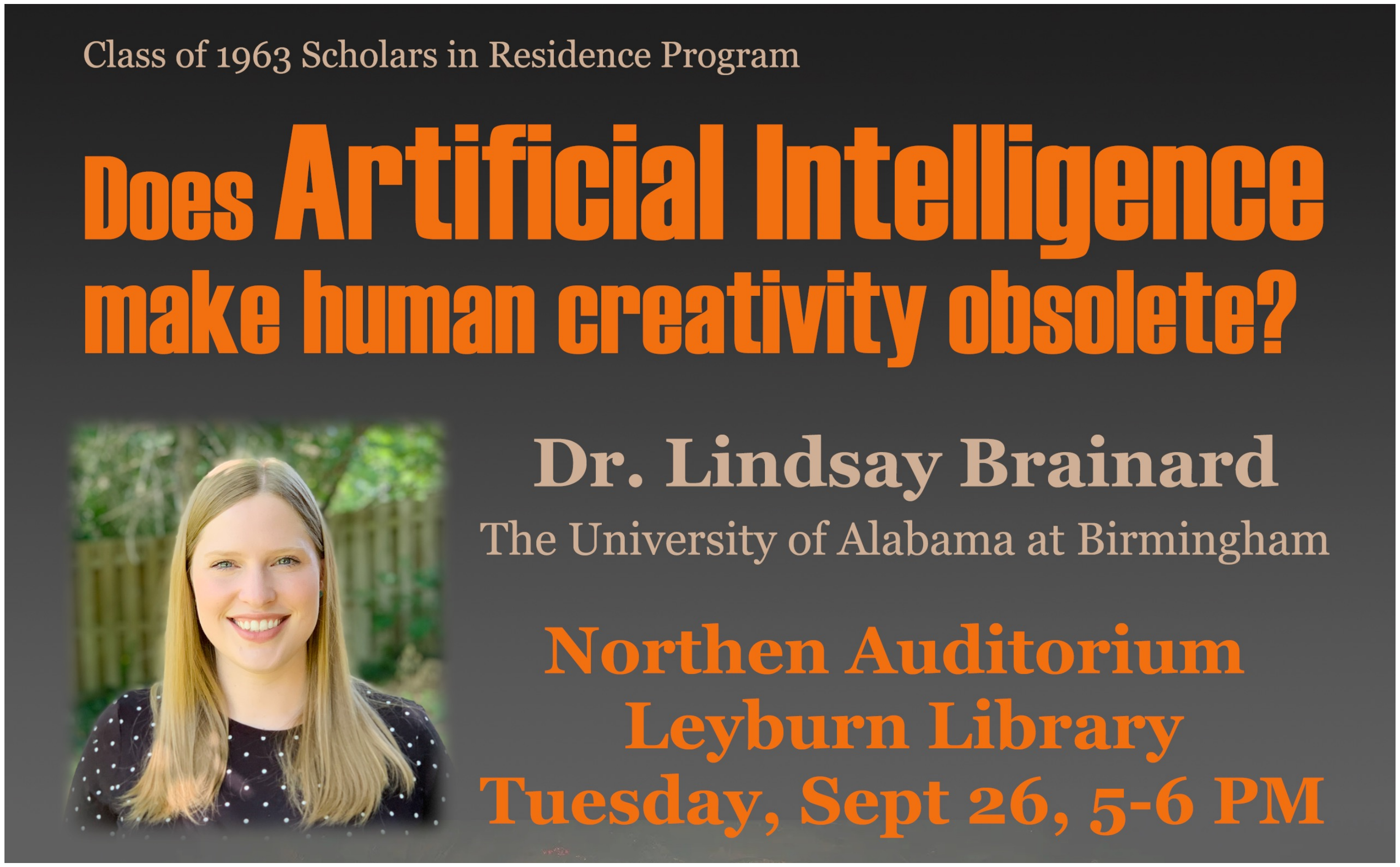
Objectives

- Using the Java Library
  - `java.lang` classes: `String` class
  - Getting user input with `java.util.Scanner`
  - Constructing objects
  - Using the Java API
  - Importing classes
- Continuing Java Fundamentals
  - Control Structures
  - Scope

Extra Credit Opportunity



Class of 1963 Scholars in Residence Program

Does Artificial Intelligence make human creativity obsolete?

