Computer Programming, I
Laboratory Manual

Experiment #5

Strings & Text Files Input
The String Data Type

The `char` type represents only one character. A `string` is a sequence of characters. To represent a string of characters, use the data type called `String`. A sequence of characters is enclosed in `double quotation` marks. Example:

```java
String message = "Welcome to Java";
```

`String` is a predefined class in the Java library, just like the classes `System` and `Scanner`. It contains many useful methods manipulating characters. The following table lists the String methods for obtaining string length, for accessing characters in the string, for concatenating strings, for converting a string to upper or lowercases, and for trimming a string.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length()</td>
<td>Returns the number of characters in this string.</td>
</tr>
<tr>
<td>charAt(index)</td>
<td>Returns the character at the specified index from this string.</td>
</tr>
<tr>
<td>concat(s1)</td>
<td>Returns a new string that concatenates this string with string s1.</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>Returns a new string with all letters in uppercase.</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>Returns a new string with all letters in lowercase.</td>
</tr>
<tr>
<td>trim()</td>
<td>Returns a new string with whitespace characters trimmed on both sides.</td>
</tr>
</tbody>
</table>

Strings are `objects` in Java. The methods in the table above can only be invoked from a specific string instance. For this reason, these methods are called `instance methods`. A non-instance method is called a `static method`. A `static method` can be invoked using the `class name`, without using an object. All the methods defined in the `Math` class are static methods. They are not tied to a specific object instance.

Getting Characters from a String

The `str.charAt(index)` method can be used to retrieve a specific character in a string `s`, where the index is between `0` and `s.length() - 1`. For example, `message.charAt(0)` returns the character `W`, as shown in the next figure. Note that the index for the first character in the string is `0`.
Experiment #5: Strings & Text Files Input

Note that attempting to access characters in a string `s` out of bounds is a common programming error. To avoid it, make sure that you do not use an index beyond `s.length() - 1`.

Reading a String from the Console

To read a string from the console, invoke the `next()` method on a Scanner object. The `next()` method reads a string that ends with a whitespace character. You can use the `nextLine()` method to read an entire line of text. The `nextLine()` method reads a string that ends with the Enter key pressed.

Also, you can read characters from the console using the following statement:

```java
char c = input.nextLine().charAt(0);
```

Ex: Write a Java program to prompt the user to enter a line and output the following:

- How many characters in the line?
- The fifth character (if exists)
- All characters in uppercase
- All characters in lowercase

Solution:
Scanner input = new Scanner(System.in);
String s = input.nextLine();
s = s.trim();

int length = s.length();
System.out.println("Length is: " + length);

if (length >= 5)
    System.out.println("Fifth character is: " + s.charAt(4));
else
    System.out.println("The fifth character does not exist");

System.out.println("Uppercase: " + s.toUpperCase());
System.out.println("Lowercase: " + s.toLowerCase());

Comparing Strings

The String class contains the following methods for comparing two strings.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals(s1)</td>
<td>Returns true if this string is equal to string s1.</td>
</tr>
<tr>
<td>equalsIgnoreCase(s1)</td>
<td>Returns true if this string is equal to string s1; it is case insensitive.</td>
</tr>
<tr>
<td>compareTo(s1)</td>
<td>Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.</td>
</tr>
<tr>
<td>compareToIgnoreCase(s1)</td>
<td>Same as compareTo except that the comparison is case insensitive.</td>
</tr>
<tr>
<td>startsWith(prefix)</td>
<td>Returns true if this string starts with the specified prefix.</td>
</tr>
<tr>
<td>endsWith(suffix)</td>
<td>Returns true if this string ends with the specified suffix.</td>
</tr>
<tr>
<td>contains(s1)</td>
<td>Returns true if s1 is a substring in this string.</td>
</tr>
</tbody>
</table>

Obtaining Substrings

You can obtain a substring from a string using the substring method. For example:

String message = "Welcome to Java"
String message = message.substring(0, 11) + "HTML";

