Set 15: Arrays Part 2

Skill 15.01: Populate an array from the command line
Skill 15.02: Interpret the *NullPointerException*
Skill 15.03: Use the split method to produce an array
Skill 15.04: Count the occurrences of a regular expression
Skill 15.05: Apply parallel arrays to manage data

Skill 15.01: Populate an array from the command line

Skill 15.01 Concepts

In our previous lesson we learned how arrays can be applied to store lists of data. In this lesson we will explore some additional features associated with arrays, along with how they can be manipulated and applied to represent complex data sets.

Believe it or not you have been declaring arrays since your first program. Consider the following code you have been using to declare the main method in all your programs,

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

Now that we know about arrays, we can see the `String args[]` is declaring args as a String array. In fact, the name of the array, *args* could be called by any legal variable name.

But where and how is this *args[]* array to be used?

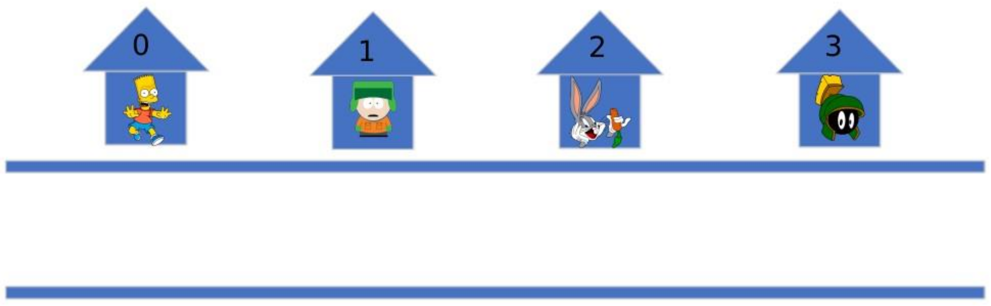
The *args[]* array allows us to pass command line arguments to the main method. Consider the main method in a class called *Street.java*, where the array name *args* has been replaced with *houses*

```
public static void main(String houses[]){}
```

The houses array could be populated directly from the command line using the following syntax

```
java Street Bart Kyle Bugs Marvin
```

After compiling *Street.java* and running the command above, our houses array has the following assignments,



[Skill 15.01: Exercise 1]()

Skill 15.02: Interpret the *NullPointerException*

Skill 15.02 Concepts

A null pointer exception results when you try to access the value of an array object that has not yet been initialized. In the following example, the *houses* array is declare, but it has not been initialized. Attempting to assign "Homer" to a location in the array results in a *NullPointerException*.

Code	Output	Explanation

`String houses[];`
`houses[6] = "Homer";`
	Exception in thread "main"
	java.lang.NullPointerException
		The array houses has been declared, but *not* initialized

Skill 15.02: Exercise 1

Skill 15.03: Use the split method to produce an array

Skill 15.03 Concepts

The split method is useful for splitting a String into separate elements and storing the result as a String array. The signature for the split method is below.

```
public String[] split (String regex)
```

- `public String[]` - the value returned when the method is implemented
- `split` - the name of the method
- `String regex` - the expression that indicates where the split should occur

The *regex* (regular expression) indicates where the split can occur. This expression can specify that the split occur at one character or multiple characters. Both scenarios are described below.

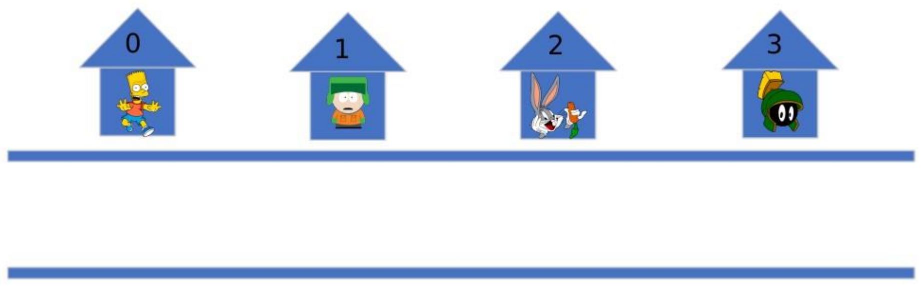
One character

Consider the String people as declared below. Because the people are separated by commas, we can easily split the String into an array by setting the *regex* expression equal to ",".

