Scarborough College
University of Toronto
Fall 1997

CSC A06F: Introduction to Computer Programming
Lecture notes 2: The elements of programs — Part 1

Reading:

- Lewis and Loftus, chapters 2 and 3.

Program structure

A Java program is a set of one or more classes.

A class is a set of one or more methods.

A method contains statements.

An example

```
import java.io.*;

// Prints a quote on the screen
public class Quote {
   public static void main (String args[]) {
      System.out.println ("  I have fallen off the wagon, ");
      System.out.println ("  for I am a slave to tea.");
   }
}
```

The `main` method

When you run a Java application it starts with the statements in the `main` method.

Every application requires a main method.

It must always be declared like this:

```
public static void main (String[] args) {
    ...
}
```

It looks for the main method in the `class` that has the same name as the file that you compile and run. (E.g., in `"Quote.java"`)

Statements

A programming **statement** is an instruction that is carried out when the program is compiled and/or executed. Three kinds:

- **Executable statement** : says to *do* something.
- **Declaration:** gives information necessary for the executable statements to work.
- **Comments:** are for humans. They are ignored by the computer.

There are *rules* , called the **syntax** of the language, that tell you how to build up valid statements.

Always use comments. Give some *insight* into why you wrote the program the way you did. Don't repeat the obvious.

Another programmer must be able to *read* and *understand* your program easily.

Anatomy of a statement

A statement is built up from **reserved words, symbols, identifiers,** and **literals** .

Reserved words

Reserved words are the official words defined in the language.

Some indicate what kind of action to take:

```
while   if   new
```

Others indicate what kind of information is being declared:

```
class   int   void   final   private
```

See Figure 2.1, page 41, for a list all the reserved words.

Symbols

Symbols serve a purpose similar to reserved words.

Some are operators (i.e., they do something to something):

```
+   -   *   .   =
```

Others are punctuation:

```
,   {}   ()   ;
```

Identifiers

Identifiers are words that you make up to name the things that your program uses and manipulates.

They have no predefined meaning in the language. (Except for `main`).

The things you can identify are:

- packages
- classes and interfaces
- methods
- variables
- constants

You can use any combination of letters, digits, the underscore (_), and the dollar sign (`$`). But a digit *cannot* be first.

Java is **case sensitive** , so `Max`, `max`, and `MAX` are all different identifiers.

Conventions for identifiers

There are conventions for identifiers that Java programmers use so that it's easy to tell what kind of thing is being identified.

Class	Begins with an uppercase letter	`Quote`
Method	Begins with lowercase	`main(...)`
Variable	Begins with lowercase	`max`
Constant	All uppercase	`MAX_STUDENTS`

Identifiers should be

- descriptive (not meaningless like `x` or `a`),
- but not overly verbose (i.e., not too long)

Make their meaning *obvious* to anyone who is reading the program.

Literals

Literals are explicit data values.

Integer literals:

```
17  -38465
```

Floating point literals:

```
3.1415927  -0.0005  59.99f
```

Character literals:

```
'a'  'A'  '$'  '\n'
```

String literals:

```
"for I am a slave to tea."
```


The Java API

Other programmers have already developed a lot of Java code that you can use.

It is called the Java **Applications Programming Interface** .

An API is any collection of pre-existing software.

The Java API provides many standard and fundamental services, like `System.out.println`, that are not predefined in the Java language.

It is divided into **packages** that you **import** in order to use.

E.g., `import java.io.*;` lets you use `System.out.println`

Software re-use is very easy with object-oriented programming.

Variables

A **variable** is a location in memory used to store a data value.

Key idea: you can change the value of a variable.

Without variables, you wouldn't be able to do much in a program.
You couldn't even get input from the user.

To use a variable, you must first declare it:

```
int count;
int max = 0;
```

Syntax of a variable declaration:

data-type variable-name;
OR
data-type variable-name = initial-value;

Referencing a variable

To use a variable, you make a **reference** to it. Its current value will be used. Example:

```
// An example of assignment to change the value of a variable
public class Assign_example {
   public static void main (String args[]) {
      int count = 193;
      System.out.println ("There were " + count + " students enrolled.");
      count = 213;
      System.out.println ("Now there are " + count);
   }
}
```

The first call to the `println` method references the variable `count`.

What does it output?

What does the second `println` do?

