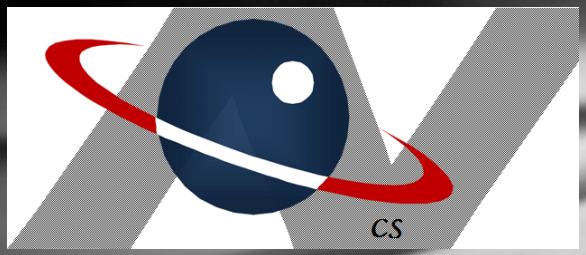
O(N) CS LESSONS

Lesson 5B – Methods – ***String class***



By John B. Owen

Topic List

- Objectives
- The String class
- Output and input with Strings
- Instantiation, immutability, reassignment
- String concatenation review
- String processing
- String method descriptions and examples
- Lesson Summary / Labs
- Contact Information for supplementary materials

Objectives

Students will:

- understand more about the String class and its various methods.
- understand how to instantiate a String object and assign values to a String object
- understand what it means to reassign values to a String object (***immutability*** concept is discussed)

Objectives

Students will:

- have a better understanding of the input and output process with Strings, especially with the commands ***next, nextLine, nextInt, nextDouble, print, println, and printf***

Objectives

Students will learn how to use methods from the String class such as:

- ***length, charAt, indexOf, equals, equalsIgnoreCase, substring, toLowerCase, toUpperCase, trim, compareTo***

String of characters

You learned in an earlier lesson that the word "string" in programming means "string of characters"

A ***String*** is actually stored as an *array* of characters in contiguous RAM memory , meaning that each character is literally "next door" to the next one.

Strings are objects

You also learned earlier that a String is not a primitive data type like int or double.

It is actually an ***object*** of the String class, which means that it not only has an array of characters as data, but it ***owns*** several important methods or functions it can use in processing its own String data.

Primitive data types such as ***int*** do not own any methods.

String vs Math

- The ***String class*** is one that defines a ***String object***, which has several important capabilities.
- Conversely, the Math class does not define and create an object, and only has "stand-alone", ***static*** methods that only work on external data sent through parameters.

String vs Math

- The String class DOES create an object, and the methods act on the internal data that belongs to the current String object.
- Strings are very important and quite prevalent in programming.

Output and input statements

- For example, everything that is input or output in Java, or for that matter, any programming language, is a String.
- Only when it is converted, or ***parsed***, inside the program does it behave like the specific data types: int, double, boolean, etc.

Output and input statements

- In the output statements ***print***, ***println***, and ***printf***, the parameters (values in parentheses) are all ***Strings***.
- The same is true for the input statements ***next*** and ***nextLine***… each one actually takes a String from the keyboard or data file.

