Numerical Data
Recitation – 01/30/2009

CS 180
Department of Computer Science,
Purdue University
Project 2

- Now posted on the class webpage.
- Due Wed, Feb. 4 at 10 pm.
- Start early!
- All questions on the class newsgroup.
- Evening consulting hours
  - MTW 7-10 p.m. in LWSN B146.
Data Types

- **Primitive Data Types**
  - Numeric
    - byte, short, int, long, float, double
  - Character
    - char
  - Logic
    - boolean

- **Reference Data Types**
  - Object

**Note:** Primitive data items are created upon declaration. No need to use `new` to allocate space.
Variables

- Three properties
  - Memory location
  - Data type
  - Name

- Example
  - `int x;  // declaration`
    - Memory space is allocated to store integer values
    - `x` is the reference name for the memory space
  - `x = 3;  // assignment`
    - The value 3 is stored at the memory space `x`
  - `x = 5;  // another assignment`
    - The value 5 is stored at `x` to replace the old value 3
Variables (cont.)

- **Another example**
  - `int y = 10;`
    - Variable `y` is declared and also initialized to 10
  - `x = y;`
    - Remember: `x` is 5 from the previous slide
    - Now the value stored in `y` is copied to the memory location of `x`: *Now x is 10!*

- **Similar to the object assignment**
  - `Student student1, student2;`
  - `student1= new Student();`
  - `student2= student1;`
  - `student2` now refers to `student1` (reference is copied from `student1` to `student2`)
Numeric Data Types

- Six data types: **byte, short, int, long, float, double**
- A mistake in the original table. See below

<table>
<thead>
<tr>
<th>Type</th>
<th>Content</th>
<th>Default Value</th>
<th>Size (bytes)</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Integer</td>
<td>0</td>
<td>1</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>Integer</td>
<td>0</td>
<td>2</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>0</td>
<td>4</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>long</td>
<td>Integer</td>
<td>0</td>
<td>8</td>
<td>-9.22337E+18</td>
<td>9.22337E+18</td>
</tr>
<tr>
<td>float</td>
<td>Real</td>
<td>0.0</td>
<td>4</td>
<td>-3.40282347 x 10^{38}</td>
<td>3.40282347 x 10^{38}</td>
</tr>
<tr>
<td>double</td>
<td>Real</td>
<td>0.0</td>
<td>8</td>
<td>-1.7977 x 10^{308}</td>
<td>1.7977 x 10^{308}</td>
</tr>
</tbody>
</table>
## Arithmetic Operations

- Arithmetic operations that can be used on the numeric data

<table>
<thead>
<tr>
<th>Operation</th>
<th>Java Operator</th>
<th>Example</th>
<th>Value (x=10, y=7, z=2.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>x + y</td>
<td>17</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>x - y</td>
<td>3</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>x * y</td>
<td>70</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>x / y</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x / z</td>
<td>4.0</td>
</tr>
<tr>
<td>Modulo division</td>
<td>%</td>
<td>x % y</td>
<td>3</td>
</tr>
</tbody>
</table>

This is integer division where the fractional part is truncated.
Division and Modulo

- **Division ‘/’**
  - Integer division
    - $5 / 4 = 1$  
    - $1 / 2 = 0$
  - When either or both numbers are float or double, then the result is a real number
    - $5.0 / 4 = 1.25$  
    - $1.0 / 2.0 = 0.5$
    - $3.2 / 1.6 = 2.0$

- **Modulo ‘%’**
  - Returns the remainder of a division, usually involves only integers
  - Examples:
    - $23 \% 5 = 3$  
    - $23 \% 25 = 23$  
    - $16 \% 2 = 0$

For both: the second number cannot be 0!
How to calculate the area?

- **Problem**
  - Given a circle with radius $r = 5$, what is the area of the circle (let $\pi = 3.14$)?
  - Hint: The formula is $S = \pi r^2$ where $S$ is the area of the circle

- **Solution**
  - `int r = 5;`
  - `final double PI = 3.14; // PI is a constant!`
  - `double s = PI*r*r; // s is double!`
  - `System.out.println("The area of the circle is: " + s);`

- **Output**
  - The area of the circle is: 78.5
Precedence Rules

When evaluating the arithmetic expressions, we need to follow the precedence rules to decide which operation/subexpression to evaluate first:

<table>
<thead>
<tr>
<th>Order</th>
<th>Group</th>
<th>Operator</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Subexpression</td>
<td>()</td>
<td>Starting with innermost ()</td>
</tr>
<tr>
<td></td>
<td>Unary operators</td>
<td>-, +</td>
<td>Left to right.</td>
</tr>
<tr>
<td></td>
<td>Multiplicative operators</td>
<td>*, /, %</td>
<td>Left to right.</td>
</tr>
<tr>
<td></td>
<td>Additive operators</td>
<td>+, -</td>
<td>Left to right.</td>
</tr>
</tbody>
</table>
What is unknown?

