Chapter 1

Introduction to Programming

1.1 Anatomy of a Program
1.1.1 A Quick Tour
1.1.2 Language Elements
1.1.3 Program Execution

1.2 The Declaration Statement
1.2.1 Variable Names
1.2.2 The Integer Types
1.2.3 Declaration and Memory
1.2.4 Other Data Types
1.2.5 Primitive and Non-Primitive Types

1.3 The Assignment Statement
1.3.1 The 1st Arithmetic Operators
1.3.2 Other Arithmetic Operators
1.3.3 Mixed Types and Casting

1.1.1 A Quick Tour
Let us take a look at a Java program. It does not matter now how the program was written; just become familiar with the terminology for describing program structures.

Note, in particular, the following four terms:
Imports, Class, Method, Style
import java.lang.System;
public class Area {
    public static void main(String[] args) {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}

Imports

Imported Class
import java.lang.System;

package java.lang;

import java.lang.*;

- Note the difference:
  import java.lang.System;
  versus
  import java.lang.*;

- And as a matter of style:
  The package naming convention calls for lowercase letters.

import java.lang.System;

public class Area {
    public static void main(String[] args) {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}

Copyright © 2006 Pearson Education Canada Inc.
import java.lang.*;
public class Area {
    public static void main(String[] args) {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}

Method Header
Method Body, a Block

Style
Class naming convention
Use title case unless an acronym, e.g. Math, UTL, StringTokenizer.

Method naming convention
Use lowercase letters but for multi-word names, capitalize the first letter of each subsequent word, e.g. main, equals, toString, isLeapYear

Block layout
Braces must align vertically and all statements must be left justified and indented by one tab position.

1.1.2 Language Elements
Still without worrying about semantics, let us identify the elements of a program:

Keywords
Identifiers
Literals
Operators
Separators
Keywords are the keywords plus the literals:
- `true`
- `false`
- `null`

Must not be a reserved word, must begin with a letter, and its character set must be: `{0-9, A-Z, a-z, _}`

Literals are recognized by the presence of a number, 'character', "characters", or one of: `true`, `false`, `null`

The character set of operators:
- `=`, `>`, `<`, `!`, `?`, `:`, `&`, `|`, `+`, `-`, `*`, `/`, `%`

Separators are:
- `,`, `;`, `…`, `( )`, `[ ]`, `{ }`

Example:
Identify the language elements in the following program...

Keywords:
- `abstract`
- `assert`
- `boolean`
- `break`
- `byte`
- `case`
- `catch`
- `char`
- `class`
- `const`
- `continue`
- `default`
- `do`
- `double`
- `else`
- `enum`
- `extends`
- `final`
- `finally`
- `float`
- `for`
- `goto`
- `if`
- `implements`
- `import`
- `instanceof`
- `int`
- `interface`
- `long`
- `native`
- `new`
- `package`
- `private`
- `protected`
- `public`
- `return`
- `short`
- `static`
- `strictfp`
- `super`
- `switch`
- `synchronized`
- `this`
- `throw`
- `throws`
- `transient`
- `try`
- `void`
- `volatile`
- `while`
import java.lang.System;
public class Area {
    public static void main(String[] args) {
        int width;  // keywords
        width = 8;  // identifiers
        int height = 3;  // literals
        int area = width * height;  // operators
        System.out.println(area);  // separators
    }
}

Keywords, Identifiers, Literals, Operators, Separators
1.1.3 Program Execution

Edit
create or edit
save the file
Compile
read source file
Java to bytecode
Area.java
Area.class
Compile-time errors
Run
VM
read one instruction
Java to bytecode
Area.class
fetch
execute

1.2 The Declaration Statement

```
type name;
```

The name of a primitive or non-primitive type, e.g. int, double...

An identifier to be associated with a memory block

The scope of the variable = the enclosing block of the declaration. The variable is not known outside its scope.

1.2.1 Variable Names

Rules and guidelines for the name:

- Must be an identifier
- Must not be in the scope of another variable with the same name
- A good name is indicative of the content that will be stored in the variable
- As a matter of style, use lowercase letters, but for multi-word names, capitalize the first letter of each subsequent word
1.2.2 The Integer Types

A type is a range of values and a set of operations on these values.

