Lecture Set #3: Java Expressions

Last time:
1. Basics of Java programs

Today:
1. Variables and types
2. Expressions in Java
3. User input

Variables …

- … are named storage locations

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5</td>
</tr>
</tbody>
</table>

- Recall that memory is a sequence of bits
- Question: How much memory to allocate for a variable's value?
- Answer: A variable must have a type specifying how much storage to allocate.

Recall Java Built-in Types

<table>
<thead>
<tr>
<th>Type name</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integers</td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>Reals</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>2</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>
Primitive Data Types In Detail

Integer Types:
- **byte**: 1 byte, Range: -128 to +127
- **short**: 2 bytes, Range: -32,000 to +32,000
- **int**: 4 bytes, Range: -2 billion to +2 billion
- **long**: 8 bytes, Range: -9 quintillion to +9 quintillion

Floating-Point Types:
- **float**: 4 bytes, -3.4x10^38 to 3.4x10^38, 7 digits of precision
- **double**: 8 bytes, -1.7x10^308 to 1.7x10^308, 15 digits of precision

Other types:
- **boolean**: 1 byte, true, false
- **char**: 2 bytes, A single (Unicode) character

Primitive-Type Constants

- Constants are also called literals
- Integer types:
  - *byte*: optional sign and digits (0-9): 12, -1, +234, 0, 1234567
  - *short*: Same as above, but followed by 'L' or 'l': -1394382953
  - *int*: Same as above, but followed by 'L' or 'l': -1394382953
  - *long*: Same as above, but followed by 'L' or 'l': -1394382953

- Floating-point types:
  - *float*: Two allowable forms:
    - Decimal notation: 3.14159, -234.421, 0.0042, -43.0
    - Scientific notation: (use E or e for base 10 exponent) 3.14159E5

- *double*: Two allowable forms:
  - Decimal notation: 3.14159, -234.421, 0.0042, -43.0
  - Scientific notation: (use E or e for base 10 exponent) 3.14159E5

  - *float*: Same as double, but followed by 'f' or 'F': 3.14159F

Note: By default, integer constants are int, unless 'L' or 'l' is used to indicate they are long. Floating constants are double, unless 'F' or 'f' is used to indicate they are float.

Character and String Constants

- **Char constants**: Single character in single quotes (‘…’) including:
  - Punctuation symbols: ‘!’, ‘@’, ‘#’, ‘$’ (except ‘ and backslash \’)
  - Escape sequences: (use below)

- **String constants**: 0 or more characters in double quotes (“…”)

- **Escape sequences**: Allows inclusion of special characters:
  - \n: new-line character (start a new line)
  - \t: tab character
  - \": double quote
  - \': single quote
  - \": backslash

- **Examples**:
  - char x = ‘\'’; → x contains a single quote
  - String s1=“\"Hi there!\"”; → s1 contains “Hi there!”
  - String s2= “C:\\WINDOWS”; → s2 contains C:\\WINDOWS
Common Numeric Operators

- **Arithmetic operators:**
  - Unary negation: \(-x\)
  - Addition/subtraction: \(x+y\) \(x-y\)
  - Multiplication/division: \(x*y\) \(x/y\)
  - Division between integer types truncates to integer: \(23/4 \rightarrow 5\)
  - Division with real types yields a real result: \(23.0/4.0 \rightarrow 5.75\)

- **Comparison operators:**
  - Equality/inequality: \(x == y\) \(x != y\)
  - Less than/greater than: \(x < y\) \(x > y\)
  - Less than or equal/greater than or equal: \(x <= y\) \(x >= y\)
  - These comparison operators return a boolean value: true or false.

Common String Operators

- **String Concatenation:** The '+' operator concatenates (joins) two strings.
  - "Go" + "Terps" \(\rightarrow\) "GoTerps"
  - When a string is concatenated with another type, the other type is first evaluated and converted into its string representation.
    - \((8 * 4) + "degrees"\) \(\rightarrow\) "32degrees"
    - \((1 + 2) + "5"\) \(\rightarrow\) "35"

- **String Comparison:** Strings have special comparison functions.
  - \(s.equals(t)\): returns true if \(s\) and \(t\) have the same characters.
  - \(s.compareTo(t)\): compares strings lexicographically (dictionary order)
    - \(\text{result} < 0\) if \(s\) precedes \(t\)
    - \(\text{result} == 0\) if \(s\) is equal to \(t\)
    - \(\text{result} > 0\) if \(s\) follows \(t\)
  - \("dilbert".compareTo("dogbert")\) \(\rightarrow\) -1 (which is less than 0)

Example 2: Basic Types

- Demonstration of "primitive types" and also the String type.
- Note that you can declare many different variables with one statement.

```java
public class Example2 {
    public static void main(String[] args) {
        int i1, i2, i3;
        double f1 = 7.3, f2 = 9.4;
        boolean b1, b2;
        char c;
        String s;

        i1 = 7;
        i2 = 3;
        i3 = i1 + i2 * 5 - 2;
        f1 = 3.1415927;
        b1 = true;
        b2 = (f2 < f1);
        c = 'X';
        s = "Hello " + "there" + " my friend.";

        System.out.println("i3 = " + i3);
        System.out.println("b1 = " + b1);
        System.out.println("b2 = " + b2);
        System.out.println("c = " + c);
        System.out.println("s = " + s);
    }
}
```
User Input in Java

- We've done output (System.out); what about input?
- Java 5.0 includes the Scanner class feature
  - Can use Scanner to create "scanner objects"
  - Scanner objects convert user input into data
- To use Scanner need to import a library:
  
  ```
  import java.util.Scanner;
  ```

```
Example5.java

import java.util.Scanner;
public class Example5 {
    public static void main(String[] args) {
        int i;
        double d;
        String s;

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        i = sc.nextInt();
        System.out.print("Enter a floating point value: ");
        d = sc.nextDouble();
        System.out.print("Enter a string: ");
        s = sc.next();

        System.out.println("Here is what you entered: ");
        System.out.println(i);
        System.out.println(d);
        System.out.println(s);
    }
}
```

Scanner Class Details

- To create a scanner object:
  ```
  new Scanner(input_source);
  ```
  - Input source can be keyboard (System.in), files, etc.
  - Object must be assigned to a variable (e.g. sc)
- Operations
  ```
  next()        Returns sequence of characters up to next whitespace
  nextBoolean() Returns value of indicated type (Reports error if type mismatch)
  nextByte()   
  nextChar()   
  nextDouble() 
  nextFloat()  
  nextInt()    
  nextLong()   
  nextShort()  
  nextLine()   Returns sequence of characters up to next carriage return
  ```
Debugging Java Programs

- Types of errors
  - "Compile time": caught by Eclipse / Java compiler
    - Syntax errors: typos, etc.
    - Type errors: misuse of variables
  - "Run time": appear during program execution
    - Division by 0
    - Wrong outputs (because of mistakes in programming)
- Eclipse helps catch compile time errors
  - Red: error
  - Yellow: warning

Example3.java – find the errors

```java
public class Example3 {
    public static void main(String[] args) {  
        int x = 7;
        int y = 12;
        double d = 72.33;
        boolean b = true;
        char c = 'c';
        String s = "cow";
        x = y + 24;
        y = 17.3;
        d = x;
        b = 17;
        c = "Here is something weird " + x + y;
    }
}
```