Numeric input statements

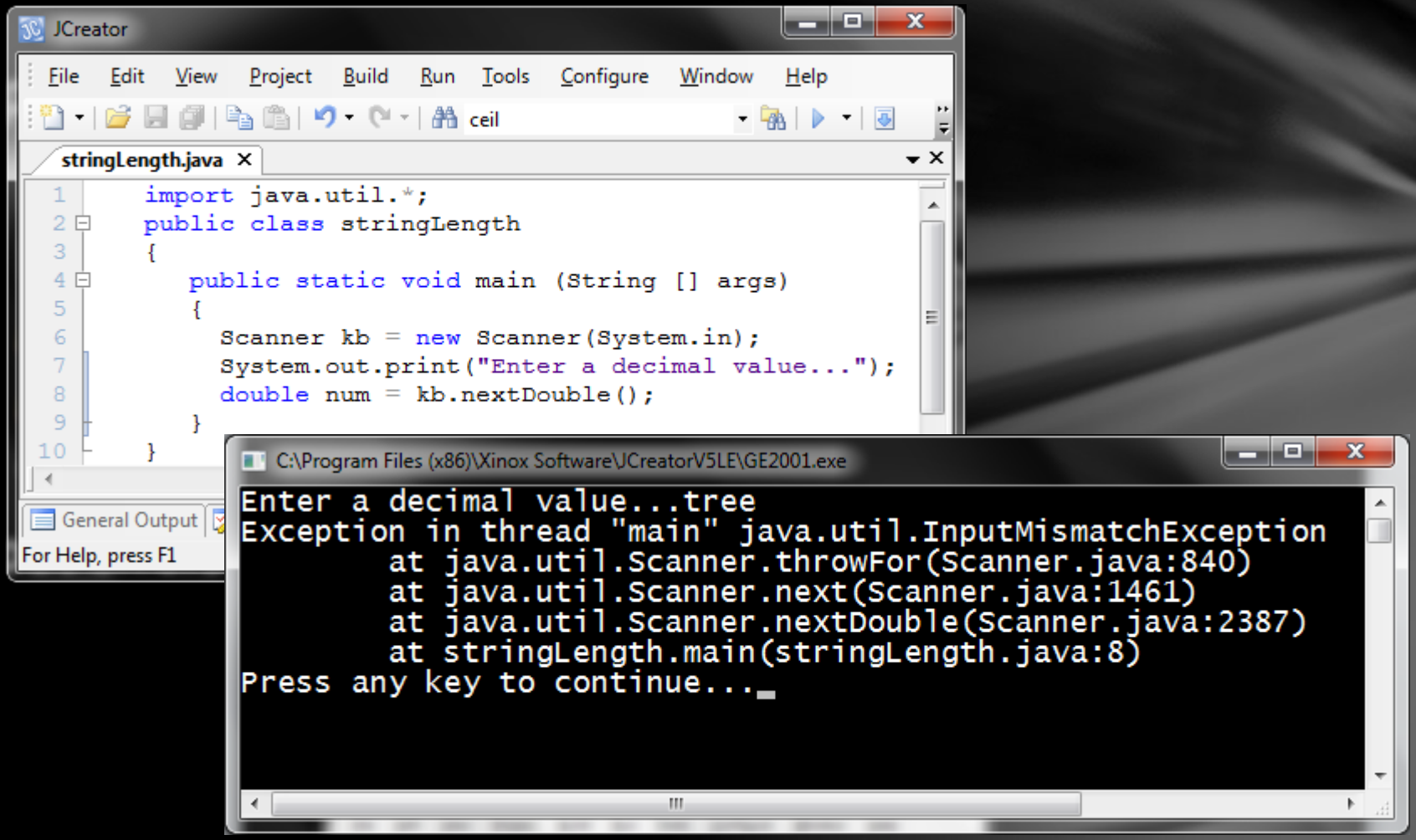
For the numeric input statements like ***nextInt*** and ***nextDouble***, you learned in Lesson 3 (Input) that two processes actually happen with each method…

- **The numeric value is input temporarily as a *String***
- **Then it is converted, or *parsed*, into the actual *numeric value***

Input errors with Strings

- If you try to input a normal word like "tree" using the ***nextInt*** or ***nextDouble*** statements, an ***InputMismatchException*** error will occur and the program will stop.
- See the next slide for an example of this.

Input errors with Strings



String instantiation

- To declare or ***instantiate*** a String object means to bring it into existence in computer memory.
- Below are two examples of this:
  - *String name = "John";*
  - *String day = new String("Monday");*
- Both of these statements ***instantiate*** new String objects.

Strings are immutable

- Unlike a primitive variable, once a String object is ***instantiated***, it cannot be changed in its current location in memory.
- The term for this is ***immutable***, which means *unchangeable*.
- *There is another very useful string class we will explore later on, the* ***StringBuilder*** *class, whose data can be altered and which owns some interesting and useful methods the String class does not.*

String reassignment

Although you cannot change a String object, it can be reassigned.

String name = "Sally";

name = "Caitlyn";

In the example above, the **name** object was ***not changed***, but it was ***reassigned*** a different String value, which IS possible.

String reassignment

- This means that a completely new memory location was created, and the variable ***name*** was re-referenced, or re-pointed to that new memory location containing the new String data.

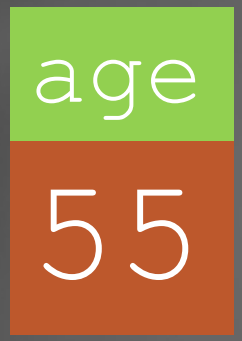
String reassignment

- The previous String object was then recycled by the internal "garbage collector".
- The next few slides examine the subtle yet important difference between primitive and object reassignment.

