
ORACLE
Academy

Java Programming

4-1

String Processing

ORACLE
Academy

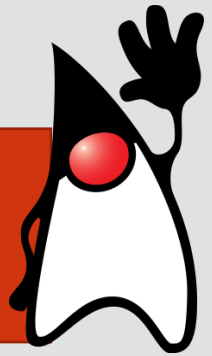
Objectives

- This lesson covers the following topics:
  - Read Strings
  - Search Strings
  - 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

A String is not a primitive data type but a class that we use as one. When created a String can hold a series of characters that can be accessed via the String's name.



Print String to Console

- Java provides an easy way of printing the contents of the String to the console
  - System.out.print() will print the contents of the String to the console and leave the cursor on the same line

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

  - System.out.println() will print the contents of the String to the console and place the cursor on the next line

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

Common String Methods

- These String methods are accessed by using dot notation with the string name – strname.length();

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

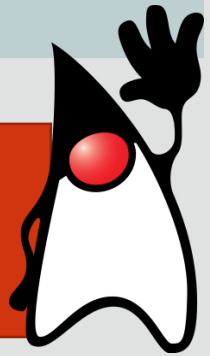
Using a FOR Loop

- 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 on a separate line

```
String str = "Sample String";

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

As a String starts it's characters at position zero it is very similar to using a for loop on an array when accessing the contents of the String.



String Example

1. Create a project named stringexample
2. Create a StringExample class that includes the following methods and declares a String field

```
public class StringExample {
   public static void main(String[] args) {
      String str = "Sample String";

      displayString(str);
   }//end method main

   static void displayString(String str)
   {
   }//end method displayString
}//end class StringExample
```

String Example

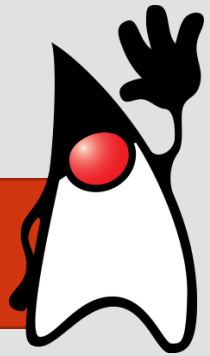
3. In the displayString() method include a console output statement that will display the String to the console and leave the cursor on the next line
4. Also in the displayString() method use a for loop to display each character on a separate line

```
static void displayString(String str)
{
   //printing the String as it is
   System.out.println(str);
   //printing each character on a separate line
   for(int index=0;index<str.length();index++){
      System.out.println(str.charAt(index));}
   //endfor
}//end method displayString
```

Benefits to Using a FOR Loop

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

The built in method length() allows you to access the data up to the end of the String without going beyond it's boundaries.



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
- A for loop makes it easy to go backwards as well as forwards

```
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**).

String Example

5. In the StringExample class include a method named displayStringBackwards
6. Call the method from main

```
public class StringExample {
   public static void main(String[] args) {
   String str = "Sample String";

   displayString(str);
   displayStringBackward(str);

   }//end method main

   static void displayStringBackward(String str)
   {
   }//end method displayStringBackward

   static void displayString(String str)
```

