CS 112 Introduction to Programming  
(Spring 2012)

Lecture #4: Built-in Types of Data  
Zhong Shao  
Department of Computer Science  
Yale University  
Office: 314 Watson  
http://flint.cs.yale.edu/cs112

Acknowledgements: some slides used in this class are taken directly or adapted from those accompanying the textbook Introduction to Programming in Java: An Interdisciplinary Approach by Robert Sedgewick and Kevin Wayne (Copyright 2002-2010)

The Computer’s View

Internally, computers store everything as 1’s and 0’s

- Example:  
  0110100 \rightarrow 104  
  0110100 \rightarrow h  
  ”hi” \rightarrow 01101000110101

How can the computer tell the difference between h and 104?

Type: A category or set of data values.
- Constrains the operations that can be performed on data
- Many languages ask the programmer to specify types  
  - Examples: integer, real number, string

A Foundation for Programming

any program you might want to write

- objects
- methods and classes
- graphics, sound, and image I/O
- arrays
- conditionals and loops
- Math
- text I/O
- primitive data types
- assignment statements

Numeric Primitive Data Types

- Integer numbers  
  - byte, short, int, long
- Real (floating point) numbers  
  - float, double

The differences among the various numeric primitive types are their storage sizes and representation format, and therefore the ranges and precision of the values they can store.

To understand this, we need to have more understanding of computer memory storage
A computer can use multiple cells (e.g., 2 bytes) to store a value. Each memory cell has a set number of bits (usually 8 bits, or one byte; a bit can represent 2 values). A computer can use multiple cells (e.g., 2 bytes) to store a value. How many values can a byte represent?

RAM is divided into many cells; each cell can be identified by a numeric address. Numeric Primitive Data

Different numeric data types have different ranges and precision:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</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>long</td>
<td>8 bytes</td>
<td>-9 x 10^38</td>
<td>9 x 10^38</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>+/- 3.4 x 10^38 with 7 significant digits</td>
<td>IEEE 754 format</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>+/- 1.7 x 10^308 with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>

Question: Can a float store all real numbers in its range?

(Frequently Used) Built-in Data Types

Data type. A set of values and operations defined on those values.

<table>
<thead>
<tr>
<th>Type</th>
<th>set of values</th>
<th>literal values</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>characters</td>
<td>‘a’</td>
<td>compare</td>
</tr>
<tr>
<td>String</td>
<td>sequences of characters</td>
<td>&quot;Hello World&quot;</td>
<td>concatenate</td>
</tr>
<tr>
<td>int</td>
<td>integers</td>
<td>17</td>
<td>add, subtract, multiply, divide</td>
</tr>
<tr>
<td>double</td>
<td>floating-point numbers</td>
<td>3.1415</td>
<td>add, subtract, multiply, divide</td>
</tr>
<tr>
<td>boolean</td>
<td>truth values</td>
<td>true, false</td>
<td>and, or, not</td>
</tr>
</tbody>
</table>
Table of variable values after each statement.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>99</td>
<td>1234</td>
</tr>
</tbody>
</table>

String data type. Useful for program input and output.

```java
public class Ruler {
  public static void main(String[] args) {
    String ruler1 = "1";
    String ruler2 = ruler1 + "2" + ruler1;
    String ruler3 = ruler2 + "3" + ruler2;
    String ruler4 = ruler3 + "4" + ruler3;
    System.out.println(ruler4);
  }
}
```

Subdivisions of a Ruler

```
% java Ruler
1 2 3 1 2 1 4 1 2 3 1 2 1
```
Integers

..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...

Integers

**int** data type. Useful for expressing algorithms.

<table>
<thead>
<tr>
<th>value</th>
<th>typical literals</th>
<th>operations</th>
<th>operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>99</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>-1234</td>
<td>-99</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1 / 0</td>
<td>1 / 0</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>3 % 5</td>
<td>3 % 5</td>
<td>%</td>
<td>%</td>
</tr>
</tbody>
</table>

**expression** | **value** | **comment**
--- | --- | ---
5 + 3 | 8 | 
5 - 3 | 2 | 
5 * 3 | 15 | 
5 / 3 | 1 | no fractional part
1 % 3 | 2 | remainder
1 / 0 | run-time error | 
3 * 5 - 2 | 13 | * has precedence
3 + 1 / 2 | 5 | / has precedence
3 - 5 - 2 | -4 | left associative
(3 - 5) - 2 | -4 | better style
3 - (5 - 2) | 0 | unambiguous

