



Accelerating Information Technology Innovation

<http://aiti.mit.edu>

Cali, Colombia
Summer 2012

Lesson 02 – Variables and Operators

Agenda

- Variables
 - Types
 - Naming
 - Assignment
- Data Types
- Type casting
- Operators

What is a Variable?

- In basic algebra, variables are symbols that can represent values in equations.
 - $y = x + 5$
- Similarly, variables in computer program represent data.
 - In computer programming, variables can represent more than just numbers

An Analogy

- Think of variables as a box that you can put values in.
- We can label the box with a name like “Box X” and re-use it many times.
- Can perform tasks on the box without caring about what’s inside:
 - “Move Box X to Shelf A”
 - “Put item Z in box”
 - “Open Box X”
 - “Remove contents from Box X”

Declaring Variables in Java

```
type name;
```

- Variables are created by declaring their type and their name as follows:
- Declaring an integer named “x” :
 - `int x;`
- Declaring a string named “greeting”:
 - `String greeting;`
- Note that we have not assigned values to these variables

Java Types: Integer Types

- Integer Types:
 - **int**: Most numbers you will deal with.
 - **long**: Big integers; science, finance, computing.
 - **short**: Smaller integers. Not as useful.
 - **byte**: Very small integers, useful for small data.

Java Types: Other Types

- Floating Point (Decimal) Types:
 - **float**: Single-precision decimal numbers
 - **double**: Double-precision decimal numbers.
 - Some phone platforms do not support FP.
- **String**: Letters, words, or sentences.
- **boolean**: True or false.
- **char**: Single Latin Alphanumeric characters

Assigning Values to Variables

```
name = value;
```

- Assign values to variables using the syntax:
- For example:
 - `x = 100;`
 - `greeting = "Jambo";`
- Illegal to assign a variable the wrong type:
 - `x = "Jambo";`
 - `x = 1.2;`
 - `greeting = 123;`
- We can declare and assign in one step:
 - `int x = 100;`
 - `String greeting = "Jambo";`

Naming Variables

- Variable names (or identifiers) may be any length, but must start with:
 - A letter (a – z, A-Z),
 - A dollar sign (\$),
 - Or, an underscore (_).
- Identifiers cannot contain special operation symbols like +, -, *, /, &, %, ^, etc.
- Certain reserved keywords in the Java language are illegal.
 - `int`, `double`, `String`, etc.

Naming Variables

- Java is case sensitive
- A rose is not a Rose is not a ROSE
- Choose variable names that are informative
 - Good: `int studentExamGrade;`
 - Bad: `int tempvar3931;`
- “Camel Case”: Start variable names with lower case and capitalize each word:
 - “camelshavehumps”.

Review

- Which of the following are valid variable names?
 - \$amount
 - 6tally
 - my*Name
 - salary
 - _score
 - first Name
 - short

Integer Types

- There are 4 primitive integer types: `byte`, `short`, `int`, `long`.
- Each type has a maximum value, based on its underlying binary representation:
 - Bytes: ± 128 (8 bits)
 - Short: $\pm 2^{15} \approx 32,000$ (16 bits)
 - Int: $\pm 2^{31} \approx 2$ billion (32 bits)
 - Long: $\pm 2^{63} \approx$ really big (64 bits)

Overflow

- What happens when if we store Bill Gates's net worth in an `int`?
 - Int: $\pm 2^{31} \approx 2$ billion (32 bits)
 - Bill's net worth: $> \$40$ billion USD
- Undefined!

Why use different types of integers?

- Why not simply use `long` values all the time?
 - Storage requirements (data memory)
 - Computational speed
 - Power efficiency
- Use smallest type that will not overflow

Floating Point Types

- Initialize doubles as you would write a decimal number:
 - `double y = 1.23;`
 - `double w = -3.21e-10; // -3.21x10-10`
 - `float z = 3.45;`
- Doubles are more precise than Floats, but may take longer to perform operations.

