

## Adapters

- Warning: what used to be called a connector is now called an adapter.

1

## Example

- Compute the price of products

2

## Pricing Collaboration

- LineItemRole
  - provided
    - double price()
  - required
    - PricerRole pricer()
    - CustomerRole customer()
    - ItemRole item()

3

## Pricing Collaboration: business rule for computing price

```
LineItemRole:  
double price() {  
    double basicPrice = pricer().basicPrice(item);  
    double discount = pricer().  
        discount(customer(),item());  
    double unitPrice =  
        basicPrice-(discount*basicPrice);  
    return unitPrice +  
        item().additionalCharge(unitPrice);  
}
```

4

## Pricing Collaboration

- ItemRole
  - provided
    - double additionalCharge(double unitPrice)
  - required
    - ChargerRole charge()

5

## Pricing Collaboration : business rule for additional charge

```
ItemRole:  
double additionalCharge(double unitPrice) {  
    return charge().cost(unitPrice, this);  
}
```

6

## Pricing Collaboration

- CustomerRole
  - provided
  - required

7

## Pricing Collaboration

- ChargerRole
  - provided
  - required
    - double cost(double, ItemRole)

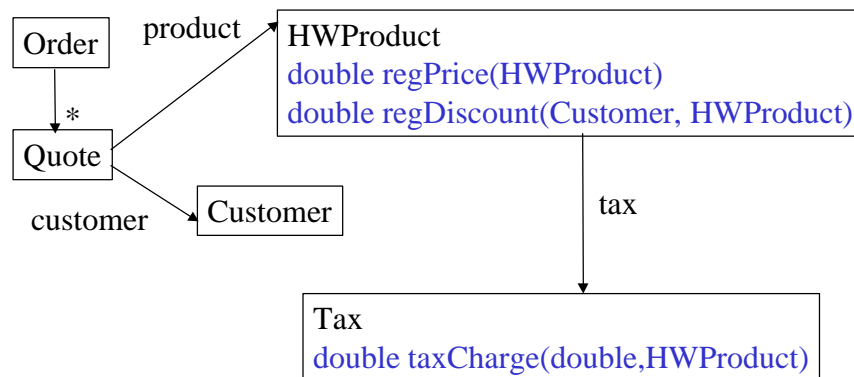
8

## Pricing Collaboration

- PricerRole
  - provided
  - required
    - double basicPrice(ItemRole)
    - double discount(CustomerRole, ItemRole)

9

## Product Application Class Model



10

## Adapter: overview

```
adapter PricingProduct adapts Product{
  adapter LineItemRole adapts Quote;
  adapter CustomerRole adapts Customer;
  adapter ItemRole adapts HWProduct;
  adapter ChargerRole adapts Tax;
  adapter PricerRole adapts HWProduct;
}
```

11

## Adapter: details

```
adapter PricingProduct adapts Product{
  adapter LineItemRole adapts Quote {
    ItemRole item(){return product();}
    CustomerRole customer(){
      return customer();}
    PricerRole pricer() {
      return product();}
  }
}
```

12

## Adapter: details

```
adapter PricingProduct adapts Product{
  adapter ItemRole adapts HWProduct {
    ChargerRole charge()
      {return tax();}
  }
}
```

13

## Adapter: details

```
adapter PricingProduct adapts Product{
  adapter ChargerRole adapts Tax {
    double cost(double unitPrice,
      ItemRole item){
      return taxCharge(unitPrice,item);
    }
  }
}
```

14

## Adapter: details

```
adapter PricingProduct adapts Product{
  adapter PricerRole adapts HWProduct{
    double basicPrice(ItemRole item) {
      return regPrice(item);}
    double discount(CustomerRole cust,
      ItemRole item) {
      return regDiscount(cust, item);}
  }
}
```

15

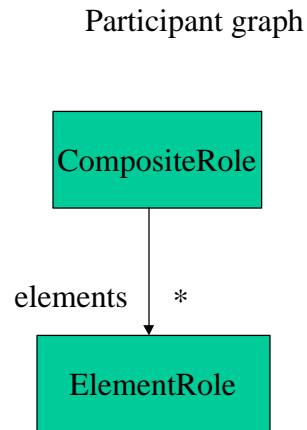
## Illustrate Layering: make Pricing a client

- We define a collaboration for Summing.
- We want to use it later for computing the price of an order using Pricing.
- An order consists of several line items and we want to sum the price of all of them.
- We first compose Summing and Pricing and apply the result to classes.

