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

Account class

Data:

int acctNumber;
double balance;
String name;

Methods:

Constructor

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

"Ted Murphy"

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

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

Using our newly defined `Account` class:

```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);
```

```java
acct1 --- acctNumber: 72354
balance: 128.41
name: "Ted Murphy"
```

---

Datatype / Client (also referred to as "servant / driver" classes)
Designing Classes

Transactions class: Creating Account objects

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

acct1.deposit (25.85);
acct1.withdraw (60, 2.50);

acct1.toString();
acct2.toString();
acct3.toString();

CSC 1051 M.A. Papalaskari, Villanova University
Transactions class:
Creating more Account objects
Account acct1 = new Account ("Ted Murphy", 72354, 102.56);

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

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

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

Using methods: withdraw()
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;
}
Another example: **withdraw()**

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

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

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

**Returning a value: getBalance()**

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

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

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

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

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

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

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

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

```java
72354   Ted Murphy      $102.56
```

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

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

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

```java
72354   Ted Murphy      $102.56
```
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

- Method control flow
- Method definition
  - Parameters
  - return statement
- Different ways of thinking about methods

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)

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);
      year ++;
  }
  ```
**Method Control Flow: example**

```java
acct2.addInterest(0.03);
```

**Method definition: Example**

- parameters
- return type
- return statement

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

**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., $a^2 + b^2$).
- 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 compute & print: $(14.8)^2 + (37.65)^2$ ?

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.

**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
generateFaceValue : 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, Major, G.P.A.</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, Width, Weight</td>
<td>Set material, Set length, Set width, Compute volume, Compute weight</td>
</tr>
<tr>
<td>Flight</td>
<td>Airlines, Right number, Origin, Destination city, Current status</td>
<td>Set flight, 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 taxes</td>
</tr>
</tbody>
</table>
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()

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.
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, e.g:
  ```java
cacct1.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 (e.g., 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</th>
<th>Variables</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public</code></td>
<td>NO (but OK for public constants)</td>
<td>Yes, for support methods only</td>
</tr>
<tr>
<td><code>private</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See also `ImInUrClassMessingUrInstanceData.java`