Floating-Point Numbers

**REAL NUMBERS**

**FLOATING-POINT NUMBERS**

---

```
public class IntOps {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int sum = a + b;
        int prod = a * b;
        int quot = a / b;
        int rem = a % b;
        System.out.println(a + " + " + b + " = " + sum);
        System.out.println(a + " * " + b + " = " + prod);
        System.out.println(a + " / " + b + " = " + quot);
        System.out.println(a + " % " + b + " = " + rem);
    }
}
```

\$ javac IntOps.java \$
\$ java IntOps 1234 99 \$
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46
1234 = 12*99 + 46
Floating-Point Numbers

double data type. Useful in scientific applications.

<table>
<thead>
<tr>
<th>values</th>
<th>operation</th>
<th>expression</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>real numbers (specified by IEEE 754 standard)</td>
<td>add</td>
<td>3.141 + .03</td>
<td>3.171</td>
</tr>
<tr>
<td></td>
<td>subtract</td>
<td>3.141 - .03</td>
<td>3.111</td>
</tr>
<tr>
<td>6.0e23 / 2.0</td>
<td>multiply</td>
<td>3.0e23</td>
<td></td>
</tr>
<tr>
<td>5.0 / 3.0</td>
<td>divide</td>
<td>1.66666666666667</td>
<td></td>
</tr>
<tr>
<td>10.0 % 3.141</td>
<td></td>
<td>0.177</td>
<td></td>
</tr>
<tr>
<td>1.0 / 0.0</td>
<td></td>
<td>Infinity</td>
<td></td>
</tr>
<tr>
<td>Math.sqrt(2.0)</td>
<td></td>
<td>1.4142135623730951</td>
<td></td>
</tr>
<tr>
<td>Math.sqrt(-1.0)</td>
<td></td>
<td>NaN</td>
<td></td>
</tr>
</tbody>
</table>

Excerpts from Java’s Math Library

public class Math

public abs(double a) absolute value of a
public max(double a, double b) maximum of a and b
public min(double a, double b) minimum of a and b
Note: abs(), max(), and min() are defined also for int, long, and float.
double sin(double theta) sine function
double cos(double theta) cosine function
double tan(double theta) tangent function
Note: 2. Angles are expressed in radians. Use toDegrees() and toRadians() to convert.
Note: Use toIntExact(), toLongExact(), and toIntAndRem() for integer functions.
double exp(double a) exponential (e^a)
double log(double a) natural log (ln a or log a)
double pow(double a, double b) raise a to the bth power (a^b)
long round(double a) round to the nearest integer
double random() random number in [0, 1)
double sqrt(double a) square root of a
double E value of e (constant)
double PI value of pi (constant)

http://download.oracle.com/java1se/6/docs/api/java/lang/Math.html

Quadratic Equation

Ex. Solve quadratic equation x^2 + bx + c = 0.

```java
public class Quadratic {
    public static void main(String[] args) {
        // parse coefficients from command-line
        double b = Double.parseDouble(args[0]);
        double c = Double.parseDouble(args[1]);
        // calculate roots
        double discriminant = b * b - 4.0 * c;
        double d = Math.sqrt(discriminant);
        double root1 = (-b + d) / 2.0;
        double root2 = (-b - d) / 2.0;
        // print them out
        System.out.println(root1);
        System.out.println(root2);
    }
}
```

Ex. Solve quadratic equation x^2 + bx + c = 0.

Testing. Some valid and invalid inputs.

```java
% java Quadratic 3.0 2.0 2.0 1.0
x^2 - 3x + 2
% java Quadratic -1.0 -1.0 1.618033988749895 4.6180339887498954
golden ratio
% java Quadratic 1.0 1.0 NaN NaN
NaN is not a number
% java Quadratic 1.0 hello java.io.NumberFormatException: hello
% java Quadratic 1.0 java.lang.ArrayIndexOutOfBoundsException: bounds exception
```
**Booleans**

Booleans are a fundamental data type in programming. They are used to control logic and flow of programs. The boolean data type can have two values: `true` or `false`. This is useful for making decisions in code, such as whether a condition is met or if a certain action should be taken.