16

# Summing Collaboration

- CompositeRole
  - provided
    - double sum()
  - required
    - Vector elements()



17

# Summing Collaboration

```
CompositeRole:
double sum() {
    total = 0;
    for (ListIterator I = elements().listIterator();
        I.hasNext();) {
        total += ((ElementRole) I.next()).value();
    }
    return total;
}
```

18

## Summing Collaboration

- ElementRole
  - provided
  - required
    - double value()

19

## How to compose Summing and Pricing?

- Use a collaboration-collaboration adapter.
- SummingPricing adapter = **collaboration adapter**  
SummingPricing uses  
Summing, Pricing with  
ElementRole matches LineItemRole,  
ElementRole.value() matches  
LineItemRole.price()

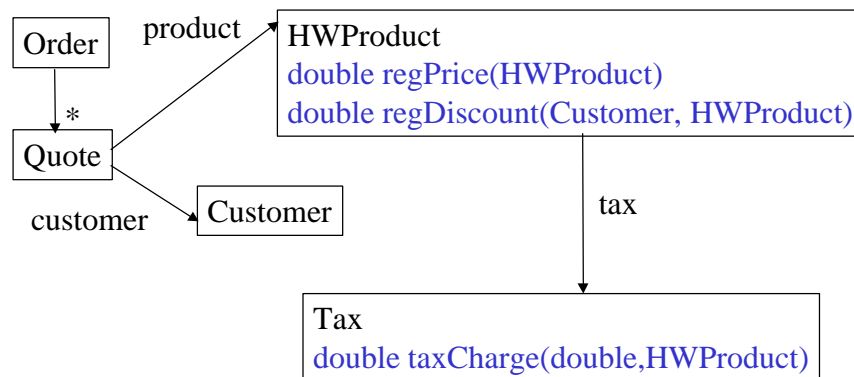
20

## Collaboration-collaboration adapter using uses

- Builds a new collaboration out of existing collaborations identifying roles and methods specified in matches clauses

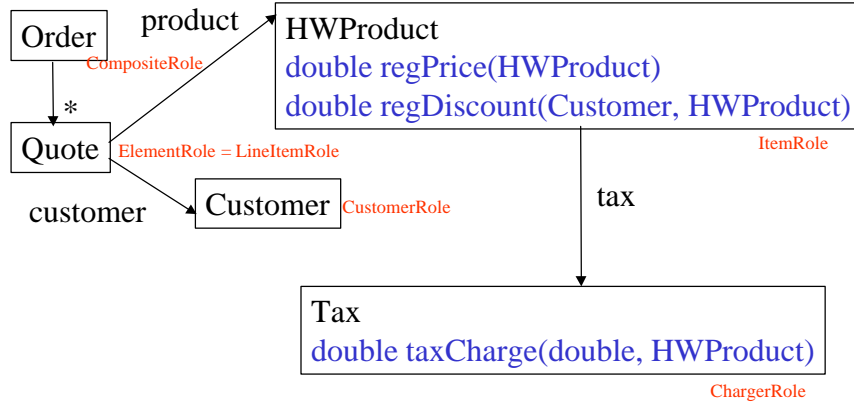
21

## Application Class Model



22

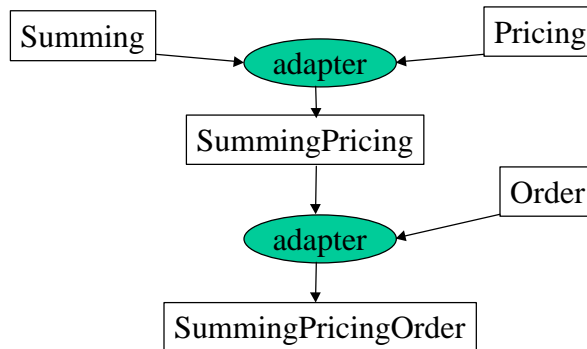
# Application Class Model with Adapter SummingPricingOrder