The range of the int type consists of all whole numbers between -2 and +2 billions (approx). int supports the four arithmetic operations plus the remainder.

The long type is very similar to int except its range is much bigger, +/-10\(^{19}\).

An integer literal has an int type unless suffixed by L (l), in which case it is long.

1.2.3 Declaration and Memory

Logical versus Physical
Address versus Content
Bytes, KB, MB, GB, TB, ...

The Memory Block
- 1-byte block at address 24
- 1-byte block at address 25
- 2-byte block at address 26
- 4-byte block at address 28

What happens when we write:

```java
int width;
```

1. A block big enough to hold an int is allocated, e.g. a 4B block at 24
2. Its address is associated with the variable name, e.g. 24 with width

Note that no initialization is involved; only an association of a name with an address.
1.2.4 Other Data Types

- **Integer**
  - Use for integer data, e.g. count.
  - 100% exact
- **Real**
  - Use for real data, e.g. amount.
  - Inherently inaccurate

### Numeric Types

- **Integer**
  - int: 4, ±2^G exact
  - long: 8, ±2^E exact
  - Integer literals are int by default unless suffixed with L

- **Real**
  - float: 4, ±10^M SD=7
  - double: 8, ±10^M SD=15
  - Real literals are recognized thru a decimal point or an exponent
  - They are double by default unless suffixed with F

### The Type boolean

- Stores the result on a condition
- Has only two possible values
- **true** and **false** are reserved words
- Boolean variables are not integers

Note: Boolean literals are the easiest to recognize!
The Character Type \texttt{char}

- A letter, digit, or symbol
- Digits versus Numbers
- Store the code, not the typeface
- The case of English: ASCII
- \texttt{char} is thus an (unsigned) integer type
- Unicode has 64K codes

Character literals are recognized by single quotes surrounding one character, e.g. \texttt{\textquoteleft A\textquoteright}.

More on Characters

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\null</td>
</tr>
<tr>
<td>32</td>
<td>\textquoteleft space\textquoteright</td>
</tr>
<tr>
<td>65-90</td>
<td>\textquoteleft A'-Z'</td>
</tr>
<tr>
<td>97-122</td>
<td>\textquoteleft a'-z'</td>
</tr>
<tr>
<td>65535</td>
<td>\null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Escape</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\null</td>
<td>The character whose code is (hex) \texttt{xxxx}</td>
</tr>
<tr>
<td>\textquoteleft</td>
<td>Single quote</td>
</tr>
<tr>
<td>\textquotedbl</td>
<td>Double quote</td>
</tr>
<tr>
<td>\textbackslash\textbackslash</td>
<td>Backslash</td>
</tr>
<tr>
<td>\textbackslash n</td>
<td>New line</td>
</tr>
<tr>
<td>\textbackslash r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\textbackslash f</td>
<td>Form Feed</td>
</tr>
<tr>
<td>\textbackslash t</td>
<td>Tab</td>
</tr>
<tr>
<td>\textbackslash b</td>
<td>Backspace</td>
</tr>
</tbody>
</table>

1.2.5 Primitive & Non-Primitive

- Primitive
  - \texttt{number}
  - \texttt{character}
  - \texttt{boolean}
  - \texttt{class}
  - \texttt{interface}
  - \texttt{array}

- Non-Primitive

Copyright © 2006 Pearson Education Canada Inc.
Java’s Primitive Type

<table>
<thead>
<tr>
<th>PRIMITIVE TYPES</th>
<th>Type</th>
<th>Size (bytes)</th>
<th>Approximate Range  min</th>
<th>max</th>
<th>S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNED</td>
<td>byte</td>
<td>1</td>
<td>-128</td>
<td>127</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>short</td>
<td>2</td>
<td>-32,768</td>
<td>32,767</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>int</td>
<td>4</td>
<td>-2×10^9</td>
<td>2×10^9</td>
<td>N/A</td>
</tr>
<tr>
<td>UNSIGNED</td>
<td>long</td>
<td>8</td>
<td>-9×10^18</td>
<td>9×10^18</td>
<td>N/A</td>
</tr>
<tr>
<td>REAL</td>
<td>char</td>
<td>2</td>
<td>0</td>
<td>65,535</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>float</td>
<td>4</td>
<td>+3.4×10^38</td>
<td>+3.4×10^38</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>8</td>
<td>-1.7×10^308</td>
<td>+1.7×10^308</td>
<td>15</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>boolean</td>
<td>1</td>
<td>true/false</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

