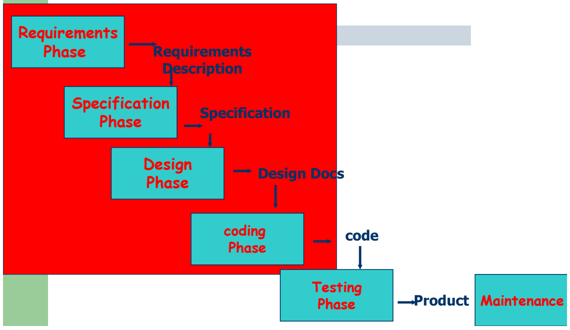
面向对象程序的分析与设计
Object-Oriented Analysis and Design

Lecture 11

Prof. S. Xu

Where We Are?



Contents

- Implementation Model
- Convert Design to Code: An Example

Implementation Model

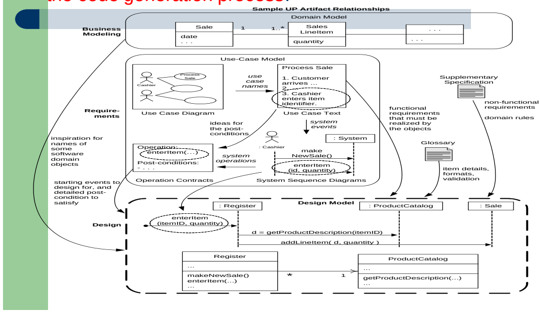
But wait - Before we get into Code

- You Created a Domain Model from requirements and use cases
- Used System Sequence Diagrams to identify system operations
- Clarified system operations with Operation Contracts
- Assigned “doing” responsibilities with Interaction Diagrams (Communication and Sequence Diagrams)
- Assigned “knowing” responsibilities with Design Class Diagrams

Depending on the system, many of these steps might just be sketches!

Okay, now. Mapping Designs to Code

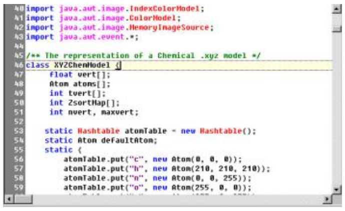
- The UML artifacts created during the design work - the interaction diagrams and DCDs - will be used as input to the code generation process.



Implementation Model

What is Implementation Model?

- Including all the implementation artifacts, such as the source code, database definitions, JSP/XML/HTML pages, and so forth.

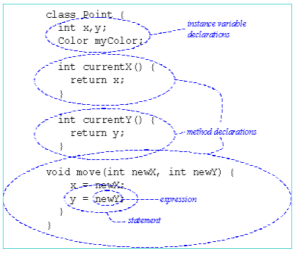


```
import java.awt.image.IndexColorModel;
import java.awt.image.ColorModel;
import java.awt.image.MemoryImageSource;
import java.awt.event.*;

/** The representation of a Chemical .xyz model */
class XYZChemModel {
      float vert[];
      Atom atoms[];
      int tvert[];
      int ZsortMap[];
      int nvert, maxvert;

    static Hashtable atomTable = new Hashtable();
    static Atom defaultAtom;
    static {
        atomTable.put("c", new Atom(0, 0, 0));
        atomTable.put("h", new Atom(210, 210, 210));
        atomTable.put("n", new Atom(0, 0, 255));
        atomTable.put("o", new Atom(255, 0, 0));
```

Mapping Designs to Code

How to conduct Implementation?