ORACLE Academy

String Example

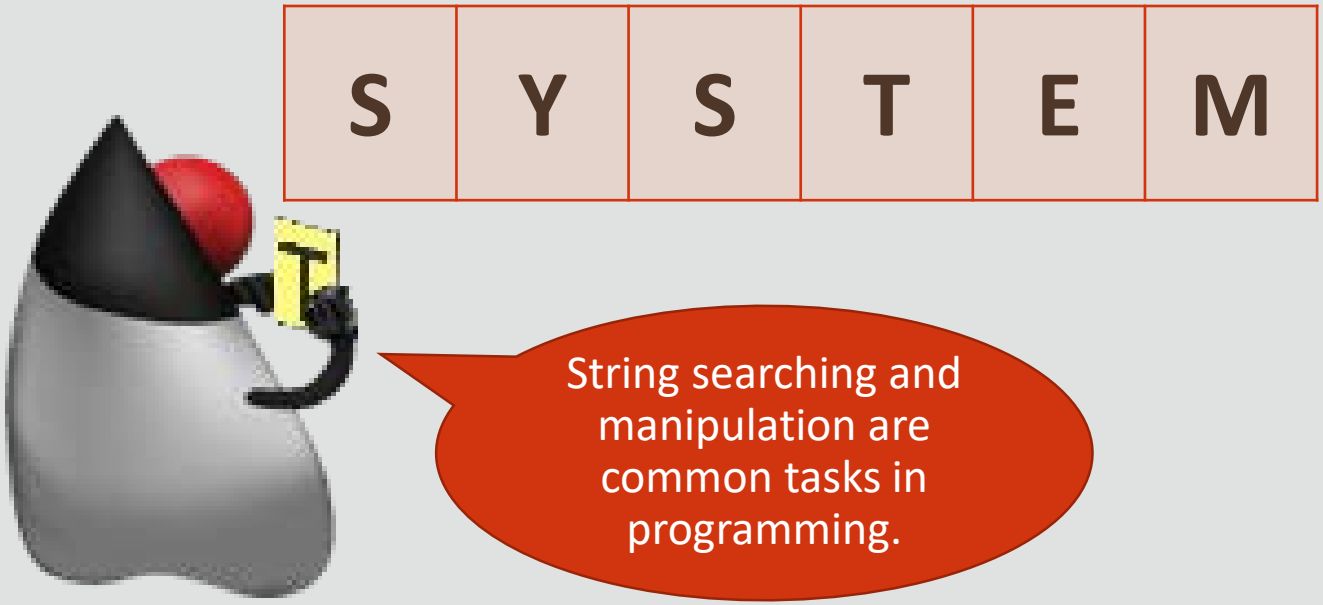
7. In the displayStringBackward() method, use a for loop to display the String backwards on a single line

```
static void displayStringBackward(String str)
{  //create a local String variable
   String strBackwards = "";

   //read the array backwards and store the characters in
strBackwards
   for(int i=str.length()-1; i>=0 ; i--){
      strBackwards+=str.substring(i,i+1);
   }//endfor
   System.out.println(strBackwards);
}//end method displayStringBackward
```

Searching and Strings

- There are a few different ways to search for a specific character or String inside of an existing 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 for loop can be used for many purposes as it allows you to work through each character of the String easily
- This code counts the number of spaces 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
```

Search through the String until the index reaches the last element, which is 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.

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

String Example

8. In the StringExample class include a method named searchString
9. Call the method from main

```
public class StringExample {
   public static void main(String[] args) {
      String str = "Sample String";

      displayString(str);
      displayStringBackward(str);
      searchString(str, 'S'); //Search for an uppercase S
   }//end method main

   static void searchString(String str, char c) {
   }//end method searchString

   static void displayStringBackward(String str)
   {
```

String Example

10. In the searchString() method use a for loop to count how many times the given character appears in the String
11. Display the result to screen

```
static void searchString(String str, char c) {
   int count=0;

   for(int i=0; i<str.length(); i++){
      if(str.charAt(i) == c)
         count++;
      //endif
   }//endfor
   System.out.println("The character " + c
                      + " appears in the text "
                      + count + " times.");
}//end method searchString
```

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

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

It is an extremely common task to split a String into multiple substrings and to put them back together in a different order. Creating a username from a users first initial and surname is an example of this.

Steps to Parsing a String with a For Loop

- First a String is required that can be parsed

```
String str = "Parse this String";
```

- A data structure must be created to store the individual sections of the parsed String, as the number of substrings is unknown an ArrayList is ideal

```
ArrayList<String> words = new ArrayList<String>();
```

- A while loop is used to work through the String until all the substrings have been removed

```
while(str.length() > 0){
  //parsing code goes here
}//endwhile
```

Steps to Parsing a String with a For Loop

- Within the while, a nested for loop is required that will search through the String until the parse character or the end of the String is found

```
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
```

When the end of the String or the space character is found, a substring is created and then added to the words ArrayList.

The substring is removed from the original String str.

The break statement then returns control to the while loop.

Steps to Parsing a String with a For Loop

- The contents of the ArrayList can then be displayed to screen using an enhanced for loop

