Mapping Designs to Code
Creating Class Definitions from DCDs

public class SalesLineItem
{
    private int quantity;

    private ProductDescription description;

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

    public Money getSubtotal() { ... }
}

ProductDescription

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

SalesLineItem

quantity : Integer

getSubtotal() : Money
Creating Methods from Interaction Diagrams (Register.enterItem)

```java
{ 
    ProductDescription desc = catalog.ProductDescription(id);
    currentSale.makeLineItem(desc, qty);
}
```
Collections

• One-to-many relationships are common.
  – For example, a Sale must maintain visibility to a group of many SalesLineItem instances.
  – In OO programming languages, these relationships are usually implemented with the introduction of a collection object, such as a List or Map, or even a simple array.

• Java libraries contain collection classes such as ArrayList and HashMap, which implement the List and Map interfaces, respectively.

• The choice of collection class is influenced by the requirements;
  – key-based lookup requires the use of a Map,
  – a growing ordered list requires a List, and so on.
A collection class is necessary to maintain attribute visibility to all the `SalesLineItem`

```java
public class Sale {
    ...
    private List lineItems = new ArrayList();
}
```
Order of Implementation

- Classes need to be implemented from least-coupled to most-coupled.

E.g.,

- possible first classes to implement are either Payment or ProductDescription;
- next are classes only dependent on the prior implementations—ProductCatalog or SalesLineItem.
Order of Implementation

1. Payment
   - amount: Money

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

3. ProductCatalog
   - getProductDesc(...)

4. SalesLineItem
   - quantity: Integer
   - makeLineItem(...)
   - getSubtotal()

5. Sale
   - isComplete: Boolean
   - time: DateTime
   - becomeComplete()
   - makeNewSale()
   - makePayment(...)
   - getTotal()

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

7. Store
   - address: Address
   - name: Text
   - addSale(...)
How to design `makeNewSale`?

by Creator and Controller

`makeNewSale`

:Register

Register creates a Sale by Creator

:Sale

linItems : List<SalesLineItem>

this execution specification is implied to be within the constructor of the Sale instance

by Creator, Sale creates an empty collection (such as a List) which will eventually hold SalesLineItem instances
enterItem

desc = getDescription(itemId)

makeLineItem(desc, qty)

create(desc, qty)

add(sl)

total = getTotal()

desc is an object reference

To be elaborated further
getTotal

- :Sale
- lineItems[i]:SalesLineItem
- :ProductDescription

Loop:
- st = getSubTotal
- pr = getPrice

tot = getTotal
makePayment, getBalance
endSale

```
public void becomeComplete()
{
    isComplete = true;
}
```
// 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;
    }
}
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;  }
}
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);
  }
}
public class SalesLineItem
{
    private ProductDescription description;
    private int quantity;

    public SalesLineItem (ProductDescription desc, int quantity )
    {
        this.description = desc;
        this.quantity = quantity;
    }

    public Money getSubtotal()
    {
        return description.getPrice().times( quantity );
    }
}
public class Sale {
    private List<SalesLineItem> lineItems = new ArrayList<>();
    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 );
}
} //end of sale
public class Register {
    private ProductCatalog catalog;
    private Sale currentSale;

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

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

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

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

    public void endSale() { currentSale.becomeComplete(); }
}
public class Store
{
    private ProductCatalog catalog = new ProductCatalog();
    private Register register = new Register( catalog );

    public Register getRegister() { return register; }  
}