
ORACLE
ACADEMY

Java Programming

3-1
String Processing

ORACLE ACADEMY

Objectives

This lesson covers the following topics:

- Read, search, and parse Strings
- Use StringBuilder to create Strings

Strings

- The String object can be interpreted as a group of characters in a single memory location.
- Strings, like arrays, begin their index at 0 and end their index at StringName.length()-1.
- There are many different ways of approaching String manipulation.

Using a FOR Loop

- One way to manipulate Strings is to use a FOR loop.
- This code segment initializes a String and increments through its characters, printing each one to the console.

```
String str = “Sample String”;

for(int index=0;index<str.length();index++){
    System.out.print(str.charAt(index));}
//endfor
```

Benefits to Using a FOR Loop

- Using the FOR loop method of incrementing through a String is beneficial if you desire to:
  - Search for a specific character or String inside of the String.
  - Read the String backwards (from last element to first element).
  - Parse the String.

Print String to Console

- An easier way to print a String to the console does not involve incrementing through the String.
- This code is shown below.

```
System.out.print(str);
```

Common String Methods

- A few other common String methods are:

String Method	Description
length()	Returns the number of characters in the String.
charAt(int i)	Returns the character at index i.
substring(int start)	Returns part of the String from index start to the end of the String.
substring(int start, int end)	Returns part of the String from index start to index end, but does not include the character at index end.
replace(char oldC, char newC)	Returns a String where all occurrences of character oldC have been replaced with newC.

Searching and Strings

- There are a few different ways to search for a specific character or String inside of the String.
- The first is a for loop, which can be altered to search, count, or replace characters or Substrings contained in Strings.

Searching and Strings Example

- The code below uses a for loop to count the number of spaces found in the String.

```
 String str = “Searching for spaces”;
int count=0;

for(int i=0;i<str.length();i++){
   if(str.charAt(i)==' ')
      count++;
   //endif
}//endfor
```

Since the index of a String begins at 0, we must begin searching for a ' ' at index 0.

Search through the String until the index reaches the last element of the String, which is at index.strlength()-1.
This means that i cannot be > or = to the str.length().
If it does exceed str.length()-1, an "index out of bounds" error will occur.

Calling Methods on the String

- Other ways to search for something in a String is by calling any of the following methods on the String.
- These methods are beneficial when working with programming problems that involve the manipulation of Strings.

String Method	Description
contains(CharSequence s)	Returns true if the String contains s.
indexOf(char ch)	Returns the index within this String of the first occurrence of the specified character and -1 if the character is not in the String.
indexOf(String str)	Returns the index within this String of the first occurrence of the specified Substring and -1 if the String does not contain the Substring str.

Reading Strings Backwards

- Typically a String is read from left to right.
- To read a String backwards, simply change the starting index and ending index of the FOR loop that increments through the String.

```
String str = “Read this backwards”;
String strBackwards = “”;

for(int i=str.length()-1; i>=0 ; i--){
   strBackwards+=str.substring(i,i+1);
}//endfor
```

Start the FOR loop at the last index of the array (which is str.length()-1), and decrease the index all the way through to the first index (which is 0).

Parsing a String

- Parsing means dividing a String into a set of Substrings.
- Typically, a sentence (stored as a String) is parsed by spaces to separate the words of the sentence rather than the whole sentence.
- This makes it easier to rearrange the words than if they were all together in one String.
- You may parse a String by any character or Substring.
- Below are two techniques for parsing a String:
  - For loop
  - Split

Steps to Parsing a String with a For Loop

- Increment through the for loop until you find the character or Substring where you wish to parse it.
- Store the parsed components.
- Update the String.
- Manipulate the parsed components as desired.

Steps to Parsing a String with a For Loop

```
import java.util.*;

public class StringManipulation {
   public static void main(String[] args) {
      //Variable declaration section
      String str = "Parse this String";
      ArrayList<String> words = new ArrayList<String>();

      while(str.length() > 0){
         for(int i=0; i<str.length(); i++){
            if(i==str.length()-1){
               words.add(str.substring(0));
               str = "";
               break;
            }
            else if(str.charAt(i)==' '){
               words.add(str.substring(0,i));
               str=str.substring(i+1);
               break;
            }//endif
         }//endfor
      }//endwhile
      for(String s : words)
         System.out.print(s + ' ');
      //endfor
   }//end of method main
}//end of class
```

