CSPP 511-01:
Introduction to Object-Oriented Programming

Harri Hakula

Ryerson 256, tel. 773-702-8584
hhakula@cs.uchicago.edu

June 26, 2000
Outline

- Variables
- Primitive types
- Integers
- Documentation and Style
Let Us Start Programming

Data Structures + Algorithms = Programs
Variables

Data is stored in variables. Data is the value of the variable. For instance:

```java
public class IntegerExample {
    public static void main(String[] args) {
        int val; // Declaration
        val = 5; // Assignment
        System.out.println( val );
        int myInt = 3; // Initialization
        myInt = 4;
        System.out.println( myInt );
    }
}
```
Variables Cntd.

Every variable is assigned a memory location. When the variable is given a value, this value is placed in the memory local assignment to that variable.

`int var;` is a variable declaration. Every variable in a JAVA program must be declared before it is used. For example:

```
{ ... var = 5; ... int var; ... }
```

would be illegal.
Variables Cntd.

Setting the value is called the assignment. This is done with the assignment operator ‘=’, the equal sign. Declaration and assignment can be combined:   `int var = 5;`

This is called the initialization clause.

It is a good programming practice to initialize all variables used in a program.
Identifiers

The name of something in a JAVA program must not start with a digit and must consist entirely of letters, digits (0 through 9), and the underscore character. Uppercase and lowercase are considered to be different characters.

There are many conventions for naming classes, methods, and variables. Anything goes as long as it’s consistent.
Types

Variable’s type determines what kind of value the variable can hold. There are two main kinds of types in JAVA, class types and primitive types. A class type or user-defined type is a type for object with both data and methods. Primitive types are for indecomposable values.
# Primitive Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Kind of Value</th>
<th>Memory Used</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true or false</td>
<td>1 bit</td>
<td>boolean y = true;</td>
</tr>
<tr>
<td>byte</td>
<td>integer</td>
<td>1 byte</td>
<td>byte b = 8;</td>
</tr>
<tr>
<td>short</td>
<td>integer</td>
<td>2 bytes</td>
<td>short s = 10;</td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>4 bytes</td>
<td>int i = 128;</td>
</tr>
<tr>
<td>long</td>
<td>integer</td>
<td>8 bytes</td>
<td>long ago = 1000;</td>
</tr>
</tbody>
</table>
## Primitive Types Cntd.

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Kind of Value</th>
<th>Memory Used</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>floating-point</td>
<td>4 bytes</td>
<td>float f = 1.0f;</td>
</tr>
<tr>
<td>double</td>
<td>floating-point</td>
<td>8 bytes</td>
<td>double d = 1.0d;</td>
</tr>
<tr>
<td>char</td>
<td>single character</td>
<td>2 bytes</td>
<td>char c = ’A’;</td>
</tr>
</tbody>
</table>
All About Integers

Arithmetic operations in the order of precedence:

1. * / %

2. + -

% is the remainder or modulus operator. \( 5 \% 2 = 1 \), five mod two equals one.

Operators at the same precedence level are evaluated left to right:

\( 3 * 5 / 3 \% 4 = 1 \).

Use parentheses to change precedence:

\( 5 * 4 + 3 \) is not equal to \( 5 * (4 + 3) \).
All About Integers Cntd.

For integers, integer division is used: $5 \div 4 = 1$. Note that since $5 \div 4 = 1$ we have $5 \div 4 = 1 \ 1/4$.

What if we want to insist on $5 \div 4 = 1.25$?

**Type cast**

```java
double res = (double) 5 / 4;
```

We force the compiler to convert the integer 5 into an equivalent double value.

**Promotion**

```java
double res = 1.0 * 5 / 4;
```

By multiplying 5 by a floating-point number of type double, 1.0, the intermediate result is automatically a double.
All About Integers Cntd.

Notational shorthand: \( a = a + 1 \); is equivalent to \( a += 1 \).

Similarly for other arithmetic operators:

3. \( *= /= %= \)

4. \( += -= \)
Increment and Decrement Operators

Increment and decrement operators are special integer arithmetic operators: `++` `--` that increase and decrease the value by one, respectively. They come in pre- and post-variations.

**Preincrement** \( a = 1; \ b = \text{++}a; \) is equivalent to

\[
\begin{align*}
\text{a} & = 1; \\
\text{a} & = \text{a} + 1; \\
\text{b} & = \text{a};
\end{align*}
\]

**Postincrement** \( a = 1; \ b = \text{a}++; \) is equivalent to

\[
\begin{align*}
\text{a} & = 1; \\
\text{b} & = \text{a}; \\
\text{a} & = \text{a} + 1;
\end{align*}
\]

Similarly for predecrement and postdecrement.
Assignment Compatibilities

The values can safely be copied without any loss of information up the type hierarchy:
byte → short → int → long → float → double.

Going downstream means loss of information and must be enforced with type casting.
double val = 1.0; float fval = (float)val;
Documentation and Style

Ideally, your code should be easily understood by your peers. It is a common practice to insert helpful comments into programs. Comments are just that, they are meant for those who read the code, not for those who execute it.

There are two kinds of comments in JAVA:
end-of-line comment (//) and
multi-line comment (/* . . . */).
Example: TwoSum.java

/*
TwoSum computes the sum of two integers.
Author: Harri Hakula
Version: 1.0
*/

public class TwoSum {
    public static void main(String[] args) {
        int a = 1, b = 2;
        double res = a + b; // Change to int!
        System.out.println(res);
    }
}

Documentation and Style Cntd.

The compiler will ignore everything between the starting /* and the closing */. This comment can span multiple lines, whereas // affects only a single line.

Selection of the comment style depends on what you have to say. Again, be consistent.

Structured comments are comments that contain information for tools used to generate documentation. In JAVA environment there exists a popular convention that many organizations use. We may or may not visit this topic again later when the individual projects begin.
Indentation

Use indentation to make your code readable by others. Again, be consistent. Especially the placement of the curly brackets is a difficult issue.
Named Constants

Constants are variables whose values do not change after they have been initialized. This can be indicated to a compiler with a final keyword:

```java
public static final int TWO = 2;
```

Named constants are universally given uppercase names.
Named Constants Cntd.

/*
TwoTimes multiplies an integer by two.
Author: Harri Hakula Version: 1.0
*/

public class TwoTimes {
    public static final int TWO = 2;
    public static void main(String[] args) {
        int a = 3;
        double res = TWO * a;  // Change to int!
        System.out.println(a + " times 2 is " + res);
    }
}