Dr. Lindsay Brainard

The University of Alabama at Birmingham

Northen Auditorium
Leyburn Library
Tuesday, Sept 26, 5-6 PM

Write response on Canvas

Review

- What are some of the primitive data types of Java?
- What is the syntax for declaring a variable in Java?
  - What is the difference between *declaring* a variable and *defining* a variable?
- What are the arithmetic and relational operators that Java supports?
  - What does ++ after a variable mean?
- 111/112 review
  - What is an **API**?
  - What are operations you can do on strings in Python?
- How did assignment 0 go?

You can and *should* review previous slides if you don't remember answers

Review: Data Types

- Java is ***strongly*** and ***statically typed***
  - Every variable must have a ***declared* type**
- All data in Java is an ***object*** – except for the ***primitive data types***:

`int`	4 bytes (-2,147,483,648 -> 2,147,483,647)
`short`	2 bytes (-32,768 -> 32,767)
`long`	8 bytes (really big integers)
`byte`	1 byte (-128 -> 127)
`float`	4 bytes (floating point)
`double`	8 bytes (floating point)
`char`	2 bytes (Unicode representation), **single** quotes
`boolean`	`true` or `false`

Fun fact: Python *unified* ints and longs → no longer has long

Assign 0

- Problems?
- Tips or tricks for others?
  - Read: what mistakes will you vow never to do again but probably will?
- Debugging part – shows that you shouldn’t write a lot of code before compiling!

INTRODUCTION TO JAVA LIBRARIES

Java Libraries

- Organized into a *hierarchy* of ***packages***
- Similar to Python's ***packages*** (`__init__.py`)

```
java
 ├─ lang
 │   ├─ Object
 │   └─ String
 ├─ net
 └─ util
     └─ Scanner
javax
org
```

Fully qualified name: `java.lang.String`

> `java.lang.*` classes included
> *by default* in all Java programs

Many, many more classes and packages

`java.lang.String` class

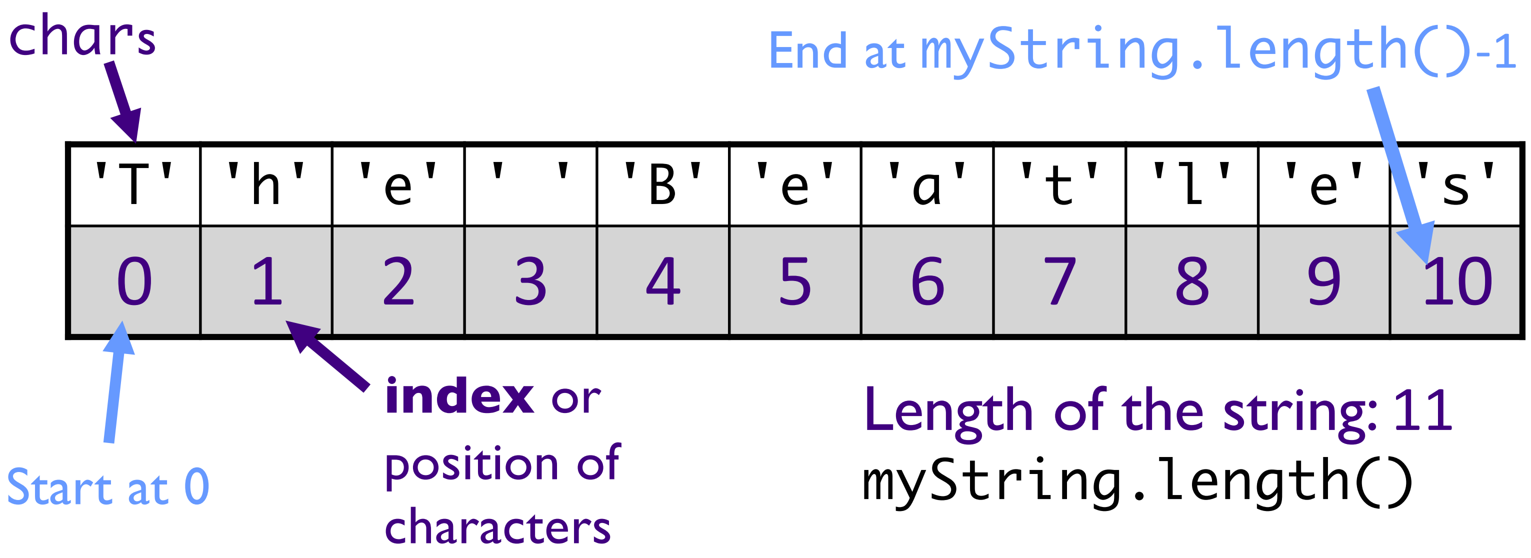
- Similar functionality to Python but accessed differently
  - Mostly through *methods*!
- **Included by default** in every Java program
- Strings are represented by **double** quotes: `""`
  - Single quotes represent `chars` only
- Examples:

```
String emptyString = "";
String niceGreeting = "Hello there.";
String badGreeting = "What do you want?";
```

