Objectives

- To know what are instance variables and class variables.
- To know when to use instance variables as oppose to when to use class variables.
- To know what are instance methods and class methods.
- To differentiate how to call instance methods as opposed to calling class methods.
- To know about the object reference object this, and be able to use it.
- To develop approach to problem solving using object oriented approach.
Program Development

• Introduction
• Problem I – Generating Sales Report
• Instance variables
• Instance Methods
• Class Variables
• Class Methods
• Overloading
• The Object Reference, this
• Constants
• Rules Governing Constants
• Problem II – Customer Billing
• Pitfalls
Introduction

• Software development can range from simple program requirements to complex requirements.
• Some outputs can look very intricate, but the programming is relatively simple
• We will use two examples to demonstrate this
  ✓ Problem I – Generating Sales Report
  ✓ Problem II – Customer Billing
Generating Sales Report

Problem I

A local furniture store ABC Furnishing, Inc, wishes to computerize its daily sales. You are required to write a Java application program to generate a summary report for each day’s sale.

The program accepts the name of the product; the quantity sold day; and the gross amount in sale. The program should also generate the total number of products sold and the gross sale for all the products.

The Figure 1 shows the format of the report.
ABC Furnishing, Inc

Sales Report for Oct 13, 2007

<table>
<thead>
<tr>
<th>Product</th>
<th>Quantity</th>
<th>Amount($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chair</td>
<td>20</td>
<td>1075.0</td>
</tr>
<tr>
<td>Table</td>
<td>20</td>
<td>1155.0</td>
</tr>
<tr>
<td>Lamp</td>
<td>10</td>
<td>175.0</td>
</tr>
<tr>
<td>Sofa</td>
<td>40</td>
<td>2255.0</td>
</tr>
</tbody>
</table>

Total pieces sold 90
Total day's sale $4,660.00

------------- End of report -------------
The General Solution

Solution

• Name of entity – Sales - other names are just as good.

• Attributes - at first glance we will need the following variables:
  ✓ The name of the product
  ✓ The quantity of product sold each day
  ✓ The amount in daily sales

• Methods - accessor methods to return the value for each of the variable.
The constructor

- **Constructor - Every set product sold has:**
  - A name,
  - The amount of money it was sold for, and
  - The quantity sold.

- **Hence, every object will reflect these three values**
Kinds of Variables and Methods

• At a second glance we realize that we also need variables:
  ✓ To accumulate the sales amount for each type of product sold, and
  ✓ To accumulate the quantity of each product sold

• The first three variables discussed are assigned their respective values each time a sale is made; i.e., every time an object is created.

• The latter two are updated regardless of which product is sold.

• This difference in idea gives rise to variable classification – instance variables and class variables
Instance Variables

- Variables that depend on the creation of objects.

- The variables in all of the examples that we have studied so far are instance variables.

- In the current exercise, the variables for the name of products, the quantity sold, and the amount of money collected, are all instance variables.

- They can only be assigned values if a transaction is carried out.
Instance Methods

• Just as we have instance variables, we also have instance methods.

• **Instance methods** are designed to access **instance variables**.

• In other words, an instance method cannot be invoked unless an instance of the class is created; and,

• An instance variable cannot be accessed by an instance method **unless** an instance of the class has been created.

• All of the methods that we have discussed so far are instance methods.
Class Variables

- Do not depend on any instance of a class
- Require class methods to access them
- Can however, be accessed by instance methods
- Only one copy of each of variable exists during program execution

By now you can see why we want to discuss class variables:

- One to accumulate the total number products, regardless of the product sold
- The other to total the amount of sales per product.
Format of Class Variables

static data_type nameOfVariable;

• In the current example let’s call these variables:
  ✓ total_quantity – to accumulate the total of all products sold, and
  ✓ total_sales – accumulate the total amount of money for all product sold

• The total of all products will be declared as follows:

  static int total_quantity;

• and for the total sales

  static double total_sales;
Format of Class Methods

- Are designed to access class variables.
- Can access literal values
- Cannot access instance variables
- Do not rely on the existence of objects either
- Use the class name to communicate with them directly
- Class methods must be prefaced with the keyword `static`

The format of the method is as follows:

```
static data_type methodName(<parameter>)
{
}
```
## Developing The class Sales

<table>
<thead>
<tr>
<th>Class Name:</th>
<th>Sales</th>
</tr>
</thead>
</table>
| **Instance Variables:** | ✓ product  
✓ daily_sale  
✓ daily_quantity  |
| **Class Variables** | ✓ total_sale  
✓ total_quantity  |
| **Constructor** | ✓ Sales(product, cost, quantity)  |
| **Instance Methods** | ✓ getProduct()  
✓ getPieces()  
✓ getSale()  |
| **Class Methods** | ✓ getTotalSale()  
✓ getTotalQuantity()  |
Design class Sales.java – Instance variables vs Class variables

