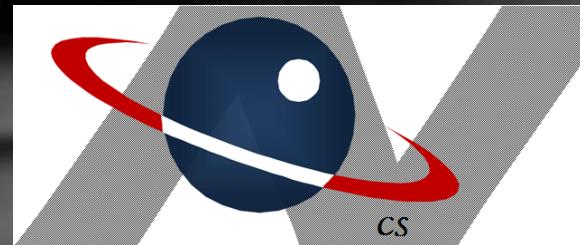


O(N) CS LESSONS

*Lesson 6D – The “split” Process
Using Loops and Arrays*



By John B. Owen

All rights reserved

©2011, revised 2014



Table of Contents

- Objectives
- File input “split” process
- args – an array of Strings
- “split” process
- Indexing
- String array processing
- Integer array transformation
- Examples
- Lesson Summary / Labs
- Contact Information for supplementary materials



Objectives

- In the previous lesson, you learned in detail about two different file input scenarios – the “unknown quantity” process, and the “initial N” process.
- A third file processing scenario is introduced in this lesson. It uses the `String` class “`split`” process, which takes in a multi-value horizontal line of data as a long `string` and “splits” the single `String` into an array of `Strings`.



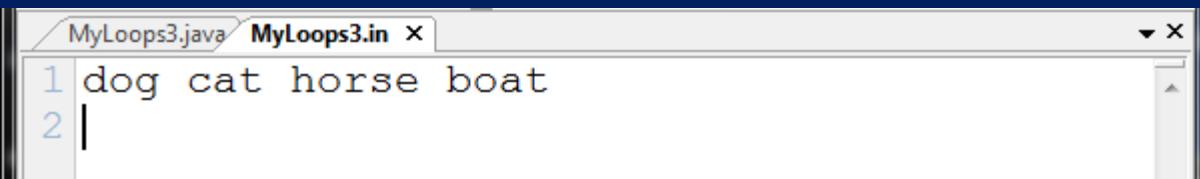
Objectives

- An introduction to arrays is provided, enough to be successful with the “split” process.
- Also introduced is the transformation process required to take an array of Strings containing mathematical values and parsing them into a parallel array of actual values.
- Labs are provided to practice this new technique.



File input “split” scenario

- This advanced technique will require some careful concentration and study, as well as learning a new and advanced data structure called an “array”.
- First let’s look at the data file and how it is organized
 - Here are several data items in a file, all on one line, separated by spaces.



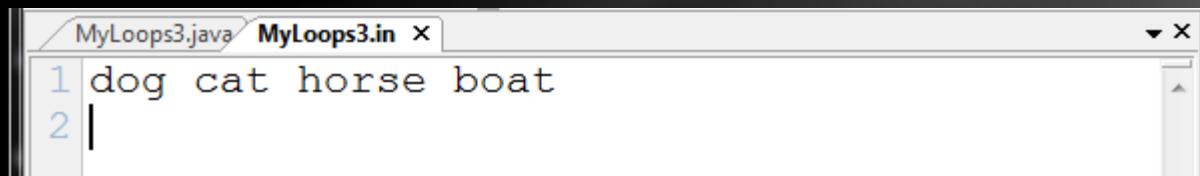
```
MyLoops3.java MyLoops3.in x
1 dog cat horse boat
2 |
```

- Although it is still possible to use the “*unknown quantity*” (File Scenario #1) technique in this situation, it’s time to learn this new technique which will take us further and provide more flexibility for data processing.



File input “split” scenario

- The four steps of this process are:
 1. Read in the entire line as a String using the ***nextLine()*** command.
 2. Use the String “split” method to divide it into several parts.
 3. Store the resulting array of Strings into a String array object.
 4. Process the array of Strings in any number of ways.



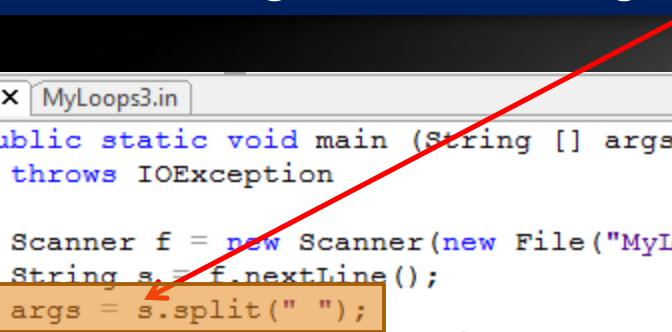
The screenshot shows a Java development environment with two tabs open. The top tab is titled "MyLoops3.java" and the bottom tab is titled "MyLoops3.in". The "MyLoops3.in" tab is active and displays the following text:

```
1 dog cat horse boat
```



File input “split” scenario

- Demonstrated below is the basic process as we just described:
 - The Scanner **f** object is constructed and linked
 - A String is read using the **nextLine()** command and assigned to **s**
 - And then...something new and strange-looking...



```
MyLoops3.java x MyLoops3.in
22
23 public static void main (String [] args)
24     throws IOException
25 {
26     Scanner f = new Scanner(new File("MyLoops3.in"));
27     String s = f.nextLine();
28     args = s.split(" ");
29     out.printf("I love my %s.\n",args[0]);
30     out.printf("I love my %s.\n",args[1]);
31     out.printf("I love my %s.\n",args[2]);
32     out.printf("I love my %s.\n",args[3]);
33 }
34 }
```

```
MyLoops3.java MyLoops3.in x
1 dog cat horse boat
2
I love my dog.
I love my cat.
I love my horse.
I love my boat.
Press any key to continue...
```



args – an array of Strings

- So far you have been dutifully typing in the main method header with the **(String [] args)** portion included, not really knowing its purpose.
- We now ~~will~~ use this feature for the split process, although it is still not the original reason for it. (*Originally it was used for mainframe computer command line processing, but that is another lesson entirely.*)



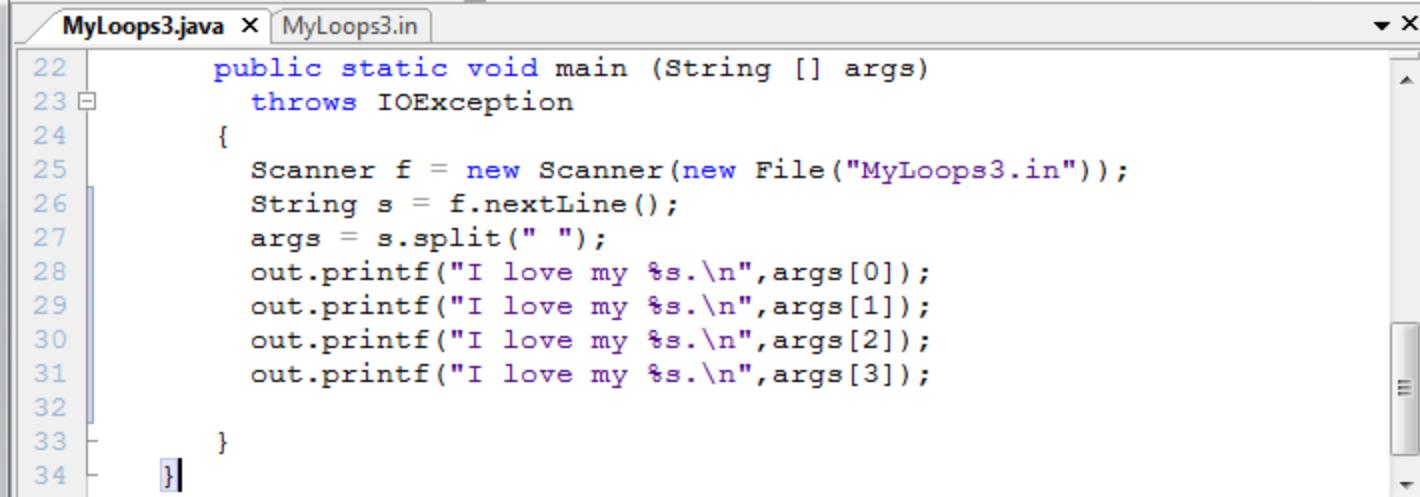
```
MyLoops3.java x MyLoops3.in
22
23 public static void main (String [] args)
24     throws IOException
25 {
26     Scanner f = new Scanner(new File("MyLoops3.in"));
27     String s = f.nextLine();
28     args = s.split(" ");
29     out.printf("I love my %s.\n",args[0]);
30     out.printf("I love my %s.\n",args[1]);
31     out.printf("I love my %s.\n",args[2]);
32     out.printf("I love my %s.\n",args[3]);
33 }
34 }
```

```
MyLoops3.java MyLoops3.in x
1 dog cat horse boat
2
I love my dog.
I love my cat.
I love my horse.
I love my boat.
Press any key to continue...
```

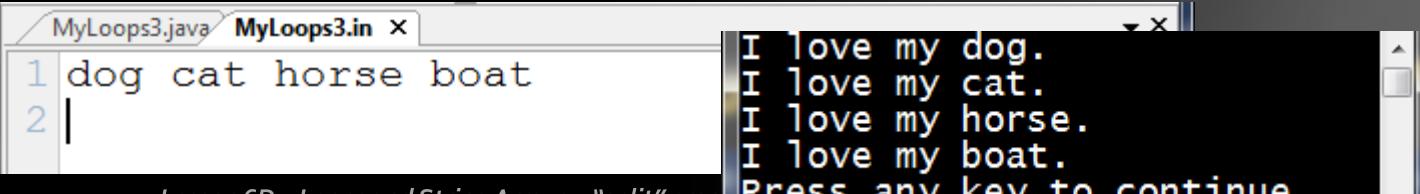


args – an array of Strings

- Essentially **args** (short for “arguments”) is a special data structure object called an array, which we will discuss in much more detail in Lesson 7.
- You can tell it is an array by the square brackets “[]” next to it.
- This simply means that it is capable of holding more than just one data element.



```
MyLoops3.java x MyLoops3.in
22 public static void main (String [] args)
23     throws IOException
24 {
25     Scanner f = new Scanner(new File("MyLoops3.in"));
26     String s = f.nextLine();
27     args = s.split(" ");
28     out.printf("I love my %s.\n",args[0]);
29     out.printf("I love my %s.\n",args[1]);
30     out.printf("I love my %s.\n",args[2]);
31     out.printf("I love my %s.\n",args[3]);
32 }
33 }
```