Strings

- A `char` at each position of `String`

```
String myString = "The Beatles";
```



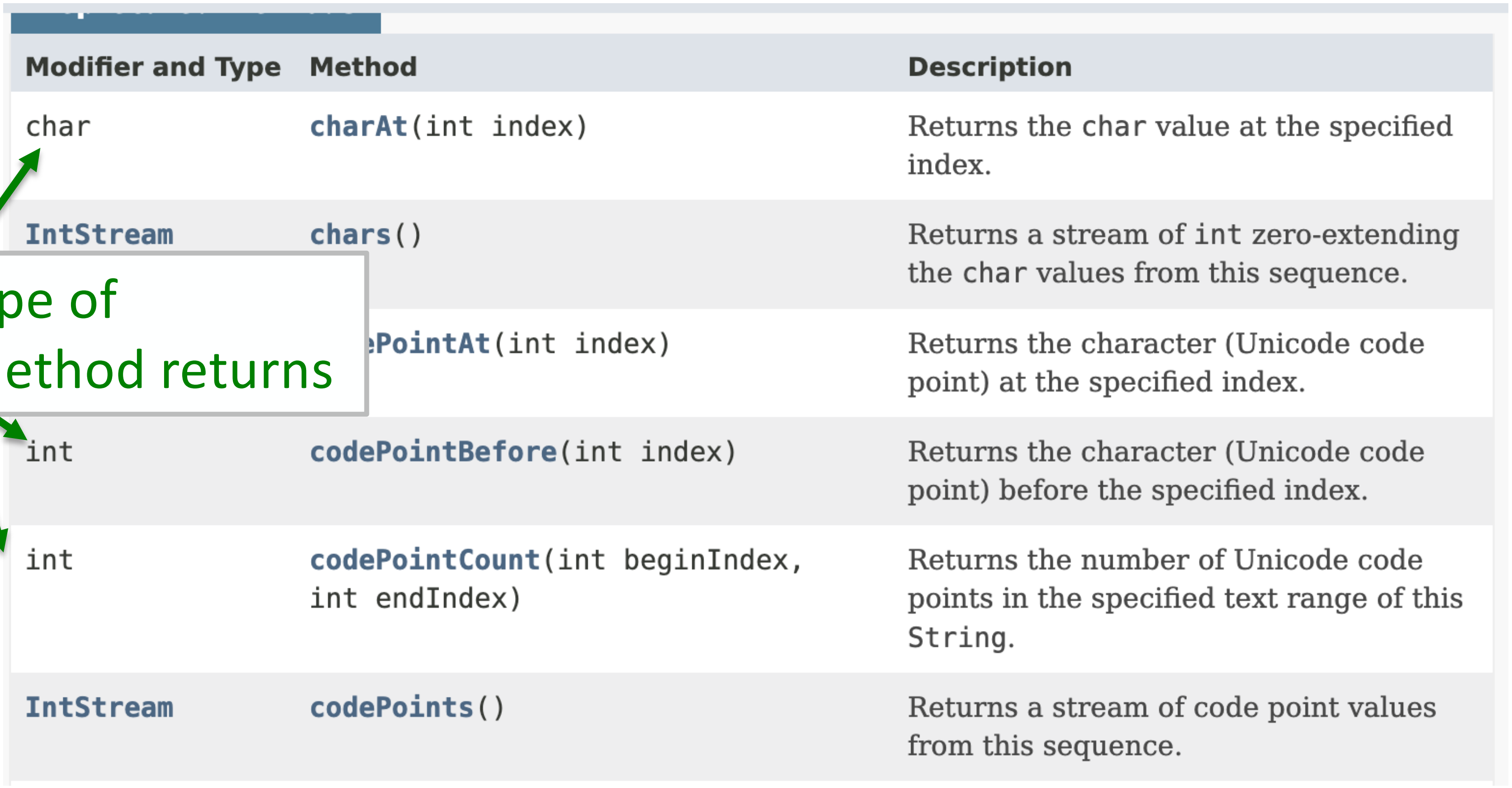
Calling a method is the same syntax as Python: `object.method()`

