Building Java Programs

Chapter 2
Lecture 2-1: Expressions and Variables

reading: 2.1 - 2.2
Data and expressions

reading: 2.1
self-check: 1-4
videos: Ch. 2 #1
Data types

- **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

- Internally, computers store everything as 1s and 0s
  - 104  →  01101000
  - "hi"  →  01101000110101
Java's primitive types

- **primitive types**: 8 simple types for numbers, text, etc.
  - Java also has **object types**, which we'll talk about later

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integers</td>
<td>42, -3, 0, 926394</td>
</tr>
<tr>
<td>double</td>
<td>real numbers</td>
<td>3.1, -0.25, 9.4e3</td>
</tr>
<tr>
<td>char</td>
<td>single text characters</td>
<td>'a', 'X', '?', '\n'</td>
</tr>
<tr>
<td>boolean</td>
<td>logical values</td>
<td>true, false</td>
</tr>
</tbody>
</table>

- Why does Java distinguish integers vs. real numbers?
Expressions

**expression**: A value or operation that computes a value.

- **Examples:**
  - $1 + 4 * 5$
  - $(7 + 2) * 6 / 3$
  - 42

- The simplest expression is a *literal value*.
- A complex expression can use operators and parentheses.
Arithmetic operators

- **operator**: Combines multiple values or expressions.
  - `+` addition
  - `-` subtraction (or negation)
  - `*` multiplication
  - `/` division
  - `%` modulus (a.k.a. remainder)

As a program runs, its expressions are evaluated.

- **1 + 1** evaluates to **2**
- `System.out.println(3 * 4);` prints **12**
  - How would we print the text **3 * 4**?
Integer division with /

- When we divide integers, the quotient is also an integer.
  - $14 \div 4$ is 3, not 3.5

\[
\begin{array}{c}
\underline{4) 14} \\
12 \\
\underline{2}
\end{array}
\quad\begin{array}{c}
\underline{10) 45} \\
40 \\
\underline{5}
\end{array}
\quad\begin{array}{c}
\underline{27) 1425} \\
135 \\
\underline{75}
\end{array}
\]

- More examples:
  - $32 \div 5$ is 6
  - $84 \div 10$ is 8
  - $156 \div 100$ is 1

- Dividing by 0 causes an error when your program runs.
Integer remainder with \% 

- The \% operator computes the remainder from integer division.
  - $14 \% 4$ is 2
  - $218 \% 5$ is 3

```
4 \) 14
 4
12
2
```
```
5 \) 218
 20
 18
 15
 3
```

- Applications of \% operator:
  - Obtain last digit of a number: $230857 \% 10$ is 7
  - Obtain last 4 digits: $658236489 \% 10000$ is 6489
  - See whether a number is odd: $7 \% 2$ is 1, $42 \% 2$ is 0

What is the result?
- $45 \% 6$
- $2 \% 2$
- $8 \% 20$
- $11 \% 0$
Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.
    
    \[
    1 - 2 - 3 \text{ is } (1 - 2) - 3 \text{ which is } -4
    \]
  - But \(*/\%\) have a higher level of precedence than \(+/\text{-}\)
    
    \[
    1 + 3 \times 4 \quad \text{is } 13
    \]
    
    \[
    6 + 8 / 2 \times 3 \\
    6 + 4 \times 3 \\
    6 + 12 \quad \text{is } 18
    \]
  - Parentheses can force a certain order of evaluation:
    
    \[
    (1 + 3) \times 4 \quad \text{is } 16
    \]
  - Spacing does not affect order of evaluation
    
    \[
    1 + 3 \times 4 - 2 \quad \text{is } 11
    \]
Precedence examples

1 * 2 + 3 * 5 % 4

1 + 8 % 3 * 2 - 9

Copyright 2008 by Pearson Education
Precedence questions

• What values result from the following expressions?