Parsing a String: Split

- Split is a method inside the String class that parses a String at specified characters, or if unspecified, spaces.
- It returns an array of Strings that contains the Substrings (or words) that parsing the String gives.
- How to call split on a String:

```
String sentence = “This is my sentence”;
String[] words = sentence.split(“ ”);
//words will look like {This,is,my,sentence}
String[] tokens = sentence.split(“i”);
//tokens will look like {Th,s ,s my sentence}
```

Split a String by More than One Character

- It is also possible to split a String by more than one specified character if you use brackets [ ] around the characters.
- Here is an example:

Notice how the brackets are used to include i and e.

```
String sentence = “This is my sentence”;

String[] tokens = sentence.split(“[ie]”);
//tokens will look like {Th,s ,s my s,nt,nc}
//each token is separated by any occurrence of
//an i or any occurrence of an e.
```

StringBuilder

- StringBuilder is a class that represents a String-like object.
- It is made of a sequence of characters, like a String.
- The difference between String and StringBuilder objects is that:
  - StringBuilder includes methods that can modify the StringBuilder once it has been created by appending, removing, replacing, or inserting characters.
  - Once created, a String cannot be changed. It is replaced by a new String instead.

StringBuilder

- To create and display the value of a StringBuilder object the following code can be used:

```
package stringBuilderDemo;

public class StringBuilderDemo {

   public static void main(String[] args) {
      //create a new StringBuilder object named str
      StringBuilder str = new StringBuilder("Oracle");

      //display the value of str to screen
      System.out.println("string = " + str);
   }
}
```

StringBuilder Objects can be modified

- It is **not** possible to make modifications to a String.
- Methods used to “modify” a String actually create a new String in memory with the specified changes, they do not modify the old one.
- This is why StringBuilders are much faster to work with: They can be modified and do not require you to create a new String with each modification.

StringBuilder and String Shared Methods

- StringBuilder shares many of the same methods with String, including but not limited to:
  - charAt(int index)
  - indexOf(String str)
  - length()
  - substring(int start, int end)

```
//shared StringBuilder and String methods
System.out.println("The length of the text is: " + str.length());
System.out.println("The character at the beginning is: " + str.charAt(0));
System.out.println("The second character is: " + str.charAt(1));
System.out.println("The position of the start of the text \"acl\" is: " +
                                                        str.indexOf("acl"));
System.out.println("The following text is included within the String: " +
                                                        str.substring(1,4));
```

StringBuilder Methods

- StringBuilder also has some methods specific to its class, including the five below:

Method	Description
append(Type t)	Is compatible with any Java type or object, appends the String representation of the Type argument to the end of the sequence.
delete(int start, int end)	Removes the character sequence included in the Substring from start to end.
insert(int offset, Type t)	Is compatible with any Java type, inserts the String representation of Type argument into the sequence.
replace(int start, int end, String str)	Replaces the characters in a Substring of this sequence with characters in str.
reverse()	Causes this character sequence to be replaced by the reverse of the sequence.

StringBuilder Example

```
// reverse characters of the StringBuilder and prints it
System.out.println("reverse = " + str.reverse());

// reverse characters of the StringBuilder and prints it
System.out.println("reverse = " + str.reverse());

//add Characters to the end of the existing string
str.append(" Java Programming");
System.out.println("string = " + str);

//delete characters from the string by specifying the start and end
 position
str.delete(7, 12);
System.out.println("string = " + str);

//insert a new string into an existing string
str.insert(7, "Java ");
System.out.println("string = " + str);

//Replace an existing section of a string with another string
str.replace(7, 23, "String Processing");
System.out.println("string = " + str);
```

Methods to Search Using a StringBuilder

- Searching using a StringBuilder can be done using either of the below methods.

Method	Description
charAt(int index)	Returns the character at index.
indexOf(String str, int fromIndex)	Returns index of first occurrence of str.

```
/*checks for the first occurrence of the letter c after position 6 in
  the String */
System.out.println("The position of the first letter c after \"Oracle\"
 is: " + str.indexOf("c", 6));
```

StringBuilder versus String

- These are some of the important differences between a StringBuilder and a String object.

StringBuilder	String
Changeable	Immutable
Easier insertion, deletion, and replacement.	Easier concatenation.
Can be more difficult to use, especially when using regular expressions (introduced in the next lesson).	Visually simpler to use, similar to primitive types rather than objects.
Use when memory needs to be conserved.	Use with simpler programs where memory is not a concern.

Terminology

Key terms used in this lesson included:

- String
- String Methods
- String Searching
- Parsing a String
- Split
- StringBuilder

Summary

In this lesson, you should have learned how to:

- Read, search, and parse Strings
- Use StringBuilder to create Strings


ORACLE
ACADEMY