AP Computer Science A: *Developing Algorithms Using Strings*

CodeHS

This Lesson Will Cover:

- Identifying and modifying standard string processing algorithms
- Creating string methods

CodeHS

Recap: String Methods

Some methods use indices!

Method	Use
`name.length()`	Returns the number of characters in a `String` object
`name.substring(2, 6)`	Returns the substring beginning at index 2 and ending at index `5`.
`name.indexOf("d")`	Returns the index of the first occurrence of d; returns `-1` if not found.
`name.equals("Karel")`	Returns `true` if name is equal to `Karel`; returns `false` otherwise
`name.compareTo("Karel")`	Returns a value `< 0` if name is less than `Karel`; returns `zero` if name is equal to `Karel`; returns a value `> 0` if name is greater than `Karel`.

Recap: String Methods

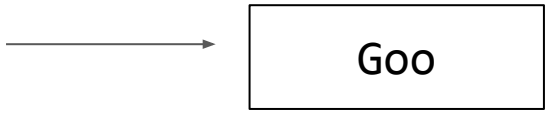
Some methods use indices!

Method	Use
`name.length()`	Returns the number of characters in a `String` object
`name.substring(2, 6)`	Returns the substring beginning at index 2 and ending at index `5`.
`name.indexOf("d")`	Returns the index of the first occurrence of d; returns `-1` if not found.
`name.equals("Karel")`	Returns `true` if name is equal to `Karel`; returns `false` otherwise
`name.compareTo("Karel")`	Returns a value `< 0` if name is less than `Karel`; returns `zero` if name is equal to `Karel`; returns a value `> 0` if name is greater than `Karel`.

Returning Substrings

.substring(int from, int to) returns a String starting at index `from` and ending at index `to - 1`.

```
String str = new String("Good day!");
//Return substring from index 0 through index 2
String firstWord = str.substring(0, 3);
System.out.println(firstWord);
```



Recap: substring()

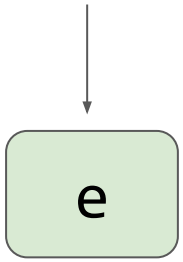
What if we called substring() as follows? Can you tell which part of the string would be selected and outputted?

```
String str = new String("CodeHS");
// use substring() to extract a substring
System.out.print("The extracted substring is : ");
System.out.println(str.substring(3,4));
```

Recap: substring()

What if we called substring() as follows? Can you tell which part of the string would be selected and outputted?

```
String str = new String("CodeHS");
// use substring() to extract a substring
System.out.print("The extracted substring is : ");
System.out.println(str.substring(3,4));
```



Recap: substring()

Remember, strings are sequences of characters:

```
String str = “CodeHS”;
```

Each character in a String has an **index.** The index is the position of the character in the String.

0	1	2	3	4	5
C	o	d	e	H	S

Recap: substring()

Strings are sequences of characters:

```
String str = “CodeHS”;
```

This call uses the index 3 inclusively and the index 4 exclusively, according to the substring implementation.

0	1	2	3	4	5
C	o	d	e	H	S

```
System.out.println(str.substring(3,4));
```

Recap: substring()

Strings are sequences of characters:

```
String str = “CodeHS”;
```

inclusive **exclusive**

0	1	2	3	4	5
C	o	d	e	H	S

```
System.out.println(str.substring(3,4));
```

Recap: substring()

Strings are sequences of characters:

```
String str = “CodeHS”;
```

inclusive **exclusive**

0	1	2	3	4	5
C	o	d	e	H	S

```
System.out.println(str.substring(3,4));
```

Recap: substring()

Strings are sequences of characters:

```
String str = “CodeHS”;
```

Since 3 is the only included index in this call to substring, the character “e” is the only value returned.

0	1	2	3	4	5
C	o	d	e	H	S

```
System.out.println(str.substring(3,4));
```

String Traversals

The substring() method is particularly useful when attempting to **traverse** Strings.

CodeHS

String Traversal

Traversing is the process of going through a String one character at a time, often using loops!