1.3 The Assignment Statement

- Pre-declared and in-scope
- Its type can hold RHS
- Its content will be overwritten

- a Literal
- a Name, or
- an Expression

1.3.1 The int Arithmetic Operators

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Kind</th>
<th>Syntax</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>+</td>
<td>infix</td>
<td>a + y</td>
<td>add y to a</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>infix</td>
<td>a - y</td>
<td>subtract y from a</td>
</tr>
<tr>
<td>&lt;</td>
<td>*</td>
<td>infix</td>
<td>a * y</td>
<td>multiply a by y</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>infix</td>
<td>a / y</td>
<td>divide a by y</td>
</tr>
<tr>
<td>&gt;</td>
<td>%</td>
<td>infix</td>
<td>a % y</td>
<td>remainder of a / y</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>prefix</td>
<td>+a</td>
<td>identity</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>prefix</td>
<td>-a</td>
<td>negate a</td>
</tr>
<tr>
<td></td>
<td>++</td>
<td>prefix</td>
<td>a++</td>
<td>a = a + 1; result = a</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>prefix</td>
<td>a--</td>
<td>a = a - 1; result = a</td>
</tr>
<tr>
<td></td>
<td>++</td>
<td>postfix</td>
<td>++a</td>
<td>result = a; a = a + 1</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>postfix</td>
<td>--a</td>
<td>result = a; a = a - 1</td>
</tr>
</tbody>
</table>
Examples

double price;
price = 17.25;
int quantity = 25;
boolean isValid = false;
double cost;
cost = price;
double extended;
extended = quantity * price;

Can combine declaration with assignment.

RHS is a variable.
RHS is an expression.

Example

5 + (4 - 3) / 5 - 2 * 3 % 4

Example

5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
Example

5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
Example

\[
5 + (4 - 3) / 5 - 2 * 3 \mod 4 \\
= 5 + 1 / 5 - 2 * 3 \mod 4 \\
= 5 + 0 - 2 * 3 \mod 4 \\
= 5 + 0 - 6 \mod 4 \\
= 5 + 0 - 2 
\]
Example

\[ 5 + (4 - 3) / 5 - 2 * 3 \% 4 \]

= \[ 5 + 1 / 5 - 2 * 3 \% 4 \]

= \[ 5 + 0 - 2 * 3 \% 4 \]

= \[ 5 + 0 - 6 \% 4 \]

= \[ 5 + 0 - 2 \]

= \[ 5 - 2 \]

= \[ 3 \]
1.3.2 Other Arithmetic Operators

Each of long, float, and double come with 11 operators with the same symbols as int; i.e. the symbols are overloaded. Note:

- The int operators satisfy closure thru circular wrapping
- The / int operator always rounds toward 0 and leads to an exception if the divisor is zero
- The sign of % is the same as that of the dividend
- The real operators satisfy closure by adding infinity and NaN. Hence, dividing by zero does not lead to exceptions
- (a * b) / c is not the same as a * (b / c) for any type
- (a + b) - c is not the same as a + (b - c) for real types

1.3.3 Mixed Types and Casting

- Promotion (aka widening conversion) is done automatically when needed
- May lead to loss of precision but the order of magnitude is preserved
- Demotion is not done automatically. Can be done manually thru a cast
- Casting is risky...avoid it.

Promotion and Demotion hierarchy:

- double
- float
- long
- int
- byte
- short
- char
Note:

- The cast operator has a precedence that is higher than * but less than ++
- The = operator has the lowest precedence of all operators
- There are shorthand operators to combine assignment with an operator:
  
  \[ x \text{ op} y \text{ is shorthand for } x = x \text{ op } y \]
  
  Ex: \( x +=1 \) is like \( x = x + 1 \) or \( x++ \)

Example

```java
int iVar = 15;
long lVar = 2;
float fVar = 7.6f - iVar / lVar;
double dVar = 1L / lVar + fVar / lVar;
int result = 100 * dVar;
```

Fix, if need be, and output result

The answer may surprise you!