Floating Point Types

- We must be careful with integer division:
 - `double z = 1/3; // z = 0.0 ... Why?`
 - At least one of the operands must be a double:
 - `double z = 1.0/3; // z = 0.333`

Type Casting

- When we want to convert one type to another, we use type casting
- The syntax is as follows:

```
(new type)variable
```

- Example code:
 - `double decimalNumber = 1.234;`
 - `int integerPart = (int)decimalNumber;`
- Results:
 - `decimalNumber == 1.234;`
 - `integerPart == 1;`

Boolean Type

- Boolean is a data type that can be used in situations where there are two options, either `true` or `false`.
- The values `true` or `false` are case-sensitive keywords. Not `True` or `TRUE`.
- Booleans will be used later for testing properties of data.
- Example:
 - `boolean monsterHungry = true;`
 - `boolean fileOpen = false;`

Character Type

- Character is a data type that can be used to store a single characters such as a letter, number, punctuation mark, or other symbol.
- Characters are a single letter enclosed in **single** quotes.
- Example:
 - `char firstLetterOfName = 'e' ;`
 - `char myQuestion = '?' ;`

String Type

- Strings are not a primitive. They are what's called an Object, which we will discuss later.
- Strings are sequences of characters surrounded by **double** quotations.
- Strings have a special append operator + that creates a new String:
 - `String` greeting = "Ho" + "la" ;
 - `String` bigGreeting = greeting + "!" ;
 - `String` numGreeting = greeting + 25 ;

Review

- What data types would you use to store the following types of information?:
 - Population of Colombia `int`
 - World Population `long`
 - Approximation of π `double`
 - Open/closed status of a file `boolean`
 - Your name `String`
 - First letter of your name `char`
 - \$237.66 `double`

A Note on Statements

- A statement is a command that causes something to happen.
- All statements are terminated by semicolons ;
- Declaring a variable is a statement.
- Method (or function) calls are statements:
 - `System.out.println("Hello, World");`
- In lecture 4, we'll learn how to control the execution flow of statements.

What are Operators?

- **Expressions** can be combinations of variables, primitives and operators that result in a value
- Operators are special symbols used for:
 - mathematical functions
 - assignment statements
 - logical comparisons
- Examples with operators:
 $3 + 5$ // uses + operator
 $14 + 5 - 4 * (5 - 3)$ // uses +, -, * operators

The Operator Groups

- There are 5 different groups of operators:
 - Arithmetic Operators
 - Assignment Operator
 - Increment / Decrement Operators
 - Relational Operators
 - Conditional Operators
- The following slides will explain the different groups in more detail.

Arithmetic Operators

- Java has the usual 5 arithmetic operators:
– +, –, ×, /, %
- Order of operations (or precedence):
 1. **P**arentheses (**B**rackets)
 2. **E**xponents (**O**der)
 3. **M**ultiplication and **D**ivision from left to right
 4. **A**ddition and **S**ubtraction from left to right

Order of Operations (Cont'd)

- Example: $10 + 15 / 5;$
- The result is different depending on whether the addition or division is performed first

$$(10 + 15) / 5 = 5$$

$$10 + (15 / 5) = 13$$

Without parentheses, Java will choose the second case

- You should be explicit and use parentheses to avoid confusion

Integer Division

- In the previous example, we were lucky that $(10 + 15) / 5$ gives an exact integer answer (5).
- But what if we divide 63 by 35?
- Depending on the data types of the variables that store the numbers, we will get different results.

Integer Division (Cont'd)

- ```
int i = 63;
int j = 35;
System.out.println(i / j);
```

---

Output: 1

- ```
double x = 63;  
double y = 35;  
System.out.println(x / y);
```

Output: 1.8

- The result of integer division is just the integer part of the quotient!

Assignment Expression

- The basic assignment operator (=) assigns the value of `expr` to `var`

```
name = value
```

- Java allows you to combine arithmetic and assignment operators into a single statement

- Examples:

`x = x + 5;` is equivalent to `x += 5;`

`y = y * 7;` is equivalent to `y *= 7;`

Increment/Decrement Operators

- `++` is called the increment operator. It is used to increase the value of a variable by 1.

For example:

`i = i + 1;` can be written as:
`++i;` or `i++;`

- `--` is called the decrement operator. It is used to decrease the value of a variable by 1.

`i = i - 1;` can be written as:
`--i;` or `i--;`

Increment Operators (cont'd)

- The increment / decrement operator has two forms :

- Prefix Form e.g `++i;` `--i;`

- Postfix Form e.g `i++;` `i--;`

Prefix increment /decrement

- The prefix form first adds/subtracts 1 from the variable and then continues to any other operator in the expression
- Example:

```
int numOranges = 5;  
int numApples = 10;  
int numFruit;  
numFruit = ++numOranges + numApples;
```

```
numFruit has value 16  
numOranges has value 6
```


Postfix Increment/ Decrement

- The postfix form `i++`, `i--` first evaluates the entire expression and then adds/subtracts 1 to the variable
- Example:

```
int numOranges = 5;  
int numApples = 10;  
int numFruit;  
numFruit = numOranges++ + numApples;
```

```
numFruit has value 15  
numOranges has value 6
```

Relational (Comparison) Operators

- Relational operators compare two values
- They produce a boolean value (**true** or **false**) depending on the relationship

OperationIs true when
$a > b$	a is greater than b
$a \geq b$	a is greater than or equal to b
$a == b$	a is equal to b
$a != b$	a is not equal to b
$a \leq b$	a is less than or equal to b
$a < b$	a is less than b

Note: ==
sign!

