OO Design

CPSC 1181 – O.O.

Jeremy Hilliker

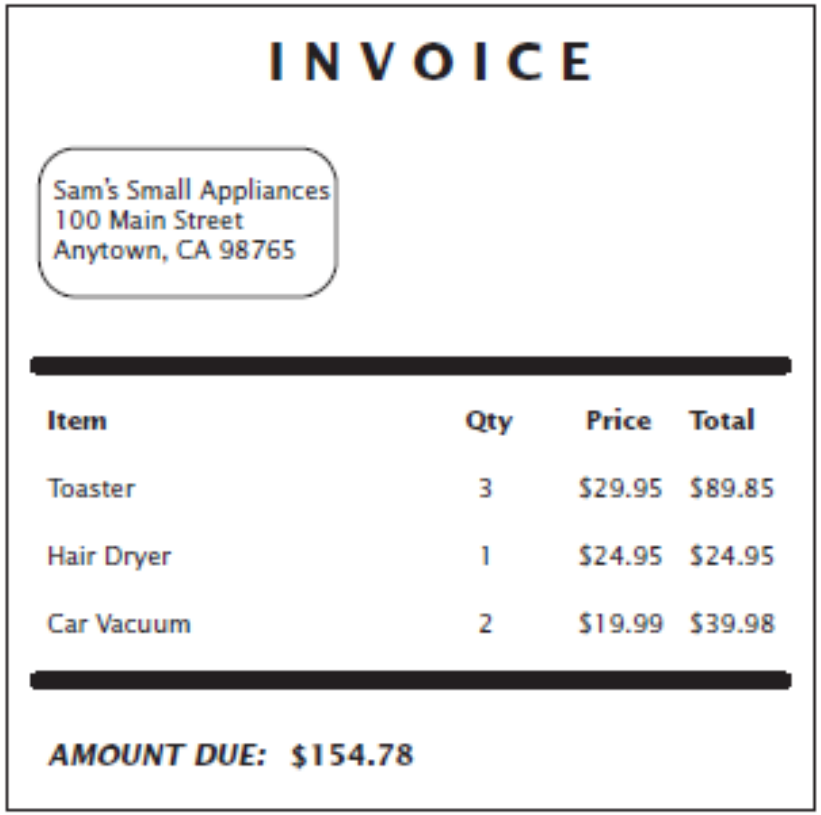
Summer 2017

Overview

- Example

- User Story
- Use Case
- Discover classes
- CRC cards
- Dev process

Example



INVOICE

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Item	Qty	Price	Total
Toaster	3	$29.95	$89.85
Hair Dryer	1	$24.95	$24.95
Car Vacuum	2	$19.99	$39.98

AMOUNT DUE: **$154.78**

User Story: Invoice

- **As a** vendor
- **I want to** create an invoice
- **So that** both the customer and I have a record of the transaction (what was sold, how many, for what price, and the amount due)

Use Case

	Customer	Vendor	System
1	Orders some items		
2		Opens a new invoice	Creates new invoice
4		Enters an item and quantity ordered	
5			Creates line item containing item, quantity, price, quantity*price
6			Updates total amount due
7		Repeat step 4 - 6 as needed	
8		Finalize invoice	Finalize invoice […]
9		Print invoice	Record invoice
10	Receive goods & invoice		

Recall: Discovering Classes

- **Nouns**
  - Classes?
    - First Class Citizen?
  - Composite attribute?
  - Represents a single concept from the domain.
  - “Is a” relationship
- Verbs
  - Behaviours (mutators)
- Questions
  - Accessors, Predicates
- Adjectives
  - Attributes (“Has a” relationship)
  - Parameters

Advice

- Write some potential classes
- Determine which ones are first class citizens
- Err on the side of too few classes
  - YAGNI, KISS
  - Can always add a class if you decide it is a first class citizen
- Concepts from the problem domain are good concepts for classes
- Not all classes can be discovered this way
  - Will need some supporting classes

Example

- Potential classes
  - Order
  - Invoice
  - Customer
    - Address
  - Line Item
    - Product
    - Quantity
    - Price
    - Total
  - Total amount due