• $9 \div 5$
• $695 \% 20$
• $7 + 6 \cdot 5$
• $7 \cdot 6 + 5$
• $248 \% 100 \div 5$
• $6 \cdot 3 - 9 \div 4$
• $(5 - 7) \cdot 4$
• $6 + (18 \% (17 - 12))$
Real numbers (type double)

- Examples: 6.022, -42.0, 2.143e17
  - Placing .0 or .after an integer makes it a double.

- The operators +-*/%() all still work with double.
  - / produces an exact answer: 15.0 / 2.0 is 7.5
  - Precedence is the same: () before */% before +-
Real number example

\[ 2.0 \times 2.4 + 2.25 \times 4.0 \div 2.0 \]

\[ 4.8 + 2.25 \times 4.0 \div 2.0 \]

\[ 4.8 + 9.0 \div 2.0 \]

\[ 4.8 + 4.5 \]

\[ 9.3 \]
Mixing types

- When `int` and `double` are mixed, the result is a `double`.
  - `4.2 * 3` is `12.6`

- The conversion is per-operator, affecting only its operands.
  
  \[
  \begin{array}{c}
  7 / 3 \times 1.2 + 3 / 2 \\
  \downarrow \\
  2 \times 1.2 + 3 / 2 \\
  \downarrow \\
  2.4 + 3 / 2 \\
  \downarrow \\
  2.4 + 1 \\
  \downarrow \\
  3.4 \\
  \end{array}
  \quad
  \begin{array}{c}
  2.0 + 10 / 3 \times 2.5 - 6 / 4 \\
  \downarrow \\
  2.0 + 3 \times 2.5 - 6 / 4 \\
  \downarrow \\
  2.0 + 7.5 - 6 / 4 \\
  \downarrow \\
  2.0 + 7.5 - 1 \\
  \downarrow \\
  9.5 - 1 \\
  \downarrow \\
  8.5 \\
  \end{array}
  \]

- `3 / 2` is `1` above, not `1.5`. 
String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

```
"hello" + 42       is  "hello42"
1 + "abc" + 2      is  "1abc2"
"abc" + 1 + 2      is  "abc12"
1 + 2 + "abc"      is  "3abc"
"abc" + 9 * 3      is  "abc27"
"1" + 1            is  "11"
4 - 1 + "abc"      is  "3abc"
```

- Use + to print a string and an expression's value together.
  - `System.out.println("Grade: " + (95.1 + 71.9) / 2);`
  - Output: Grade: 83.5
Variables

reading: 2.2
self-check: 1-15
exercises: 1-4
videos: Ch. 2 #2
Receipt example

What's bad about the following code?

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal: ");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .08 +
                           (38 + 40 + 30) * .15);
    }
}
```

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `println` statements
Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:

- Steps for using a variable:
  - *Declare it* - state its name and type
  - *Initialize it* - store a value into it
  - *Use it* - print it or use it as part of an expression
Declaration

- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

**Syntax:**

```
    type name;
```

- The name is an *identifier*.

- `int x;`
  - `x`

- `double myGPA;`
  - `myGPA`
Assignment

- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- **Syntax**: 
  
  \[
  \text{name } = \text{ expression};
  \]

- ```
  \text{int x;}
  \text{x = 3;}
  \text{double myGPA;}
  \text{myGPA = 1.0 + 2.25;}
  ```

<table>
<thead>
<tr>
<th>x</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>myGPA</td>
<td>3.25</td>
</tr>
</tbody>
</table>
Using variables

- Once given a value, a variable can be used in expressions:

```java
int x;
x = 3;
System.out.println("x is " + x);  // x is 3
System.out.println(5 * x - 1);   // 5 * 3 - 1
```

- You can assign a value more than once:

```java
int x;
x = 3;
System.out.println(x + " here");    // 3 here
x = 4 + 7;
System.out.println("now x is " + x); // now x is 11
```
Declaration-initialization

- A variable can be declaredinitialized in one statement.

- Syntax:
  \[ \text{type name} = \text{value}; \]
  - \text{double myGPA} = 3.95;
  - \text{int x} = (11 \% 3) + 12;

\[
\begin{array}{|c|}
\hline
\text{x} & 14 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
\text{myGPA} & 3.95 \\
\hline
\end{array}
\]
Assignment and algebra

- Assignment uses `=`, but it is not an algebraic equation.
  - `=` means, "store the value at right in variable at left"
  - `x = 3;` means "x becomes 3" or "x should now store 3"

- What happens here?