Below is an example,

```
String people = "Bart, Kyle, Bugs, Marvin";
String houses[] = people.split(",");
```

After implementing the code above, our houses have the following assignments,



Multiple characters

The split method can be applied to split a String at multiple locations. Consider the same String as before, but instead we have separated the people with different characters. The String can still be separated, but we need to indicate all the characters where the split should occur. In the code below each character in the *regex* expression is separated by a "|". This symbole stands for "or".

Code	**Output**

String people = "Bart~Kyle,Bugs-Marvin";
String houses[] = people.split("~|,|-");
 	
System.out.println(houses[0]);
System.out.println(houses[1]);
System.out.println(houses[2]);
System.out.println(houses[3]);
	Bart
	Kyle
	Bugs
	Marvin

Regular expression are a very important tool in programming and they can become quite complex. The link below provides more information on how to parse Strings using regular expressions.

Regular Expressions Resource

Skill 15.03: Exercise 1

Skill 15.04: Count the occurrences of a regular expression

Skill 15.04 Concepts

The split method can be used to count the number of occurrences of a specified regular expression within a String. For example, consider the following String,

```
String s = “IF THE BOX IS RED IT'S THE RIGHT ONE.”;
```

In order to count the occurrences of “THE”, we can use it as the regular expession with the split method (String sp[] = s.split(“THE”)). The underlined portions below show the three different elements of the array into which our array is “split”.

IF **THE** BOX IS RED IT'S **THE** RIGHT ONE.

The number of elements in the array is three (sp.length); therefore, the number of occurrences of "THE" is sp.length – 1.

A complication occurs if the delimiter trails the String as in the following example,

Code	Output

String s = "ENOUGH USE OF THE WORD THE";
String sp[] = s.split("THE");
System.out.println(sp.length-1);
System.out.println(sp[0]);
System.out.println(sp[1]);
System.out.println(sp[2]);
	1
	ENOUGH USE OF
	WORD
	ArrayIndexOutOfBoundsException

The value for the number of instances of "THE" is incorrect when the code above is run. This is because element sp[2] does not exist. To account for this anomoly we can concatenate a dummy character to the end of the String,

Code	Output

String s = "ENOUGH USE OF THE WORD THE" + "DUMMY";
String sp[] = s.split("THE");
System.out.println(sp.length-1);
System.out.println(sp[0]);
System.out.println(sp[1]);
System.out.println(sp[2]);
	2
	ENOUGH USE OF
	WORD
	DUMMY

Now let's consider how split handles delimiters which occur at the beginning of a String. In the following example, one would think that the length of the resultant array is one. But when the split method is implemented, two elements are created: An empty String before the "THE" and the text "ENOUGH USE OF WORD"

Code	Output

String s = "THE ENOUGH USE OF WORD";
String sp[] = s.split("THE");
System.out.println(sp.length-1);
System.out.println(sp[0]);
System.out.println(sp[1]);
	1
	""
	ENOUGH USE OF WORD

Skill 15.04: Exercise 1

Skill 15.05: Apply parallel arrays to manage data

Skill 15.05 Concepts

A group of *parallel* arrays is a data structure that uses multiple arrays to represent a singular array of records. It does so by keeping a separate, homogeneous data array for each field of the record, each having the same number of elements.
As an example, consider a grade book. The first array, *assignments*, is used to store the assignments. Then, we have a separate array for each student. Because each student has the same number of assignments, we can store them in *parallel*

	0	1	2	3	4
String assignments[]	assignment 1	assignment 2	assignment 3	assignment 4	assignment 5
char Bart[]	5	R	3	4	M
char Kyle[]	2	3	2	4	R
char Bugs[]	4	M	3	5	4
char Marvin[]	5	2	3	3	4

Now, let's suppose Bugs completes his missing assignment 2. Below illustrates how we can use the fact that all the arrays share the same indexing to update this assignment.

Code

```
char score = '3';
int index = 0;
for(int a = 0; a < assignments.length; a++){
     if(assignments[a].equals("assignment 2")){
          index = a;
     }
}
Bugs[index] = score;
System.out.println("Bugs now has a score of " + Bugs[index] + " on " +
assignments[index]);
```

Output

```
Bugs now has a score of 3 on assignment 2
```

Skill 15.05: Exercise 1