Java Arithmetic

- Evaluate the following expressions:
  1. `int x = 5 / 2 + 2;` ← 4
  2. `int x = 2 + 5 / 2;` ← 4
  3. `int x = 5 / 2.0 + 2;` ← ERROR
  4. `double x = 5 / 2.0 + 2;` ← 4.5
  5. `int x = (int) 5 / 2.0 + 2;` ← ERROR
  6. `int x = (int) (5 / 2.0 + 2);` ← 4
  7. `int x = 5 / (int) 2.0 + 2;` ← 4
  8. `int x = 5 / (int) (2.0 + 2);` ← 1
  9. `double x = 5 / 2 + 2;` ← 4.0

Understanding Java Assignments

- **Remember**: the assignment operator (=) is **not the same** as the equals sign used in mathematics.

- An assignment instruction in Java:
  ```java
  target = expression;
  ```
  is a command, saying that we should store the value of the expression to the target.

- If the target is a variable identifier, then we can change the value of what it stores this way:

Java Mixed-Type Arithmetic: Tricky Cases

- When we mix types of numbers in a math expression, the final result will be of **widest type** (unless we cast).
- However, the result we get can also depend on whether the switch is made from narrow types to wider ones.
  - This depends upon **operator precedence**, too.

- Examples:
  1. `5 / 2 * 2` ← 4
  2. `5.0 / 2 * 2` ← 5.0
  3. `5 / 2.0 * 2` ← 5.0
  4. `5 / 2 * 2.0` ← 4.0

Here, since division is performed first, and both 5 and 2 are of type `int`, we use `int` division.
Conversion to `double` type doesn’t happen until the multiplication (*)
An assignment command in Java:

\[ \text{target} = \text{expression}; \]

is processed in a fixed order:

1. Evaluate \( \text{expression} \), and compute its value, \( v \)
2. Assign value \( v \) to \( \text{target} \)

Since this is how the computer will process the command, we must do the very same when trying to understand what it will actually do.

As an example, consider the command:

\[ \text{double num} = 5 / 2; \]

which is processed as follows:

1. Evaluate expression \( 5 / 2 \): since the values are both integers, the arithmetic uses integer division, and computes the value 2 (and not 2.5)
2. Assign value 2 to \( \text{num} \) when complete, which saves result as a double

As a result, when this is complete, \( \text{num} \) is equal to 2.0

Because of the “right-hand, then left-hand” order that Java does things, it is possible to have the same variable occur on both sides of the = operator. In normal arithmetic, this rarely happens, as something like the following is essentially meaningless, since there is no number \( n \) for which it is true that:

\[ n = n + 3 \]

In Java, however, such an expression is perfectly OK, as in the following:

```
int n1 = 6 + 4 * 2;
int n2 = 7 / 2;
n1 = n1 + n2;
n2 = n1 + n2;
```

```
n1: 14
n2: 3
n1: 14 + 3 ⇒ 17
n2: 17 + 3 ⇒ 20
```

Note that the 3rd line changes what \( n1 \) is. Thus, when we get to the 4th line, we use the new value of \( n1 \) to calculate \( n2 \).
Shortcut Assignment Operators for Integers

- Along with the basic arithmetic and assignment operators, int and double arithmetic can use some shortcuts to simplify our coding.
- Each of these does some arithmetic operation, and then does the assignment of the newly computed value to the associated identifier.

