Why use Java?

- Large developer support and 3rd party libraries
- Garbage collection
- Object Oriented Framework
- Optimized language ("Just in Time" Compiling)
- Portability (can write once, run anywhere)

Hello World Program

```
public class HelloWorld {
      public static void main(String [] args) { // or String args []
            System.out.println("Hello World!");
      }
}
```

Comments

- Can comment blocks of code with `/* … */`
- Can comment single lines of code with `//`

Identifiers

- Can start with an underscore, $, or letter and can contain letters, digits, or underscore.
  - Examples: `var1, _temp, $total`
  - Normally, Java programmers do not use '_' or '$' in variable names.
- Identifiers are case sensitive.
  - `Var != var` – these are two different variable names.
- Cannot use predefined keywords
  - if, else, for, do, new, public, private, final, byte, short, int, …
  - For an entire list of Java keywords, see http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

Integers

- byte (1 byte): from -128 to 127
- short (2 bytes): from -32768 to 32767
- int (4 bytes): from -2147483648 to 2147483647
- long (8 bytes): from – 9223372036854775808 to 9223372036854775807

Floating Point Numbers

- Float (4 bytes): 3 .4e–038 to 3.4e+038
- Double (8 bytes): 1 .7e–308 to 1.7e+308
- Example: 3.14, -2.5, 6.02e23
- Normally don't use floats unless we're running out of memory.

Declaring Variables

- You can declare variables on single lines (one per line):
  - `int x;`
  - `int y;`
- You can combine multiple declarations in the same line if the variables are of the same type:
  - `int x, y;`
- Usually variable names are created with camelCasing without underscores.
  - Example: `isEmpty, numOfStudents`
- Every variable must be initialized before it is used.
- The life of a variable usually is valid within the block it was declared in between "{ }" (in general)
  - Also known as the **scope** of the variable.

Constants

- Example: `public static final int NUMBER_OF_STUDENTS = 200;`
  - "public" means anyone can use it.
  - "static" means you don't have to instantiate an object to use it and this is the only one.
  - "final" means this value can't be changed in other parts of the code.
  - "int" is the type of the variable
  - NUMBER_OF_STUDENTS is the name of the variable.
    - For constant variables, it's common for the name to be in all caps with "_" separating the words.
- Consider putting all constant variables in a single location (i.e. Class file)
  - Easy to make changes in a single location without having to scan through the entire project.
- Example:

```
public class Constants {
   public static final int NUMBER_OF_STUDENTS = 200;
   // …
}

// in main
System.out.println(Constants.NUMBER_OF_STUDENTS);
```

Assignment Statements

```
float sum = 0.0;
int a = 1, b = 2, c = 3;
```

- You cannot assign variables with different types
  - `int a = "a"; // illegal!`
  - `int x = 1; double b = x; // legal (remember why?)`
  - `double c = 1.1; int y = c; // illegal!`
- We can make it legal by **type casting** values, but we possibly loose precision
  - `int y = (int) c; // legal`

Objects

- Java allows developers to create / use their own objects.
- The "new" keyword is used to allocate memory for a new object.
- Construction:

```
Object x = new Object();
Object y  = x; // y and x refer to the same Object
y = new Object(); // y refers to a new object
x = y;   // x and y refer to the same object.
         // Nothing points to the original object. Java's
         // garbage collection automatically removes it from
         // memory.
```

- "==" compares object **references**, not values.

Expressions

Arithmetic

*	multiply
/	divide
%	modulo
+	addition
-	subtraction

Increment / Decrement

```
- x++; // post-increment – increments x by 1 after the expression is evaluated.
- ++x; // pre-increment – increments x by 1 before expression is evaluated.
- x--; // post-decrement – decrements x by 1 after the expression is evaluated.
- --x; // pre-decrement – decrements x by 1 before expression is evaluated.
```

- **Example:**

```
int x = 1;
System.out.println(x++); // prints 1
System.out.println(x); // prints 2

x = 1;
System.out.println(++x); // prints 2
System.out.println(x); // prints 2
```

Shortcut Expressions

```
v1 += v2; // v1 = v1 + v2
v1 -= v2; // v1 = v1 – v2
v1 *= v2; // v1 = v1 * v2
v1 /= v2; // v1 = v1 / v2
```

- These are nice, but use them sparingly because it can make readability hard.

```
int x = 10, y = 20, z = 30;
x += --y – z++; // ☹
```