Java API Documentation

https://docs.oracle.com/en/java/javase/17/docs/api/index.html

- API: Application Programming Interface
  - What a class can do for YOU!
  - The contract: if you call this method with these parameters, then the method will do this
- Complete documentation of every class included with the JDK
  - Every method and variable contained in class
- Bookmark it!
  - Too many classes, methods to remember them all
  - Refer to it often

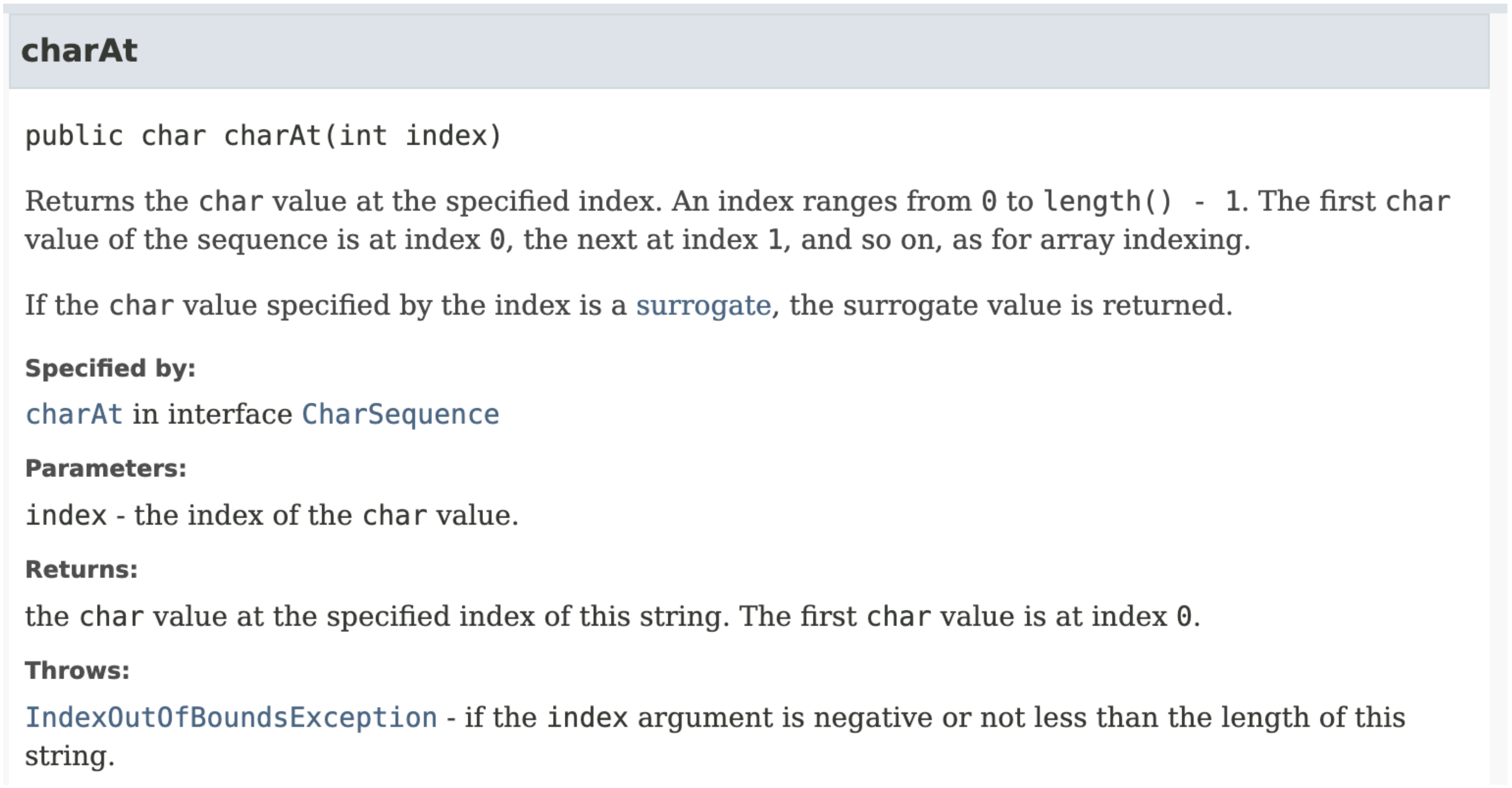
Reading JavaDocs: String Overview



Modifier and Type	Method	Description
char	charAt(int index)	Returns the char value at the specified index.
IntStream	chars()	Returns a stream of int zero-extending the char values from this sequence.
	ePointAt(int index)	Returns the character (Unicode code point) at the specified index.
int	codePointBefore(int index)	Returns the character (Unicode code point) before the specified index.
int	codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range of this String.
IntStream	codePoints()	Returns a stream of code point values from this sequence.

Data type of what method returns

Reading JavaDocs: Detail



charAt

```
public char charAt(int index)
```

Returns the `char` value at the specified index. An index ranges from `0` to `length() - 1`. The first `char` value of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

If the `char` value specified by the index is a surrogate, the surrogate value is returned.

Specified by:

`charAt` in interface `CharSequence`

Parameters:

`index` - the index of the `char` value.

