Data Types, Assignment, Casting, Constants

Lecture 5, Fri Jan 15 2010

borrowing from slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/111-10
Reading This Week

- Chap 1: 1.3-1.8
- Chap 2: 2.1-2.2, 2.5
- Chap 4: 4.1-4.2

- reminder: weekly reading questions due now!
Resources

- Demco Learning Center
  - ICICS/CS x150
  - Learning center is ramping up starting Monday
  - Normal hours: 10 am - 6 pm M-Th, 10 am - 4 pm F
    - Staffed by TAs from all 1st year courses
    - Drop by if you have any questions
More Resources

- WebCT discussion groups
  - Monitored by TAs/instructor, use to ask questions

- don’t forget to check web page first/often!
  - lecture slides, handouts, schedule, links, ....
  - http://www.cs.ubc.ca/~tmm/courses/111-10
Yet More Resources

- reminder: my office hours Fridays 4-5pm, starting today

- office location is X661 (tall wing of ICICS/CS bldg)
Recap: White Space

- **White space**
  - Blanks between identifiers and other symbols
  - Tabs and newline characters are included

- White space does not affect how program runs

- Use white space to format programs we create so they’re easier for people to understand
Recap: Errors

- Compile-time errors
  - syntax/structure
- Run-time errors
- Logical errors
  - semantics/meaning
Recap: Variables

- Variable: name for location in memory where data is stored
  - avoid having to remember numeric addresses
  - like variables in algebra class

- Variable names begin with lower case letters
  - Java convention, not compiler/syntax requirement
Recap: Data Types

- Java requires that we tell it what kind of data it is working with.
- For every variable, we have to declare a **data type**.
- Java language provides eight **primitive** data types:
  - i.e. simple, fundamental
- For more complicated things, can use data types:
  - created by others provided to us through the Java libraries
  - that we invent
    - More soon - for now, let’s stay with the primitives
- We want **a**, **b**, and **c** to be integers:
  - Here’s how we do it...
Recap: Variables and Data Types

```java
public class Test3 {
    public static void main (String[] args) {
        int a; //these
        int b; //are
        int c; //variable declarations
        b = 3;
        c = 5;
        a = b + c;
        System.out.println ("The answer is " + a);
    }
}
```
Recap: Floating Point Numbers

- significant digits
  - 42 = \(4.2 \times 10^1\)
  - 4.2 = \(4.2 \times 10^0\)
  - 42000000 = \(4.2 \times 10^7\)
  - 0.000042 = \(4.2 \times 10^{-5}\)

- only need to remember
  - nonzero digits
  - where to put the decimal point
    - floats around when multiply/divide by 10
## Data Type Sizes

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

- fixed size, so finite capacity

![Address vs Data Diagram](image)
Variable Declaration Examples

- person’s age in years
- height of mountain to nearest meter
- length of bacterium in centimeters
- number of pets at home
Variable Declaration and Assignment

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- variable name is specified by identifier
- syntax:
  - `typeName variableName;`
//*****************************************
// Test3.java       Author: Kurt
//
// Our third use of variables!
//*****************************************

public class Test3
{
    public static void main (String[] args)
    {
        int a;
        int b;
        int c;
        b = 3; // these
        c = 5; // are
        a = b + c; // assignment statements
        System.out.println ("The answer is " + a);
    }
}

Assignment
Assignment Statements

- Assignment statement assigns value to variable
  - sometimes say binds value to variable
- Assignment statement is
  - identifier
  - followed by assignment operator (=)
  - followed by expression
  - followed by semicolon (;)

```
b = 3;
c = 8;
a = b + c;
weekly_pay = pay_rate * hours_worked;
```

- Note that = is no longer a test for equality!
Assignment Statements

- Java first computes value on right side
- Then assigns value to variable given on left side

```
x = 4 + 7;     // what's in x?
```

- Old value will be overwritten if variable was assigned before

```
x = 2 + 1;     // what's in x now?
```
Assignment Statements

Here’s an occasional point of confusion:

```plaintext
a = 7; // what’s in a?
b = a; // what’s in b?
// what’s in a now???
```
Assignment Statements

Here’s an occasional point of confusion:

```java
a = 7;                 // what’s in a?
b = a;                  // what’s in b?
// what’s in a now???
System.out.println("a is " + a + " b is " + b);
```

Find out! Experiments are easy to do in CS
Here’s an occasional point of confusion:

```java
a = 7;     // what’s in a?
b = a;     // what’s in b?
           // what’s in a now???
System.out.println("a is " + a + "b is " + b);
```

- Variable values on left of = are clobbered
- Variable values on right of = are unchanged
  - copy of value assigned to a also assigned to b
  - but that doesn’t change value assigned to a
Assignment Statements

■ Here’s an occasional point of confusion:

    a = 7;   // what’s in a?
    b = a;   // what’s in b?
    // what’s in a now???
    System.out.println("a is " + a + " b is " + b);
    a = 8;
    System.out.println("a is " + a + " b is " + b);

■ Memory locations a and b are distinct
  ■ copy of value assigned to a also assigned to b
  ■ changing a later does not affect previous copy
    ■ more later
Variable Declaration and Assignment

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- variable name is specified by identifier
- syntax:
  - `typeName variableName;`
  - `typeName variableName = value;`
    - can declare and assign in one step
Expressions