Example

- Potential classes
  - ~~Order~~ (not yet)
  - **Invoice**
  - ~~Customer~~
    - **Address**
  - **Line Item**
    - Product
      - ~~Price~~ (attribute)
    - ~~Quantity~~ (attribute)
    - ~~Total~~ (calculated)
  - ~~Total amount due~~ (calculated)

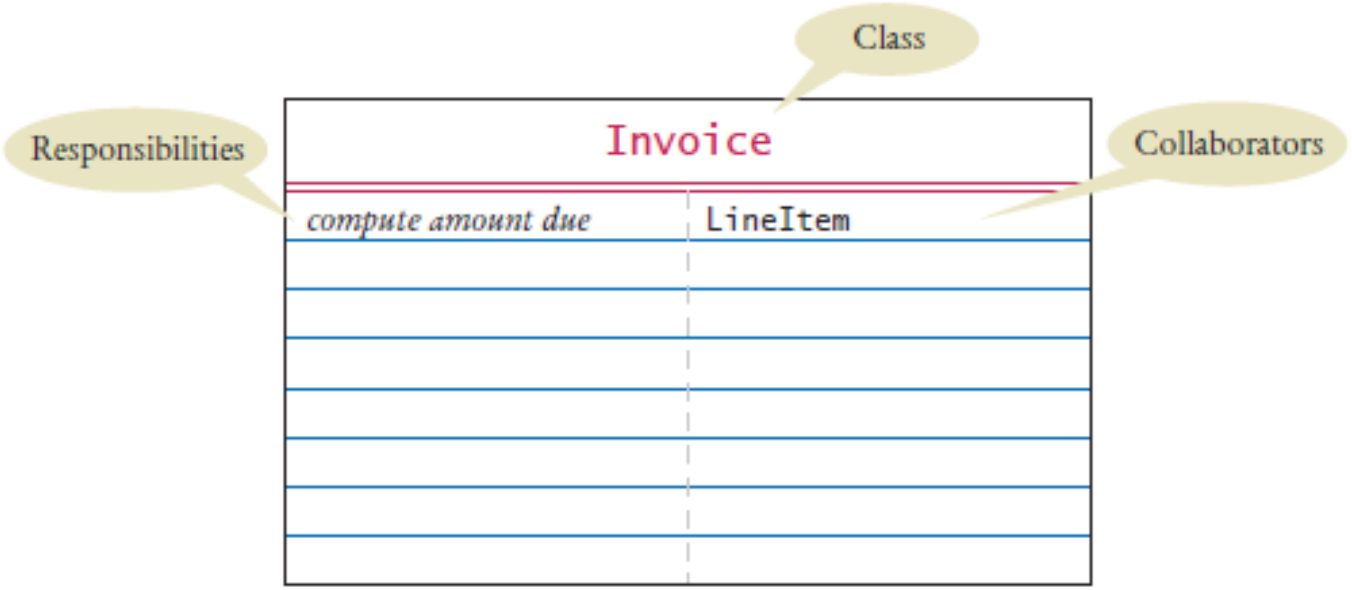
CRC Card Method

- Class-responsibility-collaboration (CRC) cards
  - K. Beck and W. Cunningham. 1989. A laboratory for teaching object oriented thinking. In *Conference proceedings on Object-oriented programming systems, languages and applications* (OOPSLA '89). ACM, New York, NY, USA, 1-6. DOI=http://dx.doi.org/10.1145/74877.74879
  - http://www.extremeprogramming.org/rules/crccards.html