23

Order o = ... ; (o lift to SummingPricingOrder).sum();

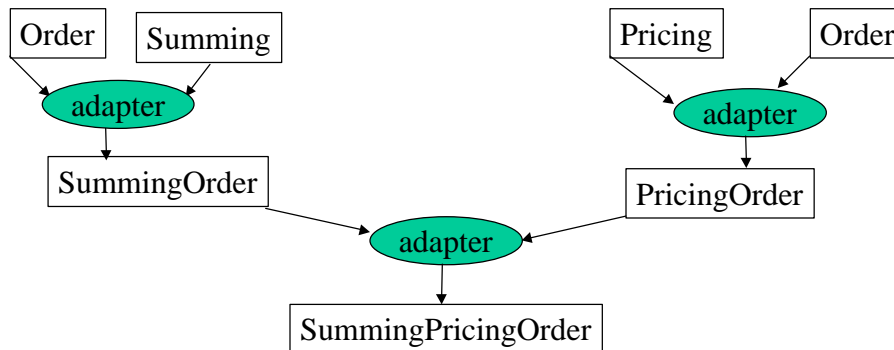
## Variant 1: simpler, better



24

Order o = ... ; (o lift to SummingPricingOrder).sum();

## Variant 2: too much detail too early



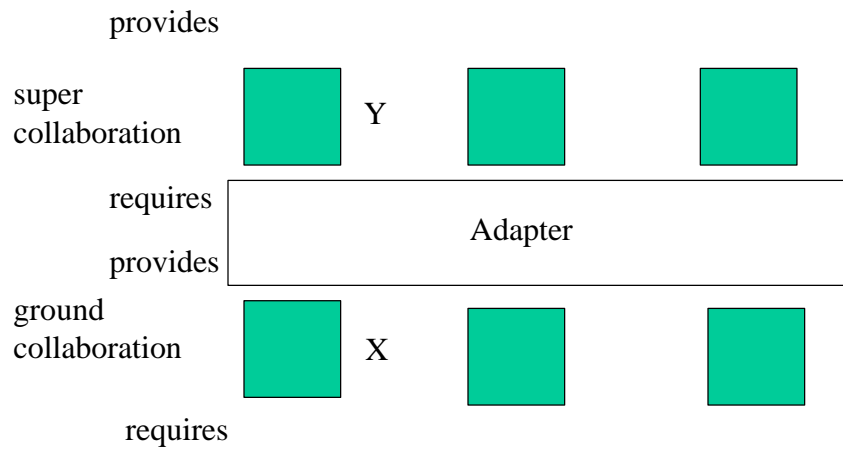
25

## Composite Adapters: Grammar

```
Adapter = "adapter" AdapterName "{ "  
  [<ground> CollabName] [<super> CollabName]  
  <fieldsAndMethods> List(MemberDefinition)  
  <helperClasses> List(ClassDef)  
  List(AdaptClause) }" .  
AdaptationClause = "adapter" AdapterName "adapts" <ground>  
  ParticipantName ["extends" <super> ParticipantName]  
  AdaptationBody.  
ParticipantName =  
  <collab> CollabName <class> ClassName.  
AdaptationBody = List(MemberDefinition).
```

