Object-Oriented Development (CIS1056-N)
Worksheet 03: Strings

Before You Start

Remember: You are not expected to complete the entire brief within the allotted two hours, but to make a start and continue outside of the class.

Ensure you have completed all assessment tasks from Worksheet 2 and are comfortable with conditions before beginning this worksheet. Attempt to complete this set of tasks before your next session. Any issues seek help from your tutors.

Introduction

So far, we have only dealt with Java's primitive types. The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are **immutable**; their values cannot be changed after they are created. String operations such as concatenate create a new String rather than modifying the existing one. String buffers support mutable strings (not covered in this worksheet).

Strings can be constructed by:

- Using the new operator.
- Directly assigning a string literal to a String Object Reference variable.

For example:

```java
String firstName = new String("James"); // Explicit construction
String lastName = "Mead";               // Implicit construction
```

The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings.

Key String Methods

The following methods are based on the Oracle String class documentation available at:

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

`charAt()`

Returns the character at the specified index (first char is at index position zero).

- **Syntax:**
```
char charAt(int index)
```
- **Example:**
```
String sample = "peter piper picked a peck";
char firstE = sample.charAt(1); // returns 'e'
```

`concat()`

This creates another String by joining the current String to the String given as the argument.

- **Syntax:**
```
String concat(String str)
```
- **Example:**
```
String first = "Java is ";
String second = "great!";
String third = first.concat(second);
```

Neither the calling String, first, nor the argument are changed. A similar result is obtained using the + operator:

```
String third = first + second;
```

`equals()`

Compares this string to another string object and returns true if both contain the same sequence of characters.

- **Syntax:**
```
boolean equals(String anotherString)
```
- **Example:**
```
String example = "BOB";
boolean b = example.equals("BOB");
System.out.println(b);
```

`equalsIgnoreCase()`

Compares this string to another string ignoring case of letters.

- **Syntax:**
```
boolean equals(String anotherString)
```
- **Example:**
```
String example = "BOB";
boolean b = example.equalsIgnoreCase("bob");
System.out.println(b);
```

`indexOf()`

This method returns the position (index) of the specified char argument. The index is counted from zero at the first character. If the character in the argument is not found, the method returns -1.

- **Syntax:**
```
int indexOf(char character)
```

- **Example:**
```
String fullName = "James Fairbairn";
int a = fullName.indexOf('F');  // returns 6
int b = fullName.indexOf('z');  // returns -1
```

There is another similar method: `int lastIndexOf(char character)`

`indexOf()` With 2 Parameters

A second version (overloaded) of the method has 2 parameters. This version searches the calling string starting from the position of the second argument.

- **Syntax:**
```
int indexOf(char character, int fromIndex)
```

- **Example:**
```
String fullName = "James Fairbairn";
System.out.println(fullName.indexOf('a', 3));
```

`length()`

Returns the length of this string. Spaces count too!

- **Syntax:**
```
int length()
```

- **Example:**
```
int result;
String name = "James Fairbairn";
result = name.length();

// Simpler use would be
System.out.println("Length: " + name.length());
```

`toLowerCase()`

This returns a new String turning any uppercase characters in the calling String to lowercase. All other characters remain the same. The state of the calling String object is not modified – you must assign the return value.

- **Syntax:**
```
String toLowerCase(char character)
```

- **Example:**
```
String capitalName = "BOB";
String smallerName = capitalName.toLowerCase();
System.out.println(smallerName); // bob
System.out.println(capitalName); // BOB (is
unchanged)
```

`toUpperCase()`

This returns a new String turning any uppercase characters in the calling String to lowercase. All other characters remain the same. The state of the calling String object is not modified – you must assign the return value.

- **Syntax:** `String toUpperCase(char character)`
- **Example:**

```
String smallerName = "bob";
String capitalName = capitalName.toUpperCase();
System.out.println(smallerName); // bob (is
unchanged)
System.out.println(capitalName); // BOB
```

`startsWith() and endsWith()`

Checks (tests) if this string starts with the specified prefix or ends with specified suffix. Both take a String argument and return a Boolean (true for a match).

- **Syntax:**

```
boolean startsWith(String prefix)
boolean endsWith(String suffix)
```

- **Example:**

```
String fullName = "James Fairbairn";
boolean a = fullName.startsWith("Jam");
boolean b = fullName.endsWith("bairn");
```

`substring()`

This extracts a chunk (substring) from a String. Returns the substring made from the character number given as the first argument, up to, but excluding the character as the second argument. Note the first character in a String is 0.