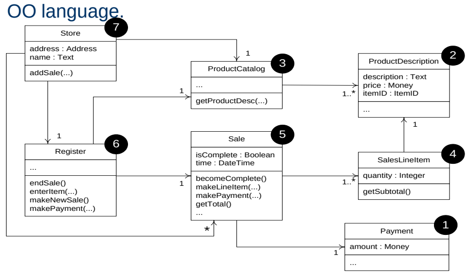
- class and interface definitions
- method definitions



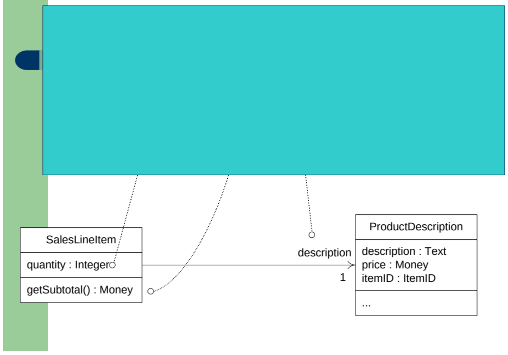
```
class Point {
  int x,y;
  Color myColor;

  int currentX() {
    return x;
  }

  int currentY() {
    return y;
  }

void move(int newX, int newY) {
    x = newX;
    y = newY;
  }
}
```

Creating Classes from DCDs

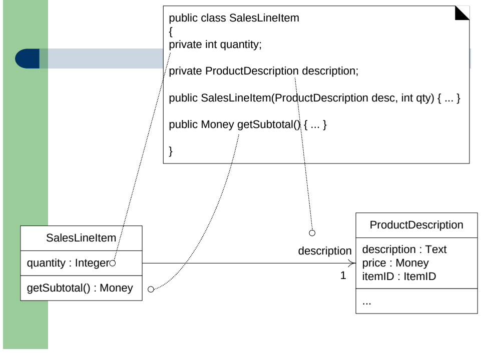
- DCDs depict the class or interface name, superclasses, operation signatures, and attributes of a class.
  - This is sufficient to create a basic class definition in an OO language.



Creating Classes from DCDs



Creating Classes from DCDs

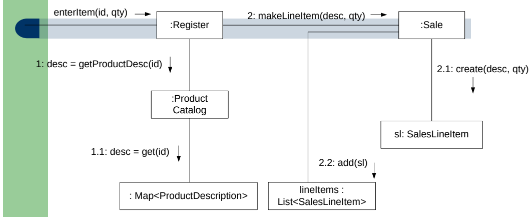


```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

public SalesLineItem(ProductDescription desc, int qty) { ... }

public Money getSubtotal() { ... }

}
```

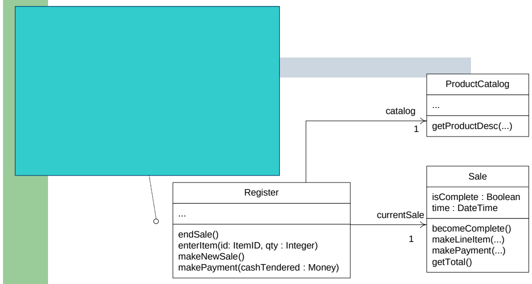
Creating Methods from Interaction Diagrams

- The sequence of the messages in an interaction diagram translates to a series of statements in the method definitions.
- The ***enterItem*** interaction diagram illustrates the Java definition of the ***enterItem*** method.
  - For the ***Register*** class.

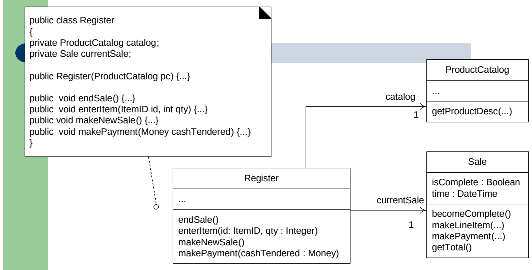
Creating Methods from Interaction Diagrams



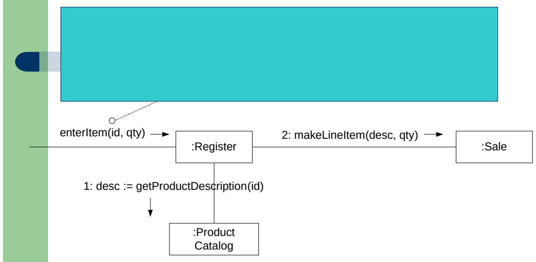
Creating Methods from Interaction Diagrams