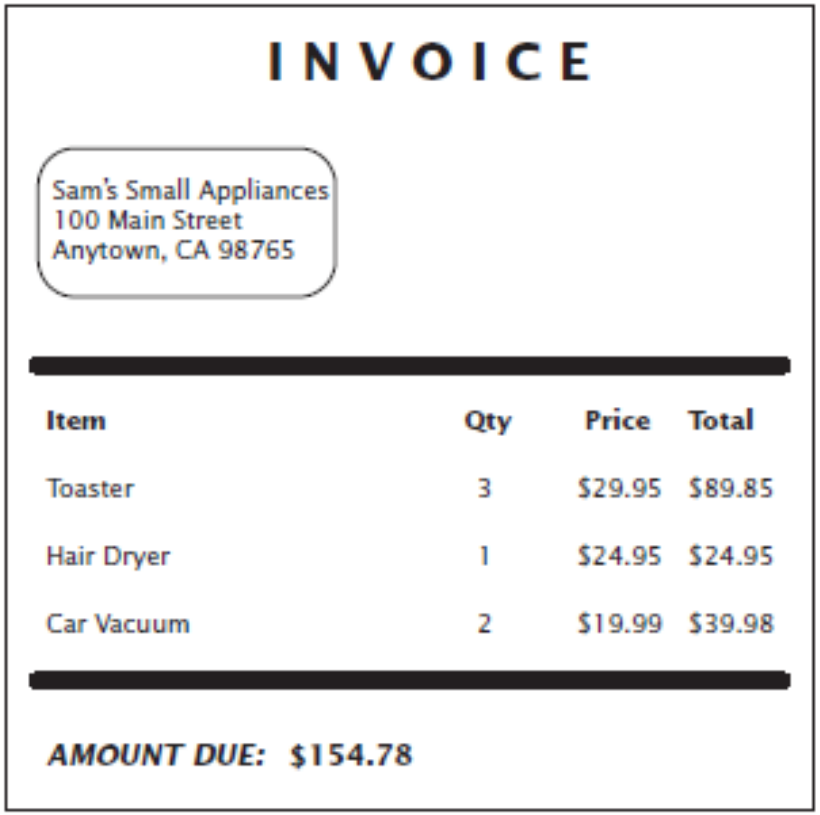
CRC Method

- After you have some (minimal) classes
- Define (minimal) behaviours
  - Verbs (mutators)
  - Questions (accessors)
  - Match them to the most appropriate object
    - Eg: need to compute total amount due
    - Who's responsible?
      - Invoice
- Indicate what other classes are needed to fulfill responsibility (collaborators)

CRC Card


Class
Responsibilities
Invoice
Collaborators
compute amount due
LineItem

Example



INVOICE

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Item	Qty	Price	Total
Toaster	3	$29.95	$89.85
Hair Dryer	1	$24.95	$24.95
Car Vacuum	2	$19.99	$39.98

AMOUNT DUE: **$154.78**

Check

- Suppose an invoice is to be saved to a file
- Name a likely collaborator
    - InvoiceFileSaver ?
    - PrintStream ?

Check

- What is a likely responsibility of the Customer class?
    - Produce the shipping address of the customer

Check

- What to do if a CRC card has a dozen responsibilities?
  - High cohesion: reword them so that they are at a higher level
  - Low cohesion: separation of concerns; break into more classes

Recall: Dependencies

			Line	Tip
Strongest	Inheritance	Is-a	Solid	Triangle
	Composition	Has-a (non-separable)	Solid	Filled diamond
	Aggregation	Has-a (separable)	Solid	Open diamond
	Association	Has-a (no ownership)	Solid	Open arrow
Weakest	Dependency	Knows-about	Dotted	Open arrow

5 Part Program Dev Process (Horstmann)

1. Gather requirements
2. Make CRC cards
3. Make UML
4. Write Javadoc
5. Implementation

We do:
- Iterate
  - Iterate
    - Write test
    - Implement
  - javadoc

Printing Invoice – Req's

- Start the development process by gathering and documenting program requirements.
- Task: Print out an invoice
- Invoice: Describes the charges for a set of products in certain quantities.
- Omit complexities
  - Dates, taxes, and invoice and customer numbers
- Print invoice
  - Billing address, all line items, amount due
- Line item
  - Description, unit price, quantity ordered, total price
- For simplicity, do not provide a user interface.

- Test program: Adds line items to the invoice and then prints it.