- **Syntax:** `String substring(int beginIndex, int endIndex)`
- **Example:**

```
String fullName = "Steven James Mead";
String middleName = fullName.substring(7, 12);
```

1. Paper Exercise Expected Outputs

Paper exercise so **no programming** (initially). What output would you expect from the following?

Expected Output One

```
String redHerring = "Baa, baa, black sheep";
int a = redHerring.length();
```

Expected Output Two

```
String festival = "Christmas";
System.out.println(festival.indexOf('s'));
System.out.println(festival.lastIndexOf('s'));
```

Expected Output Three

```
String fastfood = "pepperoni and mushroom pizza";
System.out.println(fastfood.substring(0, 6));
System.out.println(fastfood.substring(14,18));
```

Expected Output Four

```
String first = "Java is great!";
String second = "Java is ";
String third = "great!";

second = second.concat(third);
boolean result = first.equals(second);
```

Name the Method

Consider a String with a first, middle and last name e.g. “Kristin Scott Thomas”.

How can you use String methods to extract the middle name from the String?

2. Check Length

Write a program to input a string from the user and validate it to ensure the length is between 5 and 10 (inclusive).

Once you have completed your program, prepare a simple test plan to check the program meets the requirements.

3. Input String

Write a program to do the following:

a) Prompt the user to input 2 words (strings) and then print out the length of each string.

   To read a String from the keyboard use the Scanner method `next()`

   ```
   Scanner input = new Scanner(System.in);
   String name = input.next();
   ```

   This will stop reading at the first whitespace character.

   To read-in a line including spaces use the Scanner method `nextLine()`

   ```
   String fullName = input.nextLine();
   ```

b) Print the names in order of their length, so the shorter name is output first. Test this.

c) If both names are the same length, output a message stating that they are the same length.

d) Print one name all in uppercase characters and the other name all in lowercase characters.

e) Does this affect the content of the names? Check by printing out the names again at the end.

Once you have completed your program prepare a simple test plan to check the program meets the requirements.

4. MiddleNames

Write a program called **MiddleNames**, which asks the user to enter a full name (String) and displays the middle name. For example:

```
Please enter your full name: Marcy Ann Holby
The middle name is: Ann
```

The following steps could be used to extract the middle name:

a) Find the first space in the String.
   Use the `indexOf()` method to find the first space and store the index of the first space character in a variable of type int called `firstSpace`.

b) Find the last space in the String.
   Use the `lastIndexOf()` method. Store the index of the last space character in a variable of type int called `lastSpace`.

c) Extract the middle name.
   Use the `substring()` method to get all of the characters between the first space character and the last space character. Assign the return value of the `substring()` method to a new String with the identifier `middleName` and print this.

Once you have completed your program prepare a simple test plan to check the program meets the requirements.

5. Validate Size Input

Write a program that asks the user to enter a size and validates it:

- Appropriate inputs are Large, Small, L or S.
- Lower and uppercase version should be accepted.
- For valid inputs, display size as Large or Small.
- For invalid inputs, display an appropriate message.

6. Staff Number

Write a program that asks the user to enter a staff number and validates it. Staff numbers start with a letter and are followed by two digits; for example: D65

7. File extension

a) Write a program that extracts the extension from a file name. For example, an input of ModuleGuide.docx would display “The file extension is: docx”.

   Your program must be able to work with filenames that do not have an extension.

b) Extend your program to also work with Linux pathnames – for example:

Filename/Pathname	**Output**
/docs/ModuleGuide.pdf	The extension is: pdf
editor.java	The extension is: java
/docs/guide.v1/readme	The file has no extension.

8. Contents Page

Create a program that asks the user to enter:

- Page width in characters – no more than *50* (integer) – e.g. **`40`**
- Chapter Title (String) – e.g. **`Strings in Java`**
- Page Number (integer) – e.g. **`127`**

Once the values are input, it outputs the chapter title and page number formatted to fit the page width. The above example would produce the following **40** characters:

```
Strings in Java..................... 127
```

For this exercise you are **NOT** allowed to use loops.

Assume the maximum page number is 9999 and use an if-else multi-way selection to determine the size in characters of the page number (e.g. 127 is 3 characters long).

There are more efficient methods of determining the number of characters in an integer, if you have programming experience explore other methods of doing this.

Extension Activity: Validate IPv4

For students with Programming Experience

Write a program to validate an IP (IPv4) address - decimal numbers only (eg: 192.168.10.1). Each number in the list can be between 0 and 255 (inclusive). For a description of IPv4 address refer to:

https://computer.howstuffworks.com/internet/basics/what-is-an-ip-address.htm

Document History

Revision 0 (9-Oct-22): This is the initial version of the 2022/23 exercise.