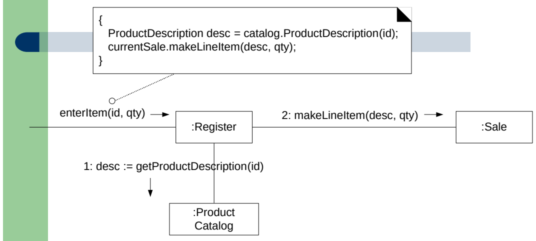
Creating Methods from Interaction Diagrams


```
public class Register
{
private ProductCatalog catalog;
private Sale currentSale;

public Register(ProductCatalog pc) {...}

public  void endSale() {...}
public  void enterItem(ItemID id, int qty) {...}
public void makeNewSale() {...}
public  void makePayment(Money cashTendered) {...}
}
```

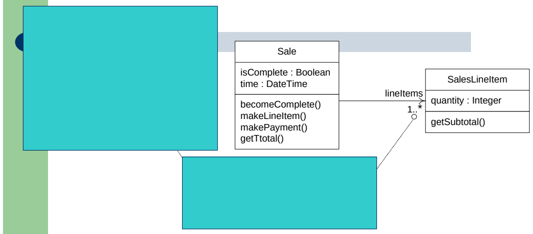
Creating Methods from Interaction Diagrams


enterItem(id, qty)
:Register
2: makeLineItem(desc, qty)
:Sale
1: desc := getProductDescription(id)
:Product Catalog

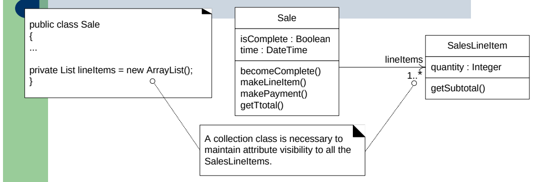
Creating Methods from Interaction Diagrams


```
{
  ProductDescription desc = catalog.ProductDescription(id);
  currentSale.makeLineItem(desc, qty);
}
```
enterItem(id, qty)
:Register
2: makeLineItem(desc, qty)
:Sale
1: desc := getProductDescription(id)
:Product Catalog

Collection Classes in Code



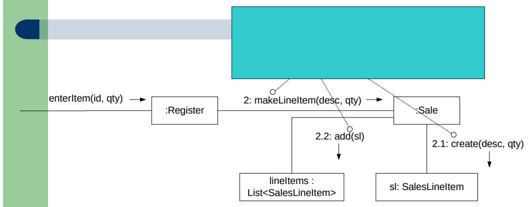
Collection Classes in Code



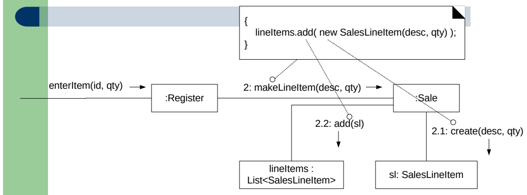
```
public class Sale
{
...

private List lineItems = new ArrayList();
}
```

A collection class is necessary to maintain attribute visibility to all the SalesLineItems.

One final example (Sale.makeLineItem method)



One final example (Sale.makeLineItem method)



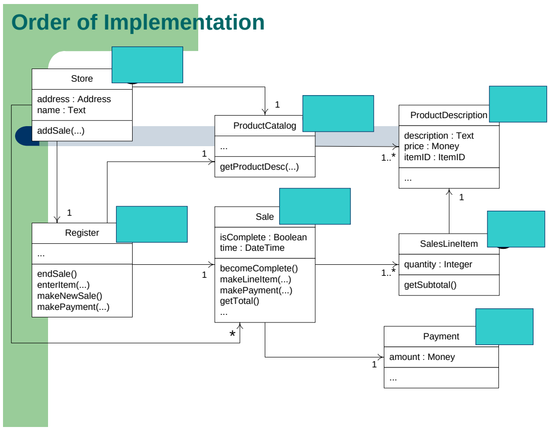
```
{
    lineItems.add( new SalesLineItem(desc, qty) );
}
```

Summary

- As demonstrated, there is a translation process
  - from UML class diagrams to class definitions, and
  - from interaction diagrams to method bodies.
- The code example for the NextGen POS case study…
  - This code defines a simple case; it is not meant to illustrate a robust, fully developed Java program with synchronization, exception handling, and so on.