Primitive vs String reassignment

```
int age = 55;
```

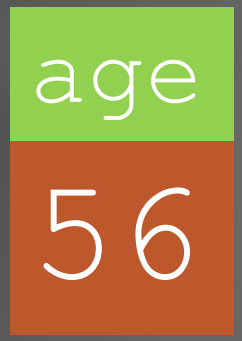
- In Lesson 2, the difference between primitive and object storage was discussed.
- A primitive, like the one shown here, is a memory location that actually stores the data.



Primitive vs String reassignment

```
int age = 55;
age = 56;
```

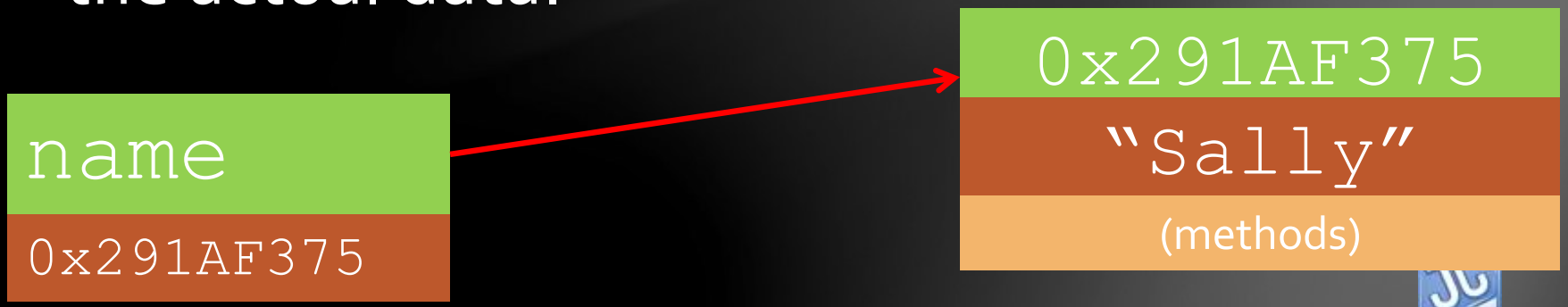
- When a primitive variable is reassigned, the actual content of the same memory location is altered.



Primitive vs String reassignment

```
String name = "Sally";
```

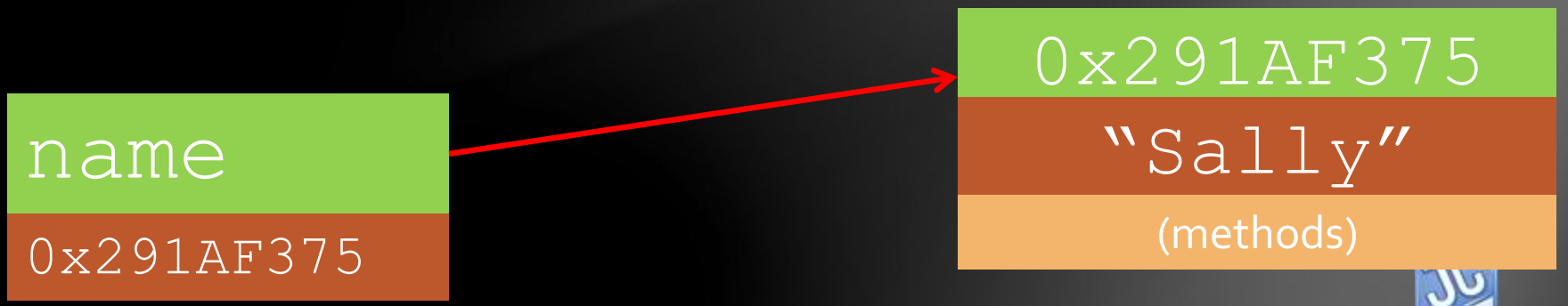
- An object, or "object reference", is actually a memory location that stores another memory location, essentially referencing, or "pointing to" that other memory location in RAM that contains the actual data.



