Java Program Structure [Basic Structure]

One class – One Method

Class name is same as source file name!

Mumble.java

```java
// Comments
public class Mumble {
    public static void main(String[] args) {
        // Code for main()
    }
}
```
Java Program Structure [Basic Structure (2)]

One class – Multiple Methods

```java
public class ____ {

    public static <type> --- (<parameter list>) {
        ...
    }
    ...

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

    // Any number of functions of this form,
    // one of which is named 'main'.
}
}
```

*Multiple methods allows splitting functionality into manageable chunks.*

*Methods cannot be declared inside other methods (nesting). Other languages allow this. Java does NOT.*

Method calls must match declaration as far as name, parameter count, parameter type, and order.

```java
public static double myMethod(int a, double b) {
    // code
}
```

Call to `myMethod`:

```java
int a = 15;
double c = 99.3;
...
double result = myMethod(a, c);
```
Console Output

System.out.print(<strings and other objects>); // Print data without a line terminator
// Example
System.out.print("Enter a number: ");

System.out.println(<strings and other objects>); // Print data with a line terminator (move to next line)
// Example
System.out.println("Value entered was: " + val); // Java converts numeric data to strings for you!

Console Input

Read from the console with a ‘Scanner’ object:

import java.util.Scanner; // Note capital ‘S’!

Scanner can be used for other types of input. For now, we can set it up to read from the keyboard by giving an argument of ‘System.in’.

Scanner console = new Scanner(System.in);

The Scanner contains many methods for reading input and testing for input types.

int data = console.nextInt(); // Read an integer from console
double floatValue = console.nextDouble(); // Read a double from console
String name = console.next(); // Read a word (ending with ‘white space’: space, tab, or newline)
String wholeLine = console.nextLine(); // Read a complete line up to a newline character

Variables, Data Types, Values, Operators, Operator Precedence

Variables are words starting with an upper or lower case letter. They hold values according to the data type with which they are declared.

int width;       // This declares width and allocates space for it (integers are 32 bits or 4 bytes)
width = 10;      // This assigns a value to variable width (does not allocate space)
int height = 60; // This both allocates space for height and assigns a value
double percentage; // This allocates space for percentage (doubles are 64 bits or 8 bytes)
percentage = 3.3;

**Basic Java data types include**:

int – Integer (whole number)
double – Floating point number (Also known as a ‘real’ number with fractional part)
boolean – true/false
char – A single character
String – A sequence of characters. This is actually a Java object rather than a simple data type.
char answer = ‘n’; // Character literals are wrapped in single quote characters!
String greeting = “Hello, there!\n”; // Note upper case ‘S’! Also, double quotes for Strings

Data Type conversion and ‘Casting’

    int a = 5;
    double b = 5.5;
    double c;
    int d;
    int e = 16;
    double f;
    c = a + b; // Converts a to double and then adds! Automatically done by Java

    d = b * a; // Converts a to double, does multiplication and casts back to int. Java warns about ‘lossy’ conversion
    d = (int)(b * a); // Converts a to double, does multiplication. We ‘force’ convert to integer with cast.
    // Java does not warn us now about ‘lossy’ conversion

    f = e / a; // e and a are integers so Java does ‘integer’ division - loses fractional part (even though f is double)
    // Force cast one variable to ‘double’ and division will be correct
    f = ((double) e) / a; // Tell java, convert ‘e’ to double, then divide. Java now converts a to double and divides.

    ‘+’ also concatenates Strings;
    String greeting = “Hello, “ + “World!”;
    String answer = “The answer is: “ + 5; // Java will convert 5 to a String, then concatenate

    String answer2 = 3 + 2 + “ is the answer”; // Generates “5 is the answer”

    String answer3 = “3” + 2 + “ is the answer”; // Generates “32 is the answer”

Operators and Operator Precedence

Most frequently used arithmetic operators:

+ - addition
- - subtraction
* - multiplication
/ - division
% - modulo

Precedence

() 
- (unary minus: i.e., 8 * -5)
*, /, %
+, -
Binary Values and ASCII

Binary numbers have digits with 2 values (0, 1). In a computer, these can represent the presence or absence of a voltage. All digital circuits work with this binary paradigm so numbers are stored using it.

Binary = Base 2

\[ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \]

16 8 4 2 1

1 0 1 1 0 = 16 + 4 + 2 = 22 (decimal)

Binary to decimal conversion

. 1 0 1 0 1 0 0 1 1 = ?
  ^ ^
  | |
  1 |
  * 2 + 0
  2 * 2 + 1 = 5
  * 2 + 0 = 10
  * 2 + 1 = 21
  * 2 + 0 = 42
  * 2 + 0 = 84
  * 2 + 1 = 169
  * 2 + 1 = 339

Decimal to Binary Conversion

57 / 2 = 28 rem 1 ^
28 / 2 = 14 rem 0 |
14 / 2 = 7 rem 0 |
  7 / 2 = 3 rem 1 |
  3 / 2 = 1 rem 1 |
  1 / 2 = 0 rem 1 |

Collect remainders bottom up!

111001

Binary Addition

1 1 1 1 1

100101111
+ 110101010

----------

1011011001 (729 decimal)
Binary Subtraction

\[
\begin{array}{c}
10101010 \\
- 1010100 \\
\hline
1010000 \\
\end{array}
\]

Boolean Values and Operators

Booleans can be represented with 1 bit (only 2 values) but Java uses 8 bits for convenience.

Logical operators can be applied to Boolean values.
Boolean operators include:

! (NOT)  
&& (AND)  
|| (OR)  

\[
\begin{array}{c|c}
A & \neg A \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
A & B & A \land B \\
\hline
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
A & B & A \lor B \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]
Methods

Methods take parameters with specific types. May have 0 parameters. May have 1 or more. Methods also may return a value (int, double, etc) or may not return a value (void). The return type, method name, parameter names and types comprise the signature of a method. A method that calls another method is the caller. The called method is the callee. Parameters are local to a function. Any changes do not affect variables in the caller. Communication between caller and callee is through arguments (caller->callee) and return values (callee -> caller).

```java
public static int bar (int a, int b) {
    int c = 3;
    int d = 4;
    a = c;
    b = d;
    return a + b;
}

public static void foo () {
    int x = 1;
    int y = 2;
    int z = bar(x, y);
}
```

No affect on x and y in foo()!
Parameters are ‘call by value’