- More on big data

Convert Design to Code: An Example



Order of Implementation

Store
address : Address
name : Text
addSale(...)

ProductCatalog
...
getProductDesc(...)

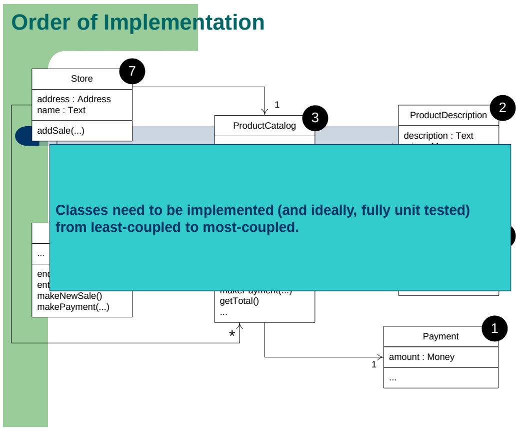
ProductDescription
description : Text
price : Money
itemID : ItemID
...

Register
...
endSale()
enterItem(...)
makeNewSale()
makePayment(...)

Sale
isComplete : Boolean
time : DateTime
becomeComplete()
makeLineItem(...)
makePayment(...)
getTotal()
...

SalesLineItem
quantity : Integer
getSubtotal()

Payment
amount : Money
...



Order of Implementation

Store
address : Address
name : Text
addSale(...)

ProductCatalog

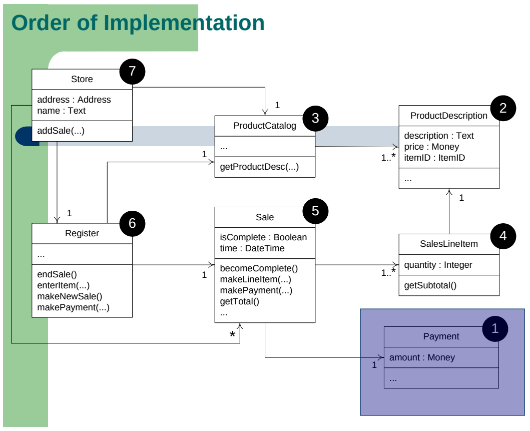
ProductDescription
description : Text

Classes need to be implemented (and ideally, fully unit tested) from least-coupled to most-coupled.

makeNewSale()
makePayment(...)

getTotal()
...

Payment
amount : Money
...



Order of Implementation

Store
address : Address
name : Text
addSale(...)

ProductCatalog
...
getProductDesc(...)

ProductDescription
description : Text
price : Money
itemID : ItemID
...

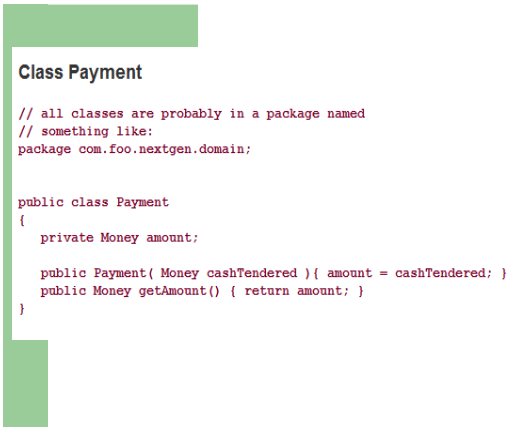
Register
...
endSale()
enterItem(...)
makeNewSale()
makePayment(...)

Sale
isComplete : Boolean
time : DateTime
becomeComplete()
makeLineItem(...)
makePayment(...)
getTotal()
...