1. public class Sales
2. {
3.     // class variables
4.     private static double total_sale;
5.     private static int total_quantity;
6. 
7.     // Instance variables
8.     private String product;
9.     private double daily_sale;
10.    private int daily_quantity;
11. }
Design class Sales.java – The constructor

1. public class Sales
2. {
3.     // class variables
4.     private static double total_sale;
5.     private static int total_quantity;
6.     // Instance variables
7.     private String product;
8.     private double daily_sale;
9.     private int daily_quantity;
10.    public Sales(String name, double cost, int amount)
11.        {
12.            product = name;
13.            daily_sale = cost;
14.            daily_quantity = amount;
15.        }
16.        total_sale = total_sale + cost;
17.        total_quantity = total_quantity + amount;
18.    }
Design class Sales.java - Instance method vs. Class method

21. // instance method
22. public double getSale()
23. {
24.    return daily_sale;
25. }
26.
27. // instance method
28. public int get Pieces()
29. {
30.    return daily_quantity;
31. }
32.
33. // instance method
34. public String getProduct()
35. {
36.    return product;
37. }
38.
39. // class method
40. public static int getTotalQuantity()
41. {
42.    return total_quantity;
43. }
44.
45. // class method
46. public static double getTotalSale()
47. {
48.    return total_sale;
49. }
50. } // End of the class definition
The class sales is now completed, we now design the client class.

The report requires:

- A heading as we have already seen, as in - ABC Furnishing, Inc
- The current date as in - Oct 13, 2007
- Underlining, as we have already seen
- Adequate spacing between values, and using ( \t and \n )
- The dollar ($) sign as in - $4660.00
• Java has hundreds of classes that you can use for various things. Example:
  ✓ The class `Date` that generates the current date.
  ✓ The class `DateFormat` which has four forms of formatting the date
  ✓ The class `NumberFormat` which formats numbers in various ways, including currency

You must import them in your program in order to use them:

```java
import java.util.Date; // Used for creating a Date object
import java.text.DateFormat; // Used for specifying the format of the date
import java.text.NumberFormat; // Used for specifying the type of currency
```
To generate the current date you must create a `Date` object, as in:

```java
Date d = new Date();
```

Formatting the date object. There are four ways to format the date:

- **SHORT** 01/30/08
- **MEDIUM** Jan 30, 2008
- **LONG** January 30, 2008
- **FULL** Wednesday, January 30, 2008

Example:

```java
DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
System.out.println("Today is: " + df.format(d));
```

**Output:** Today is: Jan 30, 2008
To format currency including US ($) symbol use the statement:

```
NumberFormat nf = NumberFormat.getCurrencyInstance();
```

For instance, to format 123.457 to US currency we write:

```
NumberFormat nf = NumberFormat.getCurrencyInstance();
System.out.println("The amount of money is: "+ nf.format(123.457));
```

Output: The amount of money is: $123.46
Design client class - TestSales.java

1. // Import the files – Date.java, DateFormat.java, and NumberFormat.java
2.
3. class TestSales
4. {
5.     public static void main(String[] arg)
6.     {
7.         // Set up the formats for date and currency
8.
9.         // Print the heading
10.
11.        // Create Sales objects
12.
13.        // Display each object's information
14.
15.        // Display summary
16.    }
17. }

import java.util.Date; // Used for creating a Date object
import java.text.DateFormat; // Used for specifying the format of the date
import java.text.NumberFormat; // Used for specifying the type of currency

class TestSales {
    public static void main(String[] arg) {
        // Set up the formatters
        Date d = new Date();
        DateFormat df = DateFormat.getDateInstance();
        NumberFormat nf = NumberFormat.getCurrencyInstance();

        // Print the heading
        System.out.println("ABC Furnishing, Inc");
        System.out.println();
        System.out.println("Sales Report for " + df.format(d));
        System.out.println("\n");
        System.out.println("Product \tQuantity\tAmount($)");
        System.out.println("-----------------------------------------");
    }
}
// Create Sales objects
Sales w1 = new Sales("Chair", 1075.00, 20);
Sales w2 = new Sales("Table", 1155.00, 20);
Sales w3 = new Sales("Lamp", 175.00, 10);
Sales w4 = new Sales("Sofa", 2255.00, 40);

// Invoke the display method to display each object's information
display(w1);
display(w2);
display(w3);
display(w4);

// Display summary
System.out.println("-----------------------------------------");
System.out.println("Total items sold " + Sales.getTotalQuantity());
System.out.println("Total sale " + nf.format( Sales.getTotalSale() ));
System.out.println("------------- End of report ------------");
System.out.println("\n");

static void display(Sales w)
{
    System.out.println(w.getProduct() + "\t" + w.getPieces() +"\t" + w.getSale());
}

The concept of polymorphism is central to OOP

Polymorphic systems exhibit same behavior under different conditions

For instance, a calculator is a polymorphic system.

