First Java Program - Output to the Screen

These notes are written assuming that the reader has never programmed in Java, but has programmed in another language in the past. In any language, one of the first tasks we learn is outputting some text to the screen. Unlike other languages where there's very little extra syntactic sugar (extra stuff you have to do to follow the rules of the language), Java has a ton of syntactic sugar in comparison. It will take a few days to learn why it's all there, but for now, there's going to be quite a bit that you just copy down without fully understanding what it's doing. Here is a Java program that prints out "Hello World!" (no quotes) to the screen:

```
// Hello.java
// My first Java program prints a greeting to the screen!

public class Hello {                                  // line 4
     public static void main( String args[] ) {  // line 5

          // Prints output                              // line 7
          System.out.println("Hello World!");    // line 8
     }                                                    // line 9
}                                                         // line 10
```

All Java programs are in classes. The name of the file must be the same as the public class defined in the file. Thus, this program above, which defines the class Hello must be stored in a file called Hello.java. For now, lines 4, 5, 9 and 10 can all be copied, roughly as is (except for changing the class and file name for different programs.)

The only line that actually does anything in this program is line 8. Java has a predefined function that prints output to the screen. That function is called println (there is another one called print that works similarly) but System.out specifies where that output will go. All functions in Java (just like most langauges) are specified with parentheses. So, after the function name, we'll see an open parenthesis, possibly followed by some stuff, then followed by a matching closed parenthesis. The end of each statement in Java must have a semicolon. (This is similar to a period in English.) The println function takes as input a String. The most simple string we can provide the function is a string literal - a fixed string value. To denote a string literal, we use matching double quotes. Everything inside the matching double quotes represent the actual string, in this case, Hello World! Now, there are

exceptions to this. For example, what if we wanted to print a double quote? If we just placed ", then the computer would think that we're ending our string literal. For this reason there are a few special characters. If we want the double quote to be part of our string literal, we write the two character sequence, \". Anytime you see a backslash inside of a string literal, it does NOT mean backslash. Rather, it's an indicator that the character to be designated is a special character. Some of the special character codes (also called escape sequences) are:

newline	- \n
tab	- \t
double quote	- \"
backslash	- \\

There are some others, but this is good for now. If you experiment with using these, you'll see how they work. Also, the difference between print and println is that println automatically adds a newline character ('\n') after whatever the println statement outputs, whereas print doesn't. To see the difference, consider these two segments of code (you can think of substituting these in the program on the previous page for line 8):

Code Segment #1

```
--------------------
System.out.print("Hello ");
System.out.println("World!");
```

Code Segment #2

```
--------------------
System.out.println("Hello ");
System.out.println("World!");
```

The output for the two code segments are:

Code Segment #1 Output

```
------------------------------
Hello World!
```

Code Segment #2 Output

```
------------------------------
Hello
World!
```

For both code segments, if we were to add a third print, that string would start printing on the following line in both. The key here is that in the second segment, the println added a newline character after the space after the 'o'. In the first code segment, this character wasn't added.

There are some other nuances about printing, but we'll handle these as they come up.

Second Java Program

Now that you have mastered your first Java program, let's move on to the next one. In this lesson we will develop a couple of key tools that you will use for the rest of the course : variables, output format, and how to read in input from the user.

Variables

The first topic we will cover is the variable. In programming, variables act as containers that can store different values. Take a look at the following program, Circle.java:

```
// Circle.java
// My second Java program calculates the area of a circle :)

public class Circle {
      public static void main( String args[] )
      {
            // Calculates the area of a circle with radius 10
            double radius, area;
            radius = 10;
            area = 3.14159*radius*radius;

            // Prints output
            System.out.print("The area of a circle with radius " +
radius);
            System.out.println(" is ", + area);
      }
}
```

In this program, we used two variables – radius and area. First the variables are declared, as in the following line:

```
double radius, area;
```

When declaring a variable, we must first tell the compiler what type of value we are expecting it to store. In this case, our variables will store values of type double. Variables also commonly store variables of type int (1, 22, -45), char ('a', 'b', 'c'), string ("Java", "BHCSI"), and boolean (true, false).

You'll notice that I have declared more than one variable on the first line of this program. Alternatively, this could also be written out as

```
double radius;
 double area;
```

but we can save ourselves time and space by writing both declarations on one line. As long as the variables you are declaring are of the same type (double, int, char, etc.), this is permissible. The general syntax is as follows:

<type> <var1>, <var2>, ... , <varn>;

You list the type, followed by each variable separated by a comma. After the last variable, you must have a semicolon to signify the end of the declaration.