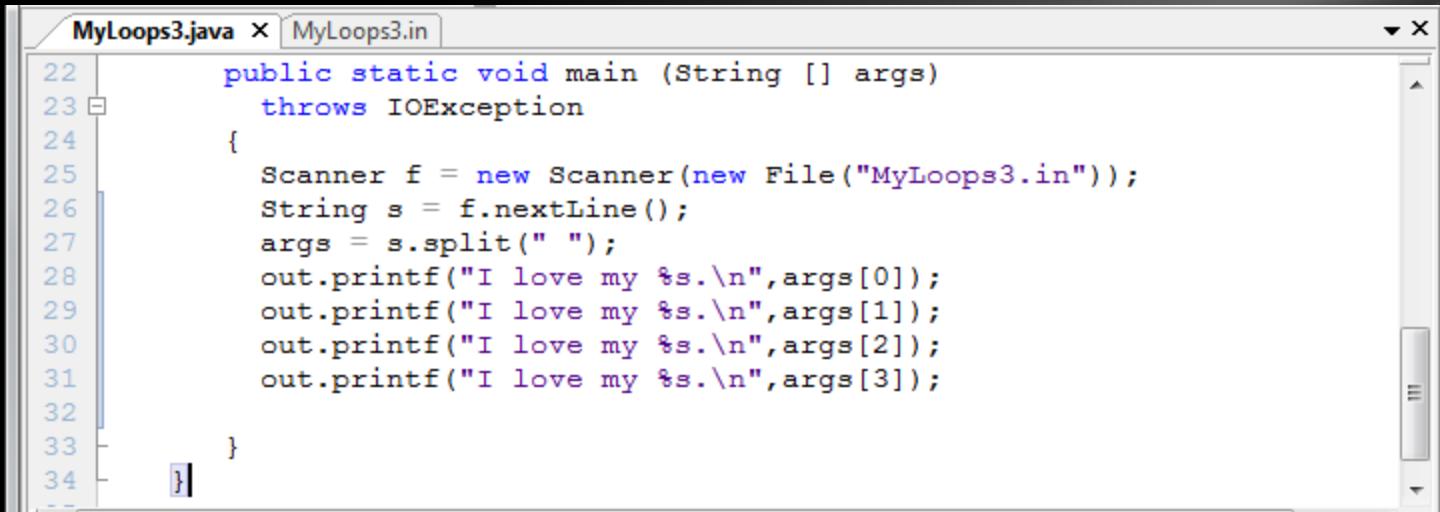


```
MyLoops3.java MyLoops3.in x
1 dog cat horse boat
2
I love my dog.
I love my cat.
I love my horse.
I love my boat.
Press any key to continue...
```



Analogy – **variable** vs **array**

- An analogy that might help your understanding is this:
 - A single element **variable** is like a normal bicycle...it has only one seat that can carry only one passenger.
 - A multi-element **array** is like a bus...it has many seats that can carry many passengers.

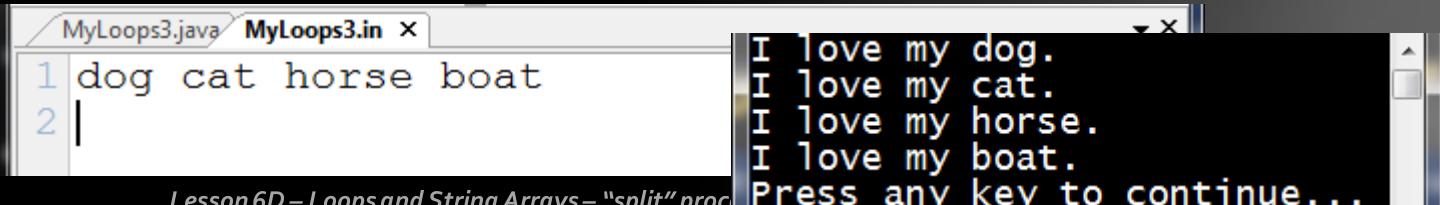


The image shows a Java development environment with two tabs open: "MyLoops3.java" and "MyLoops3.in". The "MyLoops3.java" tab contains the following code:

```
22 public static void main (String [] args)
23     throws IOException
24 {
25     Scanner f = new Scanner(new File("MyLoops3.in"));
26     String s = f.nextLine();
27     args = s.split(" ");
28     out.printf("I love my %s.\n",args[0]);
29     out.printf("I love my %s.\n",args[1]);
30     out.printf("I love my %s.\n",args[2]);
31     out.printf("I love my %s.\n",args[3]);
32 }
33 }
```

The "MyLoops3.in" tab contains the input file content:

```
dog cat horse boat
```



The image shows a terminal window with the following output:

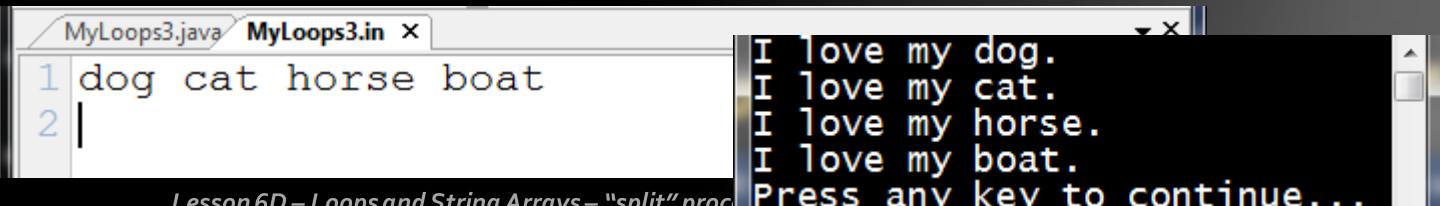
```
I love my dog.
I love my cat.
I love my horse.
I love my boat.
Press any key to continue...
```



“split” process

- Here's what is happening with the “split” command you see below.
 - The String **s** has received an entire line from the data file, containing several pieces of data, each separated by a space.
 - The “split” method is told to use the space as the “splitter”, or “delimiter”. This is indicated in the parameter list. (*Actually, ANY character can be used as a “splitter”. We'll explore that in more detail later on.*)

```
23     throws IOException
24
25     {
26         Scanner f = new Scanner(new File("MyLoops3.in"));
27         String s = f.nextLine();
28         args = s.split(" ");
29         out.printf("I love my %s.\n",args[0]);
30         out.printf("I love my %s.\n",args[1]);
31         out.printf("I love my %s.\n",args[2]);
32         out.printf("I love my %s.\n",args[3]);
33     }
34 }
```



The screenshot shows a Java IDE window with two tabs: "MyLoops3.java" and "MyLoops3.in". The "MyLoops3.java" tab contains the code above. The "MyLoops3.in" tab contains the input file content: "dog cat horse boat". Below the tabs, the terminal window shows the output of the program: four lines of text: "I love my dog.", "I love my cat.", "I love my horse.", and "I love my boat.". At the bottom of the terminal window, the text "Press any key to continue..." is visible. A red arrow points from the explanatory text in the slide to the space character in the line "args = s.split(" ")".



"split" process

- The split method searches through the entire String looking for instances of the “splitter”.
- Each time it sees one, it creates a division of the String and places it into another slot in the array.
- Since this input String has three spaces, it is divided into four parts, each part assigned to a “slot” in the array.
- Each portion is placed into the array in a numbered position, starting with position zero.

```
25     Scanner f = new Scanner(new File("MyLoops3.in"));
26     String s = f.nextLine();
27     args = s.split(" ");
28     out.printf("I love my %s.\n",args[0]);
29     out.printf("I love my %s.\n",args[1]);
30     out.printf("I love my %s.\n",args[2]);
31     out.printf("I love my %s.\n",args[3]);
32 }
```

0	1	2	3
dog	cat	horse	boat

MyLoops3.java My
1 dog cat horse boat
2 | I love my dog.
I love my cat.
I love my horse.
I love my boat.
Press any key to continue...

JC 12



“split” process

- Once the “split” process is completed, the array now contains all of the data from the original String, separated into four separate slots that are numbered from zero to 3.
- This is another example of “zero-indexing”, which we explored in an earlier lesson when we were processing the individual characters of a String.
- Get used to this...you'll see it a lot!!!***

The screenshot shows a Java code editor with the following code:

```
23     throws IOException
24 {
25     Scanner f = new Scanner(new File("MyLoops3.in"));
26     String s = f.nextLine();
27     args = s.split(" ");
28     out.printf("I love my %s.\n", args[0]);
29     out.printf("I love my %s.\n", args[1]);
30     out.printf("I love my %s.\n", args[2]);
31     out.printf("I love my %s.\n", args[3]);
32 }
```

A red arrow points from the variable **args** in the code to the first cell of a table below. The table has five columns labeled 0, 1, 2, 3. The values are: dog, cat, horse, boat. A red line highlights the entire row of the table.

0	1	2	3
dog	cat	horse	boat

The code editor also shows the input file content:

```
1 dog cat horse boat
```

The terminal window shows the output:

```
I love my dog.
I love my cat.
I love my horse.
I love my boat.
```

At the bottom, it says: **Press any key to continue...**



Indexing

- The output statements access the **args** array variable, with an additional feature...an **index** (*number inside square brackets*).
- This is required when accessing the elements of an array.
- It is not possible to retrieve the data elements of an array (for now) without this indexing feature.

The screenshot shows a Java development environment with two tabs: "MyLoops3.java" and "MyLoops3.in". The code in MyLoops3.java reads a file "MyLoops3.in" containing the words "dog", "cat", "horse", and "boat" on separate lines. It splits the input into an array of strings and prints each element followed by the word "dog". A red arrow points from the word "args" in the code to the first element of the array. Another red arrow points from the index [0] in the code to the second element of the array, which contains "cat".

o	1	2	3
dog	cat	horse	boat

The output window shows the following text:

```
I love my dog.  
I love my cat.  
I love my horse.  
I love my boat.
```

Press any key to continue...



args - clarifications

- Here are two more clarifications about the array variable **args**
 - Like the String variable we discussed in an earlier lesson, it is actually an **object reference**, which means it stores in memory the **address** of the first element in the array, not the value of the element itself.
 - Also, you can call it anything you like...**args**, **cow**, **monkey**,...it is not a "magic" word and it doesn't matter what it is named.