Returns:

the `char` value at the specified index of this string. The first `char` value is at index `0`.

Throws:

`IndexOutOfBoundsException` - if the `index` argument is negative or not less than the length of this string.

Python equivalent?

String method: charAt

```
String testString1 = "Demonstrate Strings";

char character1;
char character2 = testString1.charAt(3);
character1 = testString1.charAt(testString1.length()-2);

System.out.println(character1 + " " + character2);
```

Aside: I try to show different ways to code to show different implementations are possible, e.g., declaring variables with and without initializing it.

- Python equivalent: indexing with [<pos>]

String method: charAt

```
String testString1 = "Demonstrate Strings";

char character1;
char character2 = testString1.charAt(3);
character1 = testString1.charAt(testString1.length()-2);

System.out.println(character1 + " " + character2);
```

Displays:

```
g o
```

Python Transition Gotcha: Can’t use negative numbers for indices as in Python

- Python equivalent: indexing with [<pos>]

String methods: `substring`

- Python equivalent: *slicing*
- `String substring(int beginIndex)`
  - Returns a new String that is a substring of this string, from `beginIndex` to end of this string
- `String substring(int beginIndex, int endIndex)`
  - Returns a new String that is a substring of this string, from `beginIndex` to `endIndex-1`

```
String language = "Java!";
String subStr = language.substring(1);
String subStr2 = language.substring(2, 4);
```

subStr is "ava!"
subStr2 is "va"