The string message now becomes "Welcome to HTML"

Note that the last character (11) is excluded.
Finding a Character or a Substring in a String

The `String` class provides several versions of "indexOf" and "lastIndexOf" methods to find a character or a substring in a string, as shown in the following table:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index(ch)</td>
<td>Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>indexOf(ch, fromIndex)</td>
<td>Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>indexOf(s)</td>
<td>Returns the index of the first occurrence of string s in this string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>indexOf(s, fromIndex)</td>
<td>Returns the index of the first occurrence of string s from fromIndex in this string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>lastIndexOf(ch)</td>
<td>Returns the index of the last occurrence of ch in the string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>lastIndexOf(ch, fromIndex)</td>
<td>Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched.</td>
</tr>
<tr>
<td>lastIndexOf(s)</td>
<td>Returns the index of the last occurrence of string s. Returns -1 if not matched.</td>
</tr>
<tr>
<td>lastIndexOf(s, fromIndex)</td>
<td>Returns the index of the last occurrence of string s from fromIndex. Returns -1 if not matched.</td>
</tr>
</tbody>
</table>

Note that all the previous methods return -1 if not matched.

Examples:

"Welcome to Java".indexOf('W') returns 0
"Welcome to Java".indexOf('o') returns 4
"Welcome to Java".indexOf('o', 5) returns 9
"Welcome to Java".indexOf("come") returns 3
"Welcome to Java".indexOf("Java", 5) returns 11
"Welcome to Java".indexOf("java", 5) returns -1
"Welcome to Java".lastIndexOf('W') returns 0
"Welcome to Java".lastIndexOf('o') returns 9
"Welcome to Java".lastIndexOf('o', 5) returns 4
"Welcome to Java".lastIndexOf("come") returns 3
"Welcome to Java".lastIndexOf("Java", 5) returns -1.
"Welcome to Java".lastIndexOf("Java") returns 11
Ex: Write a Java program to prompt the user to enter his full name as “FirstName LastName”, and the program prints each word in a line.

Solution:

```java
Scanner input = new Scanner(System.in);
String name = input.nextLine();
int sp = name.indexOf(" ");
String firstName = name.substring(0, sp);
String lastName = name.substring(sp + 1);
System.out.println("First Name is: " + firstName);
System.out.println("Last Name is: " + lastName);
```

Other Solution:

```java
Scanner input = new Scanner(System.in);
String firstName = input.next();
String lastName = input.next();
System.out.println("First Name is: " + firstName);
System.out.println("Last Name is: " + lastName);
```

Conversion between Strings and Numbers

To convert a string into an int value, use the `Integer.parseInt` method, as follows:

```java
int intValue = Integer.parseInt(intString);
```

To convert a string into a double value, use the `Double.parseDouble` method, as follows:

```java
double doubleValue = Double.parseDouble(doubleString);
```

You can convert a number into a string, simply use the string concatenating operator as follows:

```java
String s = number + "";
```

Formatting Console Output

You can use the `System.out.printf` method to display formatted output on the console.
Experiment #5: Strings & Text Files Input

For example, in the following code the interest amount is currency and it should be displayed in two digits after the decimal point.

```java
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.println("Interest is $" + interest);
```

| Interest is $16.404674 |

So, we use the method `printf` with the format specifier `%5.2f`

```java
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.printf("Interest is $%5.2f", interest);
```

| Interest is $16.40 |

The 5 is the number of digits (including decimal point), and 2 is the number of digits after the decimal point. f is used to tell the formatter that the string is of type `float`.

The syntax to invoke this method is

`System.out.printf(format, item1, item2, ..., itemk)`

where `format` is a string that may consist of substrings and format specifiers.

