Java - Input and Output

When run as a console application Java is not very good at accepting input from the keyboard.

- Java thinks that any input is a number, any other input will have to be converted back to the actual keys entered.
- Input causes an error; this has to be overcome to allow any input to work.

To capture an integer and output the result requires the following code. The comment (//) lines are not required. Indentation is encouraged.

```
public class InputOutput
{
	public static void main(String[] args) throws java.io.IOException
	{
		int input; //declare an integer
		input=System.in.read();  //capture data
		System.out.println("You entered " + input);

	}
}
```

Run this code with and without the clause - throws java.io.IOException. The task list should show that the error is not caught or declared.

Modifications

1. The code still runs through and finishes with amazing speed. Have the program stop while showing the initial input by using the line - System.in.read(). Note this will not work with all IDEs. Another way top make the program wait is to set up a nice big loop where nothing happens - for(long x=0;x<1000000;x++){}

2. Mathematically the data passed back is wrong. Investigate different numbers input and fix the problem.

3. The whole set up is less than ideal due to the ENTER key itself being a keyboard character. The following code will accept a sequence of keyboard characters and convert them to a String.

```
byte buffer[] = new byte[20];  //byte is a primitive data type, here an array of 20 bytes
String temp=null;  //another data type String
//String is a class and has methods so earns a capital letter.  The primitive data type byte has
no methods and no capital letter either.  Run this code with Byte and see what happens.
System.in.read(buffer);  //read the array
temp=new String(buffer); //create String from the byte array buffer
temp=temp.trim(); // not all 20 elements of the array are used so trim them
//note trim is a method of the Class String - hence the dot operator
System.out.println("You entered " + temp);  //output.
```

Tidy up this code to accept 2 words with separate System.in.read() lines and then output them on a single line.

4. Working backwards the byte array can be adapted to any data type. Add to the existing code to accept 2 integers and add them together displaying the result on screen.

The data should be accepted a byte array and then converted to a String as before. The String then needs to be converted to an int.

```
int tempI=0; //Java likes its variables to be assigned a value when they are declared.
tempI=Integer.parseInt(temp);
```

To display the result of a calculation with System.out.println the integer will have to be converted to a String again.

```
System.out.println("You entered " + String.valueOf(iResult));  //iResult is an integer
```

5. Capture a single character and display it.

```
char read = ' '; //declare an empty character.
//capture the data as byte[] then convert to a String.
read=temp.charAt(0); //take 1st element of the String temp
```

6. Decimal numbers come in 2 flavours in Java, float for moderately long numbers and double for real physicists. Add code to capture 2 floats and add then up. Unfortunately this process id going to require use of the primitive data type float (to hold the number) and the Class Float (for some fancy manipulation).

```
float fTemp=0.0f;   //declare the float - the f at the end ensures that Java gets it right
//capture the data as byte[] then convert to a String.
Float value= Float.valueOf(temp);  // the Float Class will convert the String to Float
fTemp = value.floatValue();  // convert Float to float
//do the maths and output the result as a String.
```

7. Having got all this lot to work you will note that a lot of the code is surprisingly familiar. This is a good point to introduce a method to handle all that repetitive code, methods are the Java name for functions.

Here is a simple Java method that will add 1 to an integer

```
private static int myNumber (int test)
{
        test ++;
        return test;
}
```

The static qualifier is required for console applications with a single Class. Graphical interface applications or programs with more than 1 Class will rarely use static.

The method is called from within main

```
int check = 0;
check = myNumber(2);
```

Luckily Java methods are quite handy at accepting and returning parameters. They will accept and return Strings (there are no pointers in Java) and even objects can be passed to and from Java methods.

For this case the following lines are used more than once. They function to get a String from the keyboard that is then output or converted to a double, integer or character.

```
byte buffer[] = new byte[20];
String temp=null;
System.in.read(buffer);
temp=new String(buffer);
temp=temp.trim();
```

A suitable method can be created as follows

```
private static String getData() throws java.io.IOException
{
        byte buffer[] = new byte[20];
        String temp=null;
        System.in.read(buffer);
        temp=new String(buffer);
        temp=temp.trim();
        return temp; //return the resulting String
}
```

This code will call it for and send back a String

```
String testFunction =  getData();
```

Create this function and use it to capture data from the keyboard, the existing code can be replaced. The resulting Strings will still need to be converted to integers and such and displayed.