Data types

Expressions

Variables

Assignment

COMPI400/INFS1609 – Week 2

Data types

Data come in different types.
The type of a piece of data describes:
• What the data means.
• What we can do with it.

Primitive types

Java defines a selection of “primitive” data types.

They are called primitive because they are the basic building blocks we can use to construct more complex types.

Binary Numbers

• All data in a computer are stored as binary numbers, e.g.

\[
1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13
\]
Computer Memory

### Primitive types

<table>
<thead>
<tr>
<th>Type</th>
<th>Keyword</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>int</td>
<td>whole numbers</td>
<td>0, 1, 100, -3322</td>
</tr>
<tr>
<td>floating point number</td>
<td>float</td>
<td>fractional numbers</td>
<td>0.0f, 1.5f, -32.7f, 3.14159f</td>
</tr>
</tbody>
</table>

### Extended types

<table>
<thead>
<tr>
<th>Type</th>
<th>Keyword</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>long integer</td>
<td>long</td>
<td>whole numbers</td>
<td>0L, 9223372036854775807L</td>
</tr>
<tr>
<td>double-precision floating point number</td>
<td>double</td>
<td>fractional numbers</td>
<td>0.0000000000000000 0.0000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000000000000000 0.0000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.0000000000000000 1.0000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000000000000000 0.0000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000000000000000 0.0000000000000000</td>
</tr>
</tbody>
</table>

### Primitive types (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Keyword</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boolean</td>
<td>true or false</td>
<td>true false</td>
</tr>
<tr>
<td>character</td>
<td>char</td>
<td>characters (letters, numerals, symbols)</td>
<td>'a', 'A', '1',</td>
</tr>
</tbody>
</table>
Representing numbers

There are two types for representing numbers.
Use `int` when you know you are dealing with whole numbers (e.g., when counting things).
Use `double` when you have to deal with fractions.

Memory can encode anything

- characters, strings, database records, music, images, video, weather models, etc
- numbers can be addresses to other locations in memory
- instructions in computer program

Representing numbers

Numbers have internal limitations on their precision.
An `int` can only represent values between -2,147,483,648 and 2,147,483,647.
A `double` can represent larger values but significant digits are lost.

```
10000000000000000.0 + 1 = 10000000000000000.0
```

Expressions

An expression is a formula for computing a value.

Examples:
```
1 + 1
1.0 / 3
(1.5 * 3) + 9 - 2
```
### Integer expressions

There are five basic operators you can use to make integer expressions:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>3 + 4 = 7</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>3 - 4 = -1</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>3 * 4 = 12</td>
</tr>
<tr>
<td>/</td>
<td>integer division</td>
<td>14 / 3 = 4</td>
</tr>
<tr>
<td>%</td>
<td>remainder</td>
<td>14 % 3 = 2</td>
</tr>
</tbody>
</table>

### Float expressions

The operators have a slightly different meaning when applied to doubles and floats:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>3.5 + 4.0 = 7.5</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>3.5 - 4.0 = -0.5</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>3.0 * 4.0 = 12.0</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>14.0 / 3.0 = 4.66666</td>
</tr>
<tr>
<td>%</td>
<td>remainder</td>
<td>7.5 % 3 = 1.5</td>
</tr>
</tbody>
</table>

### Order of operations

When computing an expression the operators *, / and % take precedence over + and -.

Otherwise order is left-to-right.

\[
\begin{align*}
1 + 4 \times 3 & = 13 \\
2 - 5 / 3 & = 1 \\
2 + 4 \% 3 & = 3 \\
5 - 2 + 3 & = 6 
\end{align*}
\]
BlueJ has a Code Pad tool to allow you to test snippets of code.

Variables

When we compute a value we usually want to store it somewhere so we can use it later.

A variable is a named container for storing a single piece of data of a specified type.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>age</td>
<td>double</td>
<td>37.5</td>
</tr>
</tbody>
</table>

The value of a variable can change. When a new value is put into a variable, the old one is forgotten.

The type of a variable cannot change.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>int</td>
<td>17</td>
</tr>
<tr>
<td>age</td>
<td>double</td>
<td>38.0</td>
</tr>
</tbody>
</table>
Variable declaration

Before we use a variable we must declare it, specifying its name and type.

We can optionally specify an initial value. Otherwise it is set to zero (for ints and floats).

```
int count;  // initially 0
double age = 37.5;
```

Assignment

Once we have declared a variable we can change its value as often as we like, using an assignment statement:

```
count = 3;
// count is now 3

count = 4 + 5;
// count is now 9

count = count + 1;
// count is now 10
```

Assignment

The format of an assignment statement:

```
count = 4 + 5;
```

Semicolons

Notice that any single statement in Java has to end with a semicolon (;).

This is the way Java knows you have finished one statement and are starting the next.
A variable name can be any sequence of letters and numbers or the "_" (underscore) character.

Good naming:
1. Is meaningful.
2. Has a consistent style.

The recommend style is 'bumpyCaps' style (aka 'camelCase').
Start with a lowercase letter.
String words together with an uppercase letter starting each word:
age, numberOfChildren, taxFileNumber, oldLength

Good style makes code readable by providing meaningful names:

```
age = 37.5;
numChildren = 2;
distToTarget = 50;
```

Bad style makes code unreadable by providing obscure names or mismatched styles:

```
a = 37.5;
n_c = 2;
Dt = 50;
```
Variables and expressions

Once a variable has been declared, it can be used in expressions to compute new values:

```c
int count = 7;
int size = 3;
int spaceNeeded = count * size;
// spaceNeeded is now 21
```

Variables and expressions

A variable can even be used in an expression that is assigned to itself:

```c
int count = 7;
count = count * 2;
// count is now 14
```

Variables and expressions

This kind of operation is so common that there are shorthands:

```c
count += 2;
// count = count + 2

count -= 2;
// count = count - 2

count *= 2;
// count = count * 2

count /= 2;
// count = count / 2

count++;  // count = count + 1

count--;  // count = count - 1
```