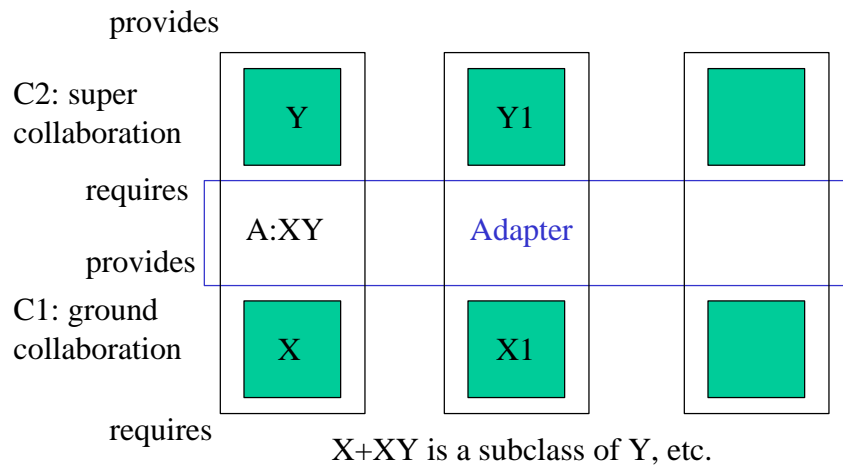
26

# Composite Adapters



27

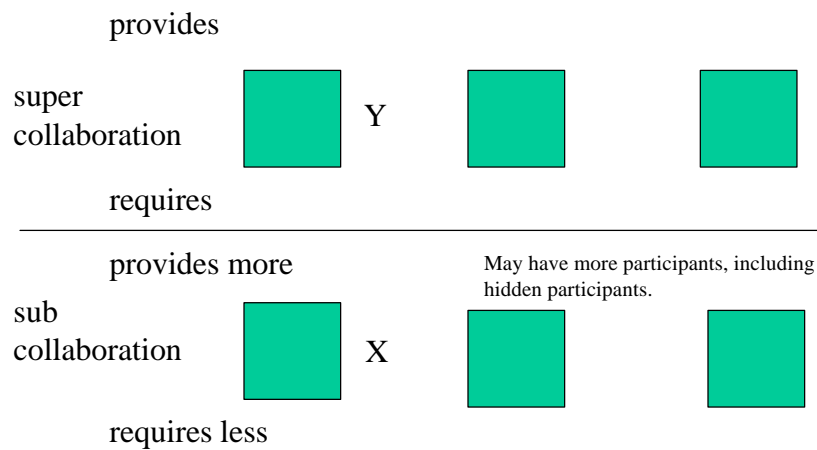
# Composite Adapters: they are partitioned



X+XY is a subclass of Y, etc.

28

## Collaboration Refinement



29

## Modeling a bank

- OpenNewBusinessRelation (ONBR)
  - a customer comes to the bank and opens a new business relationship
- OpenPartner (OP)
  - a subtask of opening a new business relationship

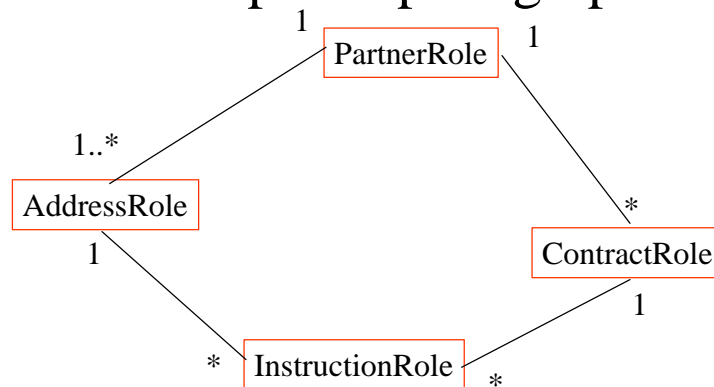
30

# ONBR Collaboration interface

```
collaboration OpenNewBusinessRelation { //ONBR
  role PartnerRole {
    P: openNewBusinessRelation();
    R: openPartner();
    R: checkPartner();}
  role AddressRole {
    R: OpenFirstAddress();}
  role ContractRole {
    R: makePrelimContract();
    R: signContract();}
  role InstructionRole {
    R: openInstructions();}
}
```

