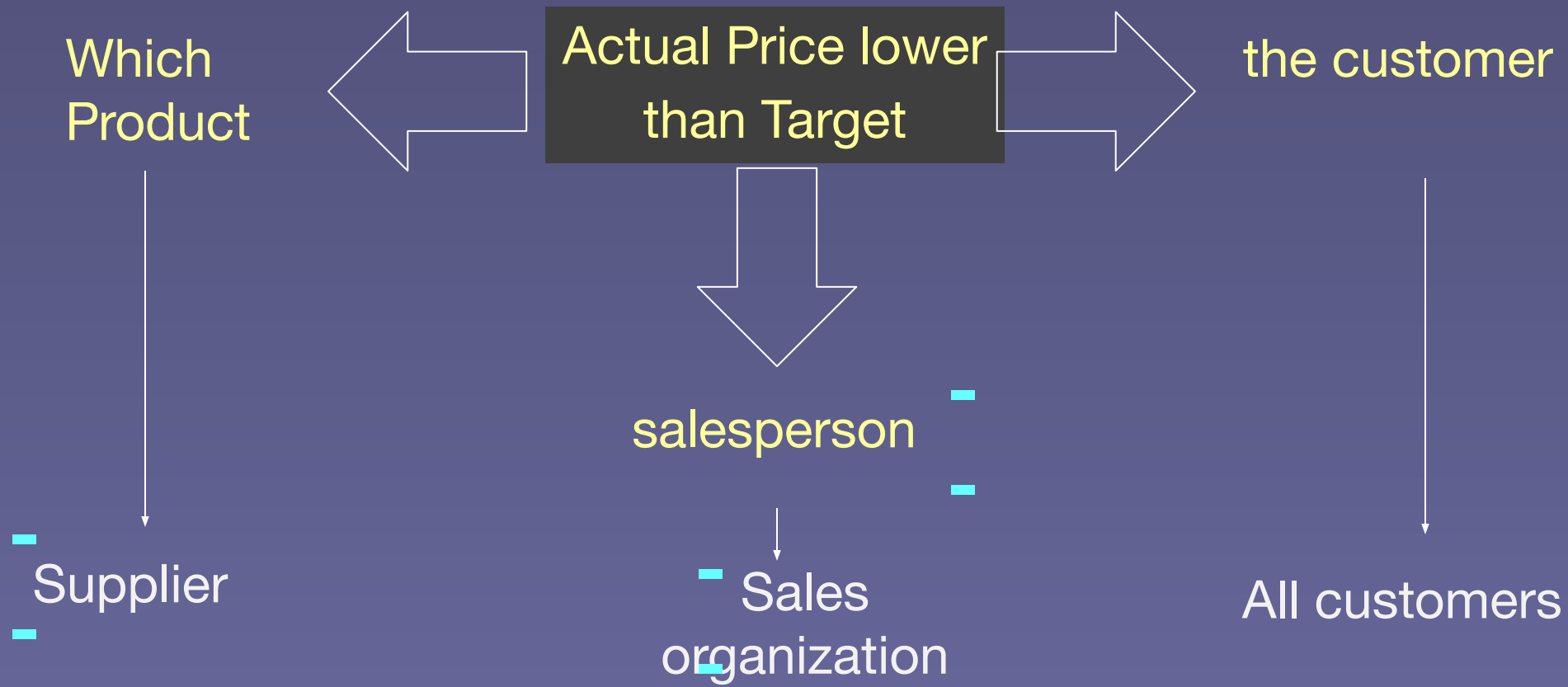
And most challenging: Are we underpricing our products?