CodeHS

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
   System.out.println(print.substring(i, i+1));
}
```

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{

}
```

Initialize the variable i to 0

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{

}
```

We want to increase i by 1 to access each character individually

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{

}
```

0	1	2	3	4	5	6	7	8
P	r	i	n	t		M	e	!

print.length() = 9, but Strings start at index 0. We put i < print.length() so i only goes up to the last index!

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
   System.out.println(print.substring(i, i+1));
}
```

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
    System.out.println(print.substring(i, i+1));
}
```

Isolates each character of the String print

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
    System.out.println(print.substring(i, i+1));
}
```

inclusive **exclusive**

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
   System.out.println(print.substring(i, i+1));
}
```

```
i = 0
```

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
   System.out.println(print.substring(i, i+1));
}
```

```
i = 0
```

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
    System.out.println(print.substring(i, i+1));
}
```

```
i = 0
```

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
    System.out.println(print.substring(0, 1));
}
```

```
i = 0
```

```
P
```

0	1	2	3	4	5	6	7	8
P	r	i	n	t		M	e	!

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
   System.out.println(print.substring(i, i+1));
}
```

```
i = 1
```

```
P
```

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
   System.out.println(print.substring(i, i+1));
}
```

```
i = 1
```

```
P
```

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
    System.out.println(print.substring(i, i+1));
}
```

```
i = 1
```

```
P
```

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
    System.out.println(print.substring(1, 2));
}
```

```
i = 1
```

```
P
r
```

0	1	2	3	4	5	6	7	8
P	r	i	n	t		M	e	!

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
    System.out.println(print.substring(2, 3));
}
```

```
i = 2
```

```
P
r
i
```

0	1	2	3	4	5	6	7	8
P	r	i	n	t		M	e	!

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
    System.out.println(print.substring(8, 9));
}
```

i = 8

```
P
r
i
n
t

M
e
!
```

0	1	2	3	4	5	6	7	8
P	r	i	n	t		M	e	!

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i < print.length(); i++)
{
   System.out.println(print.substring(8, 9));
}
```

i = 9

```
P
r
i
n
t

M
e
!
```

0	1	2	3	4	5	6	7	8
P	r	i	n	t		M	e	!

String Traversal

We can traverse Strings using loops:

```
//Prints each character in a String on a new line
String print = “Print Me!”;

for(int i = 0; i <= print.length(); i++)
{
    System.out.println(print.substring(9, 10));
}
```

i = 9

Errors:

MyProgram.java: Line 9: The index 10 appears to be out of bounds for this string.

4	5	6	7	8	
t		M	e	!	

General String Traversal

The general formula for accessing individual characters in a String using a for loop is:

```
for(int i = 0; i < string.length(); i++)
{
    String character = string.substring(i, i+1);
}
```

charAt()

We can also use `charAt(int index)` to access individual characters in a String.

Strings are Sequences of Characters

We can use the method charAt(int index) to access specific characters in a String:

```
String str = “hello”;

System.out.println(str.charAt(1));
```

1
e

General String Traversal

The general formula for accessing individual characters in a String using a for loop is:

```
for(int i = 0; i < string.length(); i++)
{
    char character = string.charAt(i);
}
```

charAt returns a char value, not a String!

String Traversal Algorithms

Using String traversals, we create useful methods algorithms!

CodeHS

String Traversal Algorithms

We can determine how many characters are Uppercase:

```
public int numUpperCase(String string)
{
    int counter = 0;
    for(int i = 0; i < string.length(); i++)
    {
        char character = string.charAt(i);
        if(isUpperCase(character))
        {
            counter++;
        }
    }
    return counter;
}
```

String Traversal Algorithms

We can determine how many characters are Uppercase:

```
public int numUpperCase(String string)
{
    int counter = 0;
    for(int i = 0; i < string.length(); i++)
    {
        char character = string.charAt(i);
        if(isUpperCase(character))
        {
            counter++;
        }
    }
    return counter;
}
```

