Java Training

Day 2: Variables, Primitive Types vs Objects & Boxed Primitives

Return to ‘HelloWorld’

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

Your `HelloWorld.java` should look something like this

Recompile and run it again to verify it still works (`javac, java` …)

Next we’ll make some changes to alter the program’s behavior….

Experiments with “Hello World!” Part #1 (and brief discussion of variables)

```
public class HelloWorld {
     public static void main(String[] args){
          String s = “Hello Variable”;
          System.out.println(s);
     }
}
```

A `String` formed from a set of quoted characters is called a *string literal*

`println()` will take any argument of type `String`, including a variable

#1: Change your HelloWorld program by creating a variable `s` and printing that in place of the literal (make the changes shown above then recompile and run)

Experiments with “Hello World!” part #2 (Concatenation, ‘+’ operator)

```
public class HelloWorld {
     public static void main(String[] args){
          String s = ???????;
          System.out.println(s);
     }
}
```

`'+'` can be used to concatenate Strings together, like so: `“S”+”O”+”S” = “SOS”`

#2: Create a concatenated `String`, store it in `s`, then print it

(for your literal strings use: `“yellow”, “brick”, ”road”`)

Part #2
(Possible Solution)

```
public class HelloWorld {
    public static void main(String[] args){
        String s = “yellow” + “brick” + “road”;
        System.out.println(s);
    }
}
```

Experiments with “Hello World!” part #3 (Concatenation in a method argument)

```
public class HelloWorld {
    public static void main(String[] args){
        System.out.println(???????);
    }
}
```

#3: Create a concatenated `String` within the call to `println()`

Use “yellow”, “brick” and “road” again

Part #3
Possible Solution

```
public class HelloWorld {
    public static void main(String[] args){
        System.out.println("yellow"+"brick"+"road");
    }
}
```

Java (like C and C++) is pretty easy-going about whitespace

Experiments with “Hello World!” part #4 (more variables)

```
public class HelloWorld {
     public static void main(String[] args){
          String s2 = ???????;
          String s = ??????;
          System.out.println(?????);
     }
}
```

#4: Add a new variable `s2` of type `String`, store a string literal in it, then concatenate `s2` with another literal into `s`, then print `s`

What happens when you concatenate “`\n`” into your `String s`?

part #4
(Possible Solution)

```
public class HelloWorld {
        public static void main(String[] args){
                String s2 = "yellow";
                String s = s2 + "brick";
                System.out.println(s);
        }
}
```

Experiments with “Hello World!” part #5 (command line arguments)

```
public class HelloWorld {
     public static void main(String[] args){
          System.out.println(?????);
     }
}
```

The `args` variable containing the command-line arguments is a `String Array` object.
Values are accessed by index using `[]`, for example:

`args[0]` is the first command-line argument
`args[1]` is the second argument
`args[2]` is the third, etc… each one is a `String` variable (like `s` and `s2` in the previous example)

#5: Modify the program so that it prints the second command-line argument, for example:

```
> java HelloWorld lions tigers bears
```

should print: `The second argument is: tigers`

Part #5
Possible Solution

```
public class HelloWorld {
    public static void main(String[] args){
        System.out.println("The second argument is: "+ args[1]);
    }
}
```

Experiments with “Hello World!” part #6 (argument challenge)

```
public class HelloWorld {
     public static void main(String[] args){
               ????????
     }
}
```

#6: Modify the program so that it prints the first three command-line arguments on separate lines, for example:

```
> java HelloWorld lions tigers bears
```

prints:

```
lions
tigers
bears
```

Part #6
(Possible Solution)

```
public class HelloWorld {
   public static void main(String[] args){
        System.out.println(args[0] +"\n"+ args[1] +"\n"+ args[2]);
   }
}
```

More about Variables

There are are other variable types, such as `int`
An `int` is for storing an integer numeric value

The `+` operator, when applied to `int` does addition (rather than concatenation as it does with Strings)

```
public class HelloWorld {
    public static void main(String[] args){
        int x = 5 + 1;
        System.out.println("The number is: " + x);
    }
}
```

- Unlike `String`, which is an object type, `int` is a **primitive** type
- Primitive types are simple and not *self-aware*, so you **can't** call methods on them (send them messages)
- Object types names (like `String`) are capitalized. Primitive types names (like `int`) are lower-case

Primitive Types & Object Types

Primitive Types

Java interprets the value of bits at the memory location as a value

```
int number = 5;
```

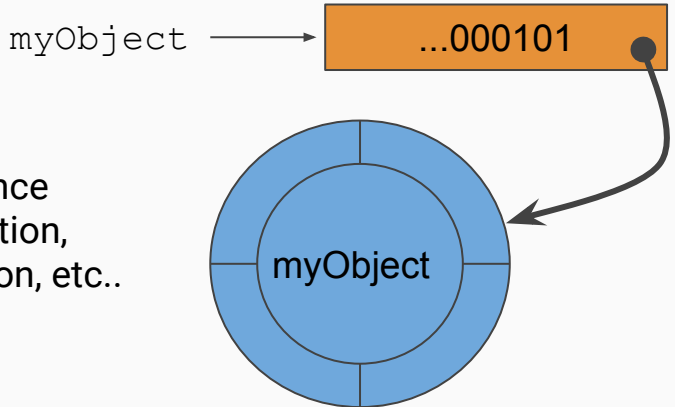
You can't *do* much with a primitive except read and modify the stored value

(These limitations are why Java has **boxed** primitives)

Object Types

The bits at a memory location are interpreted as a pointer (or reference) to information about an object

```
SomeClass myObject = new SomeClass();
```



An object reference allows introspection, method invocation, etc..

When a Primitive isn't enough: Boxed Types (explained with Integer)

Sometimes you need an whole number numeric type (an `int`), but you need it to be *smart* enough to receive messages (have methods)

For `int`, there is the boxed-type: `Integer`

```
int x = 5;                          // x is a primitive

Integer y = Integer.valueOf(x);       // boxes x into an Integer y

Integer z = Integer.valueOf("12");     // also works with String

Integer f = Integer.valueOf("pickles"); // this will fail at runtime

String s = y.toString();          // y knows how to turn itself into a String
```

When a Primitive isn’t enough: Boxed Types (explained with Integer) continued...

`Integer` and `int` play well together:

```
Integer y = 19;

int x = 3;

Integer v = x + y; // only works with int, easier that valueOf()

int n = Integer.valueOf(“42”);   // boxes, then unboxes immediately
```

“Hello World!”
2 number Adder challenge

Using what you’ve learned so far, turn HelloWorld into a two parameter integer adder so that:

```
> java HelloWorld 24 3
```

Will print:

```
The answer is: 27
```

The parts you will need:

1. The `args` parameter to `main()` and access to indexes of that array using the syntax : `args[0]`
2. The `Integer` box type and the class method `valueOf()`
3. The `println()` class method

Now change the program so the name of the class and application is `Adder` rather than `HelloWorld`