Chapter 2 Input Output

Java AP

One important feature of most applications is the ability to get input from the user. From word processors to video games, most applications take some form of user input to control what the program does. Once we have input we can output the results of processing that input. In this chapter we are going to concentrate on console input and output of numerical values, and text.

Console Output

In the variables section you may have noticed the code **System.out.println** being used although was not explained. This is something we call a function (also known as a method). We use it to output some data to the console.

System.out.println():

This function allows us to output a single line to the console. For example we can output an integral value as follows:

Example:	
	```
	public static void main(String[] args)
	{
	   int num = 45;
	   System.out.println(num);
	   int num2 = 2459;
	   System.out.println(num2);
	}
	```

Output:	45 2459

Notice that each value is on its own line. A newline is inserted after every **System.out.println** call. A newline is simply a means to say go to the start of the next line before writing more text.

Any variable type can be passed to **System.out.println()**.

System.out.print():

This method give us the same output as print line but without adding a newline at the end.

Example:

```
public static void main(String[] args)
{
   int num = 45;
   System.out.print(num);
   int num2 = 2459;
   System.out.print(num2);
}
```

Output:

```
452459
```

Since there was no newline between our values, the numbers in the output ended up connected together. We can fix this by adding a space between the two values.

To add a space, we will add a **System.out.print** between the two existing ones which will add a space character **" "**.

Example:

```
public static void main(String[] args)
{
   int num = 45;
   System.out.print(num);
   System.out.print(" ");  /*Print out a space */
   int num2 = 2459;
   System.out.print(num2);
}
```

Output:

```
45 2459
```

Any text can be printed between the two quotes. For example if we wanted to print out the number of bits in a byte we could do the following:

Example:

```
public static void main(String[] args)
{
         System.out.print("Number of bits in a byte is 8");
}
```

Output:	Number of bits in a byte is 8

Combining text and variables inside print() and println() is done by using the + operator.

<table>
<tr><td><b>Example:</b></td><td><pre>public static void main(String[] args)
{
        int numOfBitsInAByte = 8;
        System.out.print("Number of bits in a byte is " + numOfBitsInAByte);
}</pre></td></tr>
<tr><td><b>Output:</b></td><td>Number of bits in a byte is 8</td></tr>
</table>

System.out.format() :

The **format** function gives the user more formatting control. The general form of the function is:

System.out.format(format_string, ...);

where:

format_string	A user specified string
...	An optional list of expressions to print

Another look at the general form:

System.out.format(format_string, expression1, expression2, expression3, ...);

format can only print strings (words). If you wish to print a number, you have to convert it to a string first. Fortunately, **format** makes this very easy. Within *string*, there are usually one or more *conversion specifiers* that determine how to print numeric values.

Example:

```
public static void main(String[] args)
{
            int age = 18;
            float gpa = 3.78F;
            double pi = 3.1415926;

            System.out.format("Hello%n");
            System.out.format("John's age is %d%n", age);
            System.out.format("His GPA is %f%n", gpa);
            System.out.format("The value of PI is %f%n", pi);
            System.out.format("John is %d years old and his GPA is %f%n", age, gpa);
}
```

Output:

```
Hello
John's age is 18
His GPA is 3.780000
The value of PI is 3.141593
John is 18 years old and his GPA is 3.780000
```

incorrect usage:

Example:

```
public static void main(String[] args)
{
            int age = 18;
            System.out.format("John is %d years old and his GPA is %f%n", age);
}
```

Output:

```
John is 18 years old and his GPA is Exception in thread "main"
java.util.MissingFormatArgumentException: Format specifier 'f'
          at java.util.Formatter.format(Unknown Source)
          at java.io.PrintStream.format(Unknown Source)
          at Main.main(Main.java:10)
```

Example:

```
public static void main(String[] args)
{
        int age = 18;
        float gpa = 3.78F;
        System.out.format("John is %d years old%n", age, gpa);
}
```

Output:

```
John is 18 years old
```

Example:

```
public static void main(String[] args)
{
        int age = 18;
        System.out.format("John is %f years old%n", age);
}
```

Output:

```
John is Exception in thread "main" java.util.IllegalFormatConversionException: f != java.lang.Integer
        at java.util.Formatter$FormatSpecifier.failConversion(Unknown Source)
        at java.util.Formatter$FormatSpecifier.printFloat(Unknown Source)
        at java.util.Formatter$FormatSpecifier.print(Unknown Source)
        at java.util.Formatter.format(Unknown Source)
        at java.io.PrintStream.format(Unknown Source)
        at Main.main(Main.java:12)
```

Example:

```
public static void main(String[] args)
{
        float gpa = 3.78F;
        System.out.format("His GPA is %d%n", gpa);
}
```

Output:

```
His GPA is Exception in thread "main" java.util.IllegalFormatConversionException: d != java.lang.Float
        at java.util.Formatter$FormatSpecifier.failConversion(Unknown Source)
        at java.util.Formatter$FormatSpecifier.printInteger(Unknown Source)
        at java.util.Formatter$FormatSpecifier.print(Unknown Source)
        at java.util.Formatter.format(Unknown Source)
        at java.io.PrintStream.format(Unknown Source)
        at Main.main(Main.java:13)
```

Some common conversion specifiers:

Type character	input	String result
%d	**signed *int***	signed decimal integer
%u	**unsigned *int***	unsigned decimal integer
%o	**unsigned *int***	unsigned octal integer
%x, %X	**unsigned *int***	unsigned hexadecimal integer, lowercase or uppercase
%f	***float***	real number, standard notation
%e, %E	***float***	real number, scientific notation (lowercase or uppercase exponent marker)
%s	***String***	string
%c	***char***	character
%p	***Object***	object identity hash code (i.e., pointer value), in unsigned hexadecimal

PS: more conversion specifiers will be covered in future chapters.