Although it is permissible to declare many variables on a single line, it is not a good programming practice to stick every int variable you use in the same declaration. The general rule of thumb is that you should have related variables declared on a line together, but no more than about 5 or 6.

After the variables are declared, they can have values assigned to them. For example:

```
radius = 10;
area = 3.14159*radius*radius;
```

Here, radius stores the value 10. That is, whenever the compiler sees the variable `radius`, it replaces it with the value 10. So, by setting `area` equal to 3.14159*`radius`*`radius`, we are really setting it equal to 3.14159*10*10, or 314.159.

It seems redundant to declare a variable radius in this program. We could simply combine the two lines

```
radius = 10;
area = 3.14159*radius*radius;
```

into one line as follows:

```
area = 3.14159*10*10;
```

However, when written out as a variable, it is easier to understand what is happening in the program. Furthermore, once we learn to read input from the user, we will need the variable `radius` to store the value entered by the user.

Also, it seems silly to write out the value of pi. What if we had a program that used this value several times? It would be nice to be able to refer to this value (which doesn't change) in a symbolic manner, as we refer to variables. In Java, a `final` variable declaration provides such a function. Here is how we should change the initial code to use a `final` variable, or constant definition:

```
final double pi = 3.14159;
double radius, area;
radius = 10;
area = pi*radius*radius;
```

When we use a `final` variable declaration, the programmer is not allowed to change the value of the variable from that point on. Hence, the identifier pi, in this case, really stands for a constant value. Here are two reasons to use constants instead of integer literals:

1) An identifier gives a constant meaning. If you see the value 1 in a program, it is difficult to decide its meaning. However, a constant named `h2odensity` would be more clear.

2) If you happen to need to change the value of a constant when you run a program, you need only change the value in one place, not every place you actually used it. Also, if you just used literal values, it may be unclear which ones to change. (Perhaps you should only change some of the 2's to 4's, based on their function in the program.)

Output Formatting

Finally, we need a way to print out the information the program has calculated. We can do this with print statements. In our first program, we learned how to print out strings. But, in this program, we would like to print out the value of a variable, area. Here is the general syntax of a print statement:

```
System.out.println(<item1> + <item2> + ... + <itemn>);
```

Each item may either be a variable of a particular type or a literal value Typically, the only type of literal value included in a print statement is a string literal. A string literal always lies in between two double quote marks(“ ”). Basically, if you print out a string literal, whatever is in between the two quote marks gets printed out exactly as it appears. (There are a few exceptions, which we will get to.)

This choice of syntax leads to the question: how does the computer differentiate between the plus sign that means adding two numbers and the plus sign that means concatenating two strings together? Here is the answer:

An expression, as we mentioned, is evaluated from left to right. Thus, the first part of the expression that is evaluated is <item1> + <item2>. When considering how a single plus operation is interpreted, the compiler checks the types of each of the operands. If both are numeric (int, long, float or double), then the resulting answer will also be numeric. However, if at least one of the operands is a string, the operation performed is a string concatenation, which also produces a string. So, you are guaranteed, if the first item is a string, for the print statement to work as you wanted it to.

In certain cases, you will want to do arithmetic inside of a print statement. Consider the following println statements:

```
System.out.println(“2 + 5 = “ + 2 + 5);
System.out.println(2 + 5 + “ = 2 + 5);
System.out.println(2+5);
```

The corresponding output from these three lines is:

2 + 5 = 25
7 = 2 + 5
7

(Note: The output does not print in bold. I am just doing that in the notes so you can differentiate it as the output.)

In the first example, what happens is “2 + 5 = “ + 2 is interpreted as string concatenation. Thus, this expression evaluates to the string “2 + 5 = 2”.

Now, the final operation that is executed is “2 + 5 = 2” + 5, which also returns a string. In the second and third examples, the first plus operation is interpreted as addition instead of string concatenation, explaining the remaining output. But, it seems natural to write the print statement in the form of the first one. To do this, a simple fix is to use an extra set of parenthesis, as follows:

```
System.out.println(“2 + 5 = “ + (2 + 5));
```

The inner parenthesis give a higher order of operation to the second plus sign, meaning that 2+5 gets executed first. Since both of these operands are integers, an addition is performed. Thus, the output is as desired:

2 + 5 = 7

One of the things you will notice is that our program has one print statement and one println statement. Here is the difference between the two:

`Print` statements work just like a typewriter. The output of successive `print` statements goes from left to right, top to bottom. If you have a `println` statement, after everything in the `println` statement is printed out, “the cursor” advances to the left hand side of the next line, like a carriage return on a typewriter. However, with a `print` statement, “the cursor” is left directly to the right of the last character printed out. In our example, the output of executing

```
System.out.print(“The area of a circle with radius “ + radius);
System.out.println(“ is “, + area);
```

is

The area of a circle with radius 10 is 314.159

Had both statements been `println`s, the output would have been as follows,

The area of a circle with radius 10
is 314.159

Finally, if both statements had been `print` statements, the output would have all been on one line, but a subsequent `print` statement would have started on the same line, directly to the right of 314.159.

