Strings

We've been using strings in our examples, but we haven't fully explored what we can do with them. First of all, `String` is a *class* defined by Java -- we haven't talked about classes yet, but we will later one. The Java `String` class uses a character array to store the individual characters in the string, and then defines several commands that let us do things with that array. Here is how to declare a string variable:

```
String s1;
```

And here is how to initialize a string:

```
s1 = "hello";
```

We can declare and initialize a string at the same time:

```
String s1 = "hello";
```

Concatenation

String concatenation is pushing two strings together with the + sign -- we've seen how to concatenate strings when printing things out. Here are some more examples:

```
int age = 7;
char initial = 'L';
String first = "Bob";
String last = "Jones";

String about = first + " " + initial + ". " + last + ", age " + age;
```

The string `about` will hold "`Bob L. Jones, age 7`".

Notice that we can concatenate types other than strings, like ints and chars. In fact, we can do this with any variable type. The only confusion is when we're concatenating several ints – does the + mean concatenation or integer addition?

The answer, of course, is that it can mean both. The compiler interprets expressions by reading from left to right. If it sees a + and has only seen primitive types (like int a double) so far, it will assume you mean mathematical addition. If it has seen a string or object already on that line, it will assume you mean string concatenation. Here are some examples:

```
String s1 = 4+7+" hi ";        //evaluates to "11 hi"
String s2 = "hi "+4+7;          //evaluates to "hi 47"
String s3 = 4+7+" hi "+4+7;     //evaluates to "11 hi 47"
```

Length

You can get the number of characters in a string by accessing its length. Here's how:

```
String s = "Hello";
int count = s.length();          //has value 5
```

Getting a Character

A string is backed by an array of characters, but you can't access a character in a string by using array notations (`[i]`). Instead, you need to use the `charAt` command. This returns the character at a particular index in the string. Like an array, the first index in a string is 0, and the last one is the length-1. Here's an example:

```
String s = "Hello";
char c = s.charAt(1);            //has value e
```

Now that we know the `length` and `charAt` commands, we can step through every character in a string. Here's an example that prints every character in a string:

```
Scanner s = new Scanner(System.in);
System.out.print("Enter a string: ");
String str = s.nextLine();
for (int i = 0; i < str.length(); i++) {
      System.out.println("Character " + str.charAt(i));
}
```

Finding the Index of a Character/Substring

Sometimes we also want to go the other way – get back the index of a character or piece within a string. We can do this with the `indexOf` command. Here is an example:

```
String s = "cake";
int index = s.indexOf("a");          //has value 1
```

Notice that we get back 1, which is the index of "a" in the string "cake". We can also search for a bigger piece of the string:

```
int bigIndex = s.indexOf("ke");      //has value 2
```

Now we get back 2, because the piece "ke" starts at index 2 within "cake". Here are some more examples:

```
String second = "hello";
int find1 = second.indexOf("l");     //has value 2
int find2 = second.indexOf("x");     //has value -1
```

Notice that "l" appears twice in "hello", but we get back the index of the FIRST "l". Also, when we search for something that's not in the string (like "x"), we get back -1.

Substring

To extract a piece of a string, use the `substring` command:

```
String s = "hello";
String sub = s.substring(3);   //has value "lo"
```

Alternatively, we can specify the starting index (inclusive) and the ending index (non-inclusive):

```
String s = "hello";
String sub1 = s.substring(1, 3);    //has value "el"
String sub2 = s.substring(2, 5);    //has value "llo"
String sub3 = s.substring(0, 1);    //has value "h"
```

StringTokenizer

The Java `StringTokenizer` class helps you break strings into small pieces. This is helpful if you have a bunch of data stored in a string, separated by commas or spaces. First, you need to import the `java.util` library:

```
import java.util.*;
```

Then, you need to create a new `StringTokenizer`:

```
StringTokenizer st = new StringTokenizer(fullString, delimeters);
```