Controlling Size and Precision:

The conversion specifiers determine the type of the data to display. If you want to have more control over the output, you need to specify additional information in the format string.

The general form is this:

%[flags][width][.precision]type

where:

flags	Optional characters to control justification and characters used for padding. Default is right justification. A minus sign indicates left justification.
width	Optional number that controls the minimum number of characters to output.
.precision	Optional number that controls the whether or not to print a decimal point and how many digits to the right of the point to print. Cannot be used with integers.

Some useful flags:

'+'	Includes sign, whether positive or negative
'-'	Left-justified
' ' *(space)*	Non-negative values begin with a space character (' '). This flag is only useful for signed conversion results (%d and %f)
'#'	With %o: values are pre-pended with a zero ('0') With %x or %X: values are pre-pended with the prefix "0x" or "0X"
','	Includes locale-specific grouping characters

Example:

```
public static void main(String[] args)
{
        int age = 21;

        System.out.format("|%d|%5d|%-5d%n", age, age, age);
}
```

Output:

```
|21|   21|21   |
```

Example:

```
public static void main(String[] args)
{
        long blah = 461012;
        System.out.format("|%d|%n", blah);
        System.out.format("|%08d|%n", blah);
        System.out.format("|%+8d|%n", blah);
        System.out.format("|%,8d|%n", blah);
        System.out.format("|%+,8d|%n", blah);
}
```

Output:

```
|461012|
|00461012|
| +461012|
| 461,012|
|+461,012|
```

Example:

```
public static void main(String[] args)
{
          float wt = 165.89F;
          System.out.format("|%f|%10.3f|%10.3e|\n", wt, wt, wt, wt);
}
```

Output:

```
|165.889999|   165.890| 1.659e+02|
```

Example:

```
public static void main(String[] args)
{
          int oct = 80;
          int hex = 165;
          System.out.format("|%o|%#o|%#10o|%-#10o|%n", oct, oct, oct, oct);
          System.out.format("|%x|%#x|%#10x|%-#10x|%n", hex, hex, hex, hex);
}
```

Output:

```
|120|0120|      0120|0120      |
|a5|0xa5|      0xa5|0xa5      |
```

Note: ***System.out.printf*** *is equivalent to* ***System.out.format*** *and can be used instead.*

For ***System.out.format*** *or* ***System.out.printf,*** *%n is preferred over \n as it will be correct to the standard of whatever system you are running.*

For more information and examples on **System.out.format** and **System.out.printf,** you can check:
http://docs.oracle.com/javase/tutorial/java/data/numberformat.html
http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html#syntax
http://docs.oracle.com/javase/tutorial/essential/io/formatting.html

Console Input

In order to get input from the user, we are going to need a **Scanner**. We are going to use the created **Scanner** in order to get integral and floating point values from the console.

Setup 1: Creating a Scanner

```
public class Main
{
	public static void main(String[] args)
	{
		Scanner consoleInput = new Scanner(System.in);
	}
}
```

Let’s examine the following line of code: **Scanner consoleInput = new Scanner(System.in);**

Scanner consoleInput: declaring a variable of type Scanner.

new Scanner(...): create a **Scanner** and storing it in our variable.

System.in: telling the **Scanner** to get its information from the user's input (Keyboard).

Note: This way of creating variables seams weird but will be explained in much more depth in the "Classes" chapter.

If we run this small program we get the following error:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
	Scanner cannot be resolved to a type
	Scanner cannot be resolved to a type

	at Main.main(Main.java:7)
```

The error is telling us that the compiler doesn't understand what the type **Scanner** is. In order to solve this problem we need to import the Scanner definition from the java util library.

```
import java.util.Scanner;
public class Main
{
	public static void main(String[] args)
	{
		Scanner consoleInput = new Scanner(System.in);
	}
}
```

Input of Integral Values:

In order to get an int value from the user, we will use the **nextInt()** function found in the Scanner variable.

Example:

```
import java.util.Scanner;
public class Main
{
	public static void main(String[] args)
	{
		Scanner consoleInput = new Scanner(System.in);

		System.out.println("Enter your age: ");
		int age = consoleInput.nextInt();
		System.out.println("You are " + age + " years old.");
	}
}
```

Output:

```
Enter your age:
18
You are 18 years old.
```

PS: User input is entered in the console window.

Note: ***we can also use nextShort(), nextByte(), nextLong() to get the different integral types (no nextChar() though).***

Input of Floating Point Values

Floats and doubles can be input using the nextFloat() and nextDouble() functions respectively.

Example:

```
import java.util.Scanner;

public class Main
{
	public static void main(String[] args)
	{
		Scanner consoleInput = new Scanner(System.in);

		System.out.println("Enter your gpa: ");
		float gpa = consoleInput.nextFloat();
		System.out.println("Your gpa is: " + gpa);
		System.out.println("Enter a decimal number");
		double d = consoleInput.nextDouble();
		System.out.println("Your decimal number: " + d);
	}
}
```

Output:

```
Enter your gpa:
3.8
Your gpa is: 3.8
Enter a decimal number
2.71828
Your decimal number: 2.71828
```