Introduction to Classes & Objects

Lecture 7

Borrowing from slides by Alan Hu, Kurt Eiselt, Paul Carter, and Tamara Munzner
Announcements

- 1\textsuperscript{st} Midterm Exam: Wed, Oct 7, 6:30 pm
  - Location TBA
  - Two weeks from today!
  - Duration: 60 minutes
  - Material: book chapters 1-4
  - Sample exams now on WebCT
- 2\textsuperscript{nd} Midterm Exam: Wed, Nov 4, 6:30 pm
- 1\textsuperscript{st} Homework
  - Will be on WebCT later this week
Reading Assignments

- For this week, read
  - Ch 4.7 (both editions)
  - Ch 3 (both editions)
  - Note: updated reading (mistakes in some versions posted last week)!
Converting Between Types

- Which of these are legal?
  - `int shoes = 2;`
  - `double socks = 1.75;`
  - `double socks = 1;`
  - `int shoes = 1.5;`
- Integers are subset of reals
  - but reals are not subset of integers
- Java automatically converts `int` to `double` when needed (or smaller size to larger size)
Casting

- **Casting**: force Java to convert from one type to another, *even with information loss*
- Converting from real to integer
  - `int shoes = (int) 1.5;`
- Truncation: fractional part thrown away
  - `int shoes = (int) 1.75;`
  - `int shoes = (int) 1.25;`
- Rounding: must be done explicitly
  - `shoes = Math.round(1.99);`
Recap: Constants

```java
public static void main (String[] args) {
    double lightYears, milesAway;
    final int LIGHTSPEED = 186000;
    final int SECONDS_PER_YEAR = 60*60*24*365;

    final double ALPHACENT_DIST = 4.35; // to AlphaCentauri
    final double ALDEBARAN_DIST = 68; // to Aldebaran

    lightYears = ALPHACENT_DIST;
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: "+ lightYears + " miles " + milesAway);
    lightYears = ALDEBARAN_DIST;

    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: "+ lightYears + " miles " + milesAway);
}
```
Recap: Avoiding Magic Numbers

- **magic numbers**: numeric constants directly in code
  - almost always bad idea!
    - hard to understand code
    - hard to make changes
    - typos possible
  - use constants instead
Recap: Syntax vs. Semantics

- **Syntax:**
  - Rules to dictate how statements are constructed.
    - Example: open bracket needs matching close bracket
  - Somewhat like grammar in a natural language
  - If program is not syntactically correct, cannot be translated by compiler

- **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.
    ```
Recap: 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
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
Recap: 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
Today's Objectives

- Understand the rationale behind classes and objects.
- Understand the concepts of (and differences and relationships between):
  - Classes
  - Objects
  - Primitive Data Types
- Learn how to use existing classes
Programming

- Programming is all about specifying
  - data that is to be manipulated or acted upon
  - operations that can act upon data
  - order in which operations are applied to data

- So far: specify data using primitive data types
  - come with pre-defined operations like
    +, -, *, and /
Programming with Classes

- What if data we want to work with is more complex than these few primitive data types?
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: create a class
  - specifies nature of data we want to work with
  - operations that can be performed on that kind of data
- Operations defined within a class called methods
Example Class: Social Clubs
Example Class: UBC Students
Classes

- A class has a name.
- A class should describe something intuitively meaningful. Why did someone create this class?
- A class describes the data stored inside objects in the class. (Nouns)
- A class describes the legal operations that can be done to the data. (Verbs)
- Example in Book: java.awt.Rectangle
## 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: +, -, ...</td>
<td>Methods</td>
</tr>
<tr>
<td>Values belong to types.</td>
<td><strong>Objects</strong> belong to classes</td>
</tr>
<tr>
<td>E.g., 3 is an <code>int</code>, 3.14159 is a <code>double</code></td>
<td>E.g., you are a UBC Student</td>
</tr>
</tbody>
</table>
Objects Belong to Classes

- Just as 1, 2, and 3 are all integers, you are all objects of the class UBCStudent!
  - You each have names, ID numbers, etc.
  - Each is unique person, but all are students

- Social organizations example:
  - Ballroom Dance Club
  - Ski Club
  - CSSS
  - Etc.

- Sometimes called “instances” of a class.
Class Libraries

- Before making new class yourself, check to see if someone else did it already
  - libraries written by other programmers
  - many built into Java
- Examples (built into Java)
  - BigInteger (java.math.BigInteger) lets you compute with arbitrarily big integers.
  - Date (java.util.Date) lets you get the current time.
  - Calendar (java.util.Calendar) does fancy date computations.
API Documentation

- Online Java library documentation at [http://java.sun.com/j2se/1.5.0/docs/api/](http://java.sun.com/j2se/1.5.0/docs/api/)
  - textbook alone is only part of the story
  - 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
import java.math.BigInteger; // Tell Java to use standard BigInteger package

class GetRichQuick {
    public static void main(String[] args) {
        BigInteger salary = new BigInteger("1112233444555666777888999");
        BigInteger stockOptions = new
            BigInteger("100000000000000000000000000000000");
        BigInteger profitPerShare = new BigInteger("314159");

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

        System.out.println("Total Compensation = $" +
            totalCompensation.toString());
    }
}
import java.math.BigInteger; // Tell Java where to find BigInteger package

public class GetRichQuick {
    public static void main(String[] args) {
        BigInteger salary = new BigInteger("11122334455666777888999");
        BigInteger stockOptions = new
            BigInteger("100000000000000000000000000000000");
        BigInteger profitPerShare = new BigInteger("314159");

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

        System.out.println("Total Compensation = $" +
            totalCompensation.toString());
    }
}
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.,
   \[3 + 42 \times (a - b)\]
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.,
   
   $3 + 42 \times (a-b)$

A bunch of digits are an integer literal.

It’s the basic way to create an integer value
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.
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!
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
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:
  
  ```java
  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 an 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:
    ```java
    int a = 3;
    ```
  - For the BigInteger class:
    ```java
    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>, <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>
import java.math.BigInteger; // Tell Java where to find BigInteger package

public class GetRichQuick {
    public static void main(String[] args) {
        BigInteger salary = new BigInteger("111222333444555666777888999");
        BigInteger stockOptions = new
            BigInteger("100000000000000000000000000000000");
        BigInteger profitPerShare = new BigInteger("314159");
        
        BigInteger optionCompensation = stockOptions.multiply(profitPerShare);
        BigInteger totalCompensation = salary.add(optionCompensation);
        
        System.out.println("Total Compensation = "+
            totalCompensation.toString());
    }
}