A typical behavior is multiplication.

Different conditions:

- Two integers, or
- Two floating-point values, or
- An integer and floating point value, or
- A floating point value and an integer

Result is the product of the quantities
Overloading and Polymorphism (cont)

• Constructors and methods can be defined in ways that the object behaves polymorphic.

• Polymorphism can occur through the concept of overloading

• Parameter defines overloading

• Constructor overloading – two or more constructors with different parameters

• Method overloading – two or more methods with the same name, but has different parameters
public class Calculator
{
    Calculator(int x, int y)
    {
    }
    Calculator(int x, double y)
    {
    }
    Calculator(double x, double y)
    {
    }
    void add(double x)
    {
    }
    void add(int x)
    {
    }
}
The Object Reference, *this*

- When an object is created, an implicit reference variable to this object is also created.

- Its name is called *this*

- It can be used refer to variables, methods, and constructors.

- For instance, a parameter and a member variable may have the same name.

- To differentiate between them, you use the *this* variable
Using Object Reference, *this* with Variables

1. public Circle
2. {
3.     private double radius;
4. }
5. public Circle(double radius)
6. {
7.     this.radius = radius;
8. }
9. }
Use Object Reference, *this* with Constructors

- Two or more constructors may create an object when they are overloaded.
- This approach avoids you from repeating existing codes.
- You can use the implicit reference object, *this* to carry out the initialization.
- The construct is as follows:
  
  ```
  this(< parameter>)
  ```
- When used, it must be the first executable statement in the calling constructor.
- In addition, the parameters must be compatible.
Using *this* with Constructors and Variables

1. public class Manufacturing
2. {
3.     private String name;
4.     private double price;
5. 
6.     public Manufacturing(String name)
7.         {
8.             this.name = name;
9.         }
10. 
11.    public Manufacturing(String name, double price)
12.        {
13.            this(name);
14.            this.price = price;
15.        }
16.    }
In Mathematics a fixed value is called a constant.

The value for pi (π) is a typical example.

In programming we use similar concept.

For instance, you might write a program that uses sales tax rate in some calculations.

Tax rates rarely change; hence, you can code sales tax as a constant.

In Java we use the modifier final to define constants.

The format for defining a constant is as follows:

```
<modifier> final data_type NAME_OF_CONSTANT = value;
```
Naming Conventions of Constants

JLS convention - name of constants should be a descriptive sequence of:
• One or more words
• Acronyms, or
• Abbreviations
• All uppercase letters, with words separated by underscore.

Here are some identifiers representing constants:
✓ MAX_VALUE
✓ PI
✓ TAX
✓ NAME_OF_CONSTANT
• A constant MUST be declared using the keyword final

• An appropriate value MUST be assigned to the identifier representing the constant.

```java
public class Circle {
    private double radius;
    private final double PI = 3.14;
}
```
Problem II

The ABC Cheap Lodging, Inc wants a computerized printout for each of its customers. Each input consists of the room number, the number of nights, and the number of customers. A customer may request additional night stay after the initial stay has expired. When registering, if customer a simply request a room then it is assumed that it’s only one person staying one night. On the other hand the customer may specify the length of stay. In this case it is assumed that there will be only one customer for that many nights; or the customer may specify the length of stay and the number of guests. See rate table below.

<table>
<thead>
<tr>
<th>Welcome to ABC Cheap Lodging, Inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rates Table</td>
</tr>
<tr>
<td>ROOM ………………$79.95 (per person)</td>
</tr>
<tr>
<td>TAX …………………6.5% (applies to room cost only)</td>
</tr>
<tr>
<td>TELEPHONE ……… $5.75 (Flat rate)</td>
</tr>
<tr>
<td>MEAL ……………... $12.95 (per person, per day)</td>
</tr>
<tr>
<td>TIP …………………7.5% (cover all charges)</td>
</tr>
</tbody>
</table>
Program Development II

Analyze the output

C: \ Program Files \ Xinox Software \ JCreatorV3LE \ GE200...