Here, `fullString` is the string you want to break into pieces, and `delimeters` is a string that contains all the characters that separate the pieces in `fullString`. For example, if you had a list of names that were separated by commas and spaces, then `delimeters` would be `" ,"`. Here's an example:

```
String fullString = "Bob, Joe, Lisa, Katie";
StringTokenizer st = new StringTokenizer(fullString, " ,");
```

Now, you need to break apart the pieces. You can check if you've stepped through all of them by calling the `hasMoreTokens()` method, which will return true when you've read the entire string. You can get the NEXT piece of the string by calling the `nextToken()` method. Here's an example that prints all the names in `fullString`:

```
while (st.hasMoreTokens()) {
     String name = st.nextToken();
     System.out.println(name);
```

```
}
```

This will print:

Bob
Joe
Lisa
Katie

Split

The Java `split` command also helps you break strings into pieces in a similar manner as `StringTokenizer`. You can use either technique to break apart a string. The `split` command is similar to the technique we will be using in C# for this process, so it is especially good to be familiar with both. Suppose you have the following string that you want to break apart:

```
String fullString = "Bob, Joe, Lisa, Katie";
```

As before, suppose you want to extract and print each name. We can specify that the names are separated by commas and spaces, and then get a string array of the remaining tokens (in this case, the names:

```
String[] tokens = fullString.split(", ");
```

Now, we can loop through the array and print each value:

```
for (int i = 0; i < tokens.length; i++) {
     System.out.println(tokens[i]);
}
```

This will print:

Bob
Joe
Lisa
Katie

One key difference between `StringTokenizer` and `split` is how the delimeters work. In `StringTokenizer`, the delimeters are individual characters that separate the information we're interested in. When we reach a delimeter, we keep stepping through the string discarding characters until we reach a non-delimeter.

In split, we specify what separates the tokens using a *regular expression*. In the case of our example, it matched the string `", "` to our list of names, and used it as a separator. If we had

put " ," instead (space then comma), it would not have separated the names at all, because it would have found no occurrence of a space followed by a comma. So for `split`, when we provide a string as a separator, we look for that ENTIRE string.

There is a lot more we can do with regular expressions, including a generic way to process all numbers, or all words matching a particular pattern. That is beyond the scope of this course, although you will likely use regular expressions in later CIS classes. In this class, we will just provide an exact string to `split` that we want to process.

Comparing Strings

In this section we will see how to compare strings, in order to see if strings have the same characters (are equal) and to see how to order strings alphabetically (to see if one "comes before" or "comes after" another). When we have compared other types (like ints), we have used ==, <, >, <=, >=, etc. – however, this operators will not work the way we want with strings. When we try to use those operators, we will actually be comparing the *memory address* of the two strings – we want to compare their characters.

Equality

To see if two strings have the same characters, we will use the `equals` command. Here is an example:

```
String s1 = "hello";
String s2 = "hello";

if (s1.equals(s2)) {
     System.out.println("same");
}
```

This example compares the strings `s1` and `s2`. If they contain exactly the same characters, the equals command will evaluate to true. Otherwise, it will evaluate to false. Equals does NOT ignore the case of characters, so it will think that "hello" and "Hello" are not equal.

Ordering alphabetically

To how two strings would be ordered alphabetically, we will use the `compareTo` command. Here is an example:

```
String s1 = "apple";
String s2 = "banana";

if (s1.compareTo(s2) < 0) {
     System.out.println(s1 + " comes before " + s2);
}
```

This example is looking to see if `s1` comes alphabetically before `s2`. Since "apple" comes before "banana", the if-statement is true. In general, if we do:

```
s1.compareTo(s2)
```

The result will be:

0, if `s1` equals `s2`
<0, if `s1` comes alphabetically before `s2`
>0, if `s1` comes alphabetically after `s2`

More Examples

In this section, we will work several additional examples of full Java programs that involve string manipulation.

Example 1

Secret messages (such as secure email, online credit card purchases, online banking, etc.) are often transmitted using encryption. When one of these messages is sent, it is encrypted by altering the message in some way that makes it unrecognizable at face value. The recipient can then "undo" this encryption to get back the original message. The idea is that ONLY the recipient can get back the original message (usually because they know some extra piece of information about how the message was transformed), and that others snooping in will just see gibberish.