The screenshot shows a Java code editor with the following code:

```
throws IOException
{
    Scanner f = new Scanner(new File("MyLoops3.in"));
    String s = f.nextLine();
    args = s.split(" ");
    out.printf("I love my %s.\n", args[0]);
    out.printf("I love my %s.\n", args[1]);
    out.printf("I love my %s.\n", args[2]);
    out.printf("I love my %s.\n", args[3]);
}
```

A red arrow points from the word **args** in the code to the variable name in the code. Another yellow arrow points from the variable name in the code to the corresponding variable in the visualization below.

Below the code, there is a visualization of a string array. The array has four elements, indexed 0, 1, 2, and 3. The values are:

0	1	2	3
dog	cat	horse	boat

The variable **args** is highlighted in orange and its value is shown as **0x52ed761a**. The array elements are also highlighted in orange.

The output window shows the following text:

```
I love my dog.
I love my cat.
I love my horse.
I love my boat.
```

At the bottom, it says **Press any key to continue...**



Indexing – offsets from zero

- The index of zero indicates the position of the element in the first memory slot referenced by the array variable.
- An index value of 1 essentially means “1 slot offset from” the original location, 2 means “2 slots offset”, and so on.

The screenshot shows a Java IDE window with two tabs: "MyLoops3.java" and "MyLoops3.in". The code in "MyLoops3.java" is:

```
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("MyLoops3.in"));
    String s = f.nextLine();
    args = s.split(" ");
    out.printf("I love my %s.\n",args[0]);
    out.printf("I love my %s.\n",args[1]);
    out.printf("I love my %s.\n",args[2]);
    out.printf("I love my %s.\n",args[3]);
}
```

The "MyLoops3.in" file contains the string "dog cat horse boat". The output window shows the following text:

```
I love my dog.
I love my cat.
I love my horse.
I love my boat.
```

A yellow box highlights the variable `args`. A yellow arrow points from this box to the memory dump below. The memory dump shows the address `0x52ed761a` followed by four memory slots containing the words "dog", "cat", "horse", and "boat". Red arrows point from the indices 0, 1, 2, and 3 in the code to the corresponding memory slots. The memory dump also shows the address `0x52ed761a` again at the bottom.



String array processing

- And now, after all of that, we have successfully used the “split” process to harvest one line of data containing multiple elements and assign it to an array of Strings.
- Now let’s talk about different ways to process it.

The screenshot shows a Java IDE window with two tabs: "MyLoops3.java" and "MyLoops3.in". The code in MyLoops3.java is:

```
22 public static void main (String [] args)
23     throws IOException
24 {
25     Scanner f = new Scanner(new File("MyLoops3.in"));
26     String s = f.nextLine();
27     args = s.split(" ");
28     out.printf("I love my %s.\n",args[0]);
29     out.printf("I love my %s.\n",args[1]);
30     out.printf("I love my %s.\n",args[2]);
31     out.printf("I love my %s.\n",args[3]);
32 }
```

A red arrow points from the variable "args" in the code to the first element of a diagrammed array. The array has five slots, labeled 0 through 4. The elements are:

0	1	2	3
dog	cat	horse	boat

The "MyLoops3.in" tab shows the input text:

```
1 dog cat horse boat
2
```

The output window shows the printed results:

```
I love my dog.
I love my cat.
I love my horse.
I love my boat.
Press any key to continue...
```



String array processing

- In the example shown so far, all four elements are accessed directly using integers for the indexing process: 0, 1, 2, and 3.
- This is really no different than simply outputting the values using a while loop to read each one from the file and output it immediately.

The screenshot shows a Java development environment with two tabs: "MyLoops3.java" and "MyLoops3.in". The code in MyLoops3.java reads a file named "MyLoops3.in" containing the words "dog", "cat", "horse", and "boat" on separate lines. It uses a Scanner to read the file and a String array "args" to store the words. The "args" array is highlighted with an orange background and has indices 0, 1, 2, and 3 labeled below it. The output window shows the program's output: "I love my dog.", "I love my cat.", "I love my horse.", and "I love my boat.", followed by a prompt to press any key to continue.

```
MyLoops3.java X MyLoops3.in
22
23 public static void main (String [] args)
24     throws IOException
25 {
26     Scanner f = new Scanner(new File("MyLoops3.in"));
27     String s = f.nextLine();
28     args = s.split(" ");
29     out.printf("I love my %s.\n",args[0]);
30     out.printf("I love my %s.\n",args[1]);
31     out.printf("I love my %s.\n",args[2]);
32     out.printf("I love my %s.\n",args[3]);
33 }
```

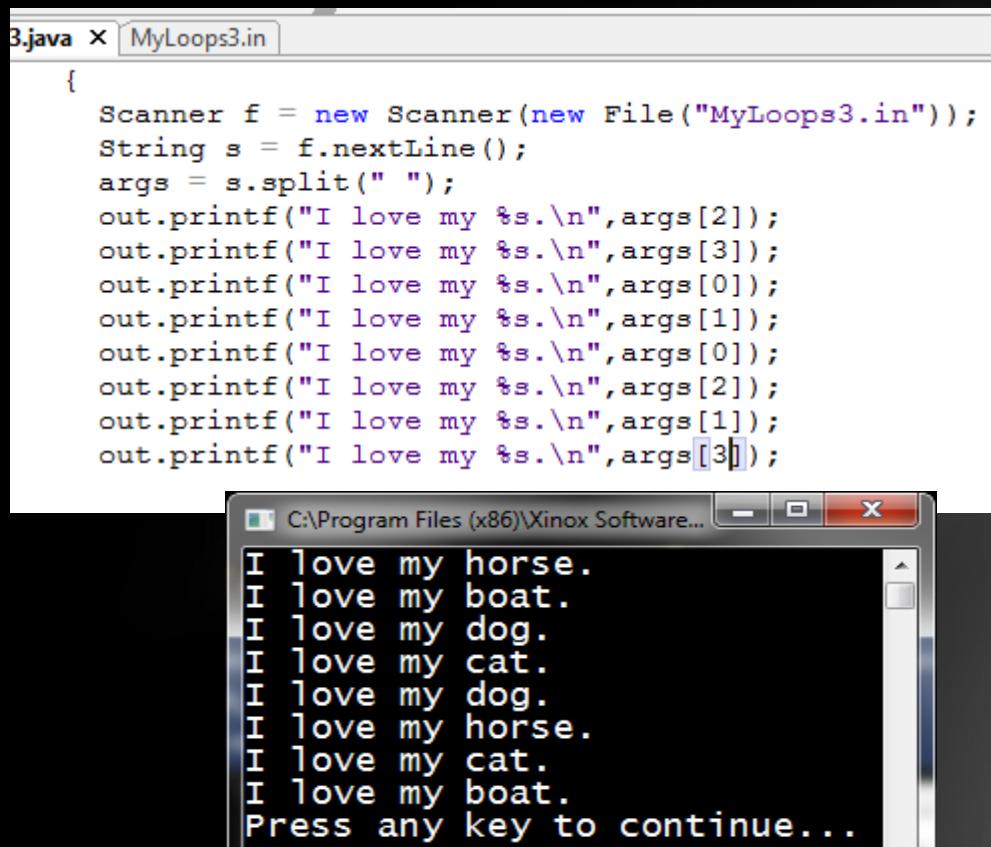
args	0	1	2	3
	dog	cat	horse	boat

```
MyLoops3.java MyLoops3.in
1 dog cat horse boat
2
I love my dog.
I love my cat.
I love my horse.
I love my boat.
Press any key to continue...
```



String array processing

- The difference is that now you can access the data elements in any order you wish, as many times as you want.
- See the examples below:



The screenshot shows a Java development environment. The code editor window has tabs for "3.java" and "MyLoops3.in". The "MyLoops3.in" tab is active, displaying a file containing the following text:

```
{  
    Scanner f = new Scanner(new File("MyLoops3.in"));  
    String s = f.nextLine();  
    args = s.split(" ");  
    out.printf("I love my %s.\n",args[2]);  
    out.printf("I love my %s.\n",args[3]);  
    out.printf("I love my %s.\n",args[0]);  
    out.printf("I love my %s.\n",args[1]);  
    out.printf("I love my %s.\n",args[0]);  
    out.printf("I love my %s.\n",args[2]);  
    out.printf("I love my %s.\n",args[1]);  
    out.printf("I love my %s.\n",args[3]);
```

Below the code editor is a terminal window titled "C:\Program Files (x86)\Xinox Software...". It displays the following output:

```
I love my horse.  
I love my boat.  
I love my dog.  
I love my cat.  
I love my dog.  
I love my horse.  
I love my cat.  
I love my boat.  
Press any key to continue...
```



Forwards and backwards

- The real power in having the data in an array is that you can use a loop to process the array.
- Below are two loops:
 - One processes the array in forwards order
 - The second outputs the array elements in reverse order

The screenshot shows a Java IDE interface with two windows. The left window displays the code for `MyLoops3.in`, which reads from a file and prints out the words "I love my dog.", "I love my cat.", "I love my horse.", "I love my boat.", "I love my boat.", "I love my horse.", "I love my cat.", and "I love my dog.". The right window shows the command prompt output for the same code, which prints the same sequence of lines followed by a prompt to press any key to continue.

```
ops3.java x MyLoops3.in
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("MyLoops3.in"));
    String s = f.nextLine();
    args = s.split(" ");
    for(int x = 0;x<args.length;x++)
        out.printf("I love my %s.\n",args[x]);
    for(int x = args.length-1;x>=0;x--)
        out.printf("I love my %s.\n",args[x]);
}
```

```
C:\Program Files (x86)\Xinox Softw
I love my dog.
I love my cat.
I love my horse.
I love my boat.
I love my boat.
I love my horse.
I love my cat.
I love my dog.
Press any key to co
```



length vs length()

- The two loops are fairly easy to understand, but there is one new feature we haven't used before.
- It looks familiar because we used a similar method with Strings, but there is one significant difference.
- ***args.length*** is a value that represents the total number of elements in the array, much like the ***String.length()*** method counts the number of characters in a String.