The code:

- `int a = 1;`
- `int b = a * 3;`
- `int c = a + b;`
- `int unknown = a + b * c % (b / (c - 1));`

The evaluation:

- `c - 1 = 4 - 1 = 3`
- `b / (c - 1) = 3 / 3 = 1`
- `b * c = 3 * 4 = 12`
- `b * c % (b / (c - 1)) = 12 % 1 = 0`
- `unknown = a + 0 = 1!`
Typecast

- Low precision -> high precision
  - byte -> short -> int -> long -> float -> double

- Implicit typecasting
  - From lower precision to higher precision
    - e.g. `double x = 3;`
    - `int x = 2.5; // wrong!`
    - `int x = (int)2.5; // right! x = 2`
    - The above is called **explicit demotion**

- Explicit typecasting
  - `double x = (double)5/2;`
  - `double y = (double)(5/2);`
  - Typecast operator: (double), cast int to double
  - Result: `x = 2.5, y = 2.0`. Why?
Constant

- Sometimes we want a value to remain the same, then we use constants

**Usage**

- `final double PI = 3.14;`
- `final int MAX_SCORE = 100;`
- “final” is a keyword; “PI”, “MAX_SCORE” are named constants; “3.14”, “100” are literal constants

**Benefits**

- Consistent and unchanged value
- Easy to manage, more readable
What does this program do?

- **Input:** a cylinder with radius 1 and height 10

- **The program:**
  - `int radius = 1;`
  - `final double PI = 3.14;`
  - `int height = 10;`
  - `double x = 2 * PI * radius;`
  - `double y = PI * radius * radius;`
  - `double result = 2 * y + x * height;`

- **So, the result is**
  - `2 * PI * radius (radius + height) = 69.08`
  - `What is this?`
How can I get numbers from strings?

- Suppose the radius and height of the cylinder are provided in the form of strings: e.g. “1” and “10”. How can we get the actual number?

- Let the strings for radius and height be r and h. Here is the code for conversion:
  - `int radius = Integer.parseInt(r);`
  - `int height = Integer.parseInt(h);`
  - If radius and height can be real numbers, the safer way to do this is:
    - `double radius = Double.parseDouble(r);`
    - `double height = Double.parseDouble(h);`
How can I print numbers nicely?

In the previous cylinder example, if we want to display the result with only two digits after the decimal point, here is what we can do:

- DecimalFormat df = new DecimalFormat("0.00");
- System.out.println(df.format(result));

Operator overloading

- The result is the **surface area** of the cylinder!
- We can display the result more clearly like this:
  - System.out.println("The surface area of the cylinder: " + TAB + df.format(result));

- Here "+" is an overloaded operator
  - Not addition here, but **concatenation**
  - Output: “The surface area of the cylinder: 69.08”
  - What’s the difference between the following two?
    - System.out.println("Surface area: " + (1 + 1));
    - System.out.println("Surface area: " + 1 + 1);
Math Class

Math class is very powerful. It provides all kinds of useful methods such as:

- `abs(a)`: absolute value of a
- `max(a, b)`: the larger of a and b
- `min(a, b)`: the smaller of a and b
- `pow(a, b)`: a raised to power b
- `round(a)`: a rounded to the nearest whole number
- `sqrt(a)`: square root of a
- `ceil(a)`: smallest whole number no less than a
- `floor(a)`: largest whole number no bigger than a
- `sin(a)`: trigonometric sine of a
- `cos(a)`: trigonometric cosine of a
- `exp(a)`: natural number $e(2.718\ldots)$ raised to power a
Sample Code

- `double a = 10.5;`
- `int b = -20;`
- `b += (int)a; // shorthand`
- `double c = Math.max(Math.abs(b), (int)(a + 1));`
- `c *= 11; // shorthand`
- `System.out.println("c = " + Math.sqrt(c));`

What is the output?

Other examples

- `Math.pow(2.0, 3.0) = 8.0`
- `Math.floor(3.4) = 3, Math.ceil(3.4) = 4`
- `Math.round(3.4) = 3, Math.round(3.6) = 4`
- `Math.sin(Math.PI/2) = 1.0`
- `Math.cos(Math.PI/2) = 0.0`
Random Numbers

- **Math.random()**
  - Generates a number (double) no less than 0 and less than 1, i.e. in the range [0, 1)

- **How to generate a random integer between min and max, i.e. in the range [min, max]?**
  - \[
  \text{int randomNumber} = \text{(int)}(\text{Math.floor(Math.random()} \times (\text{max-min+1})) + \text{min});
  \]
  - **Why is this correct?**
    - \([0, 1) \times (\text{max-min+1}) \rightarrow [0, \text{max-min+1}]
    - After applying Math.floor, the range becomes [0, max-min]
    - After adding min, the range becomes [min, max]
    - After the explicit typecasting, the random number becomes an integer between min and max
Demo: Java Error Information

- Program – test.java
  - Import java.util.*;
  - class test{
    
    public static void main(String[] args)
    {
        System.out.println("Enter an integer: ");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        System.out.println("Integer entered: "+n);
    }
  }

- Now try the following and compile. Observe the error information:
  - Remove ";"
  - Remove “int” before “n”
  - Change “scanner.nextInt()” to “Scanner.nextInt()”
  - Change “scanner.nextInt()” to “scanner.nextInt()”
What’s wrong with the following piece of code? Does it compile? How will you correct the errors?

- double x = sqrt(9.0);
- int y = Math.abs(-3.2);
- y += x;
- double z = (double) y / (1/2);
- System.out.println("z is "+ z);