- **expression** is combination of
  - one or more operators and operands
  - operator examples: +, *, /, ...
  - operand examples: numbers, variables, ...
  - usually performs a calculation
    - don’t have to be arithmetic but often are
- examples
  
  3
  7 + 2
  7 + 2 * 5
  (7 + 2) * 5
Operator Precedence

What does this expression evaluate to?

\[7 + 2 \times 5\]
## Operator Precedence

- What does this expression evaluate to? 
  
  \[ 7 + 2 * 5 \]

- Multiplication has higher **operator precedence** than addition (just like in algebra)

<table>
<thead>
<tr>
<th>precedence</th>
<th>operator</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 higher</td>
<td>+ -</td>
<td>unary plus and minus</td>
</tr>
<tr>
<td>2</td>
<td>* / %</td>
<td>multiply, divide, remainder</td>
</tr>
<tr>
<td>3 lower</td>
<td>+ -</td>
<td>add, subtract</td>
</tr>
</tbody>
</table>
Operator Precedence

- What does this expression evaluate to?
  \[ 7 + 2 \times 5 \]

- Multiplication has higher operator precedence than addition (just like in algebra)

<table>
<thead>
<tr>
<th>precedence</th>
<th>operator</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 higher</td>
<td>+, -</td>
<td>unary plus and minus</td>
</tr>
<tr>
<td>2</td>
<td>*, /, %</td>
<td>multiply, divide, remainder</td>
</tr>
<tr>
<td>3 lower</td>
<td>+, -</td>
<td>add, subtract</td>
</tr>
</tbody>
</table>

- Use parentheses to change precedence order or just clarify intent
  \[ (7 + 2) \times 5 \quad 7 \times (2 + 5) \]
Converting Between Types

Which of these are legal?

- `int shoes = 2;`
- `double socks = 1.75;`
- `double socks = 1;`
- `int shoes = 1.5;`
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
Casting

- **Casting**: convert from one type to another 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);`
Converting Between Types

```java
//**********************************************************
// Feet.java   Author: Tamara
// What type of things can be put on feet?
//**********************************************************
public class Feet{
    public static void main (String[] args){
        int shoes = 2;
        int socks = (int) 1.75;
        System.out.println("shoes = " + shoes + " socks = " + socks);
        int toes = Math.round(1.99);
        System.out.println("toes = " + toes);
    }
}
```

What’s wrong?
## Data Type Sizes

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

- Is there more to life than 4-byte ints or 8-byte doubles?
## Primitive Data Types: Numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>approx -3.4E38 (7 sig. digits)</td>
<td>approx 3.4E38 (7 sig. digits)</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

- Primary primitives are **int** and **double**
- three other integer types
- one other real type
- range of choices for storage capacity
Using Long Integers

/*****************************************
// Feet2.java   Author: Tamara
// What type of things can be put on feet?
/*******************************************/

class Feet2
{
    public static void main (String[] args)
    {
        int shoes = 2;
        int socks = (int) 1.75;
        System.out.println("shoes = " + shoes + " socks = " + socks);
        long toes = Math.round(1.99);
        System.out.println("toes = " + toes);
    }
}
Or Cast To Int

//************************************************
// Feet3.java   Author: Tamara
// What type of things can be put on feet?
//************************************************
public class Feet3
{
    public static void main (String[] args)
    {
        int shoes = 2;
        int socks = (int) 1.75;
        System.out.println("shoes = " + shoes + " socks = " +
        socks);
        int toes = (int) Math.round(1.99);
        System.out.println("toes = " + toes);
    }
}
Primitive Data Types: Non-numeric

- Character type
  - named `char`
  - Java uses the Unicode character set so each char occupies 2 bytes of memory.

- Boolean type
  - named `boolean`
  - variables of type boolean have only two valid values
    - true and false
  - often represents whether particular condition is true
  - more generally represents any data that has two states
    - yes/no, on/off
What Changes, What Doesn’t?

//*******************************************************************************
// Vroom.java Author: Tamara
// Playing with constants
//*******************************************************************************/
public class Vroom
{
  public static void main (String[] args)
  {
    double lightYears, milesAway;
    lightYears = 4.35; // to Alpha Centauri
    milesAway = lightYears * 186000 * 60 * 60 * 24 * 365;
    System.out.println("lightYears: " + lightYears + " milesAway "+ milesAway);
    lightYears = 68; // to Aldebaran
    milesAway = lightYears * 186000 * 60 * 60 * 24 * 365;
    System.out.println("lightYears: " + lightYears + " milesAway " + milesAway);
  }
}
Constants

- Things that do not vary
  - unlike variables
  - will never change

- Syntax:
  - `final typeName variableName;`
  - `final typeName variableName = value;`

- Constant names in all upper case
  - Java convention, not compiler/syntax requirement
Programming With Constants

public static void main (String[] args)
{
    double lightYears, milesAway;

    final int LIGHTSPEED = 186000;
    final int SECONDS_PER_YEAR = 60*60*24*365;

    lightYears = 4.35; // to Alpha Centauri
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: " + lightYears + "miles " + milesAway);

    lightYears = 68; // to Aldebaran
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: " + lightYears + "miles " + milesAway);
}
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
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);
}
Reading Next Week

- Rest of Chap 2
  - 2.3-4, 2.6-2.10
- Rest of Chap 4
  - 4.3-4.7