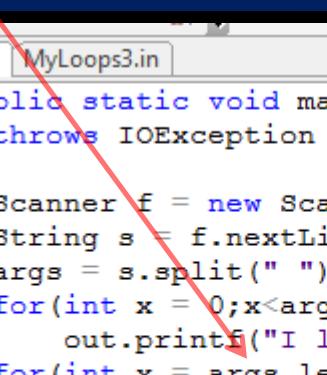
```
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("MyLoops3.in"));
    String s = f.nextLine();
    args = s.split(" ");
    for(int x = 0;x<args.length;x++)
        out.printf("I love my %s.\n",args[x]);
    for(int x = args.length-1;x>=0;x--)
        out.printf("I love my %s.\n",args[x]);
}
```

```
I love my dog.
I love my cat.
I love my horse.
I love my boat.
I love my boat.
I love my horse.
I love my cat.
I love my dog.
Press any key to co
```

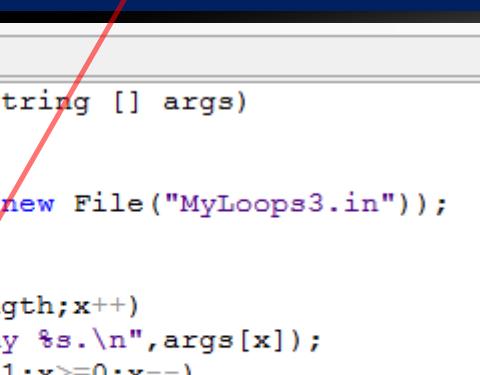


Uses of args.length

- The difference is that **args.length** does not use parentheses, and the **String.length()** method does.
- You must** keep this difference clear in your mind.
- args.length** is used in the check portion of the first loop, and is used in the start portion of the second loop.



```
ops3.java x MyLoops3.in
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("MyLoops3.in"));
    String s = f.nextLine();
    args = s.split(" ");
    for(int x = 0;x<args.length;x++)
        out.printf("I love my %s.\n",args[x]);
    for(int x = args.length-1;x>=0;x--)
        out.printf("I love my %s.\n",args[x]);
}
```



```
C:\Program Files (x86)\Xinox Softw
I love my dog.
I love my cat.
I love my horse.
I love my boat.
I love my boat.
I love my horse.
I love my cat.
I love my dog.
Press any key to co
```



Off by one

- Also notice that each time, the actual **args.length** value is never reached...**x<args.length**, and **x=args.length-1**, again because of “zero-indexing”.
- Just as in our previous study of Strings, the last element of an array is in the “length-1” position, since the first element is in the “zero” position.
- Therefore the value of **args.length** is always one step out of bounds, NOT a place you want to go, EVER!

The screenshot shows a Java development environment with two windows. On the left, the code editor displays `ops3.java` with the following content:

```
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("MyLoops3.in"));
    String s = f.nextLine();
    args = s.split(" ");
    for(int x = 0;x<args.length;x++)
        out.printf("I love my %s.\n",args[x]);
    for(int x = args.length-1;x>=0;x--)
        out.printf("I love my %s.\n",args[x]);
}
```

On the right, a terminal window titled "C:\Program Files (x86)\Xinox Softw" shows the output of the program:

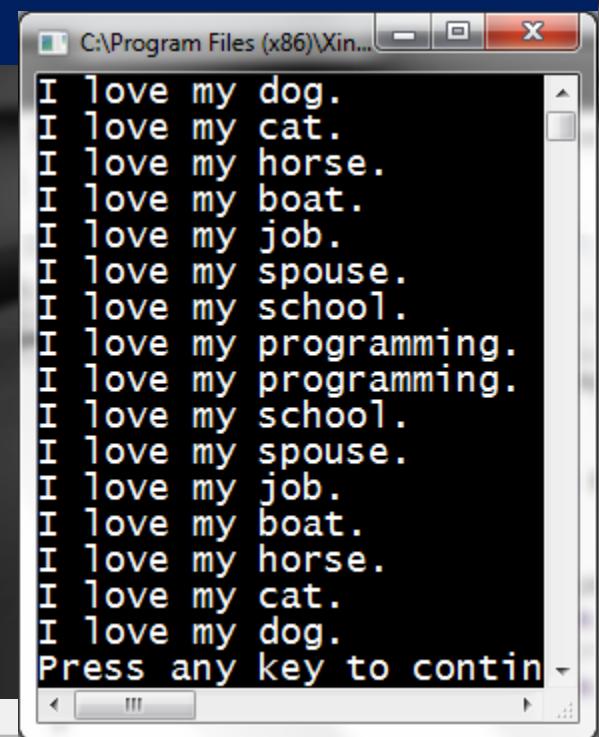
```
I love my dog.
I love my cat.
I love my horse.
I love my boat.
I love my boat.
I love my horse.
I love my cat.
I love my dog.
Press any key to co
```



Elegance and flexibility of “split” process

- And now...the TRUE power in using this “split”, array processing technique...observe!!!
- With absolutely *no change* in the source code below, but with an expansion to the data file, look at the results.

```
ops3.java x MyLoops3.in
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("MyLoops3.in"));
    String s = f.nextLine();
    args = s.split(" ");
    for(int x = 0;x<args.length;x++)
        out.printf("I love my %s.\n",args[x]);
    for(int x = args.length-1;x>=0;x--)
        out.printf("I love my %s.\n",args[x]);
}
```



```
I love my dog.
I love my cat.
I love my horse.
I love my boat.
I love my job.
I love my spouse.
I love my school.
I love my programming.
I love my programming.
I love my school.
I love my spouse.
I love my job.
I love my boat.
I love my horse.
I love my cat.
I love my dog.
Press any key to continue...
```

```
MyLoops3.java MyLoops3.in x
1 dog cat horse boat job spouse school programming
2 |
```



Elegance and flexibility of “split” process

- The number of elements in the data file increased, and the program adjusted accordingly, using the flexibility of the “split” and “length” commands.
- *Herein lies the elegance and true power of this process!!!!*

```
ops3.java x MyLoops3.in
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("MyLoops3.in"));
    String s = f.nextLine();
    args = s.split(" ");
    for(int x = 0;x<args.length;x++)
        out.printf("I love my %s.\n",args[x]);
    for(int x = args.length-1;x>=0;x--)
        out.printf("I love my %s.\n",args[x]);
}
```

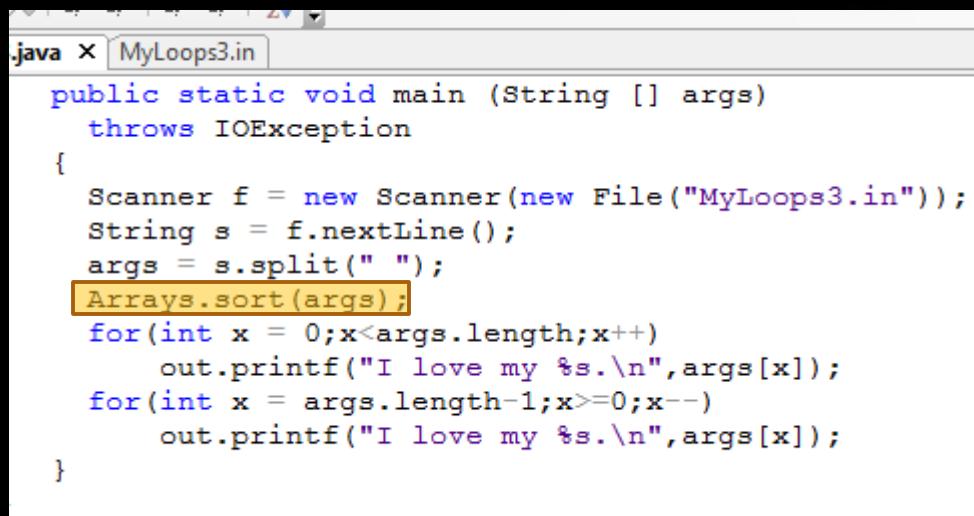
```
I love my dog.
I love my cat.
I love my horse.
I love my boat.
I love my job.
I love my spouse.
I love my school.
I love my programming.
I love my programming.
I love my school.
I love my spouse.
I love my job.
I love my boat.
I love my horse.
I love my cat.
I love my dog.
Press any key to continue . . .
```

```
MyLoops3.java MyLoops3.in x
1 dog cat horse boat job spouse school programming
2 |
```

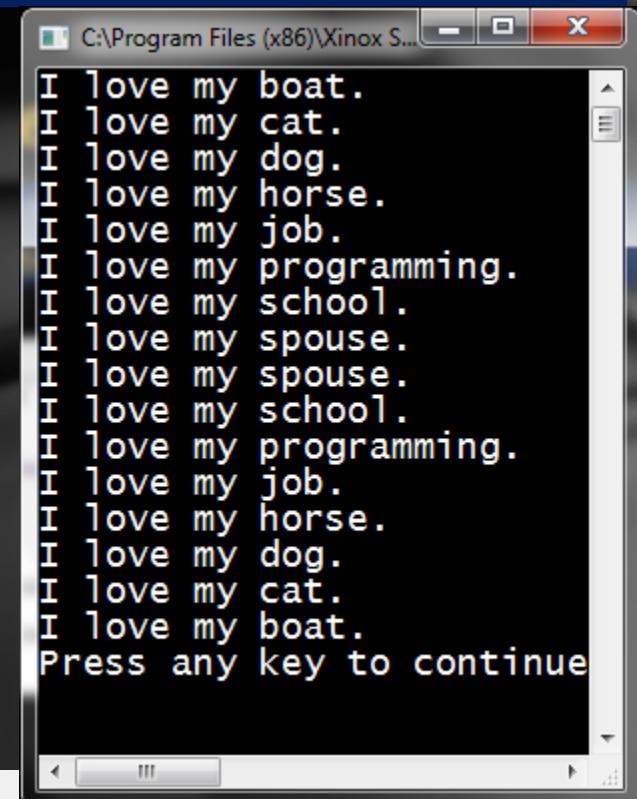


Arrays.sort

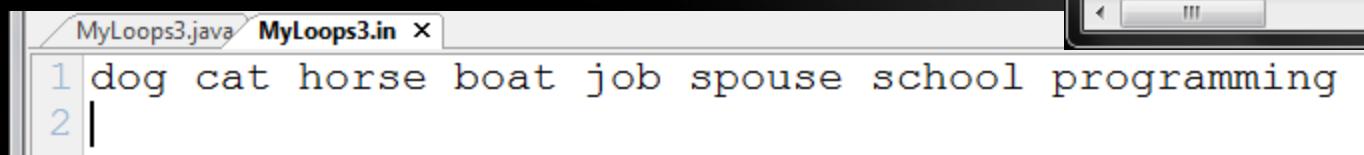
- And now, one more really cool feature about arrays...
- You can very easily sort them in alphabetical order....LOOK!!!



```
java X MyLoops3.in
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("MyLoops3.in"));
    String s = f.nextLine();
    args = s.split(" ");
    Arrays.sort(args);
    for(int x = 0;x<args.length;x++)
        out.printf("I love my %s.\n",args[x]);
    for(int x = args.length-1;x>=0;x--)
        out.printf("I love my %s.\n",args[x]);
}
```



```
C:\Program Files (x86)\Xinox S...
I love my boat.
I love my cat.
I love my dog.
I love my horse.
I love my job.
I love my programming.
I love my school.
I love my spouse.
I love my spouse.
I love my school.
I love my programming.
I love my job.
I love my horse.
I love my dog.
I love my cat.
I love my boat.
Press any key to continue
```