SalesLineItem
quantity : Integer
getSubtotal()

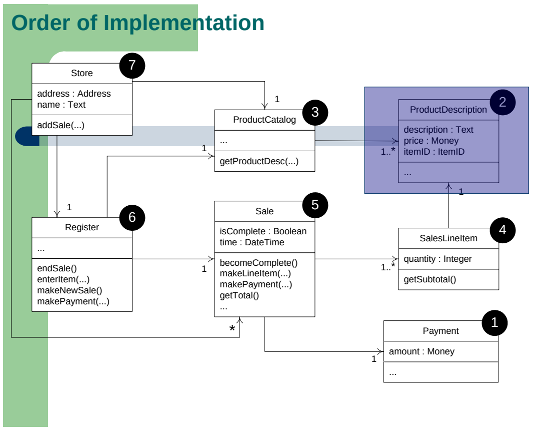
Payment
amount : Money
...

7
3
2
6
5
4
1
1
1..*
1
1
1..*
1
*
1



Class Payment

```
// all classes are probably in a package named
// something like:
package com.foo.nextgen.domain;


public class Payment
{
   private Money amount;

   public Payment( Money cashTendered ){ amount = cashTendered; }
   public Money getAmount() { return amount; }
}
```



Order of Implementation

Store
address : Address
name : Text
addSale(...)

ProductCatalog
...
getProductDesc(...)

ProductDescription
description : Text
price : Money
itemID : ItemID
...

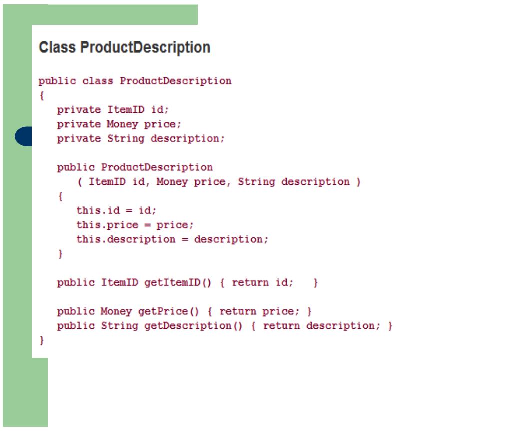
Register
...
endSale()
enterItem(...)
makeNewSale()
makePayment(...)

Sale
isComplete : Boolean
time : DateTime
becomeComplete()
makeLineItem(...)
makePayment(...)
getTotal()
...

SalesLineItem
quantity : Integer
getSubtotal()

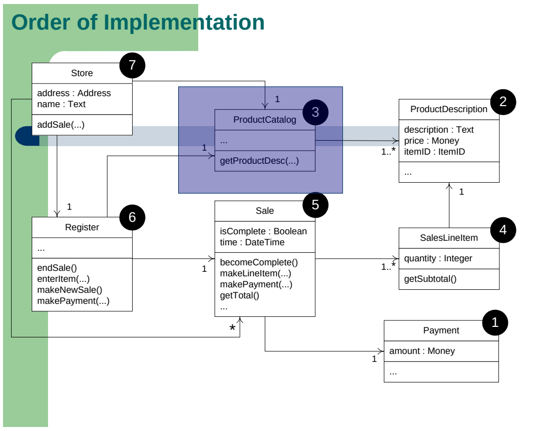
Payment
amount : Money
...

