Strings

# AP Computer Science

Credit: Slides are modified with permission from Barry Wittman at Elizabethtown College

# References & Objects

# Strings are objects

- So far we have focused mainly on primitive data types (int, double, char).
- First, we will look at some basic differences between Strings and primitive data types.
- Strings are what we will start calling an object data type.

# Initializing a String

- To declare and assign a value to a primitive we would do:

```
int num = 5;
```

- We can declare and assign a value to a String in exactly the same way.
- However, there is another way to declare and assign a value to a String.

```
String one = "Hello World!";
String two = new String("Hello World!");
```

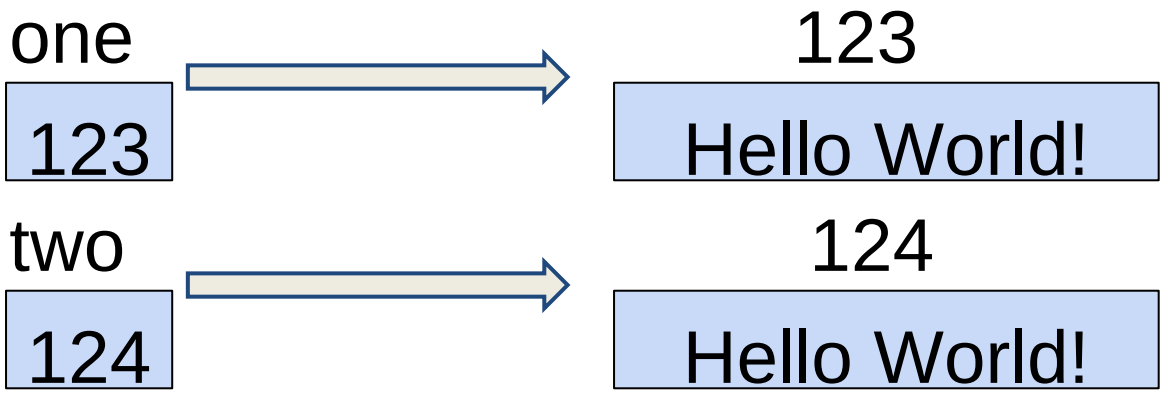
# Initializing a String

```
int num = 5;
```

