Writing Classes – Part 3

CSC 1051 – Data Structures and Algorithms I

Dr. Mary-Angela Papalaskari
Department of Computing Sciences
Villanova University

Course website:
www.csc.villanova.edu/~map/1051/
Getting to know classes so far

• Predefined classes from the Java API.

• We have defined a few classes of our own:
  – Account
  – Die
  – Book
  – Stock
Getting to know classes so far

• Predefined classes from the Java API.

• We have defined a few classes of our own:
  – Account
  – Die
  – Book
  – Stock

**Driver classes:**

  – Transactions, OnePercent (Lab 8)
  – RollingDice
  – Bookshelf (Project 8)
  – (HW exercises: Writing Classes)
## More Examples of Classes?

<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Name, Address, Major, Grade point average</td>
<td>Set address, Set major, Compute grade point average</td>
</tr>
<tr>
<td>Rectangle</td>
<td>Length, Width, Color</td>
<td>Set length, Set width, Set color</td>
</tr>
<tr>
<td>Aquarium</td>
<td>Material, Length, Width, Height</td>
<td>Set material, Set length, Set width, Set height, Compute volume, Compute filled weight</td>
</tr>
<tr>
<td>Flight</td>
<td>Airline, Flight number, Origin city, Destination city, Current status</td>
<td>Set airline, Set flight number, Determine status</td>
</tr>
<tr>
<td>Employee</td>
<td>Name, Department, Title, Salary</td>
<td>Set department, Set title, Set salary, Compute wages, Compute bonus, Compute taxes</td>
</tr>
</tbody>
</table>
Today

• Review what we learned so far
• Encapsulation
• UML diagrams
public class Transactions1
{
    //  Creates some bank accounts and requests various services.
    public static void main (String[] args)
    {
        Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
        Account acct2 = new Account ("Jane Smith", 69713, 40.00);

        System.out.println (acct1);
        System.out.println (acct2);

        acct1.deposit (25.85);
        System.out.println ();
        System.out.println (acct1);
        System.out.println (acct2);
    }
}
Review

• **class declaration**

```
long acctNumber;
double balance;
String name;
```

Data declarations

Constructor

```
deposit()
withdraw()
toString()
```

Method declarations
Review

• **Method definition**
  – **parameters**
  – **return type**
  – **return statement**

```java
char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt (sum);
    return result;
}
```

ch = obj.calc (25, count, "Hello");
Review

• using methods…
Review

Account class: Using methods

acct1.deposit (25.85);
acct1.deposit (25.85);

```java
public void deposit (double amount) {
    balance = balance + amount;
}
```

acct1

acctNumber 72354
balance 102.56
name "Ted Murphy"
Review

Account class: Using methods

```java
acct1.deposit (25.85);
//---------------------------------------------------
// Deposits the specified amount into the account.
//---------------------------------------------------
public void deposit (double amount)
{
    balance = balance + amount;
}
```

acct1

| acctNumber | 72354 |
| balance    | 102.56 |
| name       | "Ted Murphy" |
Review

Account class: Using methods

acct1.deposit (25.85);

// Deposit the specified amount into the account.
//
public void deposit (double amount)
{
    balance = balance + amount;
}
acct1.deposit (25.85);

```java
// Deposits the specified amount into the account.
public void deposit (double amount)
{
    balance = balance + amount;
}
```

Review
acct1.withdraw (60,2);
Account class: Another Example

```
acct1.withdraw (60, 2);
```

```java
// Withdraws the specified amount from the account
// and applies the fee.
public void withdraw (double amount, double fee) {
    balance = balance - amount - fee;
}
```

```
acct1
acctNumber 72354
balance 128.41
name "Ted Murphy"
```
Account class: Another Example

acct1.withdraw (60,2);

// Withdrawing the specified amount from the account and applying the fee.
public void withdraw (double amount, double fee) {
    balance = balance - amount - fee;
}

acct1

acctNumber: 72354
balance: 66.41
name: "Ted Murphy"
Account acct1 = new Account("Ted Murphy", 72354, 102.56);

```
public Account (String owner, long account, double initial) {
    acctNumber = account;
    balance = initial;
    name = owner;
}
```

acct1

acctNumber: 72354
balance: 102.56
name: "Ted Murphy"
Review

**toString() method**

```java
public String toString ()
{
    NumberFormat fmt = NumberFormat.getCurrencyInstance();
    return (acctNumber + "\t" + name + "\t" + fmt.format(balance))
}
```

```java
System.out.println(acct1);
```

<table>
<thead>
<tr>
<th>acctNumber</th>
<th>72354</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance</td>
<td>102.56</td>
</tr>
<tr>
<td>name</td>
<td>&quot;Ted Murphy&quot;</td>
</tr>
</tbody>
</table>
Invoking methods in same class

```java
public void paintSnowman(Graphics page, int x, int y) {
    // snowman code goes here
    . . .
}
```
Review

**static methods: 1) In same class**

```java
public static void doSomething()
{
    System.out.println(" At your service. ");
}
```
static methods: 2) In another class

public static void doSomething()
{
    System.out.println("At your service.");
}

OtherClass
What’s next?

