Using Classes and Objects

Lecture 7

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

Readings

Your textbook is *Big Java* (3rd Ed).
This Week’s Reading: Ch 2.6-2.10, Finish Ch 4

Labs and Tutorials

This week is Lab #2. It has a pre-lab!

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...

- Use constructors to create new objects, and explain why this is necessary.
- Identify the basic components of a method call.
- (Explain what syntax, runtime, and logical errors are.)

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: 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., a Toyota Prius is a car.</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

Example: BigInteger
- Let's write a simple program to print out how much money you'd make if your compensation package is a combination of salary and stock options.

```java
public class GetRichQuick {
    public static void main(String[] args) {
        int salary = 123456;
        int stockOptions = 10000;
        int profitPerShare = 30;

        int optionCompensation = stockOptions * profitPerShare;
        int totalCompensation = salary + optionCompensation;

        System.out.println("Total Compensation = "+
                            totalCompensation);
    }
}
```
GetRichQuick 2

public class GetRichQuick {
    public static void main(String[] args) {
        int salary = 11122233444555666777888999;
        int stockOptions = 10000000000000000000000;
        int profitPerShare = 3000000000000000000000;
        int optionCompensation = stockOptions * profitPerShare;
        int totalCompensation = salary + optionCompensation;
        System.out.println("Total Compensation = $" +
                           totalCompensation);
    }
}

GetRichQuick w/ BigInteger

import java.math.BigInteger; //Tell Java to use BigInteger pkg

public class GetRichQuick {
    public static void main(String[] args) {
        BigInteger salary = new BigInteger("11122233444555666777888999");
        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);
    }
}

GetRichQuick: Constructors

import java.math.BigInteger; //Tell Java to use BigInteger pkg

public class GetRichQuick {
    public static void main(String[] args) {
        BigInteger salary = new BigInteger("11122233444555666777888999");
        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);
    }
}

GetRichQuick: Methods

import java.math.BigInteger; //Tell Java to use BigInteger pkg

public class GetRichQuick {
    public static void main(String[] args) {
        BigInteger salary = new BigInteger("11122233444555666777888999");
        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);
    }
}

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

Literals
- With the primitive types, how do you create values with that type?
  E.g., how do we create integer values?
  1. You type some digits, like 3, or 42
  2. You combine integer-valued things with operators that work on integers, e.g.,
     ```java
     3+42*(a-b)
     ```

More Literals
- How about a value of type double?
  1. You type a bunch of digits with a decimal point, and optionally the letter e or E followed by an exponent
  2. You can combine doubles with operators that work on doubles.

Those are literals!

Long Literals
- How about values of type long?
  1. You type a bunch of digits followed by the letter l or L
  2. You combine previously created longs
**Literals – General Pattern**
- To create values of a primitive type:
  1. There’s some way to type a literal
  2. There are operators that create values of the given type.

**Literals for Classes?**
- Classes are like primitive types, except they can be defined any way you like, and they can be much more complex.
- How to create a value (an object) of a given class?
  1. Invent some way to type a literal???
  2. Operators that create objects of that class (methods).

**Constructors!**
- A constructor is the equivalent of a literal for a class. It’s how you create a new object that belongs to that class.
- Examples:
  - `new BigInteger("999999")`
  - `new Rectangle(10, 20, 30, 40)`
  - `new UBCStudent("Alan Hu",12345678,...)`

**Constructor Syntax**
- The reserved word `new`
- Followed by the name of the class
- Followed by an open parenthesis ( 
- Followed by any parameters (information needed to construct the object)
- Followed by a closing parenthesis )

**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");`

**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., <code>int</code></td>
<td>Can be arbitrarily complex</td>
</tr>
<tr>
<td>Operators: <code>+</code>, <code>-</code>, ...</td>
<td>Methods</td>
</tr>
<tr>
<td>Values belong to types. E.g., 3 is an <code>int</code>, 3.14159 is a <code>double</code></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>
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

- totalCompensation = salary.add(bonus);
- 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);
  - object
  - method

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);
```

- **object** (noun)
- **method** (verb)

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

Stuff in parentheses after the method name.
Provide additional information to method.
Can be many parameters, depending on method.
Notation like math functions $f(x,y,z) = xy+z$

Anatomy of a Method Call

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

- **object** (noun)
- **method** (verb)
- **parameters**

Change of Gears...

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

- **object** (noun)
- **method** (verb)
- **parameter**

“Add bonus to salary.”
“Hit the ball deep to the backhand corner.”
“Beat the drum lightly in triple time.”

---

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 follows 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!

Errors
- 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

Errors
- 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

Errors
- Logical error
  - Source code compiles
  - Object code runs
  - But program may still produce incorrect results because logic of your program is incorrect
    - Typically hardest problems to find

Questions?