String Concatenation

- Use + operator to concatenate Strings

```
String niceGreeting = "Hello";
String firstName = "Clark";
String lastName = "Kent";
String blankSpace = " ";

String greeting = niceGreeting + "," +
       blankSpace + firstName +
       blankSpace + lastName;

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

Note: statement doesn't end until ;

Displays `Hello, Clark Kent`

String methods: and many more!

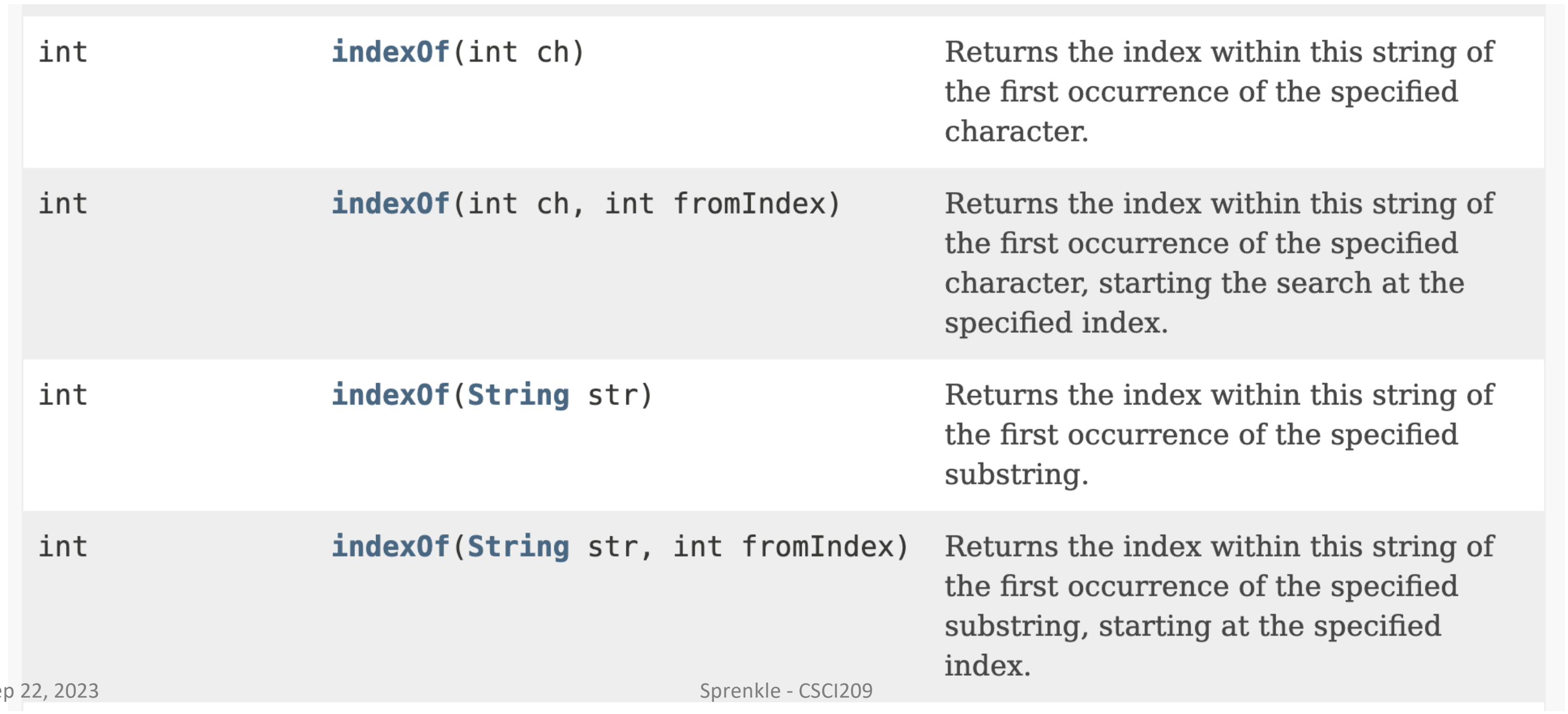
- `boolean endsWith(String suffix)`
- `boolean startsWith(String prefix)`
- `boolean equalsIgnoreCase(String other)`
- `int length()`
- `String toLowerCase()`
- `String trim()` : remove trailing and leading white space
- …

See `java.lang.String` API for all available methods:
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html

Multiple Methods with Same Name

- We saw it with `substri`
- Another example:

Different from Python!
How/why is it possible with Java?



int	indexOf(int ch)	Returns the index within this string of the first occurrence of the specified character.
int	indexOf(int ch, int fromIndex)	Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	indexOf(String str)	Returns the index within this string of the first occurrence of the specified substring.
int	indexOf(String str, int fromIndex)	Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

Leveraging an API in Java

- Consider:
  - What do I want to do?
  - What data/information do I have?
  - What information do I want back?
- Process:
  - Find a method that does what you want
    - Check if there is more than one method of the same name.
    - Don't necessarily stop when you find one that could work; may be a better fit later
    - If you can't find a method, can you break the problem down into multiple steps? Repeat process of finding method(s)
  - Look at the method's parameters and their types
    - Do you have the data in that form/data type? Can you convert it to that type?

Getting user input

JAVA.UTIL.SCANNER

Getting User Input

- To get user input, we will use the Scanner class
- Create a Scanner object by calling the ***constructor***
    - new keyword means you're *allocating memory* for an *object*

