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

1. Structure of class definition

The object:
- is like the house built from the blueprint
- is an instance of the class
- has its own data space & shares methods defined for this datatype

**Account**

- int acctNumber;
- double balance;
- String name;
- Constructor
- deposit()
- withdraw()
- getBalance()
- toString()

**Account class**

<table>
<thead>
<tr>
<th>Data</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>acctNumber</td>
<td>Constructor</td>
</tr>
<tr>
<td>balance</td>
<td>deposit()</td>
</tr>
<tr>
<td>name</td>
<td>withdraw()</td>
</tr>
<tr>
<td></td>
<td>getBalance()</td>
</tr>
<tr>
<td></td>
<td>toString()</td>
</tr>
</tbody>
</table>

**Account object**

- acctNumber 72354
- balance 102.56
- name "Ted Murphy"

Classes define DATA and METHODS i.e., a datatype
Designing Classes

2. Object instantiation

Old example:

```java
Scanner scan = new Scanner (System.in);
```

Invokes the Scanner constructor, which is a special method that sets up the object

New example:

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

Invokes the Account constructor, which is a special method that sets up the object

A new Account object is created!

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();
amount = acct1.getBalance();
acct1.deposit (25.85);
acct1.withdraw (2.50);
```

Datatype / Client (also referred to as "servant / driver" classes)
public void withdraw (double x, double fee) {
    balance = balance - x - fee;
}

public double getBalance ()
    return balance;

public String tostring ()
    NumberFormat fmt = NumberFormat.getCurrencyInstance();
    return acctNumber + \"\t\" + name + \"\t\" + fmt.format(balance);

Data, aka, Instance Variables

Constructor method

Sample Run

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>72354</td>
<td>$102.56</td>
</tr>
<tr>
<td>69713</td>
<td>$40.00</td>
</tr>
<tr>
<td>93757</td>
<td>$759.32</td>
</tr>
<tr>
<td>72354</td>
<td>$65.91</td>
</tr>
<tr>
<td>69713</td>
<td>$40.00</td>
</tr>
<tr>
<td>93757</td>
<td>$759.32</td>
</tr>
</tbody>
</table>

Transactions class: Creating Account objects

Transactions class

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 (70, 2.50);
System.out.println (acct1.tostring());
System.out.println (acct2.tostring());
System.out.println (acct3.tostring());
Transactions class:
Creating more Account objects

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

Account acct2 = new Account ("Jane Smith", 69713, 40.00);

Transactions class

acct1
accountNumber 72354
balance 102.56
name "Ted Murphy"

acct2
accountNumber 69713
balance 40.00
name "Jane Smith"

Transactions class

Using methods: deposit()

acct1.deposit (25.85);

acct1
accountNumber 72354
balance 128.41
name "Ted Murphy"

Using methods: withdraw()

acct1.withdraw (60, 2.50);

acct1
accountNumber 72354
balance 128.41
name "Ted Murphy"
Another example: **withdraw()**

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

// Withdraws the specified amount from the account
// and applies the fee

```java
public void withdraw (double x, double fee) {
    balance = balance - x - fee;
}
```

Another example:

```java
acct1.withdraw();
```

**returning a value: getBalance()**

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

```java
public double getBalance () {
    return balance;
}
```

**returning a value: toString() method**

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

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

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

```
72354   Ted Murphy      $102.56
```

**can be omitted!**

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

```
72354   Ted Murphy      $102.56
```

**returning a value: getBalance()**

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

```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
public class RollingDice {
    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("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
```
**NEXT: Focus on Methods**

- Common methods in Java
- Method control flow
- Method definition
  - Parameters
  - return statement
- UML class diagrams
- Encapsulation

**Common methods in Java classes**

- **Constructor** – always the same name as class, e.g.:
  - `public Account(String x, int y)`
  - `public Die()`
  - Always the same name as class
  - No return value
- **toString()** – returns a String corresponding to the object.
  - `public String toString()`
  - Always the exact same heading
- **getters (or accessors)** – return instance variable’s value.
  - `public int getFaceValue()`
  - No parameters
  - Return type is the same as the instance variable’s
- **setters (or mutators)** – to set or change an instance variable’s value
  - `public void setFaceValue(int value)`
  - One parameter, same type as instance variable.
  - Return type void

**Invoking methods within the same class**

- An object’s method may access any of the object’s other methods directly. Eg:

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

*Client code, eg: compound the interest for acct2 over 10 years*