Iterates through each character

String Traversal Algorithms

We can determine how many characters are Uppercase:

```
public int numUpperCase(String string)
{
    int counter = 0;
    for(int i = 0; i < string.length(); i++)
    {
        char character = string.charAt(i);
        if(isUpperCase(character))
        {
            counter++;
        }
    }
    return counter;
}
```

If the character is Uppercase, add 1 to counter

String Traversal Algorithms

We can replace substrings with other substrings:

```
public String replace(String string, String remove, String add)
{
    String newString = “”;
    for(int i = 0; i < string.length(); i++)
    {
        String character = string.substring(i, i+1);
        if(character.equals(remove))
        {
            newString += add;
        }
        else
        {
            newString += character;
        }
    }
    return newString;
}
```

String Traversal Algorithms

We can replace substrings with other substrings:

```
public String replace(String string, String remove, String add)
{
    String newString = “”;
    for(int i = 0; i < string.length(); i++)
    {
        String character = string.substring(i, i+1);
        if(character.equals(remove))
        {
            newString += add;
        }
        else
        {
            newString += character;
        }
    }
    return newString;
}
```

Because Strings are immutable, we need to create a new one to return to the program!

String Traversal Algorithms

We can replace substrings with other substrings:

```
public String replace(String string, String remove, String add)
{
    String newString = “”;
    for(int i = 0; i < string.length(); i++)
    {
        String character = string.substring(i, i+1);
        if(character.equals(remove))
        {
            newString += add;
        }
        else
        {
            newString += character;
        }
    }
    return newString;
}
```

If the character matches the character we want to replace, replace it with the character we want to add!

String Traversal Algorithms

We can replace substrings with other substrings:

```
public String replace(String string, String remove, String add)
{
    String newString = “”;
    for(int i = 0; i < string.length(); i++)
    {
        String character = string.substring(i, i+1);
        if(character.equals(remove))
        {
            newString += add;
        }
        else
        {
            newString += character;
        }
    }
    return newString;
}
```

Otherwise, just add the existing character!

String Traversal Algorithms

We can replace substrings with other substrings:

```
String original = “Peter piper picked a peck of pickled peppers”;

String alteredString = replace(original, “p”, “st”);

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



```
Peter stister sticked a steck of stickled
steststers
```

String Traversal Algorithms

We can also create this method using a while loop:

```
public String replace(String string, String remove, String add)
{
    while(string.indexOf(remove) >= 0)
    {
        int index = string.indexOf(remove);
        string = string.substring(0, index)+ add + string.substring(index+1);
    }
    return string;
}
```

String Traversal Algorithms

We can also create this method using a while loop:

```
public String replace(String string, String remove, String add)
{
    while(string.indexOf(remove) >= 0)
    {
        int index = string.indexOf(remove);
        string = string.substring(0, index)+ add + string.substring(index+1);
    }
    return string;
}
```

Using the **`indexOf()`** method, we will continue to check the string while the String to remove is in **`string`**.

String Traversal Algorithms

We can also create this method using a while loop:

```
public String replace(String string, String remove, String add)
{
    while(string.indexOf(remove) >= 0)
    {
        int index = string.indexOf(remove);
        string = string.substring(0, index)+ add + string.substring(index+1);
    }
    return string;
}
```

When the substring remove is no longer in string, **`indexOf()`** will return a **`-1`**, making the condition **`false`** and the **`while`** loop will be exited.

String Traversal Algorithms

We can also create this method using a while loop:

```
public String replace(String string, String remove, String add)
{
    while(string.indexOf(remove) >= 0)
    {
        int index = string.indexOf(remove);
        string = string.substring(0, index)+ add + string.substring(index+1);
    }
    return string;
}
```

Keeps string the same from 0, index where remove was found

Keeps string the same from the index after where remove was found

String Traversal with While Loops

AP CSA (Java (main))

```java
class MyProgram
{
    public static void main(String[] args)
    {
        String original = "Peter piper picked a peck of pickled peppers";
        String altered = replace(original, "p", "st");

