Designing Classes

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/

Where do objects come from?

*Good question!*

*We will learn how to create a class that defines a new datatype, i.e., a new type of objects*

*We need to learn:*

1. What is the structure of a class definition?
2. How to specify what happens when an object is instantiated (i.e., when the `new` operator is used)?
3. How do we define the methods that can be invoked through objects of this class?

**Example:** *Account* datatype

- represents a generic bank account

```
acct1
acctNumber 72354
balance 102.56
name "Ted Murphy"

acct2
acctNumber 69713
balance 40.00
name "Jane Smith"
```
1. Structure of class definition

- The class is the blueprint.
- Classes define DATA and METHODS, i.e., a datatype.

2. Object instantiation

Creating Objects – old example:
- We have already seen something like this:
  ```java
  Scanner scan = new Scanner (System.in);
  ```
- This invokes the `Scanner constructor`, which is a special method that sets up the object.

Creating Objects – our newly defined `Account` class:
- A new `Account` object is created!
- Invokes the `Account constructor`, which is a special method that sets up the object.

3. Method invocation

• As we have seen, once an object has been created, we can use the dot operator to invoke its methods:

```java
ans = scan.nextLine();
numChars = title.length();
```

Chapter 4: Writing Classes

• We’ve been using predefined classes from the Java API. Now we will learn to write our own classes.
  – class definitions
  – constructors
  – instance data
  – method declaration and parameter passing
  – encapsulation and Java modifiers
  – more about creating graphical objects (next week)
import java.text.NumberFormat;
public class Account {
    String name;
    double balance;
    public Account (String x, int y, double z) {
        name = x;
        acctNumber = y;
        balance = z;
    }
    public void deposit (double x) {
        balance = balance + x;
    }
    public void withdraw (double x, double fee) {
        balance = balance - x - fee;
    }
    public String tostring () {
        return acctNumber + \" Edward \" name + \" Ted Murphy\" fee;
    }
}
public class Transactions {
    public static void main (String[] args) {
        Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
        Account acct2 = new Account ("Jane Smith", 69713, 40.00);
        Account acct3 = new Account ("Edward Demsey", 93757, 759.32);
        System.out.println (acct1);
        System.out.println (acct2);
        System.out.println (acct3);
        acct1.deposit (25.85);
        acct1.withdraw (60.00, 2.50);
        System.out.println ();
        System.out.println (acct1);
        System.out.println (acct2);
        System.out.println (acct3);
    }
}

public class Transactions {
    public static void main (String[] args) {
        Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
        Account acct2 = new Account ("Jane Smith", 69713, 40.00);
        Account acct3 = new Account ("Edward Demsey", 93757, 759.32);
        System.out.println (acct1);
        System.out.println (acct2);
        System.out.println (acct3);
        acct1.deposit (25.85);
        acct1.withdraw (60.00, 2.50);
        System.out.println ();
        System.out.println (acct1);
        System.out.println (acct2);
        System.out.println (acct3);
    }
}

Transactions class: Creating Account objects

```java
Account acct1 = new Account("Ted Murphy", 72354, 102.56);
```

Transactions class: Creating more Account objects

```java
acct1 = new Account("Ted Murphy", 72354, 102.56);
```

Transactions class: Creating Account objects

```java
public Account (String x, int y, double z) {
    name = x;
    acctNumber = y;
    balance = z;
}
```

Constructor method

```java
acct1 = new Account("Ted Murphy", 72354, 102.56);
acct2 = new Account("Jane Smith", 69713, 40.00);
```
Account class: Using methods

acct1.deposit (25.85);

acct1
acctNumber 72354
balance 102.56
name "Ted Murphy"

acct1.deposit (25.85);

acct1
acctNumber 72354
balance 102.56
name "Ted Murphy"
Designing Classes

Account class: Using methods

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

Account class: Another Example

```java
acct1.withdraw (60, 2.50);
//------------------------------------------------
//  Withdraws the specified amount from the account and applies the fee.
//------------------------------------------------
public void withdraw (double x, double fee)
{
    balance = balance - x - fee;
}
```

Account class: Another Example

```java
acct1.withdraw (60, 2.50);
//-------------------------------------------------
//  Withdraws the specified amount from the account
//   and applies the fee.
//-----------------------------------------------
public void withdraw (double x, double fee)
{
    balance = balance - x - fee;
}
```

Account class: Another Example

```java
acct1.withdraw (60, 2.50);
```
Class definitions

- A class can contain data declarations and method declarations

Data declarations

(double also called fields)

Constructor

deposit()
withdraw()
getBalance()
toString()

Method declarations

(note: the constructor is also a method)

toString() method

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

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

can be omitted!

toString() method

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

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

getBalance () method

```
double amount = acct1.getBalance();
// Note: this code is not used in Transactions.java
```

```
public double getBalance()
{
    return balance;
}
```
Bank Account Example

- There are some improvements that can be made to the `Account` class.
- The design of some methods could also be more robust, such as verifying that the `amount` parameter to the `withdraw()` method is positive.
- Some of these improvements are in the book examples.
- `Account.java`, `Transactions.java` (simplified versions)
- `Account.java`, `Transactions.java` (book versions)

```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);
      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("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

Another Example: RollingDice.java
Designing Classes

UML Class Diagrams

UML = Unified Modelling Language
- Example: A UML class diagram for the RollingDice program:

Next: Focus on **Method definition**
- parameters
- return type
- return statement

char ch = obj.calc(start, 2, "ABCDE");

char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt(sum);
    return result;
}
Method Control Flow
• If the called method is in the same class, only the method name is needed

Invoking methods within the same class
• An object’s method may access any of its other methods directly. Eg:

```java
public void addInterest(double rate) {
    deposit (rate*balance);
}
```

Method Control Flow
• The called method is often part of another class or object

• Thus the dot operator is an addressing mechanism. Note that it can also be used to access an object’s or class’s data directly, for example
  – acct1.name
  – Color.black
• more on this later (encapsulation)

More Method Examples:
• Write a method with two `double` parameters `a` and `b` that computes and returns the sum of squares of its two parameters (i.e., \(a^2 + b^2\)).

How do we invoke the method to compute & print: \((14.8)^2 + (37.65)^2\) ?
More Method Examples:

- Write a method with one `int` parameter `num`, that returns a String composed of “Happy Birthday” `num` times.

- How do we invoke the method to print “happy birthday” 4 times?

---

Getting to know classes so far

- Using predefined classes from the Java API.
- Defining classes for our own datatypes.

<table>
<thead>
<tr>
<th>Datatypes:</th>
<th>Clients (Driver classes):</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Account</td>
<td>- Transactions, OnePercent</td>
</tr>
<tr>
<td>- Die</td>
<td>- RollingDice</td>
</tr>
<tr>
<td>- Shoe</td>
<td>- YouVeGotShoes (Project)</td>
</tr>
<tr>
<td>- Person</td>
<td>- PeopleBeingPeople (Lab)</td>
</tr>
</tbody>
</table>

**Next:** Static methods, graphical objects, visibility modifiers… stay tuned!