SYSC-2004

From C to Java

A Quick Intro

Toqeer Israr

(Based on Dr. Babak Esfandiari's Lecture Notes)

(inspired from John Bryant's "From C to Java" slides)

Basic Syntax

Java (but also C, C++, C#):

- statements end with a semi-colon ";"
- blocks are surrounded by curly braces: "{" and "}"
- indentation isn't necessary but strongly recommended

```
int a = 1;
System.out.println("hello");
if (a > 1) {
    System.out.println("to all of");
};
System.out.println("you");
```

Variable Declarations and value assignment

In Java, variables need to be declared, along with their *type*. Assignment of value to a variable works the same in Java and C.

Java

```
int i; //i is of type integer
i = 4;
String greeting; //greeting is a string
greeting = "Hello";

//you can also combine declaration
// and assignment
String location = "Ottawa";
```

A few built-in primitive types

Integral Types

`byte` 1 byte (-128 to 127)
`short` 2 bytes (-32,768 to 32,767)
`int` 4 bytes (-2,147,483,648 to 2,147,483,647)
`long` 8 bytes (-9,223,372,036,854,755,808 to 9,223,372,036,854,755,807)

Real Types

`float` 4 bytes (6-7 significant digits)
`double` 8 bytes (15 significant digits)

Character Types

`char` 2 bytes (Unicode code)

Boolean Types

`boolean` 1 byte (`true` or `false`)

Literal Constants

int: 1, 7, 0, -4, 99

long: 45L, 567457895L, -56L

double: 45.6, 67.9, -45.2,
34e12, -4.1e-9 (scientific notation)

float: 45.6F, 67.9F, -45.2F,
34e12F, -4.1e-9F (scientific notation)

char: 'A', 'b', 'z', etc.
'\n' newline (linefeed)
'\t' tab
'\\' backslash

boolean: true, false

Primitive Type Conversions

Java allows some conversions between primitive types. The two lists below define which conversions are possible.

1/. byte > short > int > long > float > double

2/. char > int

Conversion is automatic if the "from" type appears to the left of the "to" type in either one of the lists.

An explicit *cast* is required if the "from" type appears to the right of the "to" type in either one of the lists (because information may be lost).

Otherwise conversion is impossible.

Conversion Examples

1/. byte > short > int > long > float > double
2/. char > int

```
int a;  long s;
double x;  boolean f;  char g;

s = a; // OK - automatic conversion
a = x; // illegal - cast required
a = (int) x; // OK
a = (int) f; // illegal (with or without cast)
a = g; // OK - automatic conversion
a = 45.6; // illegal - cast required
s = (long) 45.6; // OK (though peverse)
a = 8L; // illegal - cast required (try it)
```

Floating point to integral conversions involve truncation (the fractional part is lost).

The cast operator consists of the name of the type being cast to enclosed in parentheses.

While Loops

Java's while loops (and also C, C++, C#)

```
while  (<boolean expression>) {
        <statements>
    }
```

example:

```
while (value != -999) {
  .  .  .
}
```

The <boolean expression> can be anything that evaluates to a boolean value.

For Loops

Java loops:

for (<s1>; <s2>; <s3>) {
<statements>
}

is essentially equivalent to

<s1>;
while (<s2>) {
<statements>
<s3>;
}

ex:

```
for (int i = 0; i < 10; i++) {
   System.out.println("Hello number: " + i);
}
```

The If..else...

The Java if… statement:

```
if (<boolean expression>) {
        <statements>
    }
```

The if..else.. construct:

```
if (<boolean expression>) {
        <statements>
    } else {
        <statements>
    }
```

ex:

```
if (values_read == 0) {
   .  .  .
} else {
   .  .  .
}
```

Else If

```
if  (<boolean expression_1>) {
        <statements>
    } else if (<boolean_expression_2>) {
      <statements>
    } else if (<boolean_expression_3>) {
      <statements>
    } else { // note - the "else" part is optional
      <statements>
    }
```

There's also a *switch* statement, we'll see it if/when we need it…

Expressions

An expression consists of one or more operands tied together by operators.

Operands may be:
- variables
- constants (e.g. 0)
- *method* calls (similar to function calls, more on that later)

Operators include:
- arithmetic operators (+. -. /, *, %)
- relational operators (==, !=, <, >, <=. >=)
- boolean operators (!, ||, &&)
- increment and decrement (++, --)
- casts ((type_name))

Order of Evaluation

The order in which operations are to be performed can be explicitly specified by using parentheses (**strongly recommended**). What is to follow is only provided for completeness sake. Again: **just use parentheses**!!!

Otherwise operator "precedence" and "associativity" apply.

When operations have different precedence, the one with higher precedence is performed first.

When operations have the same precedence, they are performed either left to right (if they are "left associate") or right to left (if they are "right associative").

Precedence Table

Operators	Precedence	Associativity
`! unary- ++ -- (cast)`	Highest	Right
`* / %`		Left
`+ -`		Left
`< <= > >=`		Left
`&&`		Left
`||`		Left
`= += -= *= /= %=`	Lowest	Right

Note that '=' is also an operator (the result is the value assigned).

```
a = b = c = 0;  // all three variables get 0
```

The '+=' and so on are also just as in C.

```
a += 6; // equivalent to a = a + 6
z *= m + 4;  // equivalent to z = z * (m + 4)
sum += value;  // from sample program
```

Java Output (1)

C

```
printf("Enter a value: \n");
```

Java

```
System.out.println("Enter a value: ");
```

JAVA methods "System.out.print" and "System.out.println" accept a single string argument and write this string to the screen.

The two methods differ in whether or not a "newline" is appended to the string output.

Java Output (2)

Strings may be generated by concatenating smaller strings together. Strings are concatenated using the '+' operator which, in a string context, means concatenation rather that arithmetic addition. The following two statements have exactly the same effect.

```
System.out.println ("I am here.");
System.out.println ("I" + " am" +  " here.");
```

Values of type "int" and so on get automatically converted into equivalent string values when used in a string context. In the case below, the value in "values_read" is first converted into the equivalent string. This string is then concatenated with " values were entered".

```
System.out.println
      (values_read + " values were entered.");
```

Java Input

We will avoid doing input until later. In the meantime, the BlueJ environment will allow you to pass values to methods directly.

Basic Scope Rules (1)

In C, everything was either "local" (if it was declared inside a function) or "global" (otherwise).

```
// a global constant (accessible from anywhere)
float pi = 3.14;

void f1 (. . .) {
  int d = 2; // local to f1
  .  .  .
}
```

But in OO (C++ AND Java), "global" variables are considered "smelly" and should be avoided!

```
public class Scope { //more on this line later

  private int a; //local to Scope object: more later
  public int b; //bad idea: still local to Scope object
           //but accessible via Scope object

  public double f1 (. . .) {
    int d, g; // local to f1
    .  .  .
  }
}
```

Basic Scope Rules (2)

The “public” and “private” attributes indicate whether these elements may or may not be accessed from outside the class.

The class itself must be externally visible in order for the program to be executed, and is thus declared as being “public”.

The method “f1” and the variable “a”, in this example, are of purely local interest. Nothing outside the class needs to know about them, and they are thus declared as being “private”.

Summary

This brief introduction to Java has really only scratched the surface of the language.

We have yet to get into what is really the heart and soul of Java – “object oriented programming”.

Nonetheless, you should now be, in Java, about where you were in C.