7
3
2
6
5
4
1
1
1..*
1
1
1..*
1
*
1

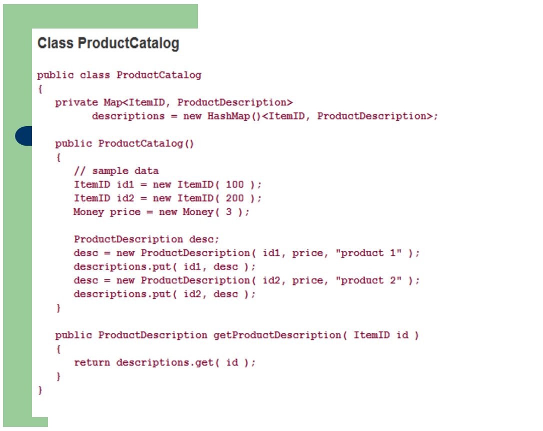


Class ProductDescription

```
public class ProductDescription
{
   private ItemID id;
   private Money price;
   private String description;

   public ProductDescription
      ( ItemID id, Money price, String description )
   {
      this.id = id;
      this.price = price;
      this.description = description;
   }

   public ItemID getItemID() { return id;   }

   public Money getPrice() { return price; }
   public String getDescription() { return description; }
}
```

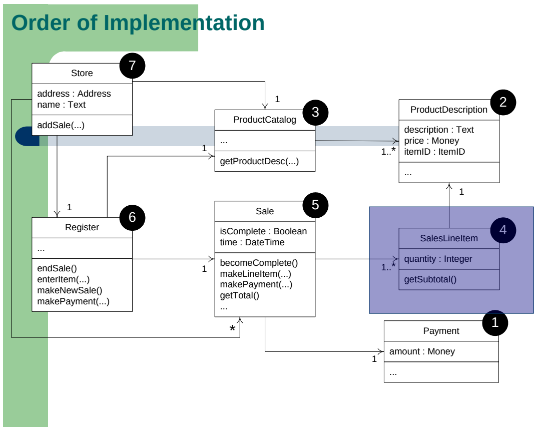


Order of Implementation

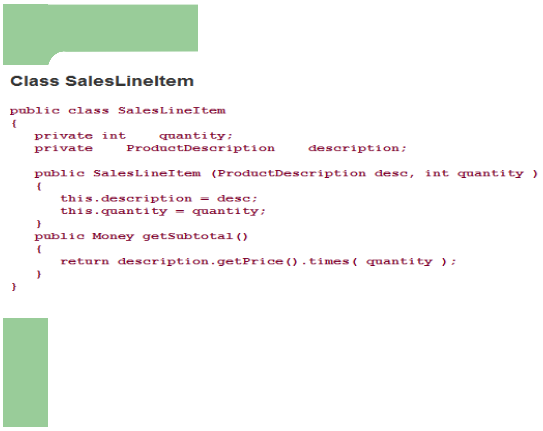


Class ProductCatalog

```
public class ProductCatalog
{
   private Map<ItemID, ProductDescription>
         descriptions = new HashMap()<ItemID, ProductDescription>;

   public ProductCatalog()
   {
      // sample data
      ItemID id1 = new ItemID( 100 );
      ItemID id2 = new ItemID( 200 );
      Money price = new Money( 3 );

      ProductDescription desc;
      desc = new ProductDescription( id1, price, "product 1" );
      descriptions.put( id1, desc );
      desc = new ProductDescription( id2, price, "product 2" );
      descriptions.put( id2, desc );
   }

   public ProductDescription getProductDescription( ItemID id )
   {
      return descriptions.get( id );
   }
}
```