The ABC Cheap Lodging, Inc
Date: Oct 14, 2007
Room#: 12 - C
Room Rate: $79.95
Length of stay 2 nights
No. of guests: 2
Room cost: $159.90
Tax : 6.5% $10.39
Subtotal $170.29
Telephone $5.75
Meal charges $25.90
Tip $15.15
TOTAL AMOUNT DUE $217.09

Thanks for staying at The ABC Cheap Lodging, Inc
Please come again !!!
• Name of entity Hotel.
• Attributes:
  ✓ Constants:
  The problem description specifies five constants as shown in the rate table.

<table>
<thead>
<tr>
<th>ROOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAX</td>
</tr>
<tr>
<td>TELEPHONE</td>
</tr>
<tr>
<td>MEAL</td>
</tr>
<tr>
<td>TIP</td>
</tr>
</tbody>
</table>

Attributes Cont:
✓ Variables
The are possibly nine attributes associated with each object (customer)

- noOfNights
- noOfGuests
- amountDue
- meal
- tax
- subtotal
- total
- tip
- roomNumber
Constructors

There are three possible constructors

• **Single parameter** (room number) – one customer staying one night.

• **Two parameters** (room number and number of nights) – one customer staying so many nights

• **Three parameters** (room number, number of nights, and number of guests) – same as the former plus the number of guests.
Methods - possible actions that can be performed
• Accessor methods require for each variable and constant.

- ROOM
- TAX
- TELEPHONE
- MEAL
- TIP

- noOfNights
- noOfGuests
- amountDue
- meal
- tax
- subtotal
- total
- tip
- roomNumber

• A mutator method that carries out the necessary calculations.
Name: Hotel

Variables
noOfNights
noOfGuest
amountDue
meal
tax
subtotal
total
tip
roomNumber

Constructors
Hotel( roomNumber)
Hotel( roomNumber, nights)
Hotel( roomNumber, nights, guest)

Methods
getAmountDue()
getTaxDue()
getSubtotal()
subtotal()
getTip()
getMeal()
getRoomNumber()
getRoomRate()
getNumberOfNights()
getNumberOfGuests()
getPhoneCharges()
getTaxRate()
calculate()
add( nights ):  

public class Hotel {
    // Class constants
    private static final double ROOM_RATE = 79.95;
    private static final double TAX_RATE = 6.5;
    private static final double TELEPHONE = 5.75;
    private static final double MEAL_COST = 12.95;
    private static final double TIP_RATE = 0.075;

    // Instance variables
    private int noOfNights;
    private int noOfGuest;
    private double amountDue;
    private double meal;
    private double tax;
    private double subtotal;
    private double total;
    private double tip;
    private String roomNumber;
21. public Hotel(String room)  
22. {  
23.     roomNumber = room;  
24.     noOfGuest = 1;  
25.     noOfNights = 1;  
26. }  
27.  
28. public Hotel(String room, int nights)  
29. {  
30.     this(room);  
31.     noOfNights = nights;  
32. }  
33.  
34. public Hotel(String room, int nights, int guest)  
35. {  
36.     this(room, nights);  
37.     noOfGuest = guest;  
38. }
39. public void add(int nights)
40. {
41.     noOfNights = noOfNights + nights;
42. }
43.
44. public void calculate()
45. {
46.     amountDue = ROOM_RATE * noOfNights * noOfGuest;
47.     tax = amountDue * TAX_RATE/100;
48.     subtotal = amountDue + tax;
49.     meal = MEAL_COST * noOfNights;
50.     tip = TIP_RATE * (subtotal + meal + TELEPHONE);
51.     total = subtotal + TELEPHONE + meal + tip;
52. }
Design client class - TestHotel.java

1. // Import the files – Date.java, DateFormat.java, and NumberFormat.java
2.
3. class TestSales
4. {
5.    public static void main(String[] arg)
6.    {
7.        // Create Sales objects
8.        // Calculate the various amount
9.        // Display each object's information
10.       // Display summary
11.    }
12.    static void displayReceipt(Hotel h)
13.    {
14.        // Set up and display heading and date for each receipt
15.        // Display expenses line by line including subtotal
16.        // Display total
17.        // Display thank you message
18.    }
19. }
import java.util.Date;
import java.text.DateFormat;
import java.text.NumberFormat;

class TestHotel
{
    public static void main(String[] arg)
    {
        // Create customer objects, calculate amounts, display receipts
        Hotel customer1 = new Hotel("12 - B");
        customer1.calculate();
        displayReceipt(customer1);

        Hotel customer2 = new Hotel("12 - C", 2);
        customer2.calculate();
        displayReceipt(customer2);
    }

    static void displayReceipt(Hotel h)
    {
        // The body for the method goes here
    }
}