The Class, String

A newer set of notes – Set #24 – examines the type char. Strings are composed of individuals chars. You should read set Set #24 before continuing to read this set of notes.

Refer to code Set17. The .pde file, String1 is discussed below. The folder StringInput shows simple string input from the keyboard.

Processing is really Java. Processing is often referred to as a "super set " of Java because it is all of Java and more. Much of the java aspects are hidden from view by the Processing IDE. We will look at some of the hidden stuff in a future class meeting.

The main organizing structure of Java – and Processing – is the class. One reason for using a class structure is the overall size and complexity of the language. A way to consider this is to think about a library. It is composed of books – lots and lots of books. But, what if every book in the library was pasted together into a single large book? That's right – one book. Think of the ramifications:

- only one person could check it out at a time
- it would be difficult to find something specific
- its shear size and weight would make it very cumbersome and be difficult to work with in any way

This same list and more are the reasons that Java is broken up into smaller units or segments. These segments are called classes. The class list in the Java API has over 2000 entries.

We have been programming inside a class since we wrote the first program on day 1. We will look at this later.

Your have used classes in your earlier work. Sound and inmage required classes. In this next look at a class, we will look at the String class of Java. Here are the high points:

1. Anything inside quotation marks (" ") is a String. Think of a string of pearls or a string of beads and picture a string of characters. Here are some examples of Strings:
   "" the empty string
   " " the singlespace as a string
   "hello world" which is one of the most famous programing Strings
2. Strings are NOT arrays. They are references to pieces of memory called objects or specifically in this discussion, String objects.
3. The characters in a String object are indexed like arrays beginning at zero (but they are not elements of an array).
4. There is a length field (variable) of the String object but it is private – we cannot directly access it. To find the length of a String, we must use the length( ) function.

5. The last character in a String has an index of the String's length( )-1

The processing API lists seven String functions (In Java, functions have a different name – they are called methods. sigh… functions, methods, parameters, arguments, sigh again…).
that may be useful to you. The JAVA API has over 2000 functions (methods) listed for the Java String class. All of these can be used by you in a Processing program because Processing is really Java.

This compares and contrasts an array of char and a String;

Category	Array of char	String
declaration	`char [ ] c;`	`String s;`
one form of initialization	`char [ ] c = {'a', 'b'};`	`String s = "hello";`
access	`println( c[ 0 ] );`	`println( s.charAt( 0 ) );`
Changing values	`c[ 0 ] = 'x';`	Illegal for Strings
Size	`println( c.length );`	`println( s.length( ) );` note parens
Index of beginning character	`println( c[ 0 ] );`	`println( s.charAt( 0 ) );`
Index of last character	`println( c[ c.length - 1 ] );`	`println( s.charAt( s.length( ) - 1 ) );`

Equality	==
	compiles, runs, and does not work properly
		==
		compiles, runs, and usually does not work properly
		`if ( s.equals(s1) )`
		`{`
		
		`}`
		compiles, runs and works properly

One question that is asked is how can the toUpperCase( ) and toLowerCase( ) functions be used. One way to use them is when we are dealing with user input (discussed later in these notes). If we ask the use to type either yes or no as a reply to a question the user can type any of the following to correctly respond:
yes Yes YES yEs yeS YeS no No NO nO
Any of these are proper inputs. Yet we do not want to have to type:

```
if ( s1.equals( "YES" ) ||
     s1.equals( "yes" ) ||
     s1.equals( "yES" ) ||
     s1.equals( "yeS" ) ||
     s1.equals( "YeS" ) ||
     s1.equals( "yEs" )  )
   {      . . .
   }
```

This is silly. What if we were asking for the capital of the state of Florida? We can use the case changing functions to simplify the code:

```
if ( s.toUpperCase().equals("YES" ) )
{
  . . .
}
```

The function **s.toUpperCase()** returns a new String and we can use the **equals()** function with this returned String. This does look a bit bizarre, but there is a simpler way to write this code:

```
String temp = s.toUpperCase( );
if ( temp.equals("YES" ) )
{
  . . .
}
```

Arrays of String References

We can build arrays of any type of data so we can build an array of Strings. The syntax is the same:

```
String [ ] labels = { "On", "Off", "In", "Out" };
```

String Input

We have used single character input for weeks. We have done this in the keyPressed( ) function. The advantage of this is that the program does not pause and wait for the input. If we have no animation ongoing when we want input, a pause in the action is fine. But, if there is animation occurring, a pause for input is not good.

Processing provides no other direct way to get user input and we do not want to explore Java's ways of getting user input so we will keep using the keyPressed() function to do our work. This code, explained in detail on the next page can get String input from the user:

```
void keyPressed( )
{
  if (keyCode == DELETE ||  keyCode == BACKSPACE )
  {
     if ( s.length() > 0 )
     {
       s = s.substring(0, s.length( )-1);
     }
  }
  else
  {
     s = s + key;
  }
}
```

Code	Comments
void keyPressed( ) {	If we assume the user might not type things correctly and will want to correct their input, we have to account for the delete (Mac) or backspace (Windows) keys. These keys are CODED so we check the keyCode value first.
if (keyCode == DELETE || keyCode == BACKSPACE)	
{	
if (s.length() > 0)	If the user typed one of these coded keys, we must first test the String to insure it is not empty. Our code will crash is we try to delete a character from an empty String.
{	
s = s.substring (0, s.length( )-1);	We cannot alter a String so we have to make an new String without the last character. The substring( ) function does this. The details of the parameters are discussed below.
}	
}	
else	If the user did not type the coded keys tested for above, this assumes they typed a character and a new String is made from the old String and the new letter using the concatenation operator +
{	
s = s + key;	
}	
}	

This code is considered NAÏVE because it does not deal with bad input such as the up arrow or some other non-printing key expect the two that are tested for in the first if – the [delete] and [backspace keys.

The parameters for the call of the **substring( )** function need some explanation:

s = s.substring (0, s.length( )-1);

The last character in the String has an index of :

```
s.length( )-1
```

If we want a new String created that does not have the last character, shouldn't we use the following for the second parameter:

```
s = s.substring (0,  s.length( )-2);
```

This is one time that our logic fails us. When we specify a range of values, there are many functions that include only the values UP TO but do not include the second value in the range.

For example, if we call the **random( )** function like this:

```
float f = random( 10, 20 );
```

we get values between:

10.0 and 19.9999999

We will never get 20.0 as a returned value.

When we set the size of the window:

```
size( 400, 400 );
```

The actual size dimension of the window is 0-399 pixels wide and 0-399 pixels high. If you do not believe us, try this code:

```
size( 400, 400 );
background( 0 );
stroke( 255, 0, 0 );
line( width 0, width, height );
```

and look very carefully at the right side of the window.

In the parameter list for substring( ):

```
s = s.substring (0,  s.length( )-1);
```

*T*he new String that is returned has character from index 0 up to BUT NOT INCLUDING the character at the index of the second parameter.

The code in the folder StopAtReturn is a bit different. This code takes input until you press the return or enter key. The user is asked a question. The answer is Ohio – which is totally wrong but given the attention Ohio had had in the latter stages of the previous presidential campaign, it seemed appropriate at the time Jim wrote this.

We leave it to you to take it apart and see what is going on. Talk to one of us if you need more explanation.

Feel free to use part or all of any of this code in your final projects.