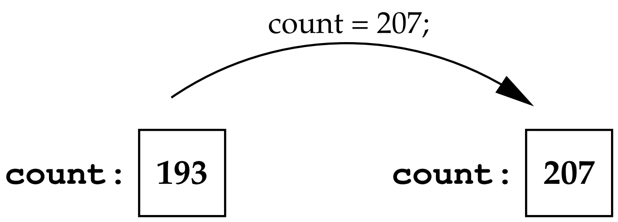
Assignment to a variable

An **assignment statement** puts the value of an expression into a variable.

```
variable-name = expression;
```

The assignment operator (=) says to evaluate the expression on the *right-hand side,* and assign the result to the variable on the *left-hand side.*

The old value is lost.



Primitive data types

A **data type** is defined by a set of values and the operations that you can perform on those values.

Every data value has a data type.

Java has eight built-in data types, which are called **primitive data types.**

Integers

Integers have no fractional part.

Four types:

Type	Storage	Min value	Max value
`byte`	8 bits	-128	127
`short`	16 bits	$-32\,768$	32 767
`int`	32 bits	$-2\,147\,483\,648$	2 147 483 647
`long`	64 bits	$-9\,223\,372\,036\,854\,775\,808$	9 223 372 036 854 775 807

Floating point numbers

Floating point numbers have a fractional part, although it can be zero.

Two types:

Type	Storage	Min value	Max value
`float`	32 bits	-3.4×10^{38}	3.4×10^{38}
`double`	64 bits	-1.7×10^{308}	1.7×10^{308}

A `float` has only 7 significant digits, e.g., can't represent 1234.5678

A `double` has 15 significant digits.

Characters

All the valid **characters** are kept in a ordered list called a **character set** . Even **control characters** like line-feed and tab, which don't have symbols.

Java uses the **Unicode** character set, which is an extension of the **ASCII** (American Standard Code for Information Exchange) character set.

Type	Storage	Values
`char`	16 bits	Anything from the Unicode set

Booleans

A **boolean** type can have one of two values: true or false (i.e., on or off, 0 or 1).

Type	Storage	Values
`boolean`	1 bit	true or false

The boolean literals: `true` and `false`.

Constants

A **constant** stores a value, but the value can never be changed.

For instance:

```
final int MAX_STUDENTS = 300;
```

declares the constant `MAX_STUDENTS` to have the value `300`. (It's just like a variable declaration, except that we add the word `final`)

Use a constant instead of a *variable* to avoid inadvertently changing its value.

Use a constant instead of a *literal* to make your programs more readable.

Expressions

An **expression** is any sequence of operators and data values that can be **evaluated** to give a new data value.

Arithmetic:

Operators: `()`, `+ -`, `* / %`, `+ -`

E.g., `5 * (2 + 3) + 7` $\Longrightarrow$ `27`

String:

Operators: `+`

E.g., `"xxx" + "yyy"` $\Longrightarrow$ `"xxxyyy"`

Boolean:

Operators: `< <= > >=`, `== !=`

E.g., `(2 < 5)` $\Longrightarrow$ `true`

Of course, an expression can have variable references in it, and anything else that gives an appropriate value (like calls to methods).

Input/Output (I/O)

Input and output is made through **streams** , which are sources or destinations for data.

The Java API lets you define many kinds of stream.

Two predefined output streams: `System.out` and `System.err`

One predefined input stream: `System.in`

Basic output to the screen is easy using the `println` and `print` methods of the `System.out` stream.

Input

To use the predefined input stream, you have to first convert it to a special kind of stream called a `BufferedReader`.

```
BufferedReader stdin = new BufferedReader
   (new InputStreamReader(System.in));
```

This creates a new stream called `stdin` out of `System.in`

A `BufferedReader` has a method called `readLine` that reads a string:

```
String firstline = stdin.readLine();
```

To convert a string to an integer, use the method `parseInt` of the `Integer` class.

```
int number = Integer.parseInt (firstline);



// OR,  all in one line!
number = Integer.parseInt (stdin.readLine);
```


The *if* statement

For making decisions that control the path of execution.

Syntax:

```
if (condition)
   statement;
```

The condition is simply a boolean expression. If it evaluates to `true`, then the statement is executed. If `false`, then the statement is skipped.

E.g.,

```
// if the number is bigger than max,
// then it's the new max

if (number > max)
   max = number;
```