Primitive vs String reassignment

```
String name = "Sally";
```

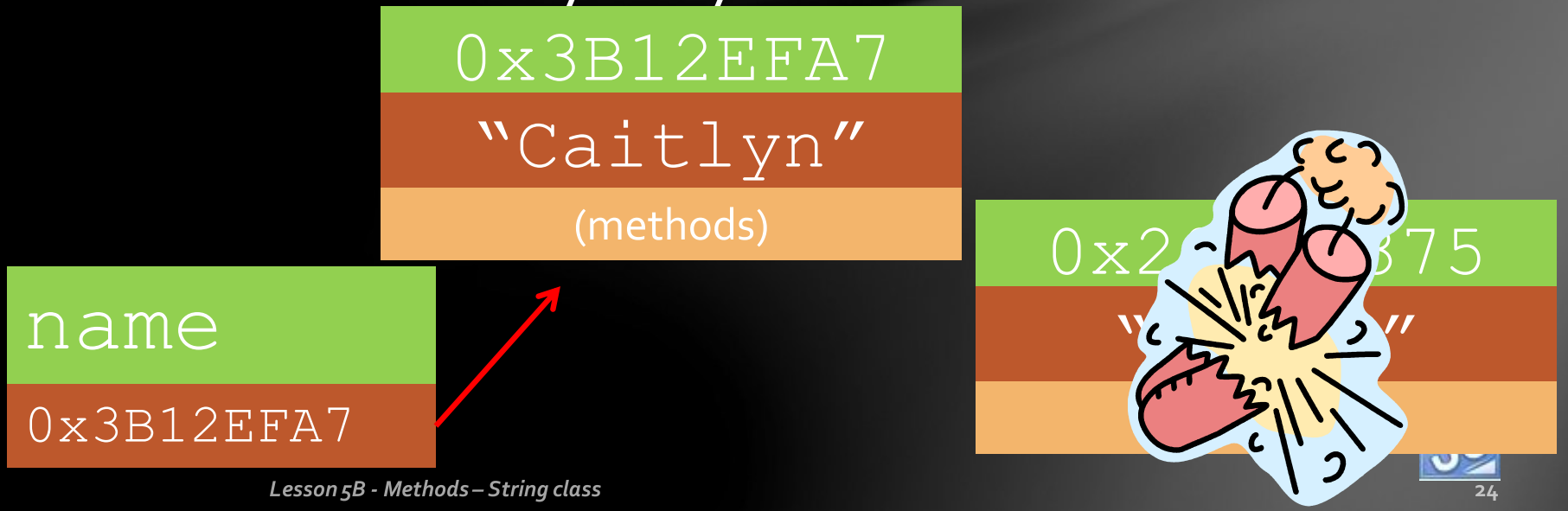
- The memory location that is referenced actually contains the String data along with all of the methods that "belong" to that data, defined by the String class.



Primitive vs String reassignment

```
name = "Caitlyn";
```

- When a reassignment occurs, an entirely new object in a new memory location is created, and the old object is destroyed and its memory recycled!



Primitive vs String reassignment

```
name = "Caitlyn";
```

- Accordingly, the memory location address is changed in the object reference, as shown below.

String concatenation review

- When the "+" operator was introduced in Lesson 2, you learned that Strings are not numerically added, but instead are "concatenated".
- *"Hello" + "World"* becomes *"HelloWorld"*…the two Strings become one new String.

String concatenation review

- If more than two String objects are concatenated, the first two are joined together into a new String, and then the third String is "added" to make another new String.
- In the expression "Hello"+" up"+" there" the first step is "Hello up", followed by "Hello up there".

String processing

String processing is a very important skill in computer programming since everything that is input or output in Java is a String.

There are several very useful methods provided by the String class that are "owned" and available for use by String objects.

String methods

Here are a few of the ones you should learn well…

- **length**
- **charAt**
- **indexOf**
- **equals**
- **equalsIgnoreCase**
- **substring**
- **toLowerCase**
- **toUpperCase**
- **trim**
- **compareTo**

String methods

- On the next several screens are the descriptions for each of these methods, followed by actual program examples.
- Each method "returns" either another String, a character, an integer value, or a boolean value.
- Also notice carefully that some methods use parameters (values inside the parentheses), and some don't.
- Even if no parameter is used, the empty parentheses "()" are still required.