```
for(String s : words)
   System.out.print(s + ' ');
//endfor
```

- You could then use the ArrayList methods to:
  - Count the number of elements created in the parse
  - Sort the elements into their natural order
  - Display them in any order you choose

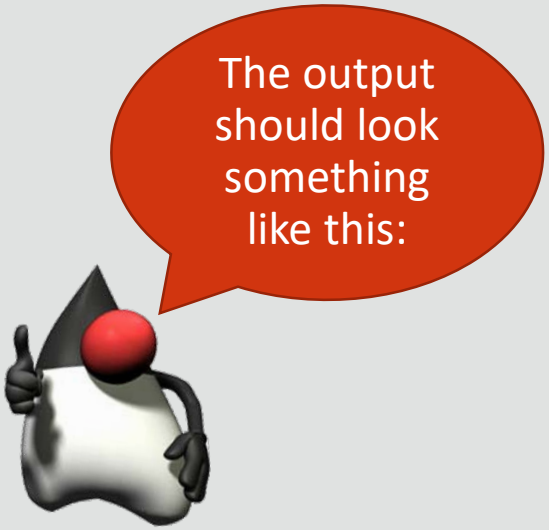
Quick Task: Create a class named StringManipulation that implements the information provided for parsing a String with a for loop!

ORACLE Academy

Parsing a String with a For Loop Solution

- The collection methods that you should have used were:

```
System.out.println("\nThere are " + words.size()
                   + " words in the original String");
Collections.sort(words);
System.out.println(words);
```




```
Parse this String
There are 3 words in the original String
[Parse, String, this]
```

ORACLE Academy

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}
for(String token : tokens)
   System.out.print(token + ", ");
//endfor
```

Parsing a String: Split (Using multiple characters)

- 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.
```

Quick Task: Create a class named StringSplit that implements the information provided for parsing a String using the split method.

ORACLE Academy

Parsing a String: Split Solution

- When all of the different split methods have been implemented the output should match the following:

```
This, is, my, sentence,

Th, s , s my sentence,

Th, s , s my s, nt, nc,
```

Test your application by changing the contents of the String and with different split values!

ORACLE Academy

Calling Methods on the String

- These methods are beneficial when working with 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.

It is often necessary to identify or find characters within a String when creating substring or usernames.

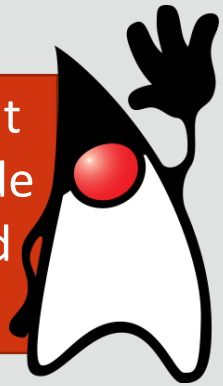
StringBuilder vs String

- 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 object once it has been created by appending, removing, replacing, or inserting characters
  - Once created, a String is immutable (cannot be changed) it is replaced by a new String instead

StringBuilder vs String

- 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

Most people don’t realize that you can’t actually change a String but that every change results in a new object being created. If your code requires a lot of changes to the contents of a String then you should use a StringBuilder instead.



StringBuilder vs String Example

1. To get a clearer understanding of the difference between StringBuilder and String objects, their hashCodes can be called
2. Create the following program that creates two fields:

```
package stringbuildervsstring;
public class StringVsBuilder {
   public static void main(String[] args) {
      //create a new StringBuilder object named str
      StringBuilder str = new StringBuilder("Test immutability");
      //create a new String object named str2
      String str2 = "Test immutability";
   }//end method main
}//end class StringVsBuilder
```

StringBuilder vs String Example

3. Display the identifying hashCodes for each object using the hashCode() method:

```
        //create a new StringBuilder object named str
        StringBuilder str = new StringBuilder("Test immutability");
        //create a new String object named str
        String str2 = "Test immutability";

        //Display the identifying hashCodes for each object
        System.out.println("StringBuilder: " + str.hashCode());
        System.out.println("String: " + str2.hashCode());

    }//end method main
}//end class StringVsBuilder
```

Both hashCodes will be displayed in the console!

StringBuilder vs String Example

4. Change the values held in both objects
5. Use an assign (=) operator for the String and the replace() method for the StringBuilder:

```
        //Display the identifying hashCodes for each object
        System.out.println("StringBuilder: " + str.hashCode());
        System.out.println("String: " + str2.hashCode());

        //change the values stored in each object
        str.replace(0, str.length(), "HelloWorld");
        str2="HelloWorld";

    }//end method main
}//end class StringVsBuilder
```

StringBuilder vs String Example

6. Display the identifying hashCodes for each object using the hashCode() method:

```
        System.out.println("String: " + str2.hashCode());

        //change the values stored in each object
        str.replace(0, str.length(), "HelloWorld");
        str2="HelloWorld";

        //Display the identifying hashCodes for each object
        System.out.println("StringBuilder: " + str.hashCode());
        System.out.println("String: " + str2.hashCode());

