COMP 202 – Introduction to Computing

Tutorial 2 – Java Expressions
Data Types

• Primitive data types include
  • Boolean
    – boolean (1 bit – true/false)
  • Numeric
    – Integral (with whole number values)
    – Floating point (decimals)
Integral Data Types

• The character – char (16 bits)
• Integer values
  – byte: 8 bits \((-2^7, 2^7 - 1)\) or \((-128, 127)\)
  – short: 16 \((-2^{15}, 2^{15} - 1)\) or \((-32768, 32767)\)
  – int: 32 \((-2^{31}, 2^{31} - 1)\)
    … or \((-2147483648, 2147483647)\)
  – long: 64 \((-2^{63}, 2^{63} - 1)\) or…
    approximately \((-922337x10^{18}, 922337x10^{18})\)
Floating Point

• **float**
  - 32 bits, 7 significant digits
  - \( \pm 3.4e+38f \)

• **double**
  - 64 bits, 15 significant digits
  - \( \pm 1.7977e+308 \)
int x;  int x;  int x, y;  int x = 1;
int y;  int y;  int z;  int y = x;
x = 1;  x = y = 1;  x = 1;  x = 14;
y = 3;  
y = x;

int x;

int y = x;
x = 1;
Useful Operators

• Addition + (unary and binary)
• Subtraction – (unary and binary)
• Multiplication * (binary)
• Division / (binary)
• Modulus, or remainder % (binary)
/ and % with Different Types

- For integers, using the division will only yield the integer quotient.
  
  ```java
  int x = 13, y = 4;
  System.out.println("quotient = " + x/y + ", remainder = "+ x%y); // q = 3, r = 1
  ```

- For floating points, it’s a different story.
  
  ```java
  double i = 8.7, j = 4.1;
  System.out.println("quotient = " + i/j + ", remainder = "+ i%j); /*jkfhgfjghjgfj*/
  ```
int x = 1;
x += (x = 1);
System.out.println(x);

• Or try this…

x = x + (x = 1);

• They yield the same result.
The Many Magical Ways to Increment/Decrement in Java

```java
int x = 0;
int x = 0;
x = x + 1;
x = x - 1;
x += 1;
x -= 1;
x++;
x--;
++x;
--x;

• What is the difference between the prefix and the postfix?
```
Precedence 1

x + y * z; //which occurs first?
Order of precedence:
Array indexing, []
Postfix increment/decrement blah++, blah--
Parameter evaluation, doThis(parameters)
All these have the highest precedence in Java, and follow a left to right association.
Precedence 2

- Prefixes. ++x, --x
- Unary operators +, -
- Not operators, ~, !

- These follow a right to left association
Moving On

• Multiplication, Division, and Remainder
• Left to right association
• They are above binary addition and subtraction, and string concatenation ‘+’
  (again left to right)
Comparative operators

• Less than (or equal to), <, <=
• Greater than (or equal to), >, >=
• These follow the previous operators in precedence, but precede…
• Equals, Does not equal, ==, !=

• Left to right association.
• Then come the and/or operators
• &&, ||
• Finally, the assignment operators
• =, +=, -=, *=, /=, %=…
• Left to right, again.
Examples

• System.out.println(1 + 2 * 3);
• int x;
• System.out.println((x=3) + 4);
• System.out.println((x=3) + x);
• x = 1;
• x += 1;
• x = 2*x + 1; // so x is 5
• System.out.println(x++);
One other thing

- If it is inside brackets, it takes precedence.
Conversion

• Say you want to make the width of a variable larger/smaller.
• Or you want to turn an integer to a floating point number.

```java
int x = 7;
long y = x;
// now y wasted 64 bits storing the number 7
```
float z = 6.44;
int w = (int)z;  // w = 6
byte v = (byte)w;
// v = 6, but this is stored in just 8 bits
int e = 2147483647;
int d = e + 1;  // not good
long c = e + 1;
Strings

• Not a primitive data type.

```java
String str1 = "Obey";
String str2 = new String("Obey");
int len1 = str1.length();
int len2 = str2.length();
int diff = len1 - len2;
str2 = "our orders";
System.out.println(str1 + " " + str2 + " or else");
System.out.println(diff); // what does this print?
```
Issues that Will Haunt You for Life

• Always make sure you count the brackets.
• Also, make sure what you want inside/outside the brackets, actually is inside/outside the brackets.
• You may find errors to occur unexpectedly; check the line above the erroneous one to verify the semicolon was typed in.
• Do not intentionally try to be fancy; often you may come back sometime later and no longer understand what you did.
• Case in point, avoid tricky commands like break, goto, and so on.