String method descriptions

Here are the descriptions for each method.

- **length()** – returns the number of characters (int) in the current String
- **charAt(x)** – returns the character (char) at position int x in the current String
- **indexOf(str)** – returns the position (int) of the String str inside the current String
- **equals(str)** – returns true or false (boolean) indicating if str is equal to the current String

String methods

- **equalsIgnoreCase(str)** – works just like equals, but considers uppercase and lowercase letters, like 'A' and 'a', to be equal (boolean).
- **compareTo(str)** – returns a positive or negative value (int) that indicates the dictionary (lexical) order of the current String compared to str

String methods

- **substring(start,end-1)** – returns a (String) portion of the current String from the start position to the end position minus 1 (both parameters of type int)
- **substring(start)** – returns a (String) portion of the current String from the (int) start position to the end of the current String

String methods

- **toLowerCase()** – returns a new String that converts all letters to lower case – any non-letters, such as digits or symbols, are unchanged
- **toUpperCase()** – returns a new String that converts all letters to upper case
- **trim()** – returns a new String that removes any empty spaces from the front and the back

length()

In this example, a String is input and its length is displayed

```
//Here is a sample program that uses
//String processing methods
Scanner kb = new Scanner(in);
out.print("Enter a word or sentence-->");
String word = kb.nextLine();
out.println(word+" is "+word.length()+" characters long.");
```

```
Enter a word or sentence-->certificate
certificate is 11 characters long.
Press any key to continue...
```

charAt(x)

In this example, a String is input and its first and last characters are displayed. Notice how the length method is used in combination with charAt.

```
{
  //Here is a sample program that uses
  //String processing methods
  Scanner kb = new Scanner(in);
  out.print("Enter a word or sentence-->");
  String word = kb.nextLine();
  out.println(word.charAt(0)+" is the first character.");
  out.println(word.charAt(word.length()-1)+" is the last character.");
}
```

```
Enter a word or sentence-->Hello there
H is the first character.
e is the last character.
Press any key to continue...
```

indexOf(str)

In this example, a word and phrase are input, then the indexOf method is demonstrated. The -1 value in the second output example means the phrase does not exist in the word. **This method can also receive a character parameter.**

```
  Scanner kb = new Scanner(in);
  out.print("Enter a word or sentence-->");
  String word = kb.nextLine();
  out.print("Enter a phrase-->");
  String phrase = kb.nextLine();
  out.println(phrase+" is at position "+word.indexOf(phrase)+" in "+word);
}
```

```
Enter a word or sentence-->dictionary
Enter a phrase-->ion
ion is at position 4 in dictionary
Press any key to continue...
```

```
Enter a word or sentence-->python
Enter a phrase-->boa
boa is at position -1 in python
Press any key to continue...
```

equals(str)

In the first example, two words are input and checked for equality, and the result is true.

In the second output, the two words look the same, but have one case difference ('h' vs 'H'), therefore are not considered equal.

```
Scanner kb = new Scanner(in);
out.print("Enter a word-->");
String word1 = kb.nextLine();
out.print("Enter another word-->");
String word2 = kb.nextLine();
out.println("It is "+word1.equals(word2)
            +" that "+word1+" is equal to "+word2);
```

```
Enter a word-->hello
Enter another word-->hello
It is true that hello is equal to hello
Press any key to continue..
```

```
Enter a word-->hello
Enter another word-->Hello
It is false that hello is equal to Hello
Press any key to continue...
```

equalsIgnoreCase(str)

However, using this method, they are considered equal since the case difference is ignored.

```
Scanner kb = new Scanner (System.in);
out.print("Enter a word-->");
String word1 = kb.next();
out.print("Enter another word-->");
String word2 = kb.next();
out.printf("It is %s that \"%s\" is equal to \"%s\".\n",
            word1.equalsIgnoreCase(word2),word1,word2);
```

```
Enter a word-->hello
Enter another word-->Hello
It is true that "hello" is equal to "Hello".
Press any key to continue...
```

compareTo(str)

