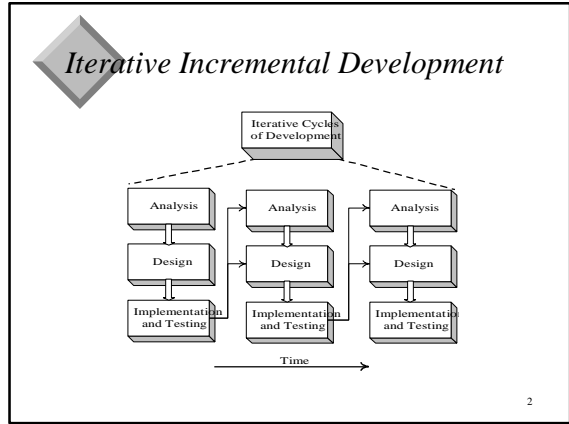


### Mapping Designs to Code (Ch. 23)

- ❖ *Rush to code* produces “brittle” systems
- ❖ The design should produce a framework in which the implementation will live
- ❖ Many detailed problems will be introduced in the coding phase, so expect them
  - iterations between Design & Impl. are common
- ❖ Exploratory programming is useful to discover workable designs, but is not Impl.

1



### Automatic Code Generation

```

public class SalesLineItem
{
    public SalesLineItem(ProductSpecification spec, int qty);
    public float subtotal();
    private int quantity;
}
    
```

C'tor args come from the *create(spec, qty)* msg to *SalesLineItem*

3

### References are suggested by associations & navigability

```

public class SalesLineItem
{
    public SalesLineItem(ProductSpecification spec, int qty);
    public float subtotal();
    private int quantity;
    private ProductSpecification prodSpec;
}
    
```

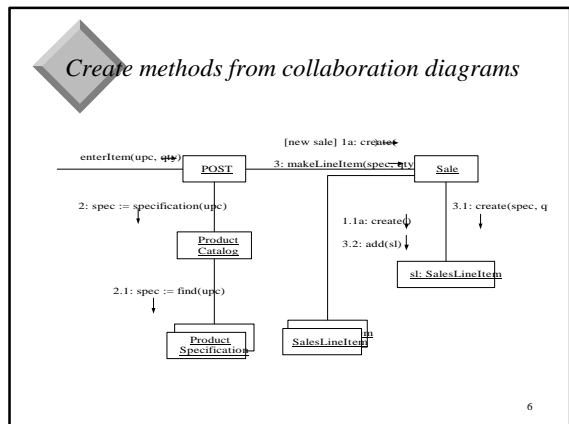
4

### Role names may be used to generate instance variables

```

public class SalesLineItem
{
    ...
    private int quantity;
    private ProductSpecification prodSpec;
}
    
```

5



### Code to Accompany Diagram

- ◆ Message 1:
  - if (isNewSale())
  - sale = new Sale();
- ◆ Message 2:
  - ProductSpecification spec =
  - new prodCatalog.specification(upc);
- ◆ Message 3:
  - sale.makeLineItem(spec, qty);
- ◆ Putting it all together:

7

### The enterItem() Method

```

public void enterItem(int upc, int qty)
{
    if (isNewSale()) { sale = new Sale; }
    ProductSpecification spec =
        prodCatalog.specification(upc);
    sale.makeLineItem(spec, qty);
}
    
```

A new method has emerged

8

### The isNewSale() Method

- ◆ Consider the following code alternatives:
  - if (isNewSale()) ...
  - if (sale == null) ...
- ◆ sale may only be null the first time through
  - private boolean isNewSale()
  - { return (sale == null ||
  - sale.isComplete());
  - }
- ◆ More robust design

9

### Adding Containers...

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

10

### and Adding Objects to it!

```

public void makeLineItem(ProductSpecification spec, int
{
    lineItems.addElement( new SalesLineItem(spec, qty);
}
    
```

11

### Possible Order of Implementation

12

**Class Payment**

```

package post;
import java.util.*;    // for Hashtable, Vector

public class Payment
{ // instance variable
  private float amount;
  // constructor
  public Payment(float cashTendered)
  { amount = cashTendered; }
  // accessor fn.
  public float getAmount()
  { return amount; }
}

```

13

**Class ProductCatalog**

```

public class ProductCatalog
{ private Hashtable productSpecifications =
  new Hashtable();
  public ProductCatalog()
  { ProductSpecification ps =
    new ProductSpecification(100, 1, "prod 1");
    productSpecifications.put(new Integer(100), ps);
    ps = new ProductSpecification(200, 1, "prod 2");
    productSpecifications.put(new Integer(200), ps);
  }

  public ProductSpecification getSpecification(int upc)
  { return (ProductSpecification)
    productSpecifications.get(new Integer(upc));
  }
}

```

14

**Class POST**

```

class POST
{ private ProductCatalog productCatalog;
  private Sale sale;

  public POST(ProductCatalog catalog)
  { productCatalog = catalog; }

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

  public void enterItem(int upc, int qty)
  { if (isNewSale()) sale = new Sale();
    ProductSpecification spec =
      productCatalog.specification(upc);
    sale.makeLineItem(spec, qty);
  }
}

```

15

**Class POST (cont.)**

```

  public void makePayment(float cashTendered)
  { sale.makePayment(cashTendered); }

  private boolean isNewSale()
  { return (sale == null) || sale.isComplete(); }

} // end of class POST

```

16

**Class ProductSpecification**

```

public class ProductSpecification
{ private int upc = 0;
  private float price = 0;
  private String description = "";

  public ProductSpecification(int upc, float price,
    String description)
  { this.upc = upc;
    this.price = price;
    this.description = description;
  }

  public int getUPC() { return upc; }
  public float getPrice() { return price; }
  public String getDescription() { return description; }
}

```

17

**Class Sale**

```

class Sale
{ private Vector lineItems = new Vector();
  private Date date = new Date();
  private boolean isComplete = false;
  private Payment payment;

  public getBalance()
  { return payment.getAmount() - total(); }

  public boolean isComplete() { return isComplete; }

  public void makeLineItem(ProductSpecification spec,
    int qty)
  { lineItems.addElement(new SaleLineItem(spec, qty));
  }
}

```

18

 *Class Sale (cont.)*

```

public float total()
{
    float total = 0;
    Enumeration e = lineItems.elements();
    while (e.hasMoreElements())
    {
        total +=
            ((SaleLineItem) e.nextElement()).subtotal();
    }
    return total;
}

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

} // end of class Sale

```

19

 *Class SaleLineItem*

```

class SaleLineItem
{
    private int quantity;
    private ProductSpecification productSpec;

    public SaleLineItem(ProductSpecification spec,
                        int qty)
    {
        productSpec = spec;
        quantity = qty;
    }

    public float subtotal()
    {
        return quantity * productSpec.getPrice();
    }
}

```

20

 *Class Store*

```


class Store
{
    private ProductCatalog productCatalog =
        new ProductCatalog();
    private POST post = new POST(productCatalog);

    public POST getPOST() { return post; }
}

```

Domain classes only, no user interface code (too involved)!

21

 *End of Phase 1 for POST System*

Next:  
Development Cycle #2

22



23