    }

    public static String replace(String string, String remove, String add)
    {
        while(string.indexOf(remove) >= 0)
        {
            int index = string.indexOf(remove);
            System.out.println("Removing " + remove + " at index: "+ index);
            System.out.println("Replacing with: "+ add);
            string = string.substring(0, index) + add + string.substring(index+1);
            System.out.println("The current string is: " +string);
        }
        return string;
    }
}
```

RUN CODE DOCS | MORE

RUN CODE STOP DEBUG

Challenge!

Let's try a challenge problem:

Create a method that reverses the order of a String.

For example, `System.out.println(reverse("Hello World"))` would yield the result: `dlroW olleH`

Challenge!

Create a method that reverses the order of a String:

Challenge!

Create a method that reverses the order of a String:

```
//Step one: create method signature
   //Step two: create new String to hold reversed
   String
   //Step three: create for loop
      //Step four: access last index of String
      //Step five: add index to new String
   //Step six: return new String
```

Challenge!

Create a method that reverses the order of a String:

```
public String reverse(String string)
   //Step two: create new String to hold reversed
   String
   //Step three: create for loop
      //Step four: access last index of String
      //Step five: add index to new String
   //Step six: return new String
```

Challenge!

Create a method that reverses the order of a String:

```
public String reverse(String string)
{
   String newString = “”;
   //Step three: create for loop
      //Step four: access last index of String
      //Step five: add index to new String
   //Step six: return new String
```

Challenge!

Create a method that reverses the order of a String:

```
public String reverse(String string)
{
    String newString = “”;
    //Step three: create for loop
```

The easiest way to do this would be to start with the **last** index of string, and move towards **index 0**

0	1	2	3	4	5
s	t	r	i	n	g

Challenge!

Create a method that reverses the order of a String:

```
public String reverse(String string)
{
   String newString = “”;
   for(int i = string.length() - 1; i >= 0; i--)
      //Step four: access last index of String
      //Step five: add index to new String
   //Step six: return new String
```

Challenge!

Create a method that reverses the order of a String:

```
public String reverse(String string)
{
    String newString = “”;
    for(int i = string.length() - 1; i >= 0; i--)
```

Initializes i to the last index of string

Stops when i reaches 0 (the first index)

Decreases the value of i by 1 each iteration

Challenge!

Create a method that reverses the order of a String:

```
public String reverse(String string)
{
   String newString = “”;
   for(int i = string.length() - 1; i >= 0; i--)
      String character = string.substring(i, i+1);
      //Step five: add index to new String
   //Step six: return new String
```

Challenge!

Create a method that reverses the order of a String:

```
public String reverse(String string)
{
   String newString = “”;
   for(int i = string.length() - 1; i >= 0; i--)
      String character = string.substring(i, i+1);
      newString += character;
   //Step six: return new String
```

Challenge!

Create a method that reverses the order of a String:

```
public String reverse(String string)
{
   String newString = “”;
   for(int i = string.length() - 1; i >= 0; i--)
   {
      String character = string.substring(i, i+1);
      newString += character;
   }
   return newString
}
```

Challenge in Action!

5.3.13: Reverse String

SAVE CONTINUE ›

```java
class MyProgram
{
    public static void main(String[] arg)
    {
        String original = "Let's reverse this string!";
        System.out.println(original);

        String newString = "";
        for(int i = original.length() - 1; i >= 0; i--)
        {
            String character = original.substring(i, i+1);
            newString += character;
        }
        System.out.println("The original string reversed = " + newString);
    }
}
```

RUN CODE | EXAMPLE | DOCS | HELP | MORE

RUN CODE STOP DEBUG

Now It's Your Turn!

CodeHS

Concepts Learned this Lesson

Term	Definition
charAt(int index)	`charAt(int index)` returns the character at the specified index.

Standards Covered

- **(LO) CON-2.F For algorithms in the context of a particular specification that involves String objects:**
    - **Identify standard algorithms.**
    - **Modify standard algorithms.**
    - **Develop an algorithm.**

- **(EK) CON-2.F.1 There are standard algorithms that utilize String traversals to:**
    - **Find if one or more substrings has a particular property**
    - **Determine the number of substrings that meet specific criteria**
    - **Create a new string with the characters reversed**