Data Types, Control Structures, and Teamwork

Overview

Data types
- Statements and expressions
- Literals and variables
- Types
- Operators

Control structures
- If
- While
- Do while
- For
- Enhanced for
- Return
- Variable scope

Teamwork
- Pair programming
- Teamwork as a process skill

Data types

Statements and expressions

A *statement* does something:

```
StdOut.println("Shut the Box");
int score = 45;
closed[n] = true;
```

An *expression* has a value:

```
9
n
closed[7] && closed[8] && closed[9]
```

If it can appear on the right side of =, it's an expression.

Nesting

Expressions can appear inside other expressions:

```
2 + (8 - 1) * 5
```

Some statements include other statements:

```
for (int i = 1; i <= 9; i++) {
  if (closed[i]) {
    StdOut.print("-");
  } else {
    StdOut.print(i);
  }
}
```

Literals and variables

A *literal expression* is a value to be used “as is”:

```
7
"You've shut the box -- you win!"
```

A *variable expression* is a name for a stored value:

```
i
score
closed
```

Types

There are eight *primitive types*:

byte, short, **int**, long
float, **double**
boolean
char

There are an infinite number of *object types*:

Array types:

int[], boolean[], double[][], …

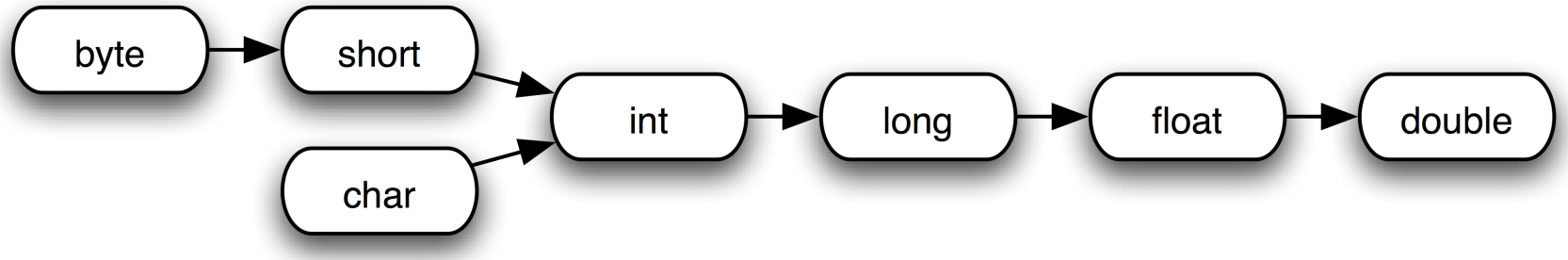
Classes built by others:

String, Color, …

Later you'll learn to define your own!

Type conversion

Java automatically converts from “less detailed” to “more detailed” primitive types:



It is therefore acceptable to write, e.g.:

```
double x = 3;
```

To convert in the other direction requires an explicit *cast*:

```
int n = (int)3.14;
```

Image: https://newcircle.com/bookshelf/java_fundamentals_tutorial/data_types

Operators

Arithmetic: `+ - * / %`

Assignment: `= ++ += -= *= /= %=`

Logical: `&& || !`

Comparison: `== != < <= > >=`

To see if two Strings are the same, use `a.equals(b)` instead of `a == b`.

Arrays should be compared element by element.

+ is also used to concatenate Strings. Non-string values are automatically converted:

`23 + "skidoo"` → `"23skidoo"`

Control structures

If

```
if (boolean expression) {
   statement
   …
}

if (boolean expression) {
   statement
   …
} else {
   statement
   …
}

if (closed[i]) {
   StdOut.print("-");
} else {
   StdOut.print(i);
}
```

While

```
while (boolean expression) {
    statement
    …
}
```

```
while (x >= 0) {
    StdOut.print(x);
    x--;
}
```

Do while

```
do {
    statement
    …
} while (boolean expression);
```

```
do {
    StdOut.print(x);
    x--;
} while (x >= 0);
```

For

```
for (statement; boolean expression; statement) {
    statement
    …
}
```

```
for (int i = 1; i <= 9; i++) {
    StdOut.print(i);
}
```

Enhanced for

```
for (type name : array) {
    statement
    ...
}
```

is exactly equivalent to:

```
for (int i = 0; i < array.length; i++) {
    type name = array[i];
    statement
    ...
}
```

```
for (int n : numbers) {
    StdOut.println(n);
}
```

Quitting early

`break` exits the innermost loop.

`continue` returns to the top of the loop.

`return` exits the current method. (If this method is `main`, `return` exits the program.)

Variable scope

A variable can only be seen in the code unit in which it is defined. Contrast:

```
for (int i = 1; i <= 9; i++) {
   StdOut.print(i);
}
```

```
int i;
for (i = 1; i <= 9; i++) {
   StdOut.print(i);
}
```

Variable scopes should be kept small.

Teamwork

Pair programming

https://www.youtube.com/watch?v=rG_U12uqRhE

Teamwork as a process skill

POGIL Interpersonal Effectiveness Videos, starting with #16.

Review

Statements do things, expressions have values; both can be nested.

Java has eight primitive types and infinitely many object types.

There are a variety of structures for controlling the flow of your program.

Working constructively and respectfully as part of a team is a skill to be deliberately developed.