Can we charge more

Range Pricing: Product

Product performs well with positive impact in different direction

Product performs with negative impact in different direction



Attributes:

Target
Floor
Ceiling
Availability
Description
Name
Product ID

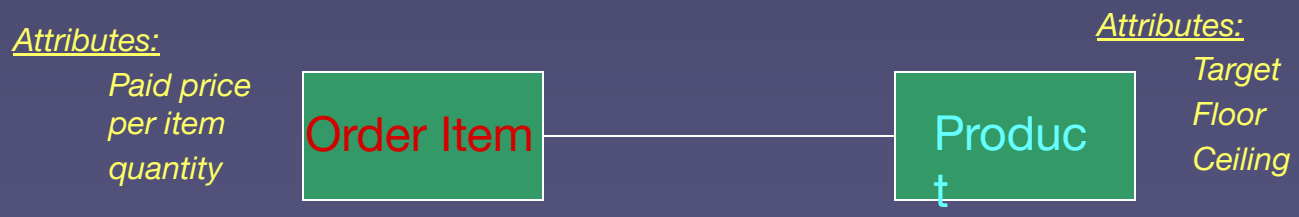

```
public class Product {

    private int floorPrice;
    private int ceilingPrice;
    private int targetPrice;
    ArrayList<OrderItem> orderitems;

    public Product(int fp, int cp, int tp) {
        floorPrice = fp;
        ceilingPrice = cp;
        targetPrice = tp;

    }
        public Product updateProduct(int fp, int cp, int tp) {
        floorPrice = fp;
        ceilingPrice = cp;
        targetPrice = tp;
        return this; //returns itself

    }
```

Attributes:

- *Target*
- *Floor*
- *Ceiling*
- *Availability*
- *Description*
- *Name*
- *Product ID*


```
public class ProductCatalog {
    String type;
    ArrayList<Product> products;
    public ProductCatalog(String n){
        type = n;
        products = new ArrayList();
    }
    public Product newProduct(int fp, int cp, int tp){
        Product p = new Product(fp, cp, tp);
        products.add(p);
        return p;
    }

}
```

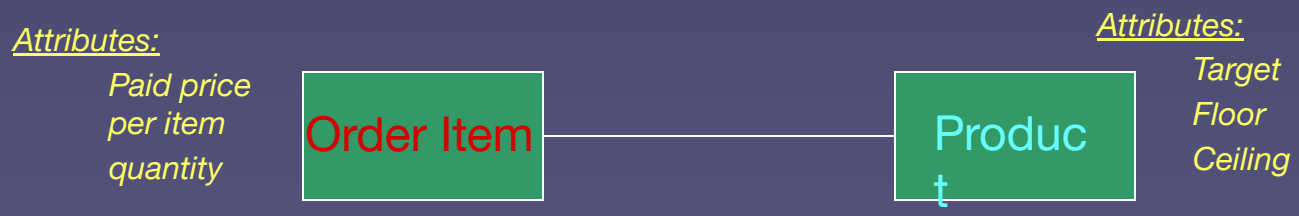
Most basic building block




```
public class OrderItem {
    Product selectedproduct;
    int ActualPrice;
    int quantity;

    public OrderItem(Product p, int q){
        selectedproduct = p;
        quantity = q;
    }
}
```

```
public class Product {

    private int floorPrice;
    private int ceilingPrice;
    private int targetPrice;
    ArrayList<OrderItem> orderitems;

    public Product(int fp, int cp, int tp) {
        floorPrice = fp;
        ceilingPrice = cp;
        targetPrice = tp;
    }
```

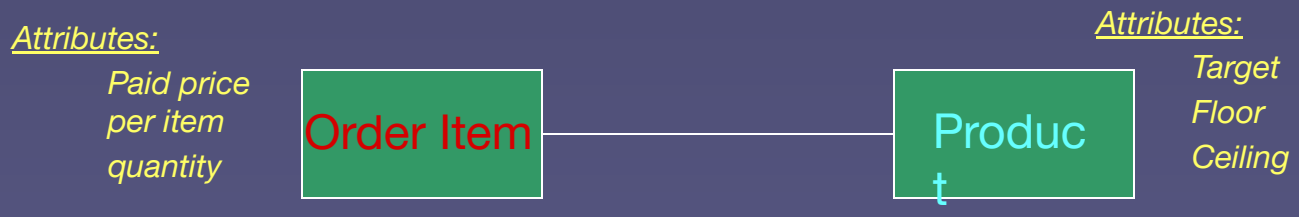
Linking demand to supply




```
public class OrderItem {
    Product selectedproduct;
    int ActualPrice;
    int quantity;

    public OrderItem(Product p, int q){
        selectedproduct = p;
        quantity = q;
    }
}
```

```
public class Product {

    private int floorPrice;
    private int ceilingPrice;
    private int targetPrice;
    ArrayList<OrderItem> orderitems;

    public Product(int fp, int cp, int tp) {
        floorPrice = fp;
        ceilingPrice = cp;
        targetPrice = tp;
    }
```