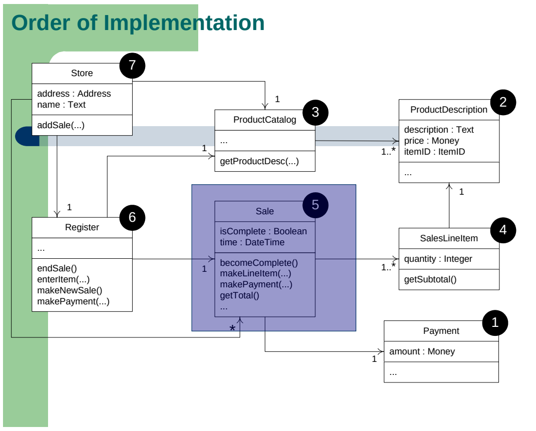


Order of Implementation

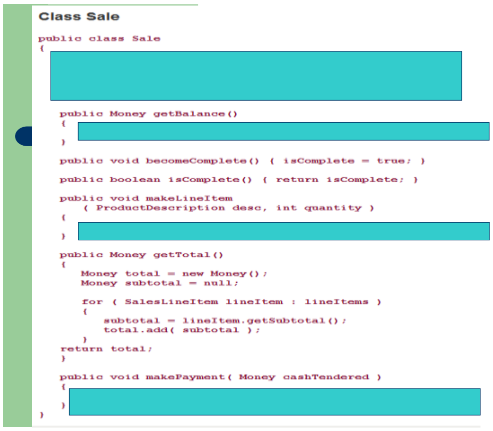


Class SalesLineItem

```
public class SalesLineItem
{
   private int    quantity;
   private    ProductDescription    description;

   public SalesLineItem (ProductDescription desc, int quantity )
   {
      this.description = desc;
      this.quantity = quantity;
   }
   public Money getSubtotal()
   {
      return description.getPrice().times( quantity );
   }
}
```



Order of Implementation



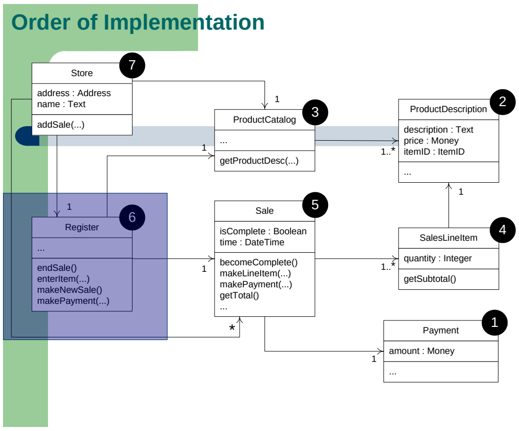
Class Sale

```
public class Sale
{

   public Money getBalance()
   {
   }

   public void becomeComplete() { isComplete = true; }

   public boolean isComplete() { return isComplete; }

   public void makeLineItem
      ( ProductDescription desc, int quantity )
   {
   }

   public Money getTotal()
   {
      Money total = new Money();
      Money subtotal = null;

      for ( SalesLineItem lineItem : lineItems )
      {
         subtotal = lineItem.getSubtotal();
         total.add( subtotal );
      }
   return total;
   }

   public void makePayment( Money cashTendered )
   {
   }
}
```

Class Sale

```
public class Sale
{
   private List<SalesLineItem> lineItems =
                              new ArrayList()<SalesLineItem>;
   private Date date = new Date();
   private boolean isComplete = false;
   private Payment payment;

   public Money getBalance()
   {
      return payment.getAmount().minus( getTotal() );
   }

   public void becomeComplete() { isComplete = true; }

   public boolean isComplete() { return isComplete; }

   public void makeLineItem
      ( ProductDescription desc, int quantity )
   {
      lineItems.add( new SalesLineItem( desc, quantity ) );
   }

   public Money getTotal()
   {
      Money total = new Money();
      Money subtotal = null;

      for ( SalesLineItem lineItem : lineItems )
      {
         subtotal = lineItem.getSubtotal();
         total.add( subtotal );
      }
   return total;
   }

   public void makePayment( Money cashTendered )
   {
      payment = new Payment( cashTendered );
   }
}
```

Order of Implementation



Store
address : Address
name : Text
addSale(...)

ProductCatalog
...
getProductDesc(...)

ProductDescription
description : Text
price : Money
itemID : ItemID
...

Register
...
endSale()
enterItem(...)
makeNewSale()
makePayment(...)

Sale
isComplete : Boolean
time : DateTime
becomeComplete()
makeLineItem(...)
makePayment(...)
getTotal()
...

SalesLineItem
quantity : Integer
getSubtotal()

Payment
amount : Money
...

