NOTICE THE DISTINCTION BETWEEN

System.out.println()

AND

System.out.print()

ONE PRINTS A NEWLINE CHARACTER, THE OTHER DOES NOT

EX.

```
// Area.java
import java.util.*;

class Area{
    Public static void main(String[] args){
        double height, width, area;
        Scanner sc = new Scanner(System.in);
        System.out.Print("Enter width: ");
        width = SC.nextDouble();
        System.out.print("Enter height: ");
        height = sc.nextDouble();
        area = width * height;
        System.out.print("The area is: ");
        System.out.println(area);
    }
}
```

AS WE CAN SEE THE Scanner class IS CAPABLE OF READING AND ASSIGNING OTHER DATA TYPES FROM STANDARD INPUT.

OUTPUT:

```
% javac Area.java
% java Area
Enter width:  25
Enter height:  35
The area is:  875.0
%
```

WHAT WOULD HAPPEN IF WE ENTER SOMETHING WHICH CANNOT BE INTERPRETED AS A Double? ONE WAY TO FIND OUT IS TO TRY IT AND SEE.

OUTPUT

```
% java Area
Enter width: abc
Exception in thread "main" java.util.InputMismatchException
        ⋮ } some other stuff
%
```

THE PROGRAM HALTS WITH AN ERROR MESSAGE.

OTHER METHODS BELONGING TO Scanner class:

nextInt( )	integers
nextDouble( )	doubles
nextBoolean( )	boolean
next( )	strings

WHAT EXACTLY IS A method (ALSO CALLED A FUNCTION)? A METHOD IS A NAMED COLLECTION OF INSTRUCTIONS.

IN THE PROGRAMS WE'VE SEEN SO FAR, WE HAVE DEFINED A SINGLE method, i.e. "main". WE HAVE ALSO USED SOME METHODS FROM OTHER STANDARD JAVA CLASSES.

WHEN WE USE A method WE SAY WE ARE CALLING THE METHOD, e.g.

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

CALLS THE println() method which BELONGS TO THE class PrintStream. System.out IS AN INSTANCE OF THE PrintStream class.

THUS WHENEVER WE CALL A METHOD WE MUST TELL JAVA WHERE TO FIND IT, SO WE WRITE

```
System.out.println()
```

RATHER THAN JUST

```
println()
```

MOST METHODS REQUIRE SOME DATA TO PERFORM THEIR SPECIFIC TASK.

THESE DATA VALUES ARE THE METHOD'S PARAMETERS, ALSO CALLED ARGUMENTS.

THE ARGUMENT TO THE ABOVE CALL TO println() IS THE STRING LITERAL

"Hello, World!"

THE STANDARD Java class Math CONTAINS A METHOD CALLED min() WHICH TAKES TWO ARGUMENTS.

EX.

```
// FindSmallest.java
import java.util.*;

class FindSmallest {
    public static void main(String[] args) {
        double x, y, smallest;
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a number: ");
        x = sc.nextDouble();
        System.out.print("Enter another number: ");
        y = sc.nextDouble();
        smallest = Math.min(x, y);
        System.out.print("The smallest is: ");
        System.out.println(smallest);
    }
}
```

WHEN A METHOD, SUCH AS min( ), TAKES MULTIPLE ARGUMENTS, THEY ARE SEPARATED BY COMMAS.

NOTE SOME METHODS TAKE NO ARGUMENTS. e.g. nextDouble( ) IN THE Scanner class. WE JUST PUT EMPTY PARENTHESES IN THIS CASE.

OUTPUT:

```
% java FindSmallest
Enter a number: 1.2
Enter another number: 3.4
The smallest is: 1.2
%
```

IN SUBSEQUENT EXAMPLES, I will NOT INCLUDE THE SURROUNDING CODE

```
// file.java
import ...

class className{
    public static void main(String[] args){
        :
        :
        :
    }
}
```

There are thousands of methods found in the standard Java classes. We have seen so far:

```
System.out.print(x)
System.out.println(x)
SC.next()
SC.nextInt()
SC.nextDouble
Math.min( )
word1.concat(word2)
```

Consider:

```
className.method(....)
        vs.
instanceOfClass.method(...)
```

Number Types

Java provides seven primitive data types for storing numerical values. These fall into two categories: integer types, and floating point types, which refer to the ways in which the numerical data is encoded.

	Type	Size	Range
Integer Types	byte	8 bits	-128 ... 127
	short	16	-32768 ... 32767
	char	16	0 ... 65536
	int	32	-2147483648 .. 2147483647
	long	64	± about 18 digits
Floating Point Types	float	32	7 dec. digits
	double	64	15 dec. digits

NOTE: A BIT IS MOST ELEMENTARY DATA VALUE THAT CAN BE STORED AND MANIPULATED. BIT STANDS FOR BINARY DIGIT: 0 OR 1.

ALSO NOTE char IS CONSIDERED AN INTEGER TYPE. THUS ONE CAN DO

```
char x = 'A';   or   char x = 65;
```

JAVA USES THE UNICODE CHARACTER SET IN THAT SYSTEM 65 IS THE CHARACTER CODE FOR 'A'. OBSERVE THAT

```
System.out.println(x);
```

PRINTS THE CHARACTER 'A', BUT NEVERTHELESS, x IS A NUMBER. (SEE P. 29)

EX. CharCode.java

```
Scanner SC = new Scanner(System.in);
char x;

System.out.print("Enter an integer (0..65536)");
x = SC.nextInt();
System.out.println("Unicode x = " + x);
```

Java does arithmetic (+, -, *, %) with only the four types

	Literal example
double	6.0
float	6.0f or 6.0F
long	6L or 6l
int	6 or [0110 (octal) 0x6 (hex)]

An expression, such as a+b, where both a and b are of one of the above types is evaluated using the arithmetic of that type. If either a or b is of type char, short, or byte, both are converted to int then evaluated. In mixed type expressions, the lower type is first converted to the higher type.

A similar conversion takes place when we do assignments.

Ex
```
int x = 6;
long y;
double z;          // these are called
y = x;             // widening conversions
z = y;             //
System.out.println(z); // Prints 6.0
```