the most basic intelligence we can have so far



Customer agreed to an actual price

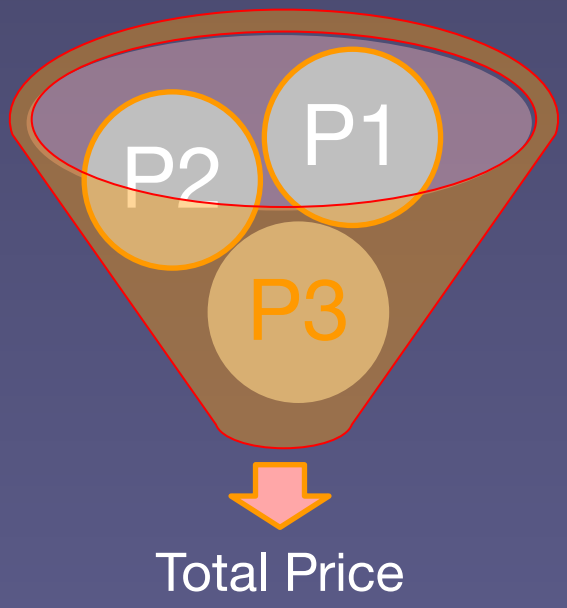
Agreed to price is

High (positive) – seller has more negotiating power

Lower (negative), customer has more negotiating power

Target (neutral) meeting item objective – equal power

Range Pricing: Solution Package



Seller wins if

Total target Price is lower than ActualPrice(P1) + ActualPrice(P2) + ActualPrice(P3) is less than the sum of the targets

Customer wins if

Total target Price is greater than ActualPrice(P1) + ActualPrice(P2) + ActualPrice(P3)

```java
public class OrderItem {

    Product selectedproduct;
    int actualPrice;
    int quantity;

    public OrderItem(Product p, int q) {
        selectedproduct = p;
        quantity = q;
    }

    public int getOrderItemTotal() {
        return actualPrice * quantity;
    }
    //returns positive if seller is making higher margin than target
    //returns negative if seller is making lower margin than target
    //otherwise zero meaning neutral
    public int calculatePricePerformance() {
        return actualPrice - selectedproduct.getTargetPrice();
    }
}
```

Attributes:

Status
Issue date
Completion date
Shipping date

Order

Attributes:

Paid price per item
quantity

Order Item

Attributes:

Availability
Description
Name
Product ID

Produc t

```
public class Order {

    ArrayList<OrderItem> orderitems;
    CustomerProfile customer;

    public Order(CustomerProfile cp){

        orderitems = new ArrayList();
        customer = cp;
    }

    public OrderItem newOrderItem(Product p, int q){
        OrderItem oi = new OrderItem(p, q);
        orderitems.add(oi);
        return oi;
    }
}
```