Math Functions

- Java's math library has many common mathematical methods you can use in your code.
  - ceil(double x) round double up (return double)
  - floor(double x) round double down (return double)
  - pow(double x, double y) x^y (return double)
  - sqrt(double x) square root of x (return double)

Examples:

```
System.out.println(Math.ceil(4.3));       // returns 5.0
System.out.println(Math.floor(4.3));      // returns 4.0
System.out.println(Math.pow(2,3));        // returns 8.0
System.out.println(Math.sqrt(100));       // returns 10.0
```

Arithmetic Conversions

- If two operands of different types are used, result is converted to the "highest" type.
  - `System.out.println(5 / 3); // returns 1`
  - `System.out.println(5.0 / 2); // returns 2.5`

Typecasting

- Temporarily changing the type of a variable during an operation.
- Example:

```
int x = 5;
int y = 2;
double answer = (double) x / y; // 2.5 even though two ints used
```

Overflow

- Java does not check for overflow
- Example:

```
int i = Integer.MAX_VALUE;
System.out.println(i); // 2147483647
System.out.println(i + 1); // -2147483648
System.out.println(Integer.MIN_VALUE); // -2147483648
// No compilation / runtime error. User must check these cases
```

Strings

- Strings are objects of class String (not a primitive type like "int")
- Strings are characters stored as an array

```
String name = "Richert";
String name = new String("Richert");
```

Escape characters

- `\t` insert tab
- `\n` insert newline
- `\'` insert single quote
- `\"` insert double quote
- `\\` insert forward slash

Examples:

```
System.out.println("a\tb\tc");
System.out.println("d\ne\nf");
System.out.println("g\'h\'i");
System.out.println("j\"k\"l");
System.out.println("m\\n\\o");
```

String methods

`.length()` – returns number of characters in the string

```
System.out.println(name.length());
```

`.substring(int start, int end)` – returns the substring starting at position start up to and not including position end

- Recall – arrays start at index 0 to n – 1.

`.toUpperCase(), .toLowerCase()`

```
String s = new String("Hello");
System.out.println(s.toUpperCase()); // HELLO
System.out.println(s.toLowerCase()); // hello
```

`.charAt(pos)`

```
System.out.println(s.charAt(2)); // 'l'
System.out.println(s.charAt(100));
// ERROR – StringIndexOutOfBoundsException
```

`.equals(String s)`

Checks if two strings have the same value, not reference.

```
System.out.println(s.equals("Hello")); // Prints true
System.out.println(s.equals("hello")); // Prints false
```

`.equalsIgnoreCase(String s)`

```
System.out.println(s.equals("Hello")); // Prints true
System.out.println(s.equals("hello")); // Prints false
```

`s1.compareTo(String s2)`

- Lexicographical comparison
- `== 0 if same`
- `> 0 if s1 > s2`
- `< 0 if s1 < s2`

```
String a = "a";
String b = "b";
String x = "A";

System.out.println(a.compareTo(a)); // == 0
System.out.println(a.compareTo(b)); // < 0
System.out.println(a.compareTo(x)); // > 0
```

Concatenation

Combine strings with '+'

```
System.out.println(3 + 4 + "Hi"); // 7Hi
System.out.println(3 + (4 + "Hi")); //34Hi
System.out.println("3" + 4 + "Hi"); //34Hi
```

Converting a String to an Integer

```
String s = "3";
int i = Integer.parseInt(s);
System.out.println(i + 5); // 8
```

Converting an Integer to a String

```
String s = Integer.toString(3);
System.out.println(s); // 3

String t = String.valueOf(4);
System.out.println(t); // 4

String u = "" + 3;
System.out.println(u); // 3
```

In-class String example:

Recall:

```
String s = "Hello";
String t = "Hello";

System.out.println(s == t); // prints true

t = new String("Hello");
System.out.println(s == t); // prints false
```

If "==" compares references and not values, then why does the first print statement return true?

- Java does some optimization by storing identical Strings that are declared in code to reference the same location in memory.
  - These strings are stored in a "String pool"
- If the String changes, then Java will allocate another section in memory for the updated string.
  - Makes sense since Java Strings are immutable (i.e. they can't be changed in memory once created).
- The "new" keyword bypasses the String pool and creates a separate memory location for the initialized String.
- For a more detailed explanation of this behavior and String pools, refer to http://stackoverflow.com/questions/2486191/what-is-the-java-string-pool-and-how-is-s-different-from-new-strings
- http://www.programcreek.com/2013/04/why-string-is-immutable-in-java