<table>
<thead>
<tr>
<th>Shortcut Assignments</th>
<th>Description</th>
<th>Notation</th>
<th>Example</th>
<th>Equivalent*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subtract and assign</td>
<td>-=</td>
<td>x -= 3;</td>
<td>x = x - 3;</td>
<td></td>
</tr>
<tr>
<td>Add and assign</td>
<td>+=</td>
<td>x += 3;</td>
<td>x = x + 3;</td>
<td></td>
</tr>
<tr>
<td>Multiply and assign</td>
<td>*=</td>
<td>x *= 3;</td>
<td>x = x * 3;</td>
<td></td>
</tr>
<tr>
<td>Divide and assign</td>
<td>/=</td>
<td>x /= 3;</td>
<td>x = x / 3;</td>
<td></td>
</tr>
<tr>
<td>Remainder and assign</td>
<td>%=</td>
<td>x %= 3;</td>
<td>x = x % 3;</td>
<td></td>
</tr>
<tr>
<td>Add one and assign</td>
<td>++</td>
<td>x++;</td>
<td>x = x + 1;</td>
<td></td>
</tr>
<tr>
<td>Subtract one and assign</td>
<td>--</td>
<td>x--;</td>
<td>x = x - 1;</td>
<td></td>
</tr>
</tbody>
</table>

(* See slides 11–12 for more detail.)

Using Shortcut Assignments

```java
int x = 10;
x++;                       \(\equiv 11\)
x += 1;                     \(\equiv 12\)
x *= 2;                     \(\equiv 24\)
x -= 1;                     \(\equiv 23\)
x--;                        \(\equiv 22\)
x /= 5;                     \(\equiv 4\)
```

One Complication: Mixed Arithmetic

- When dealing with a single type, it is OK to think of these complex assignment operators as simply abbreviations (as in the table on slide 9).
- However, these operators actually have a cast operation built into them.
- Each time we run one, it does its computation and then casts the result to guarantee it is the right type.
- That is, cast will be to the type of the identifier on the left-hand side.
- Thus, when we see something like this:
  ```java
  int x = 3;
x *= 2;     
  ```
  This is really an abbreviation for:
  ```java
  int x = 3;
x = (int)( x * 2 ); // produces value 6
  ```

Here, the cast makes no difference to the result, but it is still technically performed.

One Complication: Mixed Arithmetic

- Due to the built-in cast operation for each complex assignment operator, we can mix types when using them.
- Just need to remember that the cast happens at the end, just before the assignment operation itself.
- Thus, something like this is completely legal in Java:
  ```java
  int x = 2;
x *= 4.5;     
  ```
  This is allowed, since it is actually an abbreviation for:
  ```java
  int x = 2;
x = (int)( x * 4.5 ); // produces value 9
  ```
Sequential Program Operation

- Basic Java code operations happen one by one, line by line, in the order given by the programmer.
- Until we actually reach some instruction, none of what it does is visible to our program.
- Thus, any variables we want to use have to be declared and instantiated before we use them.

```
Oval o = new Oval( 10, 10, size, size );
int size = 300;
```

This code will not compile. The variable size must be declared and set to the value we want before we can use it in the Oval() constructor.

Sequential Program Operation

- Assignments work like other sequential code.
- Once an assignment is given to some identifier, all uses of that identifier after that point will use the assigned value.
- Nothing before the assignment is affected.
- The newly assigned value is used until another assignment occurs.

Example: what are the values of x and y after the following runs?

```
int x = 3;
int y = 5;
x = y;
y = 2;
```

- x is 3 (y doesn’t exist yet)
- x is 3; y is 5 (x changed)
- x is 5; y is 2 (y changed)

Due to the sequential nature of Java, we sometimes have to get creative.

- As seen on the previous slide, if we want to swap two values, simply using two variable identifiers won’t work.
- We need an extra variable to store “temporary information” if we are going to swap values of two variables.

```
int x = 3;
int y = 5;
int temp = x;
x = y;
y = temp;
```

- x is 3 (y doesn’t exist yet)
- x is 3; y is 5 (x changed)
- x is 5; y is 5 (no swap!)
- x is 5; y is 3; temp is 3 (swap!)
This Week & Next

Meetings next week:
- Monday, Wednesday: in the CS Lab (16 Wing)
- Tuesday, Friday: regular classroom

Program 01:
- Due: Monday, 10 February, by 11:59 PM

Reading assignment: Chapter 3
- Due: Thursday, 13 February, by 12:00 PM

Office Hours: Wing 212
- Monday/Wednesday/Friday: 11:00 AM–12:00 PM
- Tuesday: 3:15 PM–4:15 PM