Lecture 4: Java Expressions

Last time:
1. CVS and project submission
2. 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</th>
<th>Type name</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integers</td>
<td>byte</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>Reals</td>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>Other</td>
<td>char</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>
### The Memory Table Game

```java
int x;
float y;
char c;
double z;
boolean b;
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></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
  - short
  - int
  - long
  
  
  ```
  optional sign and digits (0-9):  12   -1   +234   0  1234567
  ```
  
  Same as above, but followed by ‘L’ or ‘l’: -1394382953L

- **Floating-point types:**
  - double
    
    Two allowable forms:
    
    ```
    Decimal notation:  3.14159   -234.421    0.0042    -43.0
    Scientific notation: (use E or e for base 10 exponent)
    3.145E5 = 3.145 x 10^5 = 314500.0
    1834.23e-6 = 1834.23 x 10^{-6} = 0.00183423
    ```
  
  - float
    
    Same as double, but followed by ‘f’ or ‘F’:
    3.14159F   -43.2f

- Note: By default, integer constants are **int**, unless ‘L’/‘l’ is used to indicate they are **long**. Floating constants are **double**, unless ‘F’/‘f’ is used to indicate they are **float**.

Bonnie Dorr (adapted from Rance Cleaveland)
Character and String Constants

- **Char constants**: Single character in single quotes (‘…’) including:
  - **Punctuation symbols**: ‘*’, ‘#’, ‘@’, ‘$’ (except ‘ and backslash ‘\’)
  - **Escape sequences**: (see below)
- **String constants**: 0 or more characters in double quotes (“…”)
- **Escape sequences**: Allows inclusion of ‘, ”, other special characters:
  \" double quote \n new-line character (start a new line)
  \’ single quote \t tab character
  \\ backslash
- **Examples**: char x = ‘\’ \”Hi there!\” \"C:\\WINDOWS\" → (x contains a single quote)
  → "Hi there!"
  → C:\\WINDOWS
Common Numeric Operators

- **Arithmetic operators:**
  - Unary negation: \(-x\)
  - Addition/subtraction: \(x + y\) \(x - y\)
  - Multiplication/division: \(x \times y\) \(x / y\)
    - Division between integer types **truncates** to integer: \(23/4 \rightarrow 5\)
    - \(x \% y\) returns the **remainder** of \(x\) divided by \(y\): \(23 \% 4 \rightarrow 3\)
    - 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" → "GoTerps"
  - When a string is concatenated with another type, the other type is first evaluated and **converted** into its string representation.
    - (8*4) + "degrees" → "32degrees"
    - (1 + 2) + "5" → "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)
    - result < 0 if `s` precedes `t`
    - result == 0 if `s` is equal to `t`
    - result > 0 if `s` follows `t`
  - "dilbert".compareTo("dogbert") → -1 (which is < 0)

Both functions are case-sensitive.

Note: Concatenation does not add any space.
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
public class Example3 {

    public static void main(String[] args) {
        int x = 7;
        int y = 12;
        double d = 72.33;
        boolean b = true;
        char c;
        String s;

        x = y + 24;
        y = 17.3;
        d = x;
        b = 17;
        c = "cow";
        s = "Here is something weird " + x + y;
    }
}
Example4.java: Operations

```java
public class Example4 {

    public static void main(String[] args) {
        System.out.println("Three divided by four is: " + 3/4);
        System.out.println("Three mod four is: " + 3%4);
    }
}
```
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:
  
  ```java
  import java.util.Scanner;
  ```
- Note difference between `System.out.println` vs. `System.out.print` (`println` moves to the next line after printing.)
Example5.java

```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);
    }
}
```

Create new scanner object to read from keyboard

Input an integer

Input a double

Input a string (up to white space)
Scanner Class Details

- To create a scanner object:
  ```java
  new Scanner(input_source);
  ```
  - Input source can be keyboard (```System.in```, files, etc.)
  - Object must be assigned to a variable (e.g. `sc`)

- Operations
  - `nextBoolean()`
  - `nextByte()`
  - `nextDouble()`
  - `nextFloat()`
  - `nextInt()`
  - `nextLong()`
  - `nextShort()`
  - `next()`  
    Returns sequence of characters up to next whitespace (space, carriage return, tab, etc.)
  - `nextLine()`  
    Returns sequence of characters up to next carriage return

  Returns value of indicated type (reports error if type mismatch)