CS 61B Data Structures and Programming Methodology

June 25 2008

David Sun

Announcements

- **Visit 387 Soda to arrange for after-hours access to Soda Hall.**
- **Computer Science Undergraduate Association event:**
  - Thursday June 26, 310 Soda
  - Intro to UNIX at 4 and 6 PM
  - CS61[ABC] Survival Guide, 5 PM

`static` **methods**

Today

- **Primitive types**
- **Control-of-flow**
- **Arrays**

Primitive Types

Type

- **Java is a strongly typed language**
  - every variable must have a declared type
- **A *Type* can be**
  - a class (predefined or user created)
  - a primitive type.

Integer types

- **Numbers *without* fractional parts.**

	Number of bytes	Range of values
`byte`	1	-128 to 127
`short`	2	-32,768 to 32,786
`int`	4	-2147483648 to 2147483647
`long`	8	-9,223,372,036,854,775,808 to 9223372036854775807

- **Integer types do not depend on the machine on which you will be running the Java code.**

Floating point types

- **Numbers with *fractional* parts**
  - `float:` 32 bytes, e.g., `43.2f`
  - `double:` 64 bytes, e.g., `23.5`
- `Double.NaN` (not a number)
  - To indicate errors (`0/0` or square-root of a negative number).
  - to check if `x` is NaN :

```
if (x == Double.NaN)   //wrong
 Double.isNaN(x)        //correct
```

Boolean and Char

- `boolean`
  - Two possible values: `true` or `false`
  - You cannot convert between integers and `boolean` values.
- `char`
  - `'a'` : a character
  - `"a"`: a string with a single character content

Differences Between Object and Primitive Types

	Object type	Primitive type
Variable contains a	Reference	Value
How defined?	Class definition	Built into Java
How created?	`new`	6, 3.4, true, ‘a’
How initialized?	Constructor	Default (usually zero)
How used?	Methods	Operators

Operators

- **Operations on** `int, long, short` **and** `byte` **types:**

  `+ - * /`
- **If `x` and `y` are integers:**
  - `x/y` rounds toward zero (drops the remainder).
  - `x%y` calculates the remainder of `x/y`.
- **If either x or y are floating point numbers:**
  - `x/y` is floating point division (rounds off after a certain number of digits).
- **Integer division by 0 raises a run-time exception.**
- **Floating point division by 0 yields an infinite or NaN result.**

More Operators

- **Increment and Decrement operators**

```
x = x + 1; x++;
x = x - 1; x--;
```

- **More operators in** `java.lang`
  - The `Math` class:

```
double x = Math.pow(x, a)
```

  - The `Integer` class.

```
int x = Integer.parseInt("1984");
```

  - The `Double` class:

```
double d = Double.parseDouble("2.4");
```

Boolean Operators

&& (and) ||(or) ! (not)

a	b	a && b	a || b	!a
true	true	true	true	false
true	false	false	true	
false	true	false	true	true
false	false	false	true	

```
boolean x = 2 == 4;
x = 5.2 <= 5.2;
x = 2 != 3 - 1;
x = false == (2 == 0);
```

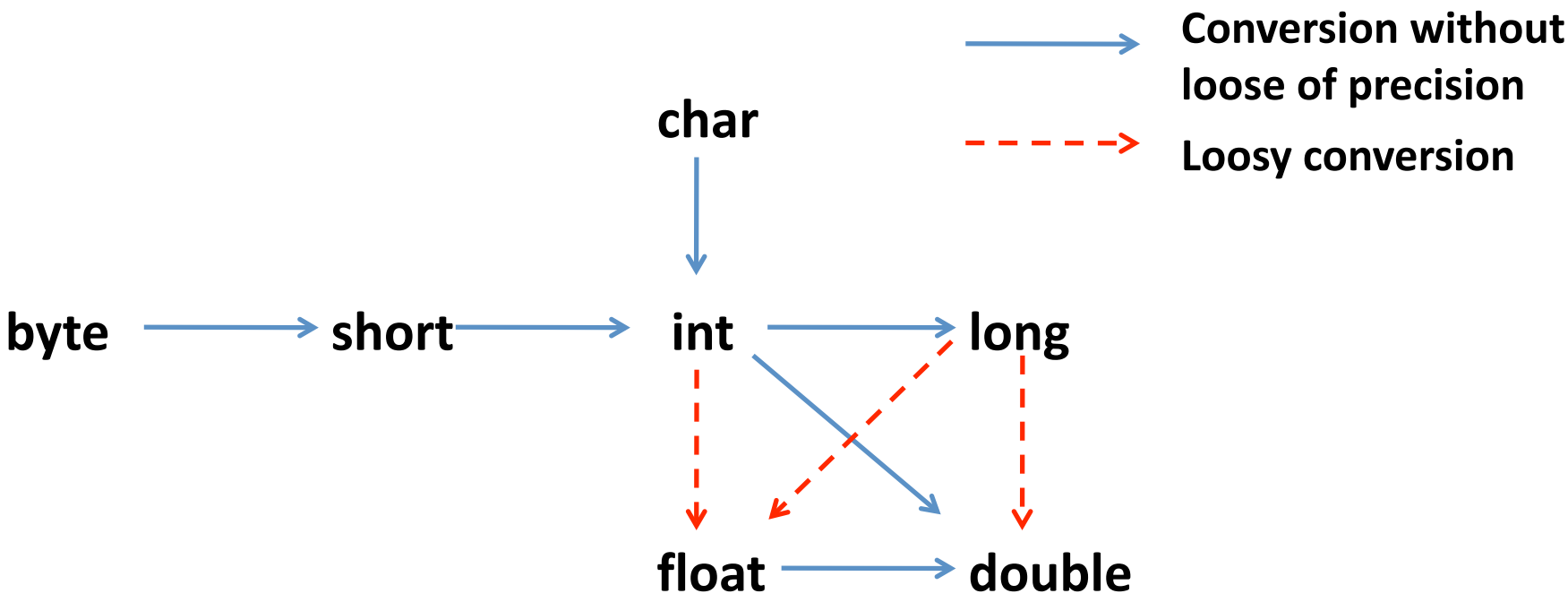
x **is false**

x **is true**

x **is false**

x **is true**

Type Conversion




```
int n = 123456789;
float f = n; //f is 1.23456792E8
```

Type Conversion (cont.)