Input

As we mentioned before, our program would be much more interesting (okay, a little bit more interesting!) if we could ask the user to enter what the radius of the circle was. We already know how to ask – just use a `print` statement. In most languages, reading in input is perfectly analogous to printing output. In Java, reading input used to be very complicated, but in the new version of Java, 1.5, it is easier. First, you must declare a Scanner to read from the keyboard as follows:

```
Scanner stdin = new Scanner(System.in);
```

In essence, `stdin` is a variable of type Scanner. You need a Scanner variable to read in input from the keyboard. Here, we are telling the `stdin` Scanner to get ready to read input from System.in, or the keyboard. Once you have this declared, there are a few different methods you may use to read in information from the user, depending on what type it is.

For example, if you know the user will be entering an integer, we can use the following statement to take the user input from the keyboard and store it in the variable `num` (assuming `num` is already declared):

```
num = stdin.nextInt();
```

Basically, the name of the method is nextInt. It is a method instead of a variable because of the parentheses following it. What this method does is read in the next Integer from the associated Scanner object (which, in this case is reading from the keyboard). Then, the method returns this integer. We then, take the returned integer and store it in num with the assignment statement.

In a very similar manner, we can read a double from the keyboard into a variable `val` as follows:

```
val = stdin.nextDouble();
```

To read in a String into the String variable name (assuming it's already declared), we do the following:

```
name = stdin.next();
```

In general, information entered by the user is read into and stored in some variable declared in the program.

Using what we have learned in today's lecture, we can improve our circle program :

```
// Circle.java
// My improved second Java program calculates the area of a
circle :)

public class Circle {
      public static void main( String args[] )
      {
            // Calculates the area of a circle using the radius
            // from the user.
            Scanner stdin = new Scanner(System.in);
            final int pi = 3.14159;
            double radius, area;
            radius = stdin.nextDouble();
            area = pi*radius*radius;

            // Prints output.
            System.out.print("The area of a circle with radius " +
radius);
            System.out.println(" is ", + area);
      }
}
```

Using the tools we have developed so far, we can write programs to prompt the user for information, use that information for calculations, and print the result of those calculations to the screen.

File Input

We can read from a file just as easily as the keyboard. In order to do this, we must set up a Scanner to read from a file instead of the keyboard. Here is an example that creates a Scanner `fin` to read from a file input.txt:

```
Scanner fin = new Scanner(new File("input.txt"));
```

Here instead of the Scanner constructor taking in System.in, it takes in a new File object indicating the file from which the Scanner `fin` will read.

The key to understand here is that when we set this up, it's as if `fin` is a bookmark to the file input.txt set to the beginning. Whenever we try to read something from the file, such as this:

```
int num = fin.nextInt();
```

then, the next token in the file is read in as an integer and stored in num. Subsequently, the "bookmark" `fin` in the file input.txt moves past the number that was just read in.

When there are no more contents to be read from the file, we should close the file with the following single statement:

```
fin.close();
```

If you try to read a token from a file and none exists, an exception will be thrown and your program will crash.

In the following program, we read in two integers from the file input.txt and print out the area of circles with those radii. It is assumed that the input file is located in the same directory as the program – make sure the input.txt and circle2.java files are in the same folder.

```
//Circle2.java
//The third Java program calculates the area of a circle with a
user-specified radius

public class Circle2 {
      public static void main( String args[] )
      {
            // Calculates the area of a circle using the radius
            // from the user.
            Scanner fin = new Scanner(new File("input.txt"));
            final int pi = 3.14159;
            double radius, area;
            radius = fin.nextDouble();
            area = pi*radius*radius;

            // Prints output for first circle.
            System.out.print(“The area of circle 1 with radius “ +
radius);
            System.out.println(“ is “, + area);

            // Gets the second radius and prints out the
            // corresponding area.
            radius = fin.nextDouble();
            area = pi*radius*radius;
            System.out.print(“The area of circle 2 with radius “ +
radius);
            System.out.println(“ is “, + area);
            fin.close();

      }
}
```