Printing Invoice – Req's

```
                  I N V O I C E

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Description                     Price Qty Total
Toaster                         29.95  3  89.85
Hair dryer                      24.95  1  24.95
Car vacuum                      19.99  2  39.98

AMOUNT DUE: $154.78
```

Example

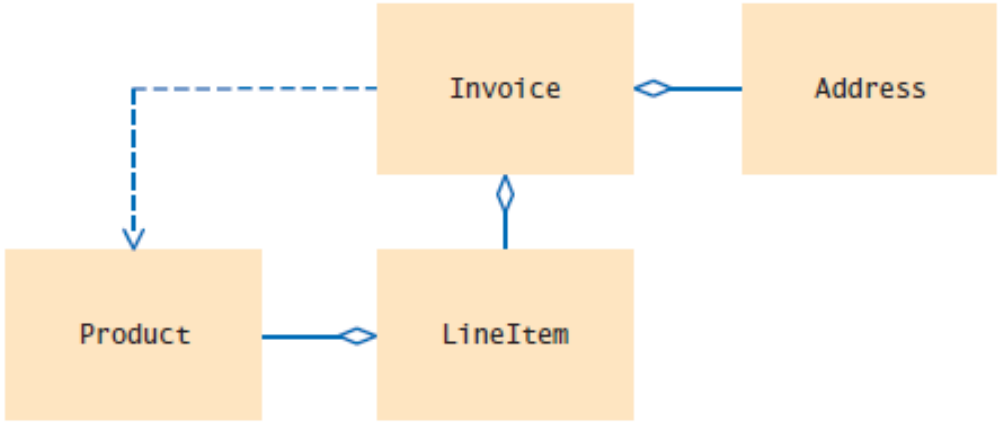
- Potential classes
  - ~~Order~~ (not yet)
  - **Invoice**
  - ~~Customer~~
    - **Address**
  - **Line Item**
    - Product
      - ~~Price~~ (attribute)
    - ~~Quantity~~ (attribute)
    - ~~Total~~ (calculated)
  - ~~Total amount due~~ (calculated)

Invoice	
format the invoice	Address
add a product and quantity	LineItem
	Product

Address
format the address

Product
get description
get unit price

LineItem	
format the item	Product
get total price	



Now Follow TDD

- Iterate
  - Iterate
    - Write test
    - Implement
  - javadoc

Check

- Which class is responsible for computing the amount due?
  - Invoice
- What are its collaborators for this task?
  - LineItem

Check

- Why did we use the toString() method instead of directly printing to console?
  - Reduces coupling.
  - No dependency on System.out
  - Can attach to other UIs

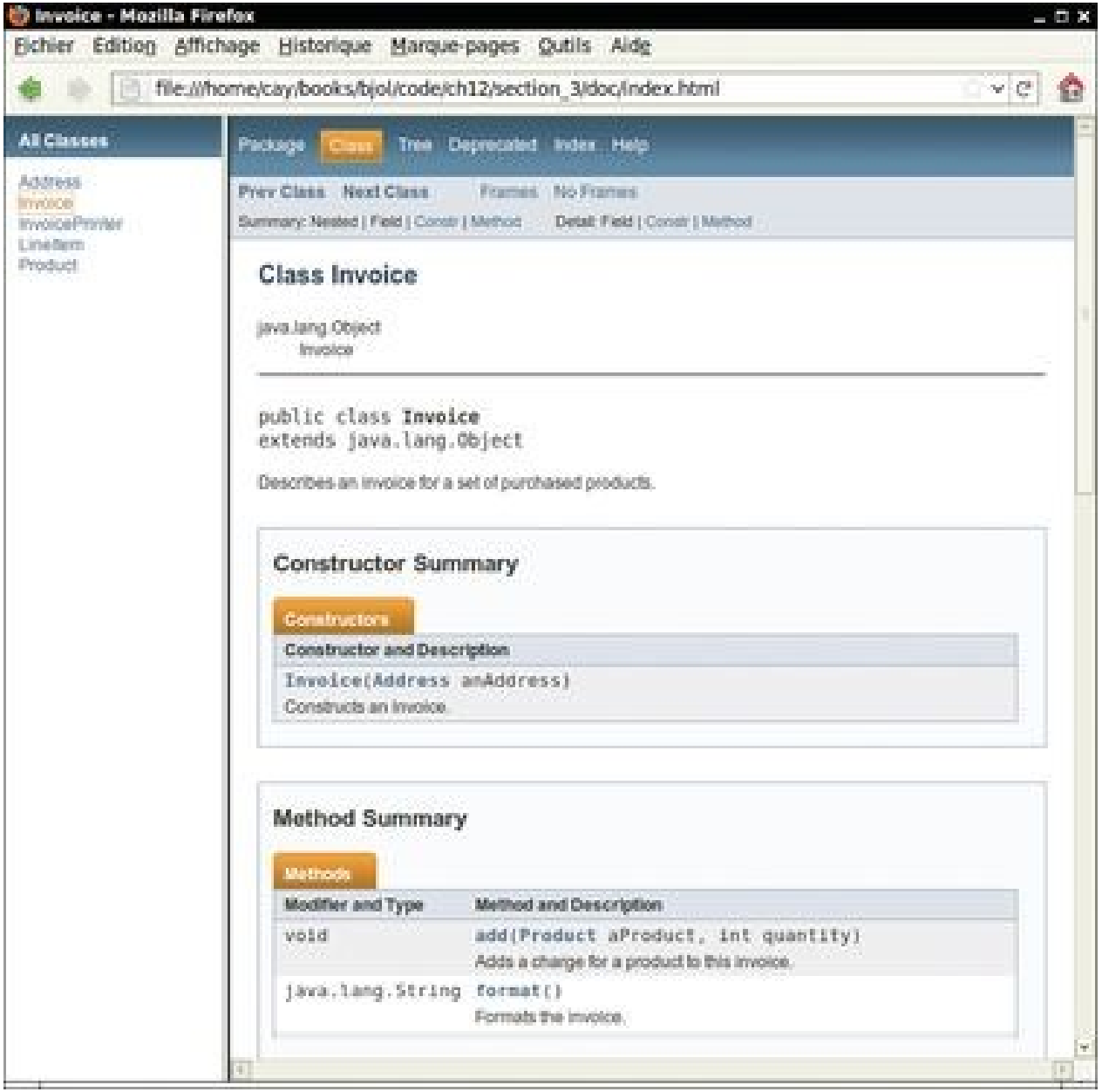
Non-TDD Follows

```java
/**
   Describes an invoice for a set of purchased products.
*/
public class Invoice {
   /**
      Adds a charge for a product to this invoice.
      @param aProduct the product that the customer ordered
      @param quantity the quantity of the product
   */
   public void add(Product aProduct, int quantity) {
   }

   /**
      Formats the invoice.
      @return the formatted invoice
   */
   public String toString() {
   }
}