```java
int x = 3;
x = x + 2;  // ???
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5</td>
</tr>
</tbody>
</table>
Assignment and types

- A variable can only store a value of its own type.
  - `int x = 2.5;  // ERROR: incompatible types`

- An `int` value can be stored in a `double` variable.
  - The value is converted into the equivalent real number.
  - `double myGPA = 4;`
  - `double avg = 11 / 2;`
    - Why does `avg` store `5.0` and not `5.5`?
Compiler errors

• A variable can't be used until it is assigned a value.
  
  ```java
  int x;
  System.out.println(x);  // ERROR: x has no value
  ```

• You may not declare the same variable twice.
  
  ```java
  int x;
  int x;  // ERROR: x already exists
  ```

  ```java
  int x = 3;
  int x = 5;  // ERROR: x already exists
  ```

• How can this code be fixed?
Printing a variable's value

- Use + to print a string and a variable's value on one line.

```java
double grade = (95.1 + 71.9 + 82.6) / 3.0;
System.out.println("Your grade was " + grade);

int students = 11 + 17 + 4 + 19 + 14;
System.out.println("There are " + students + " students in the course.");
```

- Output:

  Your grade was 83.2
  There are 65 students in the course.
Receipt question

Improve the receipt program using variables.

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                         (38 + 40 + 30) * .15 +
                         (38 + 40 + 30) * .08);
    }
}
```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}
Building Java Programs

Chapter 2
Lecture 2-2: The **for** Loop

**reading:** 2.3
self-check: 12-26
exercises: 2-14
videos: Ch. 2 #3
Increment and decrement

shortcuts to increase or decrease a variable's value by 1

**Shorthand**

variable++;  
variable--;

**Equivalent longer version**

variable = variable + 1;  
variable = variable - 1;

```
int x = 2;
x++; // x = x + 1;  // x now stores 3

double gpa = 2.5;
gpa--; // gpa = gpa - 1;  // gpa now stores 1.5
```
# Modify-and-assign operators

**shortcuts to modify a variable's value**

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>variable += value;</code></td>
<td><code>variable = variable + value;</code></td>
</tr>
<tr>
<td><code>variable -= value;</code></td>
<td><code>variable = variable - value;</code></td>
</tr>
<tr>
<td><code>variable *= value;</code></td>
<td><code>variable = variable * value;</code></td>
</tr>
<tr>
<td><code>variable /= value;</code></td>
<td><code>variable = variable / value;</code></td>
</tr>
<tr>
<td><code>variable %= value;</code></td>
<td><code>variable = variable % value;</code></td>
</tr>
</tbody>
</table>

```
x += 3;
// x = x + 3;
gpa -= 0.5;
// gpa = gpa - 0.5;
number *= 2;
// number = number * 2;
```
Repetition over a range

System.out.println("1 squared = " + 1 * 1);
System.out.println("2 squared = " + 2 * 2);
System.out.println("3 squared = " + 3 * 3);
System.out.println("4 squared = " + 4 * 4);
System.out.println("5 squared = " + 5 * 5);
System.out.println("6 squared = " + 6 * 6);

- Intuition: "I want to print a line for each number from 1 to 6"

- There's a statement, the for loop, that does just that!

    for (int i = 1; i <= 6; i++) {
        System.out.println(i + " squared = " + (i * i));
    }

- "For each integer i from 1 through 6, print ..."
for loop syntax

for (initialization; test; update) {
    statement;
    statement;
    ...
    statement;
}

- Perform **initialization** once.
- Repeat the following:
  - Check if the **test** is true. If not, **stop**.
  - Execute the **statements**.
  - Perform the **update**.
Initialization