- **Automatic casting**
  - If one operand is double, the other will be auto-casted to double
  - If one operand is float, the other will be auto-casted to float
  - If one operand is long, the other will be auto-casted to long
- **Manual casting**

```
double n = 9.997;
int x = n; //compile time error
int x = (int) n; //x is now 9.
int y = (int) Math.round(x); //x is now 10
```

Flow-of-control

Conditional Statement

```
if (condition) statements;
if (condition)
  statement1;
else
  statement2;
```

- **Examples**

switch

- Long chains of `if-then-else` can be simplified with a `switch` statement.
- Example.
- Case labels must be integers. You cannot test strings.

```
String input = …;
switch (input) //ERROR
{
  case "A": //ERROR
   …
   break;
  …
}
```

while-Loops

```
while (condition) statement
```

- **Examples**

do-while-Loops

- **Similar `while`-loops except the loop-body is executed at least once.**

```
do {
   s = keybd.readLine();
   process(s);
 } while (s.length() > 0);
```

`for`-loops

- **Loop controlled by a counter or similar variable that's updated after every iteration.**

```
for(initialize; test; update;)
  statement;

initialize;
while (test) {
  statement;
  next;
}
```

Breaking Control Flow

`break`: **breaks from the inner most loop enclosing the break and continues executing at the code after the loop.**

```
public void aMethod()
{
  while(true) {
      do_something();
     if (condition)
          break;
  }
  do_other_things();
}
```

Breaking Control Flow

`return` **: causes the method to end immediately and return to the calling method.**

```
public void aMethod()
{
  while(true) {
      do_something();
     if (condition)
         return;
  }
}
```

Breaking Control Flow

`continue` **: transfers control to the header of the inner most enclosing loop:**

```
public void aMethod()
  {
    while(true) {
        do_something();
       if (condition)
            continue;
       do_something();
    }
  }
```

`for` and `while` loops Revisited

```
int i = 0;
while (i < 10) {
    if (condition(i))
          continue;
    call(i);
    i++;
}
```

```
for (int i = 0; i < 10; i++) {
   if (condition(i))
       continue;
   call(i);
}
```

What's the difference?

More on Loops

```
do {
    String s = keybd.readLine();
    process(s);
} while (s.length() > 0);

String s = keybd.readLine();
while (s.length() > 0) {
   process(s);
   s = keybd.readLine();
 }

while (true) {
   String s = keybd.readLine();
   if (s.length() < 0)
       break;
   process(s);
 }
```

Arrays

Arrays

- **Data structure that stores a collection of variables of the same type.**
- **Individual variables are accessed through an integer *index.***
  - The variables is always indexed from 0 in increments of one.

Declaring Array

- **Declare an array variable by specifying an array type:**

```
char[]  c;
c = new char[4];
c[0] = 'b'; // Store the character 'b' at
            //index 0.
c[1] = 'l';
c[2] = 'u';
c[3] = 'e';
```

- **What if you access element 4?**
- **Once an array is created its size remains constant.**
- **The length of an array is stored in the field** `c.length`**.**

Array Example

Array Initializers

- **Shorthand to create an array and supply initial value to the array:**

```
int [] smallPrimes = { 2, 3, 5, 7, 11, 13};
smallPrimes = new int [] {17,19,23, 29,31, 37};

int [] anonymous = [] {17,19,23, 29,31, 37};
smallPrimes = anonymous;
```

Array Copying

- **If you copy one array variable into another, both variables refer to the same array**
- `System.arraycopy(…)` **copies all the values from one array into another.**

Command-Line Parameters

- **The** `String[] args` **of the** `main` **method receives the string arguments specified on the command line.**
- **Example**

Multi-Dimensional Arrays

- **Array of references to arrays**
- **Used for tables or complex arrangements.**
- **Rectangular Arrays:**

```
double [][] table = new double[7][7];
int[][] magicSquare =
   {
        {16, 3, 2, 13},
        {5, 10, 11, 8},
        {9, 6, 7, 12},
        {4, 15, 14, 1}
   }
```

Triangular Arrays

- **Pascal's Triangle**
  - Each entry is the sum of the two nearest entries in the row immediate above.
  - Binomial Coefficients: coefficients of the polynomial (x + 1) ^ i
  - E.g. (x + 1) ^ 4 = 1x^4 + 4x^3 + 6x^2 + 4x + 1
- **Example**

Arrays of Objects

- **Java does not create the objects automatically when you create an array of objects.**
- **The array contains space for references to the objects.**

```
Human[] h = new Human[5];
h[0] = new Human(12, “Wendy”);
h[1] = new Human(11, “David”);
```

Next Class

- **Reading:**
  - Head First Java Chapter 10
- **Next Class:**
  - Lists and Recursion
  - JUnit