```

```java
/**
   Describes a quantity of an article to purchase and its price.
*/
public class LineItem {
   /**
      Computes the total cost of this line item.
      @return the total price
   */
   public double getTotalPrice() {
   }

   /**
      Formats this item.
      @return a formatted string of this line item
   */
   public String toString() {
   }
}
```

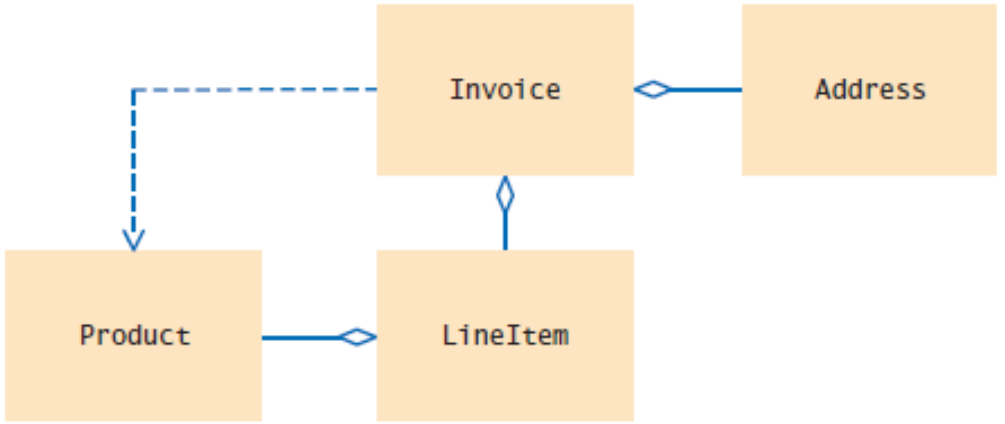
```java
/**
   Describes a product with a description and a price.
*/
public class Product {
   /**
      Gets the product description.
      @return the description
   */
   public String getDescription() {
   }

   /**
      Gets the product price.
      @return the unit price
   */
   public double getPrice() {
   }
}
```

```
1  /**
2     Describes a mailing address.
3  */
4  public class Address {
5     /**
6        Formats the address.
7        @return the address as a string with three lines
8     */
9     public String format() {
10    }
11 }
12
```