    }//end method main
}//end class StringVsBuilder
```

You can see in the output that the StringBuilder remains the same object (same hashCode) whereas the String is a completely new object (different hashCode).

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.

When code requires manipulation of Strings then the information in this table should be considered.

StringBuilder and String Shared Methods

- StringBuilder shares many of the same methods with String, including but not limited to:

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.
`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.

StringBuilder Example

- StringBuilder is a class that allows much more flexibility with the characters held within it than a String does

1. Write the following code to create and display the value of a StringBuilder object:

```
package stringbuilderdemo;
public class StringBuilderDemo {

   public static void main(String[] args) {
      //create a new StringBuilder object named str
      StringBuilder str = new StringBuilder("Learning Java
                                          with Oracle");
      //display the value of str to screen
      System.out.println("string = " + str);
   }//end method main
}//end class StringBuilderDemo
```

StringBuilder Example

2. Add the following code to the bottom of the StringBuilderDemo program to see them in operation

```
//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));
```

At this point you won't see any difference to what would have happened if a normal String had been used.

StringBuilder Methods

- StringBuilder has methods specific to its class:

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.

More methods can be found in the Java API

StringBuilder Example

3. Add the following code to see the reverse() method in operation, It is called twice to reset the order!

```
// 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());
```

4. Use the append() method to add text to the end of the existing value

```
//add Characters to the end of the existing string
str.append(" is fun");
System.out.println("Appended string = " + str);
```

- Output

```
reverse = elcarO htiw avaJ gninraeL
reverse = Learning Java with Oracle
Appended string = Learning Java with Oracle is fun
```

ORACLE Academy

StringBuilder Example

5. Use the delete() method by specifying the start and end position in the StringBuilder object of the characters that are to be removed

```
//delete characters by specifying the start and end position
str.delete(8, 13);
System.out.println("Deleted string = " + str);
```

6. Use the insert() method to add text at a specified point in the StringBuilder object

```
//insert a new string into an existing string
str.insert(8, " Java Programming");
System.out.println("Inserted string = " + str);
```

- Output

```
Deleted string = Learning with Oracle is fun
Inserted string = Learning Java Programming with Oracle is fun
```

StringBuilder Example

7. Use the replace() method to replace the characters in a given substring with a new sequence

```
//Replace an existing section of a string with another string
str.replace(13, 25, " String Processing");
System.out.println("Replaced string = " + str);
```

Output:

```
string = Learning Java with Oracle
The length of the text is: 25
The character at the beginning is: L
The second character is: e
The position of the start of the text "acl" is: 21
The following text is included within the String: ear
reverse = elcarO htiw avaJ gninraeL
reverse = Learning Java with Oracle
Appended string = Learning Java with Oracle is fun
Deleted string = Learning with Oracle is fun
Inserted string = Learning Java Programming with Oracle is fun
Replaced string = Learning Java String Processing with Oracle is fun
```

Methods to Search Using a StringBuilder

- Searching using a StringBuilder can be done using the StringBuilder methods

Method	Description
`charAt(int index)`	Returns the character at index.
`indexOf(String str)`	Returns index of first occurrence of str or minus one (-1) if the String is not found.
`indexOf(String str, int fromIndex)`	Returns index of first occurrence of str or minus one (-1) if the String is not found.

It is important to get to know the available methods and how to apply them. Using the proposals drop down menu in Eclipse (appears after you include dot notation) can help you.

StringBuilder Example

8. Use the indexOf() method to identify the first occurrence of a character in the StringBuilder

```
//display the position of a given String value
System.out.println(str.indexOf("a"));
System.out.println(str.indexOf("Q"));
```

9. Use the other indexOf() method to identify the first occurrence of a letter “a” after Learning

```
//display the position of a given String value
//after a specified location
System.out.println(str.indexOf("a", 7));
```

- Output

```
2
-1
10
```

ORACLE Academy

Terminology

- Key terms used in this lesson included:
  - Parsing a String
  - Split
  - String
  - StringBuilder
  - String Methods
  - String Searching

Summary

- In this lesson, you should have learned how to:
  - Read Strings
  - Search Strings
  - Parse Strings
  - Use StringBuilder to create Strings

ORACLE Academy


ORACLE
Academy