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?
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;

- **Constructor**
  - deposit()
  - withdraw()
  - getBalance()
  - toString()

- **Methods**

**Account** object

- acctNumber: 72354
- balance: 102.56
- name: "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

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

CSC 1051 M.A. Papalaskari, Villanova University
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.
2. Object instantiation

Creating Objects – 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.
2. Object instantiation

Creating Objects – our newly defined `Account` class:

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

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();
```
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);
```
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)
Transactions.java
Author: MA Papalaskari
// (based on Lewis/Loftus example)
// Demonstrates the creation and use of multiple Account objects.

public class Transactions {
    //-----------------------------------------------------------------
    //  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);
        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, 2.50);

        System.out.println();
        System.out.println(acct1);
        System.out.println(acct2);
        System.out.println(acct3);
    }
}

Datatype / Client (also referred to as "slave / driver" classes)

Account.java
Author: Lewis/Loftus
//       Simplified code by MA Papalaskari
//  Represents a bank account with methods deposit and withdraw.

import java.text.NumberFormat;
public class Account {
    int acctNumber;
    double balance;
    String name;

    //---------------------------------------------------------------
    //  Sets up the account by defining its owner's name, account
    //   number, and initial balance.
    //---------------------------------------------------------------
    public Account(String x, int y, double z) {
        name = x;
        acctNumber = y;
        balance = z;
    }

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

    //---------------------------------------------------------------
    //  Withdraws the specified amount x from the account.
    //---------------------------------------------------------------
    public void withdraw(double x) {
        balance = balance - x;
    }

    //---------------------------------------------------------------
    //  Gets the current balance of the account.
    //---------------------------------------------------------------
    public double getBalance() {
        return balance;
    }

    //---------------------------------------------------------------
    //  Returns a string representation of the account.
    //---------------------------------------------------------------
    public String toString() {
        return name + " Account " + acctNumber + " Balance: 
            " + String.valueOf(new NumberFormat("#.##", java.text.DecimalFormatSymbols.getInstance()).format(balance));
    }
}

Transactions
Client

CSC 1051 M.A. Papalaskari, Villanova University
************

Account.java
Author: Lewis/Loftus
Simplified code by MA Papalaskari
Represents a bank account with methods deposit and withdraw.
************

import java.text.NumberFormat;

public class Account {
    int acctNumber;
    double balance;
    String name;

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

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

---

Constructor:
- deposit()
- withdraw()
- getBalance()
- toString()
// Withdrawing the specified amount from the account and applies the fee.
public void withdraw (double x, double fee)
{
    balance = balance - x - fee;
}

// Returns the current balance of the account.
public double getBalance ()
{
    return balance;
}

// Returns a one-line description of the account as a string.
public String toString()
{
    NumberFormat fmt = NumberFormat.getCurrencyInstance();
    return (acctNumber + "\t" + name + "\t" + fmt.format(balance));
}
public class Transactions
{
  // 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);
    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, 2.50);

    System.out.println (acct1);
    System.out.println (acct2);
    System.out.println (acct3);
  }
}
public class Transactions {
  
  //---// 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);
    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, 2.50);
    System.out.println ();
    System.out.println (acct1);
    System.out.println (acct2);
    System.out.println (acct3);
  }
}
Creating Account objects

Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
Transactions class: Creating Account objects

Transactions class

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

Constructor method

public Account (String x, int y, double z)
{
    name = x;
    acctNumber = y;
    balance = z;
}
Account acct1 = new Account ("Ted Murphy", 72354, 102.56);

```java
public class Account {
    private String name;
    private int acctNumber;
    private double balance;

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

    // Constructor method

    // Other methods...
}
```
Transactions class:

Creating more Account objects

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

Account acct2 = new Account ("Jane Smith", 69713, 40.00);
Account class: Using methods

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

acct1.deposit (25.85);

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

acct1

```java
acctNumber 72354
balance 102.56
name "Ted Murphy"
```
Account class: Using methods

acct1.deposit (25.85);

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

acct1

acctNumber  72354
balance     102.56
name         "Ted Murphy"
Account class: Using methods

acct1.deposit (25.85);

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

acct1.withdraw (60, 2.50);

acct1

acctNumber: 72354
balance: 128.41
name: "Ted Murphy"
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;
}
```

CSC 1051 M.A. Papalaskari, Villanova University
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;
}
```
Class definitions

• A class can contain data declarations and method declarations

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

**Constructor**

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

Data declarations
(also called *fields*)

Method declarations
(note: the constructor is also a method)
**toString() method**

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

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

 acct1

- acctNumber: 72354
- balance: 102.56
- name: "Ted Murphy"
**toString() method**

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

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

acct1

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

CSC 1051 M.A. Papalaskari, Villanova University
double amount = acct1.getBalance();
// Note: this code is not used in Transactions.java

public double getBalance ()
{
    return balance;
}

acct1

acctNumber 72354
balance 102.56
name "Ted Murphy"
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)
# 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, 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>
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("Die One: " + die1 + ", Die Two: " + die2);
        System.out.println("New sum: " + sum);
    }
}
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;
    }

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

continue
continue

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

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

//-----------------------------------------------------------------------------
//  Returns a string representation of this die.
//-----------------------------------------------------------------------------
public String toString()
{
    String result = Integer.toString(faceValue);
    return result;
}
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
```
UML class diagram for Transactions program?
Next: Focus on **Method definition**

- parameters
- return type
- return statement

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

```java
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

CSC 1051 M.A. Papalaskari, Villanova University
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 its 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);
```
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 \texttt{int} parameter \texttt{num}, that returns a String composed of “Happy Birthday” \texttt{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.

**datatypes:**
- Account
- Die
- Shoe
- Person

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

**Next:** visibility modifiers, graphical objects,
Encapsulation

- An encapsulated object can be thought of as a \textit{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.
Violating Encapsulation - **WRONG**

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

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

*CSC 1051 M.A. Papalaskari, Villanova University*
Use Accessors & Mutators - **RIGHT**

- Indirect access through methods
- **accessors and mutators** ("getters" and "setters")
- Usually named `getX()` and `setX()`

**Transactions.java**

```java
int x1 = acct1.getBalance();
```

**Account.java**

```java
deposit()
withdraw()
getBalance()
```
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, eg:

\[
\text{acct1.name} = \text{"Bernie"};
\]

• Run the program to verify that the name on that account has changed.
• Now modify Account.java – insert the modifier \texttt{private} in front of the instance variable declaration:

\[
\text{private String name;}
\]

• 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);
```
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>public</th>
<th>private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td><strong>NO</strong></td>
<td><strong>Yes</strong></td>
</tr>
<tr>
<td></td>
<td><em>(but OK for public constants)</em></td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td><strong>Yes</strong></td>
<td><strong>Yes, for support methods only</strong></td>
</tr>
</tbody>
</table>

See also [ImInUrClassMessinUrInstanceData.java](#)