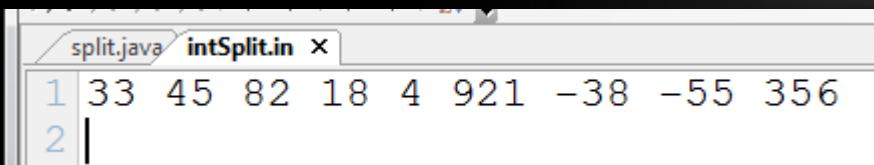


```
MyLoops3.java MyLoops3.in X
1 dog cat horse boat job spouse school programming
2 |
```

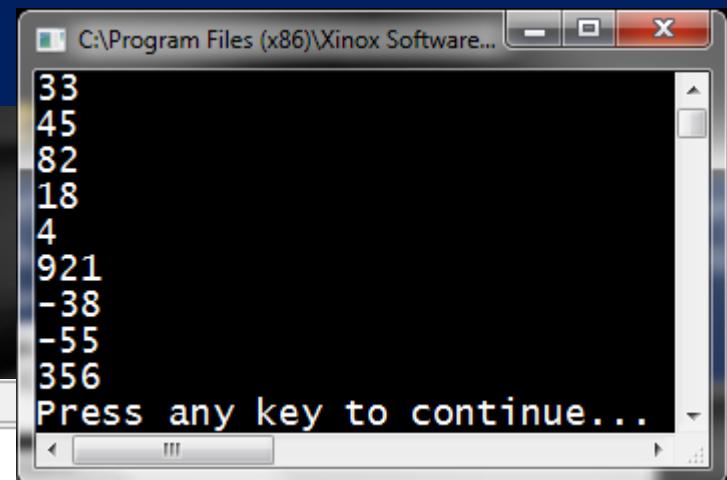


Integer array processing

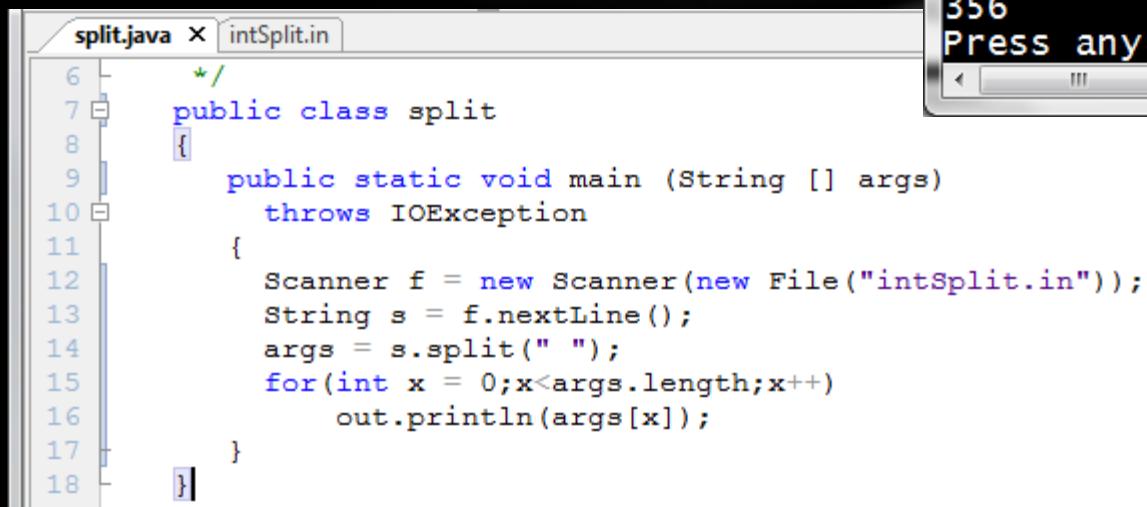
- Now let's further explore this new process using an array of integers.
- Below is a new data file, a program to process it using the split method, and a simple vertical output of the data.



```
split.java intSplit.in x
1 33 45 82 18 4 921 -38 -55 356
2 |
```



```
C:\Program Files (x86)\Xinox Software...
33
45
82
18
4
921
-38
-55
356
Press any key to continue...
```

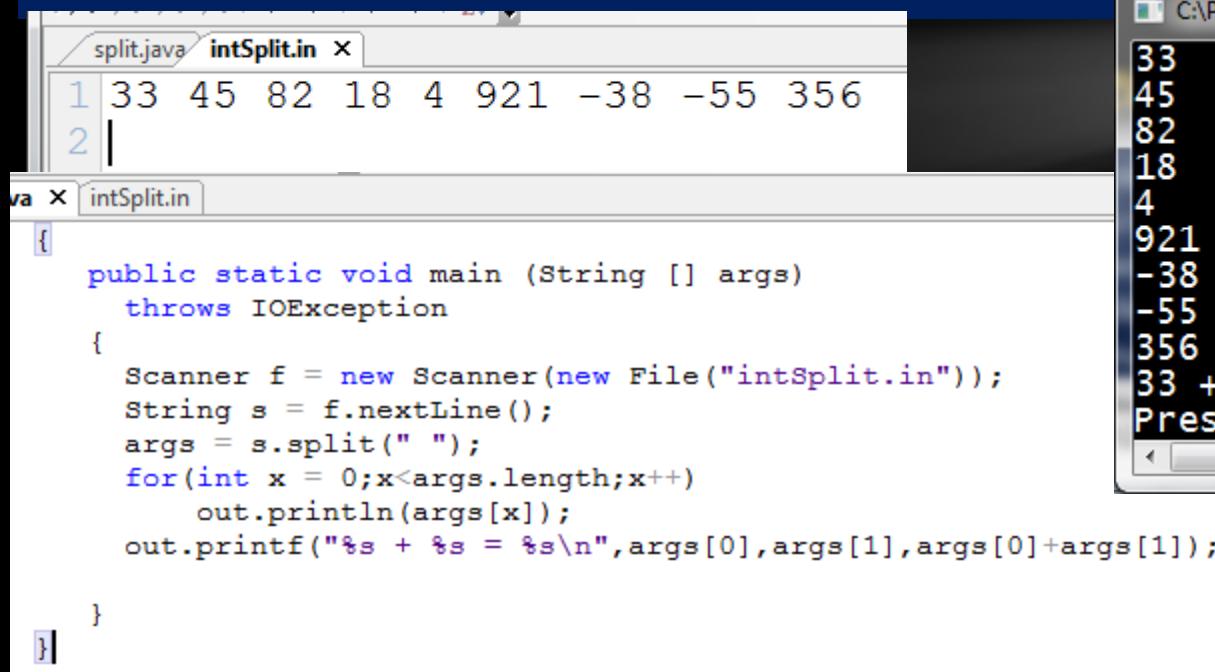


```
split.java x intSplit.in
6 */
7 public class split
8 {
9     public static void main (String [] args)
10    throws IOException
11    {
12        Scanner f = new Scanner(new File("intSplit.in"));
13        String s = f.nextLine();
14        args = s.split(" ");
15        for(int x = 0;x<args.length;x++)
16            out.println(args[x]);
17        }
18    }
```



Integer array processing

- It looks simple enough, but there is a problem...
- Can you see it!!!
- This is not an array of integers!
- If you add the first two values, the result is a **concatenation**, not a mathematical addition!

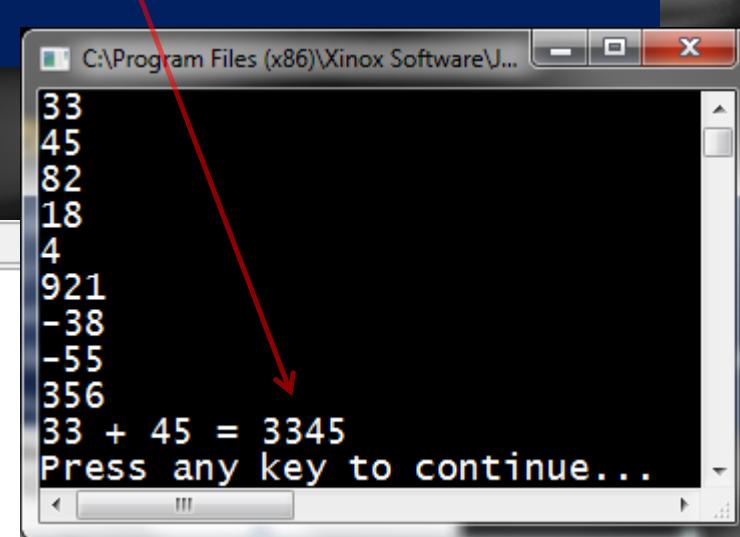


A screenshot of a Java development environment. On the left, there are two tabs: "split.java" and "intSplit.in". The "intSplit.in" tab contains the following text:

```
1 33 45 82 18 4 921 -38 -55 356
```

The "split.java" tab shows the following Java code:

```
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("intSplit.in"));
    String s = f.nextLine();
    args = s.split(" ");
    for(int x = 0;x<args.length;x++)
        out.println(args[x]);
    out.printf("%s + %s = %s\n",args[0],args[1],args[0]+args[1]);
}
```



A screenshot of a terminal window titled "C:\Program Files (x86)\Xinox Software\...". The window displays the following text:

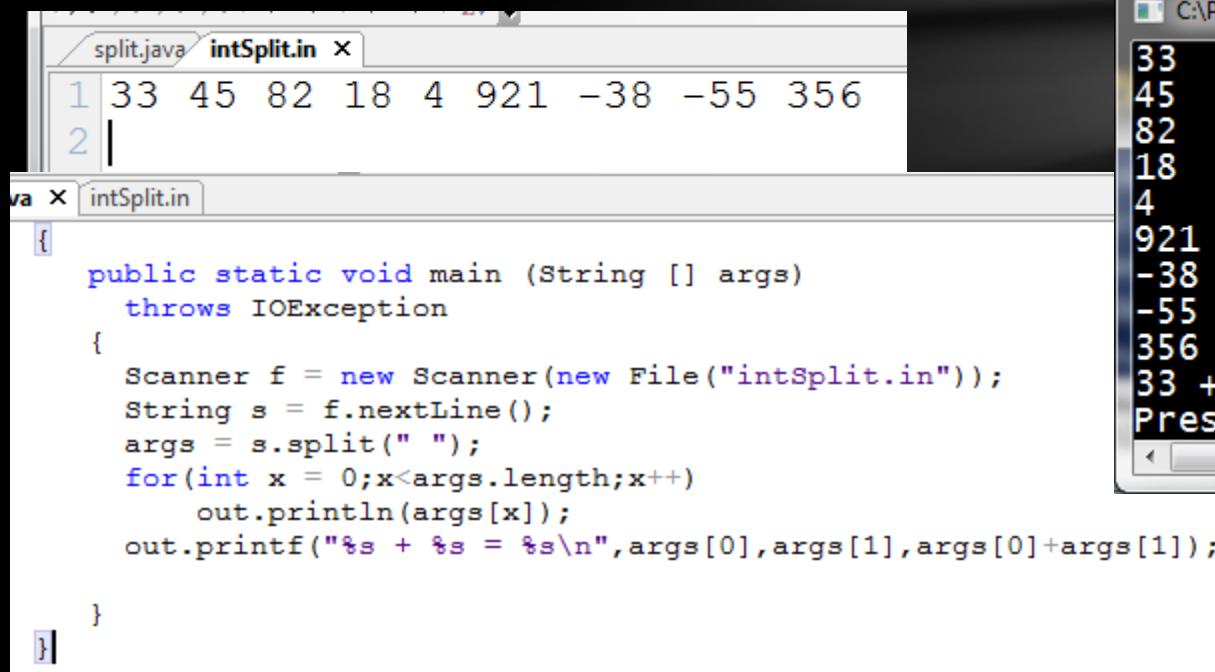
```
33
45
82
18
4
921
-38
-55
356
33 + 45 = 3345
Press any key to continue...
```

A red arrow points from the text "33 + 45 = 3345" in the terminal window to the corresponding line in the code editor.



Integer array processing

- The values look like integers, but are still indeed Strings in a String array.
- What must be done is a ***transformation*** from a String array to an int array, which is what we will do next.

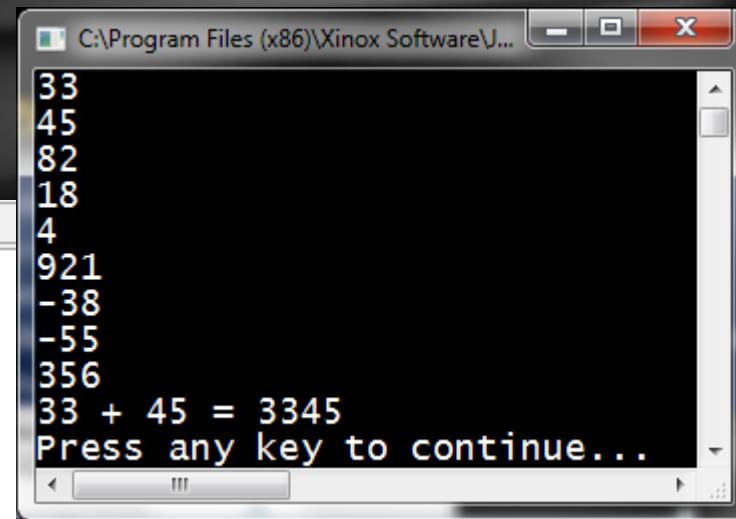


A screenshot of a Java development environment. In the top left, there are tabs for "split.java" and "intSplit.in". Below them, the code editor shows the following Java code:

```
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("intSplit.in"));
    String s = f.nextLine();
    args = s.split(" ");
    for(int x = 0;x<args.length;x++)
        out.println(args[x]);
    out.printf("%s + %s = %s\n",args[0],args[1],args[0]+args[1]);
}
```

The "intSplit.in" file contains the following integers separated by spaces:

```
33 45 82 18 4 921 -38 -55 356
```



A screenshot of a terminal window titled "C:\Program Files (x86)\Xinox Software\...". The window displays the following output:

```
33
45
82
18
4
921
-38
-55
356
33 + 45 = 3345
Press any key to continue...
```



Integer array processing

- First, a **clarification**....there is no direct way for the “split” process to take a line of mathematical values – integers or decimals – and put them into an array of their own type.
- **“split” only creates String arrays....**

The screenshot shows a Java development environment. On the left, there are two tabs: "split.java" and "intSplit.in". The "intSplit.in" tab contains a single line of text with space-separated integers: 33 45 82 18 4 921 -38 -55 356. Below these tabs is a code editor window with the following Java code:

```
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("intSplit.in"));
    String s = f.nextLine();
    args = s.split(" ");
    for(int x = 0;x<args.length;x++)
        out.println(args[x]);
    out.printf("%s + %s = %s\n",args[0],args[1],args[0]+args[1]);
}
```

To the right of the code editor is a terminal window showing the output of the program. The integers from the input file are printed one per line. Below the integers, the sum of the first two integers is calculated and printed. The terminal window has a title bar "C:\Program Files (x86)\Xinox Software\J..." and a message "Press any key to continue..." at the bottom.



Integer array processing

- However, the “split” process is the first of a three-step process required to create an array of integers.
- The **second** and **third** steps are to **create** and **fill** a new array of integers exactly the same length as the array of Strings created by the “split” process.
- That is shown here...***STUDY THIS EXAMPLE CAREFULLY!!!***

The screenshot shows a Java development environment with two windows. The top window displays the contents of a file named `intSplit.in`, which contains the following integers separated by spaces:

```
33 45 82 18 4 921 -38 -55 356
```

The bottom window shows the code for a Java program named `split.java` that reads from `intSplit.in` and prints each integer to the console. The code uses a `Scanner` to read the file, `String` to store the line, and `args` to store the split string. It then creates an `int` array `ints` and fills it with integers from the `args` array. Finally, it prints each integer and the sum of the first two integers.

```
split.java x intSplit.in
11
12     Scanner f = new Scanner(new File("intSplit.in"));
13     String s = f.nextLine();
14     args = s.split(" ");
15     int [] ints = new int[args.length];
16     for(int x = 0;x<args.length;x++)
17         ints[x]=Integer.parseInt(args[x]);
18     for(int x = 0;x<ints.length;x++)
19         out.println(ints[x]);
20     out.printf("%d + %d = %d\n",ints[0],ints[1],ints[0]+ints[1]);
21
```

C:\Program Files (x86)\Xinox Software\

```
33
45
82
18
4
921
-38
-55
356
33 + 45 = 78
Press any key to continue . . .
```



Integer array processing

- As you can see by the output, the values are indeed integers now because an array of integers is being processed, as evidenced by the final math statement.
- Let's notice several key parts of this "transformation" process as shown below.

```
split.java intSplit.in x
1 33 45 82 18 4 921 -38 -55 356
2 |
```

```
split.java x intSplit.in
11
12     Scanner f = new Scanner(new File("intSplit.in"));
13     String s = f.nextLine();
14     args = s.split(" ");
15     int [] ints = new int[args.length];
16     for(int x = 0,x<args.length,x++)
17         ints[x]=Integer.parseInt(args[x]);
18     for(int x = 0;x<ints.length;x++)
19         out.println(ints[x]);
20     out.printf("%d + %d = %d\n",ints[0],ints[1],ints[0]+ints[1]);
```

```
C:\Program Files (x86)\Xinox Software\ 33
45
82
18
4
921
-38
-55
356
33 + 45 = 78
Press any key to cont
```



Integer array processing

- After the “split” process creates the array of Strings, a new array of **integers** is created using the magic word **new**. This second step involves several new concepts we must understand
 - `int [] ints = new int[args.length];`

The screenshot shows a Java development environment with two windows. The top window displays the contents of a file named `intSplit.in`, which contains the following integers separated by spaces:

```
33 45 82 18 4 921 -38 -55 356
```

The bottom window shows the code for a Java program named `split.java`. The code reads integers from the file `intSplit.in` and prints them to the console. A specific line of code is highlighted in yellow:

```
int [] ints = new int[args.length];
```

The output window shows the integers from the file and the result of their addition:

```
33  
45  
82  
18  
4  
921  
-38  
-55  
356  
33 + 45 = 78  
Press any key to continue . . .
```



Integer array processing

- `int [] ints = new int[args.length];`
- This command first creates an *int array object reference* (`int [] ints`), similar to `String [] args`. However, instead of an array of Strings, an array of integers is what we want. The name `ints` is NOT “magic” and can be anything you want.

The screenshot shows a Java IDE interface with two tabs: "split.java" and "intSplit.in". The "split.java" tab contains the following code:

```
split.java intSplit.in x
1 33 45 82 18 4 921 -38 -55 356
2
13
14
15
16
17
18
19
20
21
String s = f.nextLine();
args = s.split(" ");
int [] ints = new int[args.length];
for(int x = 0;x<args.length;x++)
    ints[x]=Integer.parseInt(args[x]);
for(int x = 0;x<ints.length;x++)
    out.println(ints[x]);
out.printf("%d + %d = %d\n",ints[0],ints[1],ints[0]+ints[1]);
```

The "intSplit.in" tab contains the input file content:

```
33
45
82
18
4
921
-38
-55
356
```

To the right of the code editor, the output window displays the integers and their sum:

```
33
45
82
18
4
921
-38
-55
356
33 + 45 = 78
Press any key to cont
```



Integer array processing

- `int [] ints = new int[args.length];`
- Then the word **new** is used, a reserved word in Java used to construct new memory for an object.
- **int[args.length]** tells the compiler what type of memory and how much is required...an array of integers the same length as the **args** array that was just created.