In this example, we will write a very basic encryption algorithm. For each letter in the message, we will replace it with the letter that comes three after it in the alphabet. For example, 'A' becomes 'D', 'B' becomes 'E', etc. When we get to the end of the alphabet, we will wrap around to the beginning (so 'Z' becomes 'C'). This program will ask for a message and if the user wants to encrypt or decrypt ("undo" the encryption), and will then print out the resulting message.

```
import java.io.*;
public class Encrypt {
     public static void main(String[] args) {
          Scanner s = new Scanner(System.in);
          System.out.print("Enter a message: ");
          String msg = s.nextLine();
          System.out.println("Enter (e)ncrypt or (d)ecrypt: ");
          char eOrD = (s.nextLine()).charAt(0);

          String result = "";
          String alpha = "abcdefghijklmnopqrstuvwxyz";

          //this program assumes all lower-case characters
```

```
            if (eOrD == 'e') {
                  for (int i = 0; i < msg.length(); i++) {
                        String cur = msg.substring(i,i+1);
                        //get where that letter is in the alphabet
                        int index = alpha.indexOf(cur);
                        //shift it down 3 (wrapping around)
                        index = (index + 3) % 26;
                        //get letter at the new spot, add to result
                        cur = alpha.substring(index, index+1);
                        result += cur;
                  }
            }
            else if (eOrD == 'd') {
                  for (int i = 0; i < msg.length(); i++) {
                        String cur = msg.substring(i,i+1);
                        //get where that letter is in the alphabet
                        int index = alpha.indexOf(cur);
                        //shift it back 3 (wrapping, making sure not <0)
                        index = (index - 3 + 26) % 26;
                        //get letter at the new spot, add to result
                        cur = alpha.substring(index, index+1);
                        result += cur;
                  }
            }
            else {
                  System.out.println("Invalid input");
            }

            System.out.println("Result: " + result);
      }
}
```

Example 2

In this example, we will write a program that gets an input string, and then prints all possible substrings of that string.

```
import java.util.*;

public class Substrings {
      public static void main(String[] args) {
            Scanner s = new Scanner(System.in);
            System.out.print("Enter a string: ");
            String str = s.nextLine();

            //loop through all starting positions
            for (int i = 0; i < str.length(); i++) {
```

```
                    //loop through all STOPPING positions if we start at i
                    for (int j = i+1; j <= str.length(); j++) {
                          //get substring from i up to but not including j
                          String piece = str.substring(i,j);
                          System.out.println(piece);
                    }
              }
        }
}
```

Example 3

In this example, we will write a Pig-Latin converter, where the user enters a sentence and the program translates it to Pig-Latin. The rules we'll use (there are several variations, but we'll keep it simple) for converting an English word to Pig-Latin are as follows:

- If a word begins with a consonant, the consonant is moved to the end of the word and "ay" is added, like this:
  "cake" -> "ake-cay"
- If a word begins with a vowel, "way" is added to the end of the word, like this:
  "is" -> "is-way"

```
import java.util.*;

//this program assumes letters are lower-case and that words are separated only by spaces
public class PigLatin {
      public static void main(String[] args) {
            Scanner s = new Scanner(System.in);
            System.out.println("Enter a sentence: ");
            String str = s.nextLine();

            String result = "";
            //y can be a vowel too, but we're ignoring this case
            String vowels = "aeiou";

            //get each word
            StringTokenizer tok = new StringTokenizer(str, " ");
            while (tok.hasMoreTokens()) {
                  String word = tok.nextToken();
                  String first = word.substring(0,1);

                  //if word begins with a vowel
                  if (vowels.indexOf(first) >= 0) {
                        //add "way" to end of word
                        result += word + "-way ";
                  }
                  else {
                        //word is a consonant
```

```
                String rest=word.substring(1,word.length());
                result += rest + "-" + first + "ay ";
            }
        }

        System.out.println("In Pig-Latin: " + result);
    }
}
```