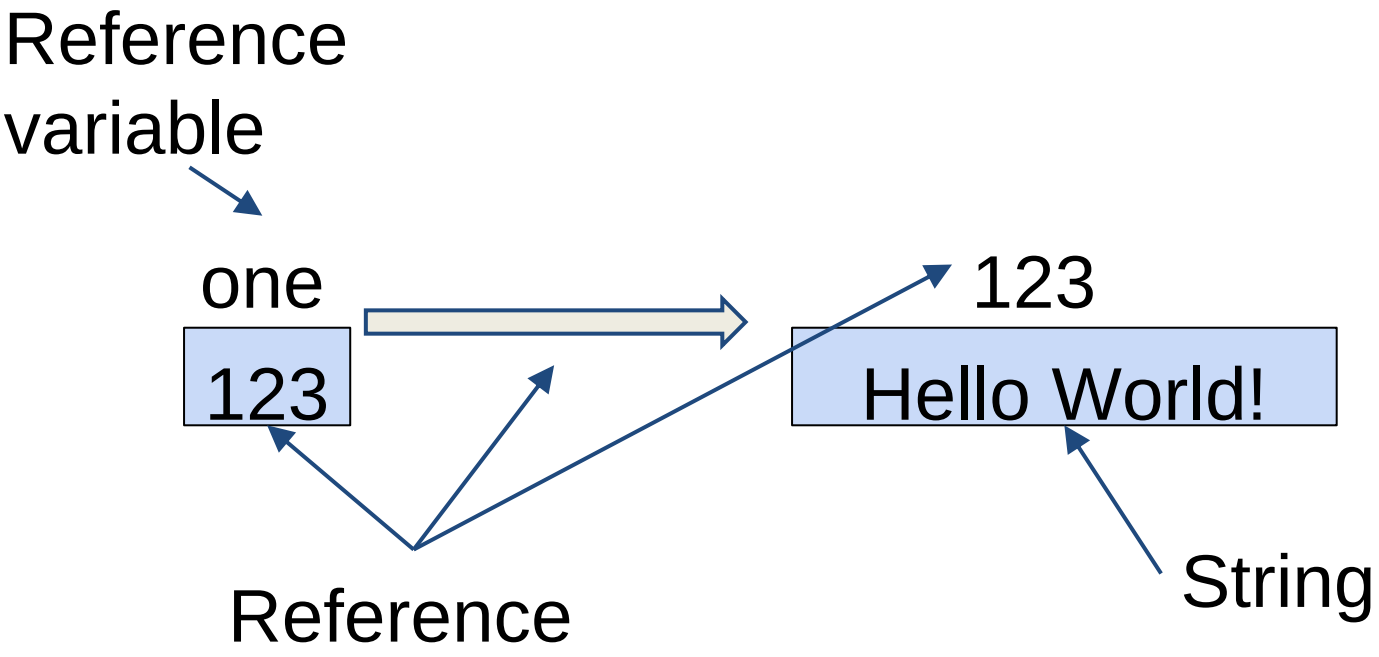
num

5

```
String one = "Hello World!";
String two = new String("Hello World!");
```



# Initializing a String




- For a String the identifier is called a **reference variable** which stores a **reference** to a location in memory where the String is located.

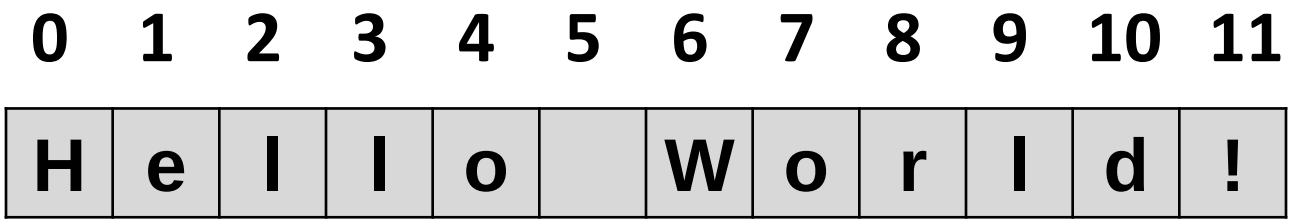
# References

- The best analogy I have heard of for the reference/String relationship is your cell phone number and you.
- Someone can “find” you and ask you to do something by calling your cell phone.

# Strings

# Strings

- Strings are **immutable** – meaning you cannot change the String (the object)
- You can however re-assign a new String to a reference variable (the reference)
- You use indexes to access individual or groups of characters in a String



# Using String Methods

- Proper syntax for calling a String method gives first the name of the String, a dot, the name of the method, and then any arguments

```
stringName.MethodName(arg1, arg2);
```

- String methods always **return** a new String (remember Strings are immutable!)
- Remember to **assign** the method to a variable

# Concatenating Strings

- You can concatenate or “add” two Strings together using the + operator

```
String word1 = “We will play”;
String word2 = “ with Turtles next!”;
String word3 = word1 + word2;
System.out.println(word3);
```

| Output |
| --- |
| We will play with Turtles next! |

# length()

- The length() method returns the number of characters in a String (including whitespace)
- Note the first index is 0
- This means the index of the last character and the length are different

```
String word = "Hello";
int wordLength = word.length();
System.out.println(wordLength);
```

| Output |
| --- |
| 5 |

```
String wordTwo = "Today is a good day!";
int wordTwoLength = wordTwo.length();
System.out.println(wordTwoLength);
```

| Output |
| --- |
| 20 |

# substring()

- substring() returns a section of the String
- You can combine length() and substring()

```
String word = "Hello World";
String newWord = word.substring(6);
System.out.println(newWord);
```