When two Strings are compared, an integer value is returned that represents the ***difference*** between the first occurrence of characters in the Strings that are different. The -1 value means that the first word comes before the second word by a difference of 1, since 'a' is one place before 'b'.

```
Scanner kb = new Scanner(in);
out.print("Enter a word-->");
String word1 = kb.nextLine();
out.print("Enter another word-->");
String word2 = kb.nextLine();
out.println("The difference between "+word1+" and "
            +word2+" is "+word1.compareTo(word2));
```

```
Enter a word-->apple
Enter another word-->banana
The difference between apple and banana is -1
Press any key to continue...
```

compareTo(str)

The resulting value of positive 1 in this example means that the first word comes after the second word by a difference of 1.

```
Scanner kb = new Scanner(in);
out.print("Enter a word-->");
String word1 = kb.nextLine();
out.print("Enter another word-->");
String word2 = kb.nextLine();
out.println("The difference between "+word1+" and "
            +word2+" is "+word1.compareTo(word2));
```

```
Enter a word-->banana
Enter another word-->apple
The difference between banana and apple is 1
Press any key to continue...
```

compareTo(str)

Here are more examples to study.

```
Scanner kb = new Scanner(in);
out.print("Enter a word-->");
String word1 = kb.nextLine();
out.print("Enter another word-->");
String word2 = kb.nextLine();
out.println("The difference between "+word1+" and "
            +word2+" is "+word1.compareTo(word2));
```

```
Enter a word-->oatmeal
Enter another word-->raisin
The difference between oatmeal and raisin is -3
Press any key to continue...
```

```
Enter a word-->zebra
Enter another word-->anteater
The difference between zebra and anteater is 25
Press any key to continue...
```

compareTo(str)

The first occurrence difference in these examples comes **after** the first letters.

```
Scanner kb = new Scanner(in);
out.print("Enter a word-->");
String word1 = kb.nextLine();
out.print("Enter another word-->");
String word2 = kb.nextLine();
out.println("The difference between "+word1+" and "
            +word2+" is "+word1.compareTo(word2));
```

```
Enter a word-->banana
Enter another word-->bat
The difference between banana and bat is -6
Press any key to continue...
```

'n' – 't' = -6

```
Enter a word-->king
Enter another word-->kindergarten
The difference between king and kindergarten is 3
Press any key to continue...
```

'g' – 'd' = 3

substring(x,y) or *substring(x)*

There are two versions of substring, both of which create new strings that are a portion of the original string. In the two-parameter version, the values 1 and 4 indicate the beginning and "end+1" of the substring, with the last character or the substring actually being in the position <u>one less</u> than 4. **STUDY THIS VERY CAREFULLY!!!**

The one-parameter version starts at that position and takes the rest of the string.

```
Scanner kb = new Scanner(in);
out.print("Enter a word-->");
String word = kb.nextLine();
out.println(word.substring(1,4));
out.println(word.substring(3));
```

```
Enter a word-->cracker
rac
cker
Press any key to continue...
```

trim()

Here is how trim works. It simply creates a new String with any spaces at the beginning and end of the String removed. However, it does not remove imbedded spaces from the original string.

```
  String word = " Hello World ";
  out.println(">"+word+"<");
  out.println(">"+word.trim()+"<");
}
```

```
> Hello World <
>Hello World<
Press any key to continue...
```

toLowerCase(), toUpperCase()

These examples are fairly obvious. Notice that only the letters are affected. Digits, spaces, and symbols are not changed.

```
Scanner kb = new Scanner(in);
out.print("Enter a word-->");
String word = kb.nextLine();
out.println(word.toLowerCase());
out.println(word.toUpperCase());
```

```
Enter a word-->121 N Santa Clara
121 n santa clara
121 N SANTA CLARA
Press any key to continue...
```

Lesson Summary

- In this lesson, you learned more about the String class, discussing the concept of "immutability", reviewing the input and output process, and exploring all of the different methods that belong to String objects, such as ***length, charAt, indexOf, equals, equalsIgnoreCase, substring, toLowerCase, toUpperCase, trim, compareTo***

Labs

- The following labs are all intended to help you practice using the String class methods.
- Be sure that the output is precise and matches EXACTLY, especially according to spacing and indentation.