Invoice - Mozilla Firefox
Fichier Edition Affichage Historique Marque-pages Outils Aide
file:///home/cay/books/bjol/code/ch12/section_3/doc/index.html
All Classes
Address
Invoice
InvoicePrinter
LineItem
Product
Package Class Tree Deprecated Index Help
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method
Class Invoice
java.lang.Object
Invoice
public class Invoice
extends java.lang.Object
Describes an invoice for a set of purchased products.
Constructor Summary
Constructors
Constructor and Description
Invoice(Address anAddress)
Constructs an invoice.
Method Summary
Methods
Modifier and Type
Method and Description
void
add(Product aProduct, int quantity)
Adds a charge for a product to this invoice.
java.lang.String
format()
Formats the invoice.

Implementation: Variables

- The UML will indicate instance variables
  - Aggregation
  - Composition
  - Possibly relationships



Implementation: Variables

```
1   public class Invoice {
2       private Address billingAddress;
3       private ArrayList<LineItem> items;
4       // ...
5   }
6
7   public class LineItem {
8       private int quantity;
9       private Product theProduct;
10      // ...
11  }
12
```

Implementation: Methods

- Should be straight-forward

```
1  public class LineItem {
2      // ...
3      /**
4         Computes the total cost of this line item.
5         @return the total price
6      */
7      public double getTotalPrice() {
8         return theProduct.getPrice() * quantity;
9      }
10     // ...
11 }
12
```

```
1  import java.util.ArrayList;
2
3  /**
4     Describes an invoice for a set of purchased products.
5  */
6  public class Invoice
7  {
8     private Address billingAddress;
9     private ArrayList<LineItem> items;
10
11    /**
12       Constructs an invoice.
13       @param anAddress the billing address
14    */
15    public Invoice(Address anAddress)
16    {
17       items = new ArrayList<LineItem>();
18       billingAddress = anAddress;
19    }
20
21    /**
22       Adds a charge for a product to this invoice.
23       @param aProduct the product that the customer ordered
24       @param quantity the quantity of the product
25    */
26    public void add(Product aProduct, int quantity)
27    {
28       LineItem anItem = new LineItem(aProduct, quantity);
29       items.add(anItem);
```

```
1  /**
2     Describes a quantity of an article to purchase.
3  */
4  public class LineItem
5  {
6     private int quantity;
7     private Product theProduct;
8
9     /**
10       Constructs an item from the product and quantity.
11       @param aProduct the product
12       @param aQuantity the item quantity
13    */
14    public LineItem(Product aProduct, int aQuantity)
15    {
16       theProduct = aProduct;
17       quantity = aQuantity;
18    }
19
20    /**
21       Computes the total cost of this line item.
22       @return the total price
23    */
24    public double getTotalPrice()
25    {
26       return theProduct.getPrice() * quantity;
27    }
28
29    /**
```

```
1  /**
2     Describes a product with a description and a price.
3  */
4  public class Product
5  {
6     private String description;
7     private double price;
8
9     /**
10       Constructs a product from a description and a price.
11       @param aDescription the product description
12       @param aPrice the product price
13    */
14    public Product(String aDescription, double aPrice)
15    {
16       description = aDescription;
17       price = aPrice;
18    }
19
20    /**
21       Gets the product description.
22       @return the description
23    */
24    public String getDescription()
25    {
26       return description;
27    }
28
29    /**
```

```
1  /**
2     Describes a mailing address.
3  */
4  public class Address
5  {
6     private String name;
7     private String street;
8     private String city;
9     private String state;
10    private String zip;
11
12    /**
13       Constructs a mailing address.
14       @param aName the recipient name
15       @param aStreet the street
16       @param aCity the city
17       @param aState the two-letter state code
18       @param aZip the ZIP postal code
19    */
20    public Address(String aName, String aStreet,
21          String aCity, String aState, String aZip)
22    {
23       name = aName;
24       street = aStreet;
25       city = aCity;
26       state = aState;
27       zip = aZip;
28    }
```

Recap

- Example

- User Story
- Use Case
- Discover classes
- CRC cards
- Dev process