**Comparisons**

Comparisons are operations that take two operands of one type (e.g., `int`) and produce a result of type `boolean`. They are used to compare values and determine if one is greater than, less than, equal to, or not equal to another.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Meaning</th>
<th><code>true</code></th>
<th><code>false</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equal</td>
<td>2 == 2</td>
<td>2 == 3</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal</td>
<td>3 != 2</td>
<td>2 != 2</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
<td>2 &lt; 13</td>
<td>2 &lt; 2</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal</td>
<td>2 &lt;= 2</td>
<td>3 &lt;= 2</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
<td>13 &gt; 2</td>
<td>2 &gt; 13</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal</td>
<td>3 &gt;= 2</td>
<td>2 &gt;= 3</td>
</tr>
</tbody>
</table>

**Leap Year**

A leap year is a year that is divisible by 4, but not by 100 unless it is also divisible by 400. The logic can be implemented in code as follows:

```java
public class LeapYear {
    public static void main(String[] args) {
        int year = Integer.parseInt(args[0]);
        boolean isLeapYear = (year % 4 == 0) && (year % 100 != 0) || (year % 400 == 0);
        System.out.println(isLeapYear);
    }
}
```

Examples:
- `java LeapYear 2004` prints `true`
- `java LeapYear 1900` prints `false`
- `java LeapYear 2000` prints `true`
**Type Conversion**

*Type conversion.* Convert value from one data type to another.
- **Automatic:** no loss of precision; or with strings.
- **Explicit:** cast; or method.

<table>
<thead>
<tr>
<th>expression</th>
<th>expression type</th>
<th>expression value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1234&quot; + 99</td>
<td>String</td>
<td>&quot;123499&quot;</td>
</tr>
<tr>
<td>Integer.parseInt(&quot;123&quot;)</td>
<td>int</td>
<td>123</td>
</tr>
<tr>
<td>(int) 2.71828</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>Math.round(2.71828)</td>
<td>Long</td>
<td>3</td>
</tr>
<tr>
<td>(int) Math.round(2.71828)</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>(int) Math.round(3.14159)</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>11 * 0.3</td>
<td>double</td>
<td>3.3</td>
</tr>
<tr>
<td>(int) 11 * 0.3</td>
<td>double</td>
<td>3.3</td>
</tr>
<tr>
<td>11 * (int) 0.3</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>(int) (11 * 0.3)</td>
<td>int</td>
<td>3</td>
</tr>
</tbody>
</table>

**Random Integer**

*Ex.* Generate a pseudo-random number between 0 and $N-1$.

```java
public class RandomInt {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double x = Math.random(); // (random to int) double to int
        int n = (int) (x * N); // double between 0 and 1.0
        System.out.println("random integer is " + n);
    }
}
```

- java RandomInt 6
  - random integer is 3
  - java RandomInt 6
  - random integer is 0
  - java RandomInt 10000
  - random integer is 3184

**Real Life Example: Data Range Error**

*Historical example: Ariane 5 explosion in 1996*  
(http://www.youtube.com/watch?v=kVUrqduUyEpI; http://www.ima.umn.edu/~arnold/disasters/ariane.html)

- *Reason:* trying to store a 64-bit real number (a double) to a 16-bit integer led to the crash
Real Life Example: Precision Error

The Patriot Missile Failure in 1991

- Perfect detection of a Scud missile, but the intercepting Patriot missed the target
- Reason:
  - a computer cannot represent 0.1 precisely; for a 24-bit floating point number they used, it is off by 0.000000095.
  - After 100 hours in operation, it is off by 0.34 seconds (=0.000000095*100 hours * 60 min /hour * 60 sec/min * 10), leading to an error of about 600 meters

(www.ima.umn.edu/~arnold/disasters/patriot.html)

Summary

A data type is a set of values and operations on those values.

- String text processing.
- double, int mathematical calculation.
- boolean decision making.

In Java, you must:

- Declare type of values.
- Convert between types when necessary.

Why do we need types?

- Type conversion must be done at some level.
- Compiler can help do it correctly.
- Ex 1: in 1996, Ariane 5 rocket exploded after takeoff because of bad type conversion.
- Ex 2: The Patriot Missile Failure in 1991