Lab 5B-1

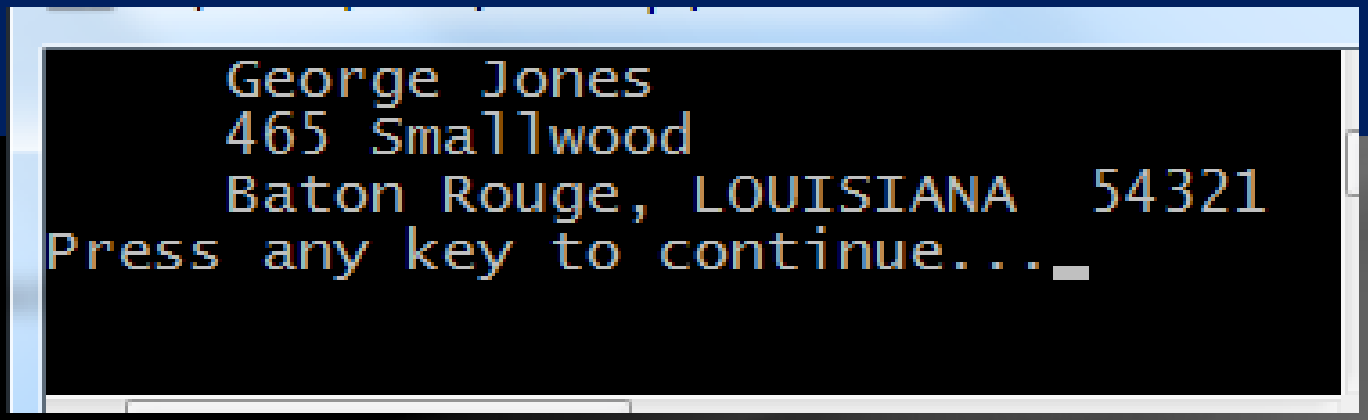
WAP to input from a data file your return address information (first name, last name, street address, city, state, and zip code, each input as a String on a separate line) and output it in correct form as shown on the next slide. Indent each line 5 spaces. Use your String methods to abbreviate and capitalize the state. Be sure there are two spaces between the state and zip code. Use your own personal data for your input. *Remember to handle the whitespace issue when **nextLine** follows **next**!*

```
Input: (data file "lab5B1.in")
George
Jones
465 Smallwood
Baton Rouge
Louisiana
54321
```

Use your own name and address for the data in this file…

Lab 5B-1

Output:



Lab 5B-2

WAP to input a sentence and output it:

- In all uppercase
- In all lowercase
- In original form

Then output the length of the sentence.

```
Input Data File ("lab5B2.in"):
DC is beautiful when the 3750 Japanese cherry trees are in bloom.
The subtotal of $56.73, plus tax and tip of about 25%, equaled $70.91.
JaCk AnD jIlL wErE nOt ToO bRiGht

Output (next slide):
```

Lab 5B-2

Output:

```
DC IS BEAUTIFUL WHEN THE 3750 JAPANESE CHERRY TREES ARE IN BLOOM.
dc is beautiful when the 3750 japanese cherry trees are in bloom.
DC is beautiful when the 3750 Japanese cherry trees are in bloom.
Length = 65 characters.

THE SUBTOTAL OF $56.73, PLUS TAX AND TIP OF ABOUT 25%, EQUALED $70.91.
the subtotal of $56.73, plus tax and tip of about 25%, equaled $70.91.
The subtotal of $56.73, plus tax and tip of about 25%, equaled $70.91.
Length = 70 characters.

JACK AND JILL WERE NOT TOO BRIGHT
jack and jill were not too bright
JaCk AnD jIlL wErE nOt ToO bRiGht
Length = 33 characters.

Press any key to continue...
```

Lab 5B-3

WAP to input two strings and check if they are equal, again if they are equal regardless of case, and then output the value returned when you compare them.

```
Input Data File ("lab5B3.in"):
pants Pants
shoes shirt
Zoot-suit ascot

Output (next slide):
```