- encapsulation and Java modifiers
- UML diagrams
Encapsulation

• An encapsulated object can be thought of as a *black box* -- its inner workings are hidden from the client

• The client invokes the interface methods and they manage the instance data
Violating Encapsulation

• It is possible for a class to access the instance data of another class directly
Violating Encapsulation - Example

- It is possible for a class to access the instance data of another class directly - *but don’t do this!*

- See Account.java (modified)
- See ImInUrClassMessingUrInstanceData.java

```java
acct1.name = "Joe";
```
Visibility Modifiers

• In Java, we enforce encapsulation through the appropriate use of *visibility modifiers*:
  - `public` – can be referenced from other classes
  - `private` – can be referenced only within that class:
  - `protected` – involves inheritance (discussed later)

• Data declared without a visibility modifier have *default visibility* and can be referenced by any class in the same package

• An overview of all Java modifiers is presented in Appendix E
public constants are ok - Example

Account acct1 = new Account ("Sartre", 72354, 102.56);

System.out.println (acct1);

System.out.println ("Interest rate = " + acct1.RATE);
## Visibility Modifiers – the **RULES**

<table>
<thead>
<tr>
<th></th>
<th>public</th>
<th>private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>NO</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>(but OK for public constants)</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>Yes</td>
<td>Yes, for support methods only</td>
</tr>
</tbody>
</table>
Encapsulation – Accessing the data

- Indirect access through methods
- Accessors and mutators (“getters” and “setters”)
- Usually named `getX()` and `setX()`

Example

`acct1.getBalance()`
Encapsulation – Accessing the data

- Mutators (setters) can restrict access to the data, as appropriate

- Example:

```java
public void setQuantity(int num)
{
    if (num<0)
    {
        System.out.println("*Error in setQuantity()");
        System.out.println("negative quantity.");
        System.out.println("quantity not changed.");
    }
    else
    {
        quantity = num;
    }
}
```
More Exercises on writing classes: Encapsulation and UML diagrams
Example: RollingDice.java (Driver)

```java
//********************************************************************
// RollingDice.java        Author: Lewis/Loftus
//
// Demonstrates the creation and use of a user-defined class.
//********************************************************************

public class RollingDice
{
    //-----------------------------------------------------------------
    // Creates two Die objects and rolls them several times.
    //-----------------------------------------------------------------
    public static void main (String[] args)
    {
        Die die1, die2;
        int sum;

        die1 = new Die();
        die2 = new Die();

        die1.roll();
        die2.roll();
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);

        continue
    }
}
```
continue

die1.roll();
die2.setFaceValue(4);
System.out.println("Die One: " + die1 + ", Die Two: " + die2);

sum = die1.getFaceValue() + die2.getFaceValue();
System.out.println("Sum: " + sum);

sum = die1.roll() + die2.roll();
System.out.println("Die One: " + die1 + ", Die Two: " + die2);
System.out.println("New sum: " + sum);
```java
continue

die1.roll();
die2.setFaceValue(4);
System.out.println("Die One: "+die1+", Die Two: "+die2);
sum = die1.getFaceValue() + die2.getFaceValue();
System.out.println("Sum: "+sum);

sum = die1.roll() + die2.roll();
System.out.println("Die One: "+die1+", Die Two: "+die2);
System.out.println("New sum: "+sum);
```
/**
 * Die.java
 * Author: Lewis/Loftus
 *
 * Represents one die (singular of dice) with faces showing values between 1 and 6.
 */

public class Die {
    private final int MAX = 6; // maximum face value

    private int faceValue; // current value showing on the die

    // Constructor: Sets the initial face value.
    public Die() {
        faceValue = 1;
    }
}
Example: Die.java

// Rolls the die and returns the result.
public int roll()
{
    faceValue = (int)(Math.random() * MAX) + 1;
    return faceValue;
}

// Face value mutator.
public void setFaceValue(int value)
{
    faceValue = value;
}

// Face value accessor.
public int getFaceValue()
{
    return faceValue;
}

continue
Example: Die.java

```java
continue

// Returns a string representation of this die.
public String toString()
{
    String result = Integer.toString(faceValue);

    return result;
}
```
UML Class Diagrams

• A UML class diagram for the RollingDice program:
UML Class Diagrams

• A UML class diagram for the RollingDice program:

```
RollingDice

main (args : String[]) : void

Die

faceValue : int

roll() : int
setFaceValue (int value) : void
getFaceValue() : int
toString() : String

die 1

faceValue

4
```
Homework

• Review Sections 4.1- 4.5
• Review the handout exercises and do the extra exercise with the Dog class at the end
• Review the example that violates encapsulation and make sure you understand it:
  – See Account.java (modified)
  – See ImInUrClassMessingUrInstanceData.java

• Do Exercises EX 4.1, 4.4, 4.10

Some slides in this presentation are adapted from the slides accompanying Java Software Solutions by Lewis & Loftus