31

# ONBR Collaboration structure ONBR participant graph



32

## ONBR Collaboration behavior

```
collaboration OpenNewBusinessRelation {
  role PartnerRole {
    openNewBusinessRelation(){
      this.checkPartner();
      contracts.makePrelimContract();
      this.openPartner();
      addresses.OpenFirstAddress();
      addresses.instructions.openInstructions();}
  };
}
```

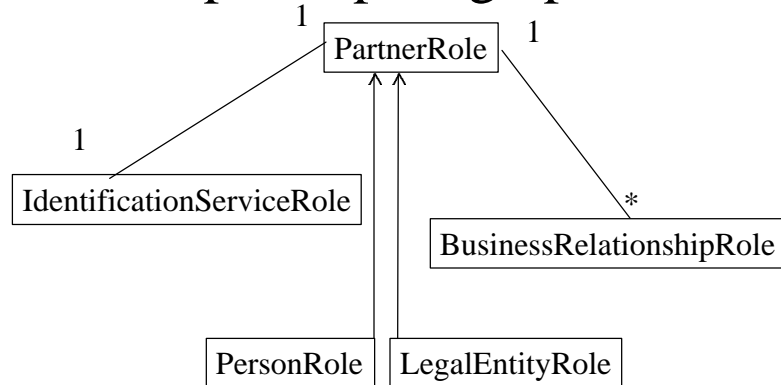
33

## OP Collaboration interface

```
Collaboration OpenPartner {
  role PartnerRole {
    P: openPartner();}
  role PersonRole extends PartnerRole {
    R: openPartner(); }
  role LegalEntityRole extends PartnerRole {
    R: openPartner(); }
  role IdentificationServiceRole {
    R: assignPartnerId();
    R: assignSecretId();}
  role BusinessRelationshipRole {
    R: openBusinessRelationship();}
}
```

34

## OP Collaboration structure OP participant graph



35

## OP Collaboration implementation

```
Collaboration OpenPartner { //OP
  role PartnerRole {
    openPartner(){
      identification.assignIdentification();
      this.openPartner();
      businessRelation.openBusinessRelationship();}
  }
  role IdentificationServiceRole {
    assignIdentification(){
      assignPartnerID();
      if (secret) assignSecretID();}
  }
}
```

36



## ONBR+OP behavior

```
adapter ONBR_OP {
  adapter PartnerRole
  adapt ONBR.PartnerRole played by OP.PartnerRole {
    //adaptation body
    openPartner(){this.openPartner();};
  }
  adapter AddressRole
  adapts ONBR.AddressRole {}
  adapter IdentificationServiceRole
  adapts OP.IdentificationServiceRole {}
  etc.
}
// seems too lengthy
```

39

## ONBR+OP behavior: shorter version

```
adapter ONBR_OP {
  OpenNewBusinessRelation(ONBR) uses OpenPartner(OP)
  adapter PartnerRole
  adapts ONBR.PartnerRole extends OP.PartnerRole {
    //adaptation body
    openPartner(){this.openPartner();};
  }
}
```

40

## Least-common-multiple Greatest-common-divisor

- $2*3*5 = 30$ ,  $3*7*11 = 231$
- $\text{LCM}(30,231) = 2*3*5*7*11 = 2310$
- $\text{GCD}(30,231) = 3$

41

## Collaborations and adapters

- A collaboration should contain the “intersection” of all adaptations of itself. Similar to the greatest common divisor. Adapters add and override, they don't delete.
- Collaboration
  - Adapter1
  - Adapter2

42

## Specializing a collaboration

- Refine Coll1

```
adapter A {  
  adapt Coll1.R1 called S1 {  
    // adaptation body  
  }  
}
```

results in collaboration A.

43

## Adapter Usage

- collaboration-to-collaboration
  - adapts Coll1.R played by Coll2.S called RS
  - by matching
    - adapter A uses Coll1, Coll2, ...
- collaboration-to-classes
  - adapts Coll1.R played by Class1
- collaboration refinement
  - adapts Coll1.R called S

44