Lab 5B-3

Output:

```
It is false that "pants" is equal to "Pants".
It is true that "pants" is equal to "Pants", ignoring the case.
"pants" compared to "Pants" returns the value 32.

It is false that "shoes" is equal to "shirt".
It is false that "shoes" is equal to "shirt", ignoring the case.
"shoes" compared to "shirt" returns the value 6.

It is false that "Zoot-suit" is equal to "ascot".
It is false that "Zoot-suit" is equal to "ascot", ignoring the case.
"Zoot-suit" compared to "ascot" returns the value -7.

Press any key to continue...
```

Lab 5B-4

WAP to input a string and a character and return the position of the character found in the string, or indicate that the character is not in the string. *Hint: use if else*

```
Input Data File ("lab5B4.in"):
dog
g
beautifully
l
orangutan
z
Output:
```

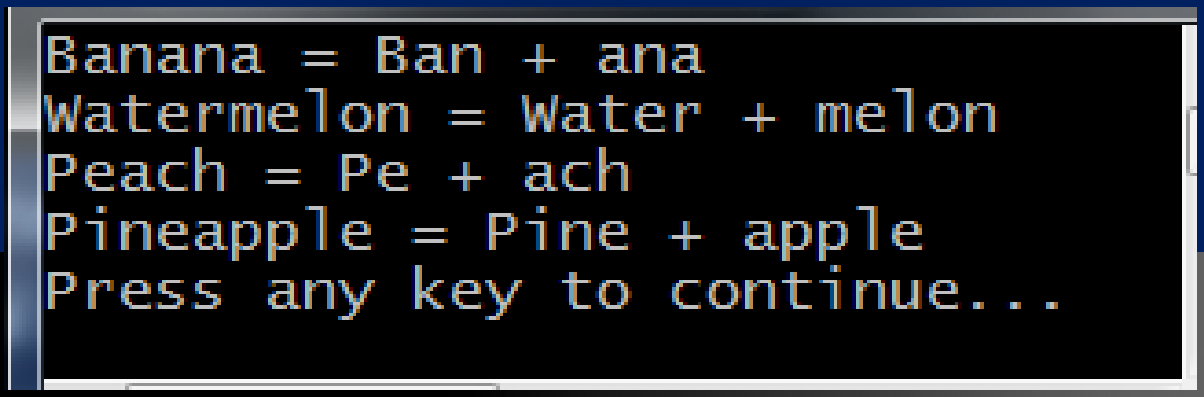
```
The position of 'g' in the word "dog" is 2.
The position of 'l' in the word "beautifully" is 8.
The word "orangutan" does not contain the letter 'z'.
Press any key to continue...
```

Lab 5B-5

WAP to input a string and output the string in two halves. First output the original string, then an equal sign, then the two halves separated by a '+' sign, as shown below. If the word has an odd number of letters, output the shorter half first.

Hint: use length and substring to help write this program.

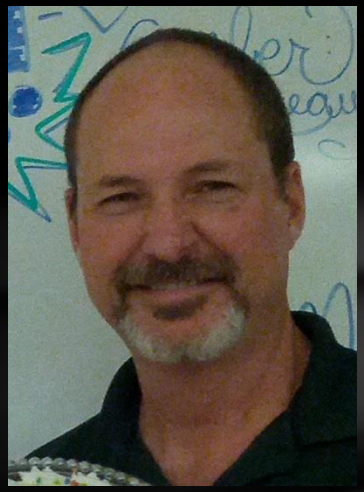
```
Input Data File ("lab5B5.in"):
Banana
Watermelon
Peach
Pineapple

Output:
```



CONGRATULATIONS!

- You now understand more about the String class and how to use the methods to do String processing.
- *The Lesson 5C will show you how to write your own "stand-alone" utility methods.*

Thank you, and have fun!



To order supplementary materials for all the lessons in this package, such as lab solutions, quizzes, tests, and unit reviews, visit the O(N)CS Lessons website, or contact me at

John B. Owen
captainjbo@gmail.com