Using Classes and Objects

Lecture 8

Some slides borrowed from Kurt Eiselt, Tamara Munzner, and Steve Wolfman. Some learning goals from Beth Simon.

Readings

This Week’s Reading: Ch 2.6-2.10, Finish Ch 4

Next Week’s Reading: Ch 3.1-3.8
(Major conceptual jump…)

Labs and Tutorials

This week is Lab #2.

Lab #3 is up.

Midterms – Save the Dates!

- Midterm #1 is 5:30-6:30pm on February 10 (Tuesday) in Woodward IRC 2
- Midterm #2 is 6-7pm on March 11 (Wednesday) in Woodward IRC 2

You must notify me well in advance (I will specify deadlines as the dates approach) if you have an unavoidable conflict with a midterm.

Extra Credit Survey

- Dr. Ben Yu is studying attitudes towards learning in introductory CS classes.
- Three surveys to complete during the term.
- Survey 1 is on WebCT now. Due January 28
- These are completely optional.

- However, to encourage participation, I will give 1 pt of extra credit, applied to your labs, if you complete all three surveys in time.

Learning Goals

By the end of the next several lectures you will be able to...

- Explain the concepts of “class”, “object”, and “method” as used in computer programming.
- Find pre-defined classes in the Java library, understand the documentation, and use the methods for basic classes.
- Write programs that use common Java classes like BigInteger, String, and Scanner.
Learning Goals
By the end of class today you will be able to...

- Identify the basic components of a method call.
- Describe the following concepts relating to methods in Java: parameters, implicit and explicit parameters, return value, static methods, and the void return type.
- Read Java documentation about methods, figure out the correct types for all parameters and the return value, and therefore write programs that call the method correctly.

(last paragraph of Learning Goals)

Last Time...

- We got the basic vocabulary of object-oriented programming: classes, objects, and methods.
- We worked through an example program using a class from the Java library (specifically, the BigInteger class, but the important things are the general patterns of how to use a class, more than the specifics of the BigInteger class).
- We studied constructors in more depth: the syntax for a constructor, what it does, and why we have them.

Review: Programming with Classes

- What if data we want to work with is more complex than these few primitive data types?
  - We can make our own data type:
    - specifies nature of data we want to work with
    - operations that can be performed on that kind of data
  - (Other people can make data types, too, and we can use their work.)

Review: Object-Oriented Terminology

- For historical reasons, there are special terms for these new data types we create:
  - The new data types are called **classes**.
  - Values with the new data type are called **objects**.
  - The operators for the new data type are called **methods**.
  - (The programming style we are teaching, which is the dominant style in industry today, is called “object-oriented programming”.)

Review: Primitive Types vs. Classes

<table>
<thead>
<tr>
<th>Primitive Types</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-defined in Java</td>
<td>Written by other programmers or by you</td>
</tr>
<tr>
<td>Simplest things, e.g., int</td>
<td>Can be arbitrarily complex</td>
</tr>
<tr>
<td>Operators: +, -, ...</td>
<td>Methods</td>
</tr>
<tr>
<td>Values belong to types. E.g., 3 is an int, 3.14159 is a double</td>
<td>Objects belong to classes E.g., you are a UBC student</td>
</tr>
<tr>
<td>Literals</td>
<td>Constructors</td>
</tr>
</tbody>
</table>

Review: API Documentation

- Online Java library documentation at http://java.sun.com/javase/6/docs/api
  - let’s take a look!
- Everything we need to know: critical details
  - and often many things far beyond current need
- Classes in libraries are often referred to as Application Programming Interfaces
  - or just API
Review: Using classes

- Import the name of the class you want:
  import java.math.BigInteger;
- Declare variables using the class name:
  BigInteger salary;
- Create values (objects) of the new type/class by using constructors:
  salary = new BigInteger("111222333");
- Operate on the objects using methods:
  totalCompensation = salary.add(bonus);

Review: GetRichQuick

import java.math.BigInteger; //Tell Java to use BigInteger pkg
public class GetRichQuick {
    public static void main(String[] args) {
        BigInteger salary = new BigInteger("111222333444555666777888999");
        BigInteger stockOptions = new BigInteger("10000000000000000000000");
        BigInteger profitPerShare = new BigInteger("3000000000000000000000");
        BigInteger optionCompensation = stockOptions.multiply(profitPerShare);
        BigInteger totalCompensation = salary.add(optionCompensation);
        System.out.println("Total Compensation = $" +
                           totalCompensation);
    }
}

Review: Using Constructors

- Use a constructor just as you’d use a literal.
  Example:
  For the int type:
    int a = 3;
  For the BigInteger class:
    BigInteger a = new BigInteger("3");

Focus on Constructors

- Import the name of the class you want:
  import java.math.BigInteger;
- Declare variables using the class name:
  BigInteger salary;
- Create values (objects) of the new type/class by using constructors:
  salary = new BigInteger("111222333");
- Operate on the objects using methods:
  totalCompensation = salary.add(bonus);

Focus on Methods

- Import the name of the class you want:
  import java.math.BigInteger;
- Declare variables using the class name:
  BigInteger salary;
- Create values (objects) of the new type/class by using constructors:
  salary = new BigInteger("111222333");
- Operate on the objects using methods:
  totalCompensation = salary.add(bonus);

Anatomy of a Method Call

- People say “call” or “invoke” a method:
  - As in “call a plumber”, “call a doctor”, “invoke the Charter of Rights”, or “invoke demons from Hell”
  - Shared meaning: getting someone or something else to get something done.
  - (“Something else” is the Java code by the person who created the class. More next week.)
Anatomy of a Method Call

totalCompensation = salary.add(bonus);

The object determines what methods are available to act on it. E.g., BigInteger objects have an add method.