Reading Input from Text File

The simplest mechanism for reading an external text file is to use `Scanner` Class. You already know to use Scanner for reading console input. Instead of `System.in`, you first connect the file with a `File` object (defined in `java.io` package), then use that File object to construct a Scanner object.

```java
File f = new File("input.txt");
Scanner input = new Scanner(f);
```

This Scanner object (`input`) reads text from the file "input.txt." You can use the Scanner methods, such as `next()`, `nextInt()`, `nextLine()`, `nextDouble()`, to read data from the input file.
input.next(); // read a string (one word only) delimited by a 
// white-space (space, tab, newline)
input.nextInt(); // read an integer
input.nextDouble(); // read a double
input.nextLine(); // read a string (the whole line)

Note that the file “input.txt” must be in the main directory of the project, otherwise, you should specify the path of it.

Ex: Write a program to reads student’s information (full name, id, email, and mobile) from a file, and prints them to the console.

Solution:

Scanner input = new Scanner(new File("input.txt"));
String name = input.nextLine();
int id = input.nextInt();
String email = input.next();
int mobile = input.nextInt();
System.out.println(name);
System.out.println(id);
System.out.println(email);
System.out.println(mobile);
Lab Work

Ex1: Write a program that reads an English word and checks if it contains vowels (a, e, i, o, u). If so, output “Contains vowels”, else, output “No vowels found”.

Solution:

```java
String word = input.nextLine().toLowerCase();
if (word.contains("a") || word.contains("e")
   || word.contains("i") || word.contains("o")
   || word.contains("u")) {
    System.out.println("Contains vowels");
} else {
    System.out.println("No vowels found");
}
```

Ex2: Write a program to prompt the user to enter a simple equation with one operator (*, /, +, -, %), and two operands (decimal numbers) (Ex. “5 * 6.3”), and outputs the result of the expression. Note that there can be more than one white space between operator and operands.

Solution:

```java
Scanner input = new Scanner(System.in);
String s = input.nextLine().trim();
double d1 = Double.parseDouble(s.substring(0, s.indexOf(' ')));
double d2 = Double.parseDouble(s.substring(s.lastIndexOf(' ') + 1));
String op = s.substring(s.indexOf(' '), s.lastIndexOf(' ')).trim();
double result = 0;
switch (op) {
    case "*":
        result = d1 * d2;
        break;
    case "/":
        result = d1 / d2;
        break;
    case "+":
        result = d1 + d2;
        break;
    case "-":
        result = d1 - d2;
        break;
    case "%":
        result = d1 % d2;
}
System.out.println("Result: " + result);
```
Other Solution:

Scanner input = new Scanner(System.in);
\textbf{double} d1 = Double.parseDouble(input.next());
String op = input.next();
\textbf{double} d2 = Double.parseDouble(input.next());
\textbf{double} result = 0;
switch (op) {
    case "+":
        result = d1 + d2;
        break;
    case "/":
        result = d1 / d2;
        break;
    case "-":
        result = d1 - d2;
        break;
    case "%":
        result = d1 % d2;
    case "*":
        result = d1 * d2;
        break;
    case ":
        result = d1 / d2;
        break;
    case "-":
        result = d1 - d2;
        break;
    case "%":
        result = d1 % d2;
    case "+":
        result = d1 + d2;
        break;
    case "/":
        result = d1 / d2;
        break;
    case "-":
        result = d1 - d2;
        break;
    case "%":
        result = d1 % d2;
}
System.out.println("Result: "+ result);
Homework

1. Write an application that compares two strings input by the user. Output whether the first string is less than, equal to or greater than the second string.

2. Write an application that inputs a line of text and outputs the text twice. Once in all uppercase letters and once in all lowercase letters. Print the length of the text. The first character of the text. (don't forget to trim the text before using it).

3. Write a program that prompts the user to enter two strings and prints whether the second string is a substring of the first string.

4. Write a program to reads a logic equation from a file, and outputs the result of the equation to the console. The file should contain one equation with one operator (and, or, not). The operands must be (True, False).

Here is a sample file:

```
input.txt - Notepad

File Edit Format View Help

x = True and False

Ln 2, Col 1
```

And here is the console output:

```
x = false

Process finished with exit code 0
```

Good Luck