The screenshot shows a Java IDE interface with two tabs: "split.java" and "intSplit.in". The "intSplit.in" tab contains the following input file content:

```
1 33 45 82 18 4 921 -38 -55 356
```

The "split.java" tab contains the following code:

```
13
14
15
16
17
18
19
20
21
2
String s = f.nextLine();
args = s.split(" ");
int [] ints = new int[args.length];
for(int x = 0;x<args.length,x++)
    ints[x]=Integer.parseInt(args[x]);
for(int x = 0;x<ints.length;x++)
    out.println(ints[x]);
out.printf("%d + %d = %d\n",ints[0],ints[1],ints[0]+ints[1]);
```

A yellow box highlights the line `int [] ints = new int[args.length];`. The output window on the right shows the integers from the file and the result of their sum:

```
33
45
82
18
4
921
-38
-55
356
33 + 45 = 78
Press any key to cont
```

JC

Integer array processing

- The final step in the transformation process is the use of a loop to take each element in the String array **args**, which “looks” like an integer, but is still a String, and **“parse”** it into an actual integer value, using the **Integer.parseInt()** process introduced in an earlier lesson (3A) on the input process.

The screenshot shows a Java development environment with two windows. The top window displays the contents of the file `intSplit.in`, which contains the integers 33, 45, 82, 18, 4, 921, -38, -55, and 356. The bottom window shows the code for `split.java`. The code reads integers from the file `intSplit.in` using a `Scanner` and stores them in a string array `args`. It then creates a new integer array `ints` and uses a `for` loop to iterate through `args`. Inside the loop, it calls `Integer.parseInt(args[x])` to convert the string elements into integers and store them in `ints`. Finally, it prints each integer in `ints` and calculates the sum of the first two integers. A yellow box highlights the line `ints[x]=Integer.parseInt(args[x]);`.

```
split.java intSplit.in x
1 33 45 82 18 4 921 -38 -55 356
2 |
```

```
split.java x intSplit.in
11 {
12     Scanner f = new Scanner(new File("intSplit.in"));
13     String s = f.nextLine();
14     args = s.split(" ");
15     int [] ints = new int[args.length];
16     for(int x = 0;x<args.length;x++)
17         ints[x]=Integer.parseInt(args[x]);
18     for(int x = 0;x<ints.length;x++)
19         out.println(ints[x]);
20     out.printf("%d + %d = %d\n",ints[0],ints[1],ints[0]+ints[1]);
21 }
```

```
33
45
82
18
4
921
-38
-55
356
33 + 45 = 78
Press any key to cont
```



Integer array processing

- When this loop process is completed, the new int array is now filled with integer values that do indeed behave like integers, as you can see in the correct *mathematical calculation* below.

The screenshot shows a Java development environment with three windows:

- split.java**: The main Java source code file containing the following code:

```
11
12     Scanner f = new Scanner(new File("intSplit.in"));
13     String s = f.nextLine();
14     args = s.split(" ");
15     int [] ints = new int[args.length];
16     for(int x = 0;x<args.length;x++)
17         ints[x]=Integer.parseInt(args[x]);
18     for(int x = 0;x<ints.length;x++)
19         out.println(ints[x]);
20     out.printf("%d + %d = %d\n",ints[0],ints[1],ints[0]+ints[1]);
```
- intSplit.in**: An input file window showing the following integer sequence:

```
1 33 45 82 18 4 921 -38 -55 356
```
- intSplit.in**: An output console window showing the integers from the input file followed by a mathematical calculation:

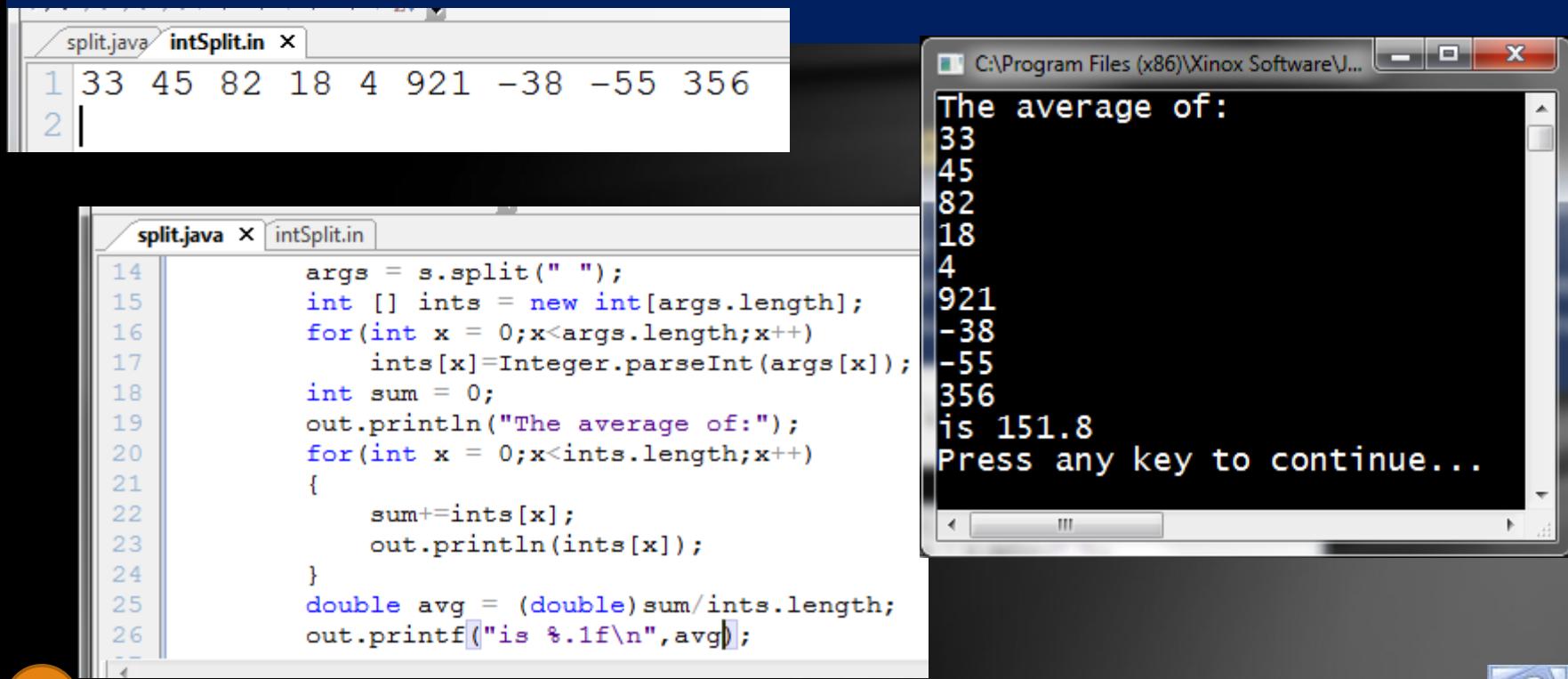
```
33
45
82
18
4
921
-38
-55
356
33 + 45 = 78
Press any key to cont
```

A red arrow points from the mathematical calculation in the output console to the highlighted line of code in the Java source file.



Example 1: average of ints

- The beauty of this process is that you can now process the data numerous times because it is stored in memory as an array of integers.
- Below is a program that will calculate the average of the integers and output the results, a classic programming technique using arrays.



```
split.java intSplit.in
1 33 45 82 18 4 921 -38 -55 356
2 |
```

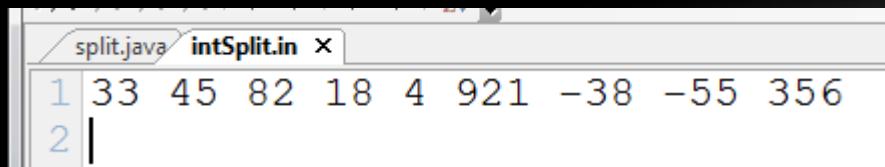
```
split.java x intSplit.in
14 args = s.split(" ");
15 int [] ints = new int[args.length];
16 for(int x = 0;x<args.length;x++)
17     ints[x]=Integer.parseInt(args[x]);
18 int sum = 0;
19 out.println("The average of:");
20 for(int x = 0;x<ints.length;x++)
21 {
22     sum+=ints[x];
23     out.println(ints[x]);
24 }
25 double avg = (double)sum/ints.length;
26 out.printf("is %.1f\n",avg);
```

```
C:\Program Files (x86)\Xinox Software\...
The average of:
33
45
82
18
4
921
-38
-55
356
is 151.8
Press any key to continue...
```