```java
for (int i = 1; i <= 6; i++) {
    System.out.println(i + " squared = " + (i * i));
}
```

- Tells Java what variable to use in the loop
  - Called a *loop counter*
    - Can use any variable name, not just `i`
    - Can start at any value, not just 1
for (int i = 1; i <= 6; i++) {
    System.out.println(i + " squared = " + (i * i));
}

• Tests the loop counter variable against a bound
  • Uses comparison operators:
    < less than
    <= less than or equal to
    > greater than
    >= greater than or equal to
Update

```java
for (int i = 1; i <= 6; i++) {
    System.out.println(i + " squared = " + (i * i));
}
```

- Changes loop counter's value after each repetition
  - Without an update, you would have an infinite loop

- Can be any expression:
  ```java
  for (int i = 1; i <= 9; i += 2) {
      System.out.println(i);
  }
  ```
Loop walkthrough

```java
for (int i = 1; i <= 4; i++) {
    System.out.println(i + " squared = " + (i * i));
}
System.out.println("Whoo!");
```

Output:
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!

![Flowchart for loop walkthrough](chart.png)
General repetition

System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("S-M-R-T");
System.out.println("I mean S-M-A-R-T");

- The loop's body doesn't have to use the counter variable:

    for (int i = 1; i <= 5; i++) {   // repeat 5 times
        System.out.println("I am so smart");
    }
System.out.println("S-M-R-T");
System.out.println("I mean S-M-A-R-T");
Multi-line loop body

System.out.println("+-----+);
for (int i = 1; i <= 3; i++) {
    System.out.println("\ "/");
    System.out.println("/ ");
}
System.out.println("+-----+");

- Output:
  +-----+
  |     |
  |     |
  |     |
  |     |
  +-----+
Expressions for counter

int highTemp = 5;
for (int i = -3; i <= highTemp / 2; i++) {
    System.out.println(i * 1.8 + 32);
}

• Output:
  26.6
  28.4
  30.2
  32.0
  33.8
  35.6
System.out.print

- Prints without moving to a new line
  - allows you to print partial messages on the same line

```java
int highestTemp = 5;
for (int i = -3; i <= highestTemp / 2; i++) {
    System.out.print((i * 1.8 + 32) + "  ");
}
```

- Output:
  26.6  28.4  30.2  32.0  33.8  35.6
Counting down

- The **update** can use -- to make the loop count down.
  - The **test** must say > instead of <

```java
System.out.print("T-minus ");
for (int i = 10; i >= 1; i--) {
    System.out.print(i + ", ");
}
System.out.println("blastoff!");
```

- **Output:**
  T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
Mapping loops to numbers

for (int count = 1; count <= 5; count++) {
    ...
}

- What statement in the body would cause the loop to print:
  4 7 10 13 16

for (int count = 1; count <= 5; count++) {
    System.out.print(3 * count + 1 + " ");
}
Slope-intercept

for (int count = 1; count <= 5; count++) {
    ...
}

- What statement in the body would cause the loop to print:
  2 7 12 17 22

- Much like a slope-intercept problem:
  - count is $x$
  - the printed number is $y$
  - The line passes through points:
    (1, 2), (2, 7), (3, 12), (4, 17), (5, 22)

- What is the equation of the line?
Loop tables

- What statement in the body would cause the loop to print:  
  2 7 12 17 22

- To see patterns, make a table of count and the numbers.
  - Each time count goes up by 1, the number should go up by 5.
  - But count * 5 is too great by 3, so we subtract 3.

<table>
<thead>
<tr>
<th>count</th>
<th>number to print</th>
<th>5 * count</th>
<th>5 * count - 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>25</td>
<td>22</td>
</tr>
</tbody>
</table>
Loop tables question

• What statement in the body would cause the loop to print: 17 13 9 5 1

• Let's create the loop table together.
  • Each time count goes up 1, the number printed should ...
  • But this multiple is off by a margin of ...

<table>
<thead>
<tr>
<th>count</th>
<th>number to print</th>
<th>-4 * count</th>
<th>-4 * count + 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>-4</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>-8</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>-12</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-16</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-20</td>
<td>1</td>
</tr>
</tbody>
</table>