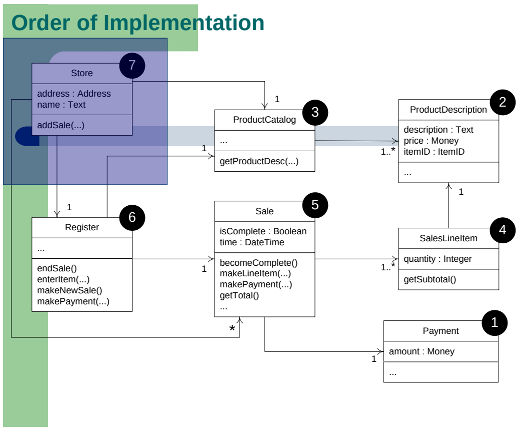
Class Register

```
public class Register
{
   private ProductCatalog catalog;
   private Sale currentSale;

   public Register( ProductCatalog catalog )
   {
      this.catalog = catalog;
   }

   public void endSale()
   {

   }

   public void enterItem( ItemID id, int quantity )
   {

   }

   public void makeNewSale()
   {

   }

   public void makePayment( Money cashTendered )
   {

   }
}
```

Class Register

```
public class Register
{
   private ProductCatalog catalog;
   private Sale currentSale;

   public Register( ProductCatalog catalog )
   {
      this.catalog = catalog;
   }

   public void endSale()
   {
      currentSale.becomeComplete();
   }

   public void enterItem( ItemID id, int quantity )
   {
      ProductDescription desc = catalog.getProductDescription( id );

      currentSale.makeLineItem( desc, quantity );
   }

   public void makeNewSale()
   {
      currentSale = new Sale();
   }

   public void makePayment( Money cashTendered )
   {
      currentSale.makePayment( cashTendered );
   }
}
```

Order of Implementation



Store
address : Address
name : Text
addSale(...)

ProductCatalog
...
getProductDesc(...)

ProductDescription
description : Text
price : Money
itemID : ItemID
...

Register
...
endSale()
enterItem(...)
makeNewSale()
makePayment(...)

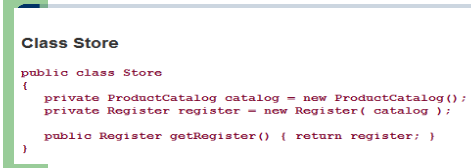
Sale
isComplete : Boolean
time : DateTime
becomeComplete()
makeLineItem(...)
makePayment(...)
getTotal()
...

SalesLineItem
quantity : Integer
getSubtotal()

Payment
amount : Money
...

Class Store

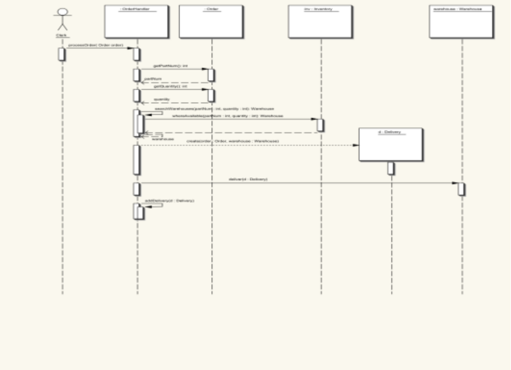
```
public class Store
{

}
```



Class Store

```
public class Store
{
   private ProductCatalog catalog = new ProductCatalog();
   private Register register = new Register( catalog );

   public Register getRegister() { return register; }
}
```

Exercise


Clerk
: OrderHandler
: Order
inv : Inventory
warehouse : Warehouse
processOrder(Order order)
getPartNum(): int
partNum
getQuantity(): int
quantity
searchWarehouses(partNum : int, quantity : int): Warehouse
whereAvailable(partNum : int, quantity : int): Warehouse
warehouse
create(order : Order, warehouse : Warehouse)
d : Delivery
deliver(d : Delivery)
addDelivery(d : Delivery)

Exercise

- Write the code for the Java method processOrder as specified by the sequence diagram above. Place it within the appropriate class, and include any attributes and methods that also belong to that class that are needed by the processOrder method. You do not have to provide all the code for the other methods in the class — just method signatures will suffice.

- More on Internet of Things