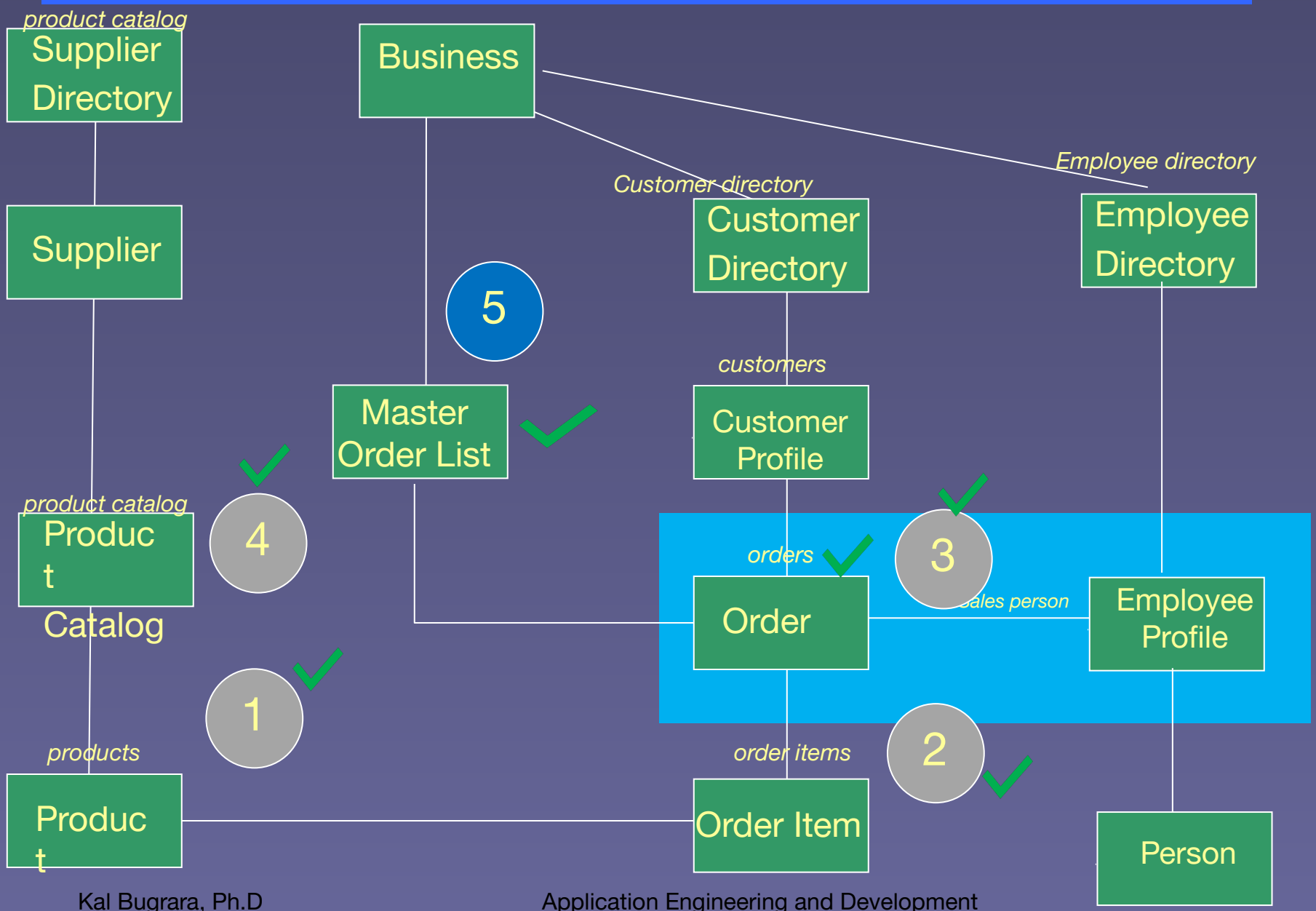
Oder expanded


```
public class Order {
    ArrayList<OrderItem> orderitems;
    CustomerProfile customer;
    public Order(CustomerProfile cp){
        orderitems = new ArrayList();
        customer = cp;
    }

    public OrderItem newOrderItem(Product p, int q){
        OrderItem oi = new OrderItem(p, q);
        orderitems.add(oi);
        return oi;
    }
    public int getOrderTotal(){
        int sum = 0;
        for(OrderItem oi: orderitems){
            sum = sum + oi.getOrderItemTotal();
        }
        return sum;
    }

        public int getOrderPricePerformance(){
        int sum = 0;
        for(OrderItem oi: orderitems){
            sum = sum + oi.calculatePricePerformance();
        }
        return sum;
    }
}
```

