Designing Classes – part 2

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 are also defining a few classes of our own:
  – Account
  – Die
  – Person
  – Book
Getting to know classes so far

• Predefined classes from the Java API.

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

Driver classes:
  – Transactions
  – RollingDice
  – PeopleBeingPeople
  – Bookshelf (Project 6)
## 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>
Next:

• Review what we learned so far
• Focus on method definition
• Encapsulation
• UML diagrams
Review

• **driver classes**

```java
// Transactions.java   Author: MA Papalaskari
// (based on Lewis/Loftus example)
// Demonstrates the creation and use of multiple Account objects.
//********************************************************************
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 (acct1);
        System.out.println (acct2);
    }
}
```
Review

• **class declaration**

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

- **Data declarations**
- **Method declarations**
  - Constructor
  - deposit()
  - withdraw()
  - toString()
Review

• using methods...
Account class: Using methods

acct1.deposit (25.85);
Account class: Using methods

acct1.deposit (25.85);

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

// Deposits the specified amount into the account.
public void deposit (double amount) {
    balance = balance + amount;
}
Account class: Using methods

acct1.deposit (25.85);

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

Review

acct1

acctNumber 72354

balance 102.56

name "Ted Murphy"
Review

Account class: Using methods

 acct1.deposit (25.85);

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

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>

CSC 1051 M.A. Papalaskari, Villanova University
Review

Account class: Another Example

acct1.withdraw (60,2);

acct1

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

acct1.withdraw (60, 2);

// 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"
 acct1.withdraw (60,2);

public void withdraw (double amount, double fee)
{
    balance = balance - amount - fee;
}
Account acct1 = new Account("Ted Murphy", 72354, 102.56);

public Account (String owner, long account, double initial) {
    acctNumber = account;
    balance = initial;
    name = owner;
}
Review **toString()** method

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

```
acct1
```

```
System.out.println(acct1);
```
Invoking methods in same class

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

`paintSnowman(page, 200, 150);`
Review

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

```java
OtherClass.doSomething();
```

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

CSC 1051 M.A. Papalaskari, Villanova University
Review

*static methods: 2) In same class*

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

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

CSC 1051 M.A. Papalaskari, Villanova University
Method Control Flow

- If the called method is in the same class, only the method name is needed.
Method Control Flow

- The called method is often part of another class or object
More Method Examples:

• Write a method that has one `int` parameter `num`, and prints “Happy Birthday” `num` times

• Write a method with two `double` parameters `a` and `b` that computes and returns the sum of squares: $a^2 + b^2$ of its two `int` parameters

• Write a method that has one `int` parameter `num`, and returns the String “Happy Birthday” `num` times
public class RollingDice
{
    // pretended comments
    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);
    }
}
Example: RollingDice.java (Driver)

```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);
}
```
Example: RollingDice.java (Driver)

```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);
```

Sample Run

```
Die One: 5, Die Two: 2
Die One: 1, Die Two: 4
Sum: 5
Die One: 4, Die Two: 2
New sum: 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;
    }
}
```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;
}
```
Example: Die.java

```java
continue

//  Returns a string representation of this die.

public String toString()
{
    String result = Integer.toString(faceValue);

    return result;
}
```
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
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 faceValue

4
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";
```

Transactions.java

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

Account.java

```java
deposit()
withdraw()
addInterest()
```
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);
public constants are ok - Example

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

System.out.println (acct1);
System.out.println ("Interest rate = " + acct1.RATE);

• Normally, constants are declared as static.
• If RATE had been declared as follows:
  final static double RATE = ;
• Then the last statement in this program would have been:
  System.out.println ("Interest rate = " + Account.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 (but OK for public constants)</td>
<td>Yes</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**

```java
acct1.getBalance()  
```

**Transactions.java**

```java
deposit()  
withdraw()  
addInterest()  
```

**Account.java**

```java
name  
acctNumber  
balance  
```
Encapsulation – Mutators (setters) can restrict access to the data, as appropriate

• Example: Say a class has an instance variable:

```java
private int quantity;
```

• The mutator may also work to ensure that `quantity`, does not become negative:

```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;
    }
}
```