Anatomy of a Method Call

totalCompensation = salary.add(bonus);

“Add to salary.”
“Hit the ball.”
“Beat the drum.”

Anatomy of a Method Call

totalCompensation = salary.add(bonus);

Stuff in parentheses after the method name. Provide additional information to method. Can be many parameters, depending on method. Notation like math functions: \( \cos(\theta), f(x,y,z) \)
Anatomy of a Method Call

```java
totalCompensation = salary.add(bonus);
```

Parameters provide information to the method. The object also provides information. Shouldn’t we count the object as a parameter?

Explicit parameters are between ( ). Object itself is the implicit parameter. Implicit parameter also specifies which method.

The method call evaluates to some value: in this case, the BigInteger representing the sum of salary and bonus.

We say the method call returns a value of type BigInteger, or returns a BigInteger.

Return value is just a value/expression that you can use like any other value or expression. In this case, we assign it to a variable.

Is this OK?

```java
BigInteger totalCompensation = salary.add(
    stockOptions.multiply(profitPerShare)
);
```

How about this?

```java
BigInteger totalCompensation =
    (stockOptions.multiply(profitPerShare)).
    add(salary);
```
How about this?

```java
BigInteger totalCompensation =
    stockOptions.multiply(profitPerShare)) .
    add(salary);
```

Return value from this method call is an object of type BigInteger.

Objects of type BigInteger have an add method.

No Explicit Parameters?

- Can a method call have no explicit parameters?
- Sure! If the method doesn’t need them.
  E.g., absolute value for BigIntegers:
    ```java
dollarVolume = gainOrLoss.abs();
```
- Just put nothing between parentheses.
- Java documentation tells you what parameters are needed. (The implicit parameter is called this.)

No Implicit Parameter?

- Can a method have no implicit parameter?
- Sure! These are called static methods.
  - (The name is historical. Things that get created when the program runs are called dynamic. Things that the compiler creates at compile time are called static. Objects get created when the program runs (using constructors), so things that are associated with objects are called dynamic. So a method that’s not attached to an object is called static)

Everything the method needs must be an explicit parameter:

```java
long a = Math.round(1.75);
```
Static Methods
- Static methods have no implicit parameter.
- How do we call a static method, if we don’t name an object?

How about no parameters at all?
- What can you do if you don’t specify anything to work on?

What about no return value?
- totalCompensation = salary.add(bonus)

Sometimes you want a return value:
- “Add the salary and the bonus together.”
- Sometimes you don’t want a return value:
  - “Kick me.”
  - transcript.addGrade(“CPSC 111”, 98);
  - The method changes the state of the object (the implicit parameter).

void methods
- A method with no return value is said to have type void (as in emptiness).
  - In the Java documentation, the return type will be shown as void.
- Since it has no return value, you can’t use it in an expression.
- Instead, you use it as a statement.

void methods
- Instead, you use it as a statement:
  - System.out.println(“Hello, World!”);
  - Rectangle box = new Rectangle();
  - box.setLocation(100, 200);
  - box.translate(10, 10);

(See java.lang.System and java.awt.Rectangle)
Different Styles of Methods

- Contrast the add method for java.awt.Rectangle and java.math.BigInteger

- Neither way is better than the other. The way a class is designed should be natural for how programmers think about the concept represented by the class.

- Check the documentation to know how to use the methods.

Questions?

Change of Gears...

Program Development

- Use an editor to create your Java program
  - often called source code
  - code used interchangeably with program or instructions in the computer world

- Another program, a compiler or an interpreter, translates source code into target language or object code, which is often machine language

- Finally, your computer can execute object code

Syntax

- Rules to dictate how statements are constructed.
  - Example: open bracket needs matching close bracket
  - If program is not syntactically correct, cannot be translated by compiler
  - Different than humans dealing with natural languages like English. Consider statement with incorrect syntax (grammar)
    
    for weeks, rained in Vancouver it hasn’t
    
  - we still have pretty good shot at figuring out meaning

Semantics

- What will happen when statement is executed
- Programming languages have well-defined semantics, no ambiguity
- Different than natural languages like English. Consider statement:
  
  Mary counted on her computer.
  
  How could we interpret this?

- Programming languages cannot allow for such ambiguities or computer would not know which interpretation to execute
Errors

- Computers follow our instructions exactly
- If program produces the wrong result, it’s the programmer’s fault
  - unless the user inputs incorrect data
  - then cannot expect program to output correct results: “Garbage in, garbage out” (GIGO)
- Debugging: process of finding and correcting errors
  - Unfortunately can be very time consuming!

- Error at compile time (during translation)
  - you did not follow syntax rules that say how Java elements must be combined to form valid Java statements

- Error at run time (during execution)
  - Source code compiles
    - Syntactically (structurally) correct
  - But program tried something computers cannot do
    - like divide a number by zero.
  - Typically program will crash: halt prematurely

Questions?