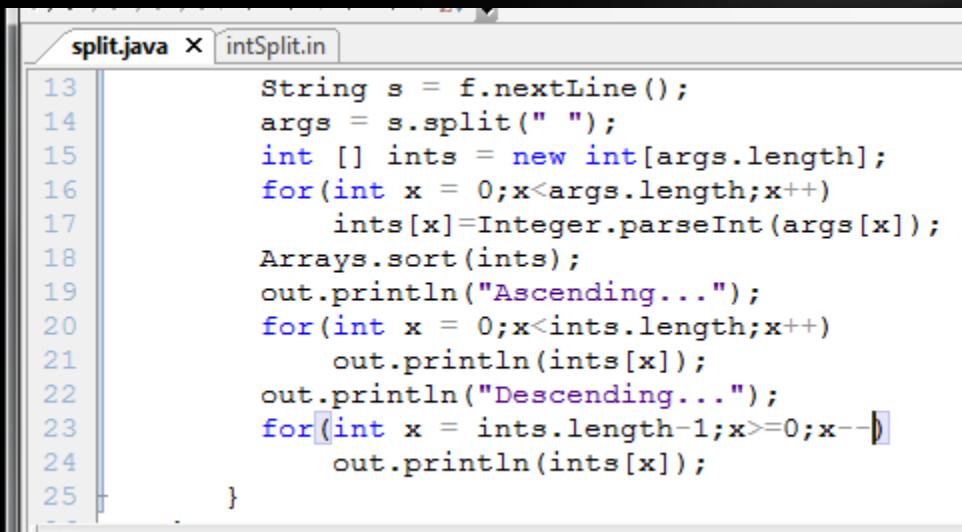


Example 2: sorted ints

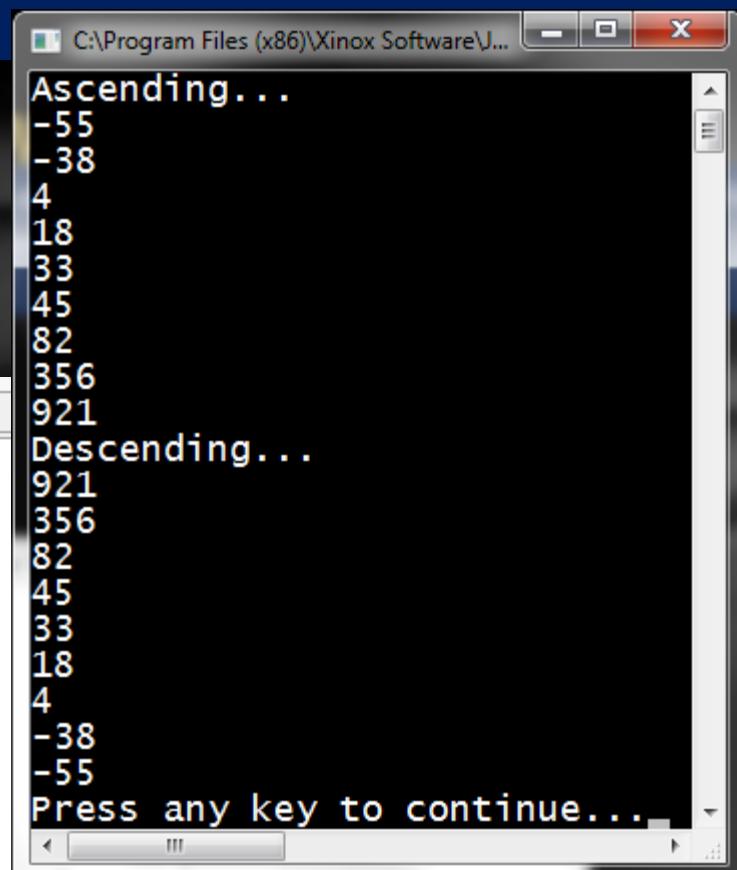
- You can also apply the **Arrays.sort** process we used earlier, only this time the sorting process is by numerical value, not alpha order.



```
split.java intSplit.in x
1 33 45 82 18 4 921 -38 -55 356
2 |
```



```
split.java x intSplit.in
13     String s = f.nextLine();
14     args = s.split(" ");
15     int [] ints = new int[args.length];
16     for(int x = 0;x<args.length;x++)
17         ints[x]=Integer.parseInt(args[x]);
18     Arrays.sort(ints);
19     out.println("Ascending...");
20     for(int x = 0;x<ints.length;x++)
21         out.println(ints[x]);
22     out.println("Descending...");
23     for(int x = ints.length-1;x>=0;x--)
24         out.println(ints[x]);
25 }
```



```
C:\Program Files (x86)\Xinox Software\J...
Ascending...
-55
-38
4
18
33
45
82
356
921
Descending...
921
356
82
45
33
18
4
-38
-55
Press any key to continue...
```



Example 3: method to transform

- In this example we will define a **method** to do the transformation process shown earlier , one that receives a String of integers as a parameter, and then returns an array of integers.

```
split.java intSplit.in x  
1 33 45 82 18 4 921 -38 -55 356  
2 |
```

```
Ascending...  
-55  
-38  
4  
18  
33  
45  
82  
356  
921  
Descending...  
921  
356  
82  
45  
33  
18  
4  
-38  
-55  
Press any key to continue..
```

```
/*Method to receive a String containing  
 *integers and return an integer array.  
 */  
public static int [] intArr (String arr)  
{  
    String [] arrs = arr.split(" ");  
    int [] ints = new int[arrs.length];  
    for(int x = 0;x<arrs.length;x++)  
        ints[x]=Integer.parseInt(arrs[x]);  
    return ints;  
}  
public static void main (String [] args)  
    throws IOException  
{  
    Scanner f = new Scanner(new File("intSplit.in"));  
    String s = f.nextLine();  
    int [] ints = intArr(s);  
    Arrays.sort(ints);  
    out.println("Ascending...");  
    for(int x = 0;x<ints.length;x++)  
        out.println(ints[x]);  
    out.println("Descending...");  
    for(int x = ints.length-1;x>=0;x--)  
        out.println(ints[x]);  
}
```



Example 4: method to average

- Here we will use a method to calculate the average of a received array of integers, like example 1 did. We will call the Example 3 method **intArr** to first transform the input String, then **avgArr** to calculate and return the average.

The screenshot shows a Java development environment with two windows. The left window displays the output of the program, which reads integers from a file named 'intSplit.in' and calculates their average. The right window shows the source code of the 'split.java' file, which includes a method 'avgArr' for calculating the average of an integer array and a 'main' method for reading input from a file and printing the results.

```
split.java  intSplit.in x
1 33 45 82 18 4 921 -38 -55 356
2

split.java x  splitLab6D_4.in
44 }
45 /**
46  *Method to receive an integer array
47  *and return the average of all the integers
48  *in the array.
49 */
50 public static double avgArr (int [] ints)
51 {
52     int sum = 0;
53     for(int x = 0;x<ints.length;x++)
54         sum+=ints[x];
55     double avg = (double)sum/ints.length;
56     return avg;
57 }
58 public static void main (String [] args)
59     throws IOException
60 {
61     Scanner f = new Scanner(new File("intSplit.in"));
62     String s = f.nextLine();
63     int [] ints = intArr(s);
64     for(int x = 0;x<ints.length;x++)
65         out.printf("%d\n",ints[x]);
66     out.printf("Average = %.1f\n",avgArr(ints));
67 }
```

Output window content:

```
33
45
82
18
4
921
-38
-55
356
Average = 151.8
Press any key to continue
```



Lesson Summary

- In this lesson, you learned a new and powerful process called “split”, in which a line of data can be placed into an array of Strings and transformed into a numerical array as well.
- Now it is time to practice with several examples.



Labs

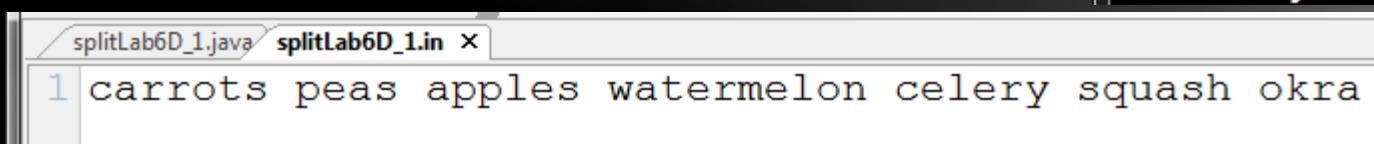
- Split will be the name of this series of labs. As you did before, create a separate folder and file called Split and do your work there.



Lab 1 – *AlphaOmega*

- WAP that reads a line of words from a data file and outputs the following elements:
 - The first word in the original array
 - The last word in the original array
 - The entire array in original order
 - The first word in alphabetical order
 - The last word in alphabetical order
 - The entire array in REVERSE alphabetical order.

```
First = carrots
Last  = okra
Original order:
carrots
peas
apples
watermelon
celery
squash
okra
Alpha = apples
Omega = watermelon
Descending alpha order:
watermelon
squash
peas
okra
celery
carrots
apples
Press any key to continue
```



The screenshot shows a Java development environment. The code editor has two tabs: "splitLab6D_1.java" and "splitLab6D_1.in". The "splitLab6D_1.java" tab contains the Java source code for the program. The "splitLab6D_1.in" tab contains the input file content, which is a single line of words: "carrots peas apples watermelon celery squash okra".



Lab 2 – ***dblArr***

- WAM called ***dblArr*** that receives a String containing a line of decimal values and returns an array of doubles. For help, refer to example 3 shown earlier in the lesson. Also, you'll need to remember how to parse double values, covered in a previous lesson.

The screenshot shows a Java development environment. In the top window, titled 'split.java', there is a code editor with two tabs: 'split.java' and 'splitLab6D_4.in'. The code editor contains the following code:

```
split.java
splitLab6D_4.in x
1 78.3 12.59 23.4 94 112.5 87.4253
2 |
```

In the bottom window, titled 'C:\Program Files (x86)\Xinox Software\JCr...', the terminal output is displayed:

```
78.3
12.6
23.4
94.0
112.5
87.4
```



Lab 3 – *avgArr* (*overloaded*)

- WAM called *avgArr* that receives an array of doubles and returns the average. This method will be *almost identical* to the one shown in example 4 shown earlier. It is actually an example of “overloading”, where a method name is the same, but the parameter signature is different.
- Below is the method header and the method call from main.

```
}

/**Method to receive a double array
 *and return the average of all the values
 *in the array.
*/
public static double avgArr (double [] dubs)
{
}

public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("splitLab6D_4.in"));
    String s = f.nextLine();
    double [] dubs = dblArr(s);
    for(int x = 0;x<dubs.length;x++)
        out.printf("%.1f\n",dubs[x]);
    out.printf("Average = %.1f\n",avgArr(dubs));
}
```



The screenshot shows a Java development environment with two tabs: "Split.java" and "splitLab6D_4.in". The "splitLab6D_4.in" tab contains the input data: 78.3 12.59 23.4 94 112.5 88.4253. The "Split.java" tab shows the code. In the terminal window below, the program outputs each value on a new line and then calculates and prints the average as 68.2.

```
78.3
12.6
23.4
94.0
112.5
88.4
Average = 68.2
Press any key to continue . . .
```

JavaDoc

- Complete the documentation for all of the methods, and then run the JavaDoc utility.

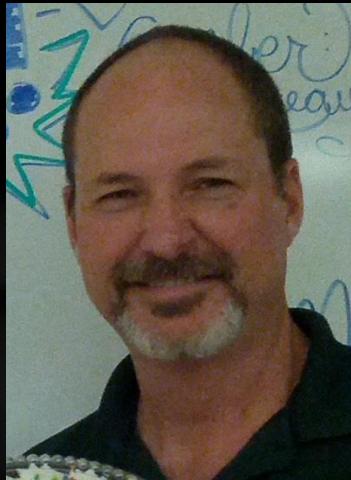


CONGRATULATIONS!

- You now know how to use the String class split method to transform String data into String arrays and value arrays.
- *Lesson 6E will explore nested loops, a key process necessary for processing matrix arrays (2D, 3D, etc), as well as outputting various patterns.*



Thank you, and have fun!



To order supplementary materials for all the lessons in this package, such as lab solutions, quizzes, tests, and unit reviews, visit the [O\(N\)CS Lessons](#) website, or contact me at

[John B. Owen](mailto:captainjbo@gmail.com)
captainjbo@gmail.com



10/10/2014