| Output |
| --- |
| World |

```
String word1 = "Today is a";
String word2 = " good day!";
String word3 = word1 + word2;
int index = word2.length() + 1;
System.out.println(index);
out.println(word3.substring(index,15));
```

| Output |
| --- |
| 11<br>good |

# charAt()

- charAt() returns a character at a certain index

```
String wordTwo = "Today is a good day!";
char charTwo = wordTwo.charAt(11);
System.out.println(charTwo);
```

| Output |
| --- |
| g |

- How would you return the last char in a String?

```
String word1 = "We will play";
String word2 = " with Turtles next!";
String word3 = word1 + word2;
out.println(word3.charAt(word3.length()-1));
```

| Output |
| --- |
| ! |

# indexOf()

- indexOf() returns the index of a specified char
  - It reads left to right
- lastIndexOf() does the same except reads from right to left

```
String word = "Hello World";
int loc = word.indexOf('o');
System.out.println(loc);
```

Output

4

```
String wordTwo = "Hello World!";
int locOne = wordTwo.lastIndexOf('o');
System.out.println(locOne);
```

Output

7

# indexOf()

- What happens when the char is not present in the String?
- When this happens, the value of -1 is returned to signify that the character is not in the String

```
String word1 = "We will play";
String word2 = " with Turtles next!";
String word3 = word1 + word2;
System.out.println(word3.indexOf('f'));
```

| Output |
| --- |
| -1 |

# compareTo()

- compareTo() returns the difference in the Strings based upon the ASCII character values
- If the Strings are the same, you receive 0
- If the first characters are the same, it continues checking to the right until it finds a difference
- One nice trick to keep track of whether the value returned is positive or negative is to imagine a **-** sign above the compareTo()

```
String word = "Hello World";
String newWord = "Hello World";
System.out.println(word.compareTo(newWord));
```

| Output |
| --- |
| 0 |

# compareTo()

- What is the output of these two sets of code?

```
String word1 = "hello World";
String word2 = "Hello World";
System.out.println(word1.compareTo(word2));
```

Output

32

```
String word3 = "Hello World";
String word4 = "Hello world";
System.out.println(word3.compareTo(word4));
```

Output

-32

# equals()

- equals() tests for equality of two Strings
- equals() compares the actual Strings

```
String word = "Hello World";
String newWord = "Hello World";
System.out.println(word.equals(newWord));
```

Output

true

```
String word1 = "Hello World";
String word2 = "hello World";
System.out.println(word2.equals(word1));
```

Output

false

# equals()

- This is distinctly different than = or as we will later see ==
- == is actually comparing the reference

```
String word = "Hello World";
String newWord = "Hello World";
System.out.println(word==newWord);
```

| Output |
| --- |
| true |

```
String word1 = "Hello World";
String newWord1;
newWord1 = new String("Hello World");
System.out.println(word1==newWord1);
```

| Output |
| --- |
| false |

# Returning non-String Values

- For the CodingBat.com labs today you will need to complete some return methods.
- The return type will be String, but many of you will want to include other values.
- Here is how you can fix this problem:

```
public String add(String a, String b)
{
    char one = a.charAt(1);
    char two = b.charAt(1);
    return "" + one + two; //adding the
        //empty string solves this problem
}
```

# String Methods

| Method | Description | Returns |
|---|---|---|
| length() | Returns the length of this string (number of characters). | int |
| substring(int from) | Returns a section of the string starting at the location + 1 | String |
| substring(int from, int to) | Returns a section of the string starting at the first location + 1 and including the second location | String |
| charAt(int index) | Returns the char value at the specified index. | char |
| indexOf(String str) | Returns the index within this string of the first occurrence of the specified substring. | int |
| lastIndexOf(String str) | Returns the index within this string of the last occurrence of the specified substring. | int |
| compareTo(String other) | Compares two strings lexicographically. | difference in ASCII values |
| equals(String other) | Compares this string to another String. | true/false |

Here is the entire **String library** with all methods available.