```
Scanner sc = new Scanner(System.in);
```

What is this?

- Need to ***import*** the class because it's ***not*** part of java.lang package
    - Imports go at the top of the program, before class definition

```
import java.util.Scanner;
```

Generalizing

- Constructing a new object:

```
DataType object = new DataType(arguments,…);
```

- Importing classes
  - If the class does not belong to `java.lang` package, need to import it

Scanner

- Makes reading/parsing input easier
- Breaks its input into *tokens* using a *delimiter pattern*, which matches *whitespace*

What is a "delimiter pattern"?
What is "whitespace"?

- Converts resulting tokens into values of different types using `nextXXX`()

Example Code Using Scanner

```
long tempLong;

// create the scanner for the console
Scanner sc = new Scanner(System.in);

// read in an integer and a String
int i = sc.nextInt();
String restOfLine = sc.nextLine();

// read in a long
tempLong = sc.nextLong();

sc.close();
```

`java.util.Scanner`

- Many constructors
  - Read from file, input stream, string …
- Many methods
  - `nextXXXX` (int, long, line)
  - Skipping patterns, matching patterns, etc.
  - Can change token delimiter from default of whitespace
- Close the Scanner when you're done with it

Using Scanner

```java
public static void main(String[] args) {

   // open the Scanner on the console input, System.in
   Scanner scan = new Scanner(System.in);

   scan.useDelimiter("\n"); // breaks up by lines, useful for console I/O

   System.out.print("Please enter the width of a rectangle: ");
   int width = scan.nextInt();

   System.out.print("Please enter the height of a rectangle: ");
   int length = scan.nextInt();
  scan.close();

   System.out.println("The area of your square is " + length * width + ".");
}
```

ConsoleUsingScannerDemo.java

Using Scanner

- What does the useDelimeter method in last slide do?
- Try running the program with and without that line, entering 5 4 as input

Effective Java: Code Inefficiency

- I showed this:

```
String s = "text";
```

- Instead of this

```
String s = new String("text"); // DON’T DO THIS
```

Why didn’t we talk about ***constructing*** a String?

Effective Java: Code Inefficiency

- I showed this:

```
String s = "text";
```

- Instead of this

```
String s = new String("text"); // DON’T DO THIS
```

Why didn’t we talk about ***constructing*** a String?

Would create *two* strings

`StringBuilders` and `Strings`

- Strings are read-only or immutable
  - Same as Python
- More efficient to use `StringBuilder` to manipulate a `String`
- Instead of creating a new String using
  - ~~`String str = prevStr + " more!";`~~
- Use

```
StringBuilder str = new StringBuilder( prevStr );
str.append(" more!");
```

- Many `StringBuilder` methods
  - `toString()` to get the resultant string back

Looking Ahead

- Assignment 1 due Monday before class
- Read textbook (see web site)