Order price intelligence

Item price intelligence

Next Steps


product catalog
Supplier Directory
Business
Employee directory
Customer directory
Customer Directory
Employee Directory
Supplier
5
customers
Master Order List
Customer Profile
product catalog
Produc t Catalog
4
orders
3
sales person
Order
Employee Profile
1
2
products
order items
Produc t
Order Item
Person

Top Customers

Managing the business

Product Report

Sales Person Report


Top Customers
Managing the business
Top Products
Top Sales Performers

Product Intelligence

Number of sales above target;
Number of sales below target;
Product price performance; //total profit above target
Actual sales volume;
Rank;

```java
public class OrderItem {

    Product selectedproduct;
    int actualPrice;
    int quantity;

    public OrderItem(Product p, int q) {
        selectedproduct = p;
        p.addOrderItem(this); //make sure product links back to the item
        quantity = q;
    }

    public int getOrderItemTotal() {
        return actualPrice * quantity;
    }
    //returns positive if seller is making higher margin than target
    //returns negative if seller is making lower margin than target
    //otherwise zero meaning neutral
    public int calculatePricePerformance() {
        return actualPrice - selectedproduct.getTargetPrice();
    }
    public boolean isActualAboveTarget(){
        if (actualPrice> selectedproduct.getTargetPrice()) return true;
        else return false;

}
```

```java
public class Product {
    private int floorPrice;
    private int ceilingPrice;
    private int targetPrice;
    ArrayList<OrderItem> orderitems;
    public Product(int fp, int cp, int tp) {
        floorPrice = fp;
        ceilingPrice = cp;
        targetPrice = tp;
        orderitems = new ArrayList();
    }
        public Product updateProduct(int fp, int cp, int tp) {
        floorPrice = fp;
        ceilingPrice = cp;
        targetPrice = tp;
        return this; //returns itself
    }
    public int getTargetPrice() {return targetPrice;}
    public void addOrderItem(OrderItem oi){
        orderitems.add(oi);
    }
    //revenues won or lost because of actual vs target difference
    public int getNumberOfProductSalesAboveTarget(){
        int sum = 0;
        for (OrderItem oi: orderitems){
            if(oi.isActualAboveTarget()==true) sum = sum +1;
        }
        return sum;
    }
```

```
public class Order {
    ArrayList<OrderItem> orderitems;
    CustomerProfile customer;
    public Order(CustomerProfile cp){
        orderitems = new ArrayList();
        customer = cp;
    }
    public OrderItem newOrderItem(Product p, int q){
        OrderItem oi = new OrderItem(p, q);
        orderitems.add(oi);
        return oi;
    }
    public int getOrderTotal(){
        int sum = 0;
        for(OrderItem oi: orderitems){
            sum = sum + oi.getOrderItemTotal();
        }
        return sum;
    }
        public int getOrderPricePerformance(){
        int sum = 0;
        for(OrderItem oi: orderitems){
            sum = sum + oi.calculatePricePerformance();      //positive and negative values
        }
        return sum;
    }
        public int getNumberOfOrderItemsAboveTarget(){
        int sum = 0;
        for (OrderItem oi: orderitems){
```

```
public class Order {
    ArrayList<OrderItem> orderitems;
    CustomerProfile customer;
    public Order(CustomerProfile cp){
        orderitems = new ArrayList();
        customer = cp;
    }
    public OrderItem newOrderItem(Product p, int q){...5 lines }
    public int getOrderTotal(){
        int sum = 0;
        for(OrderItem oi: orderitems){
            sum = sum + oi.getOrderItemTotal();
        }
        return sum;
    }
        public int getOrderPricePerformance(){...7 lines }
        public int getNumberOfOrderItemsAboveTarget(){
        int sum = 0;
        for (OrderItem oi: orderitems){
            if(oi.isActualAboveTarget()==true) sum = sum +1;
        }
        return sum;
    }
}
```

Sales Model

Self-serve model where customer prepares own order