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()`

**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**
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!**

- acct1
  - acctNumber: 72354
  - balance: 102.56
  - name: "Ted Murphy"
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();
```
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);
```
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;
    }
    //---------------------------------------------------------------
    // Returns the current balance.
    //---------------------------------------------------------------
    public double getBalance(){
        return balance;
    }
    //---------------------------------------------------------------
    // Returns a string representation of the account.
    //---------------------------------------------------------------
    public String toString(){
        NumberFormat fmt = java.text.NumberFormat.getCurrencyInstance();
        return "Name: " + name + " Acct #: " + acctNumber + " Balance: 
               " + fmt.format(balance);";
    }
}
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;
    }

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

  //-----------------------------------------------------------------
  //  Withdraws 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));
  }
  
  Constructor
  deposit()
  withdraw()
  getBalance()
  toString()
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, 2.50);

        System.out.println ();
        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);
    System.out.println (acct1);
    System.out.println (acct2);
    System.out.println (acct3);
  }
}
Transactions class:

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);

Account class

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

Constructor method

Constructor

deposit()
withdraw()
getBalance()
toString()
Transactions class:

Creating Account objects

Transactions class

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

Account class

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

Constructor method

double balance;
int acctNumber;
String name;

Constructor

deposit()
withdraw()
getBalance()
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);
Account class: Using methods

\[ \text{acct1}.\text{deposit} \ (25.85); \]

 acct1

 acctNumber 72354
 balance 102.56
 name

 "Ted Murphy"

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

acctNumber: 72354
balance: 102.56
name: Ted Murphy
acct1.deposit(25.85);

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

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

 acct1

 acctNumber 72354

 balance 128.41

 name

 "Ted Murphy"
acct1.withdraw (60, 2.50);

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

"Ted Murphy"
Account class: Another Example

acct1.withdraw (60, 2.50);

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

acct1

acctNumber 72354
balance 65.91
name "Ted Murphy"
Class definitions

• A class can contain data declarations and method declarations

```
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)*

CSC 1051 M.A. Papalaskari, Villanova University
System.out.println(acct1.toString());

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

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

 acct1 →

 acctNumber 72354

 balance 102.56

 name "Ted Murphy"

CSC 1051 M.A. Papalaskari, Villanova University
**getBalance() method**

```java
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"

CSC 1051 M.A. Papalaskari, Villanova University
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</td>
<td>Set address</td>
</tr>
<tr>
<td></td>
<td>Address</td>
<td>Set major</td>
</tr>
<tr>
<td></td>
<td>Major</td>
<td>Compute grade point average</td>
</tr>
<tr>
<td></td>
<td>Grade point average</td>
<td></td>
</tr>
<tr>
<td>Rectangle</td>
<td>Length</td>
<td>Set length</td>
</tr>
<tr>
<td></td>
<td>Width</td>
<td>Set width</td>
</tr>
<tr>
<td></td>
<td>Color</td>
<td>Set color</td>
</tr>
<tr>
<td>Aquarium</td>
<td>Material</td>
<td>Set material</td>
</tr>
<tr>
<td></td>
<td>Length</td>
<td>Set length</td>
</tr>
<tr>
<td></td>
<td>Width</td>
<td>Set width</td>
</tr>
<tr>
<td></td>
<td>Height</td>
<td>Set height</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute volume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute filled weight</td>
</tr>
<tr>
<td>Flight</td>
<td>Airline</td>
<td>Set airline</td>
</tr>
<tr>
<td></td>
<td>Flight number</td>
<td>Set flight number</td>
</tr>
<tr>
<td></td>
<td>Origin city</td>
<td>Determine status</td>
</tr>
<tr>
<td></td>
<td>Destination city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Current status</td>
<td></td>
</tr>
<tr>
<td>Employee</td>
<td>Name</td>
<td>Set department</td>
</tr>
<tr>
<td></td>
<td>Department</td>
<td>Set title</td>
</tr>
<tr>
<td></td>
<td>Title</td>
<td>Set salary</td>
</tr>
<tr>
<td></td>
<td>Salary</td>
<td>Compute wages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute bonus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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);
    }
}
Die.java
Author: Lewis/Loftus

// Represents one die (singular of dice) with faces showing values between 1 and 6.

public class Die
{
    private final int MAX = 6; // maximum face value

    private int faceValue; // current value showing on the die

    public Die()
    {
        faceValue = 1;
    }

    public int roll()
    {
        faceValue = (int)(Math.random() * MAX) + 1;
        return faceValue;
    }
}

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
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);
}
```
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.

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

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

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