Examples of Relational Operations

```
int x = 3;  
int y = 5;  
boolean result;
```

1) `result = (x > y);`

`result` is assigned the value `false` because
3 is `not greater` than 5

2) `result = (15 == x*y);`

now `result` is assigned the value `true` because the product of
3 and 5 `equals` 15

3) `result = (x != x*y);`

now `result` is assigned the value `true` because the product of
`x` and `y` (15) is `not equal` to `x` (3)

Conditional Operators

Symbol	Name
&&	AND
	OR
!	NOT

- Conditional operators can be referred to as `boolean` operators, because they are only used to combine expressions that have a value of `true` or `false`.

Truth Table for Conditional Operators

x	y	x && y	x y	!x
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Examples of Conditional Operators

```
boolean x = true;  
boolean y = false;  
boolean result;
```

- Let `result = (x && y);`

`result` is assigned the value `false`

- Let `result = ((x || y) && x);`

<code>(x y)</code>	evaluates to true
<code>(true && x)</code>	evaluates to true

now `result` is assigned the value `true`

Using && and ||

- `false && ...`
- `true || ...`
- Java performs *short circuit evaluation*
 - Evaluate && and || expressions from left to right
 - Stop when you are guaranteed a value

Short-Circuit Evaluation

```
(a && (b++ > 3));
```

What happens if `a` is `false`?

- Java will not evaluate the right-hand expression `(b++ > 3)` if the left-hand operator `a` is `false`, since the result is already determined in this case to be `false`. This means `b` will not be incremented!

```
(x || y);
```

What happens if `x` is `true`?

- Similarly, Java will not evaluate the right-hand operator `y` if the left-hand operator `x` is `true`, since the result is already determined in this case to be `true`.

Review

1) What is the value of `result`?

```
int x = 8;  
int y = 2;  
boolean result = (15 == x * y);
```

2) What is the value of `result`?

```
boolean x = 7;  
boolean result = (x < 8) && (x > 4);
```

3) What is the value of `z`?

```
int x = 5;  
int y = 10;  
int z = y++ + x + ++y;
```

Appendix I: Reserved Keywords

<code>abstract</code>	<code>assert</code>	<code>boolean</code>	<code>break</code>	<code>byte</code>
<code>case</code>	<code>catch</code>	<code>char</code>	<code>class</code>	<code>const</code>
<code>continue</code>	<code>default</code>	<code>do</code>	<code>double</code>	<code>else</code>
<code>extends</code>	<code>final</code>	<code>finally</code>	<code>float</code>	<code>for</code>
<code>goto</code>	<code>if</code>	<code>implements</code>	<code>import</code>	<code>instanceof</code>
<code>int</code>	<code>interfac e</code>	<code>long</code>	<code>native</code>	<code>new</code>
<code>package</code>	<code>private</code>	<code>protected</code>	<code>public</code>	<code>return</code>
<code>short</code>	<code>static</code>	<code>strictfp</code>	<code>super</code>	<code>switch</code>
<code>synchronized</code>	<code>this</code>	<code>throw</code>	<code>throws</code>	<code>transient</code>
<code>try</code>	<code>void</code>	<code>violate</code>	<code>while</code>	

Appendix II: Primitive Data Types

This table shows all primitive data types along with their sizes and formats:

Data Type	Description
<code>byte</code>	Variables of this kind can have a value from: -128 to +127 and occupy 8 bits in memory
<code>short</code>	Variables of this kind can have a value from: -32768 to +32767 and occupy 16 bits in memory
<code>int</code>	Variables of this kind can have a value from: -2147483648 to +2147483647 and occupy 32 bits in memory
<code>long</code>	Variables of this kind can have a value from: -9223372036854775808 to +9223372036854775807 and occupy 64 bits in memory

Appendix II: Primitive Data Types

Real Numbers

Data Type	Description
<code>float</code>	Variables of this kind can have a value from: 1.4e(-45) to 3.4e(+38)
<code>double</code>	Variables of this kind can have a value from: 4.9e(-324) to 1.7e(+308)

Other Primitive Data Types

<code>char</code>	Variables of this kind can have a value from: A single character
<code>boolean</code>	Variables of this kind can have a value from: <i>True or False</i>