```java
int year = 1;
while (year <= 10) {
    acct2.addInterest(0.03);
    acct2.deposit(0.03*balance);
    year ++;
}
```

**Method Control Flow: example**

- If the called method is in the same class, only the method name is needed.
Designing Classes

Method Control Flow (detail)

public void deposit(double x) {
    balance += x;
}

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

In general:

• 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)

Method definition: Example

– parameters
– return type
– return statement

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

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

Method Example:

Problem: Create a method levyTax() with no parameters. The method should levy tax on the account according to its value: no tax below $1000; 15% above $1000 but below $100,000; 30% above $100,000. The method should return the amount of tax levied

Client code example:
double tax1 = acct1.levyTax();
uncleSam.deposit(tax1);

Method code:
Getting to know classes so far

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

**Datatypes:**
- Account
- Die
- Shoe
- Person

**Clients (Driver classes):**
- Transactions, OnePercent
- RollingDice
- YouVeGotShoes (Project)
- PeopleBeingPeople (Lab)

Next: UML class diagrams, visibility modifiers, graphical objects,

---

### UML Class Diagrams

UML = Unified Modelling Language

- Example: A UML class diagram for the RollingDice program:

```
RollingDice

main (args : String[]) : void

Die

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

---

### Examples of datatypes (Classes)

<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Name, Address, Grade point average</td>
<td>Set address, Set major, Compute grade point average</td>
</tr>
<tr>
<td>Rectangles</td>
<td>Length, Width, Color</td>
<td>Set length, Set width, Set color</td>
</tr>
<tr>
<td>Aquarium</td>
<td>Material, Length, Weight</td>
<td>Set material, Set length, Set width, Compute volume, Compute weight</td>
</tr>
<tr>
<td>Flight</td>
<td>Airline, Right number, 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, Compute wages, Compute bonus, Compute bonus</td>
</tr>
</tbody>
</table>

---

CSC 1051 M.A. Papalaskari, Villanova University
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 which in turn manage the instance data.

Violating Encapsulation

It is possible for a class to access the instance data of another class directly – *but it's not a good idea!*

Use Accessors & Mutators - **RIGHT**

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

- Revisit the Account example (use our simplified versions from the course website or Lab 8)
- Add some code to the client (Transactions or Onepercent) to modify the value of an instance variable, eg:
  ```java
  acct1.name = “Bernie”;
  ```
- Run the program to verify that the name on that account has changed.
- Now modify Account.java – insert the modifier `private` in front of the instance variable declaration:
  ```java
  private String name;
  ```
- Without changing anything in the client, re-compile the Account class and run your program again. Note the error you get:
  ```
  Error: ____________________________________________________
  ```

Public Constants... OK

Example: The Account class can have a constant for the interest rate:
```java
public final double RATE = 0.015;
```

A client (eg, OnePercent.java) can access this constant directly:
```java
System.out.print("Interest rate = “ + acct1.RATE);
```

static Public Constants... BETTER

It is better to declare constants as `static`
```java
public final static double RATE = 0.015;
```

This way, a client can access the constants without creating an object, using the class name:
```java
System.out.print("Interest rate = “ + Account.RATE);
```

Visibility Modifiers – the RULES

<table>
<thead>
<tr>
<th>Visibility Modifier</th>
<th>Variables</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>NO</td>
<td>Yes</td>
</tr>
<tr>
<td>private</td>
<td>Yes</td>
<td>Yes, for support methods only</td>
</tr>
<tr>
<td>static</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

See also **ImInUrClassMessingUrInstanceData.java**