Java Basics:
The String Class

Objects

- So far, we have seen:
  - *variables*, which represent data (categorized by **types**)
  - *methods*, or functions, which represent behavior
- It is possible to create new types that are combinations of these two entities
  - Such types are called *object types* or *class types*
  - Languages such as Java in which you can do this are called *object-oriented* programming languages
- We will learn how to use some of Java's objects
  - In a later module we will learn to create our own types of objects

OOP Terminology

- *object*: An encapsulation of data and behavior
  - Placing data and associated methods together

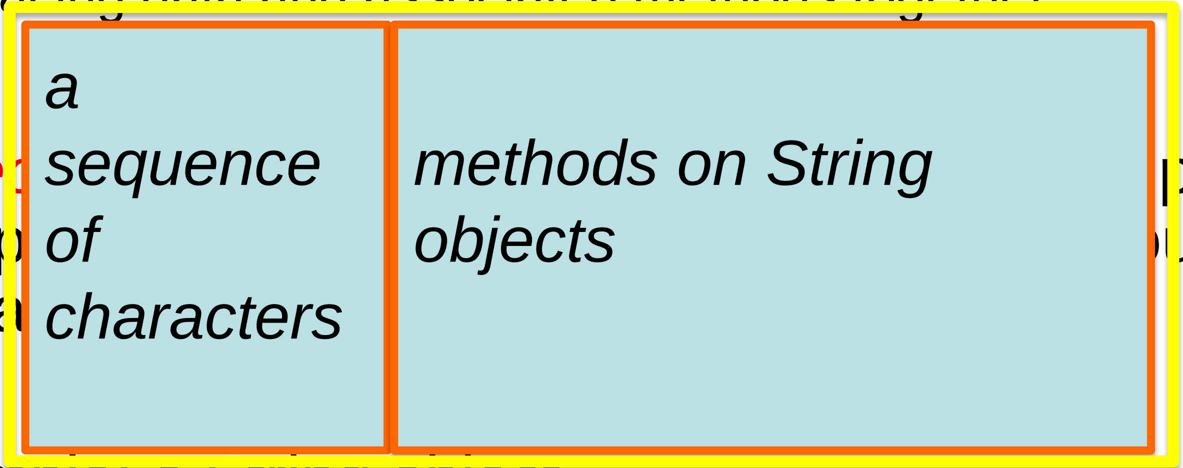
- *object-oriented programming (OOP)*: Writing programs that perform most of their useful behavior through interactions with objects

- Example: a String object

OOP Terminology

- *object*: An encapsulation of data and behavior
  - Placing data and associated methods together

- *obje* programs
  that p ugh
  intera



- Example: a String object

Overview of Java String class

class String

- A class file that describes the data and methods associated with String objects
  - a sequence of characters
  - methods such as `concat`, `contains`, `split`

String *object*

- "apple"
- "Hello World"
- "This is a long String"

Calling methods of objects

- Objects contain methods that can be called by your program.
  - For example, a `String`'s methods manipulate or process the text of that `String` in useful ways
  - We must specify which object we want to manipulate, and then write the method's name
- You call a class method by invoking it on a class object
  - Uses what we call “dot notation”
  - General syntax:
    objectName **.** ***methodName*** (***parameters***)
  - The results will be different from one object to another

Java String class

Assign String literal "cde" to a variable

```
String str1 = "cde";
```

Concatenate & print "cdexyz"

```
out.println(str1.concat("xyz"));
```

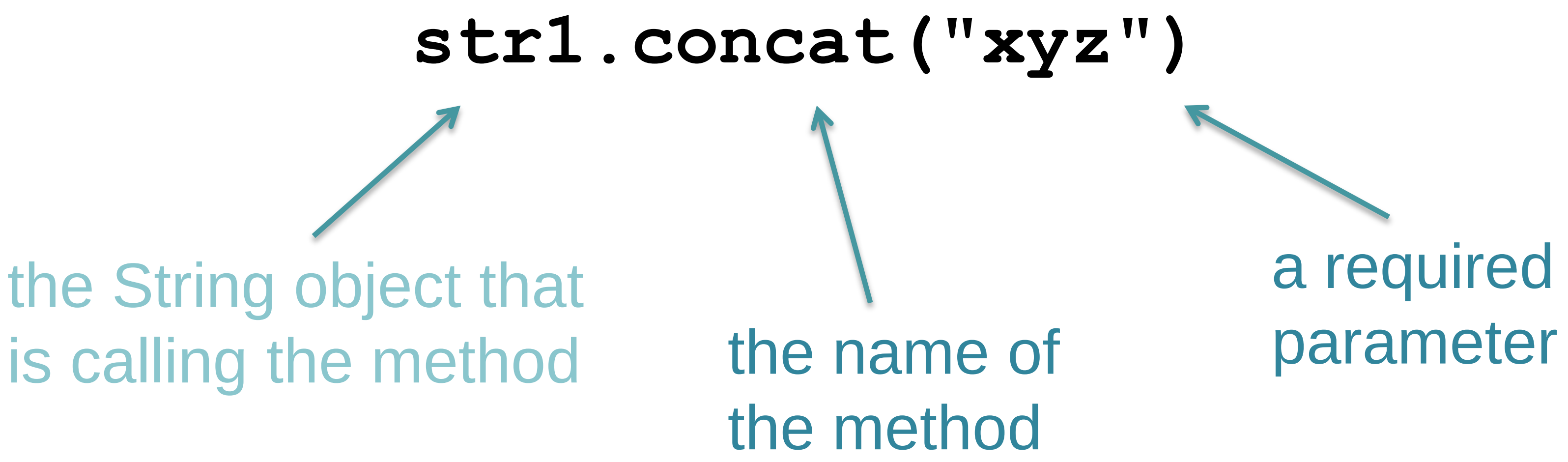
*Extract substring beginning at **a** and ending **before c***

```
String str2 = "abc".substring(0,2);
```

Prints "ab"

```
out.println(str2);
```

Using methods with objects



```
str1.concat("xyz")
```

Using methods with objects

```
str1.concat("xyz");
```

This method doesn’t change str1

Using methods with objects

```
str1.concat("xyz");
```

This method *returns* a new string that is the concatenation of str1 and “xyz”

Using methods with objects

```
String newStr = str1.concat("xyz");
```

This method *returns* a new string that is the concatenation of str1 and “xyz”

Using methods with objects

```
out.print(str1.concat("xyz"));
```

This method *returns* a new string that is the concatenation of str1 and “xyz”

Other examples of String methods

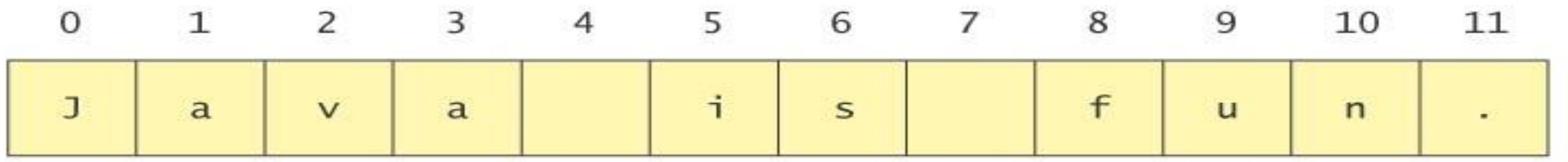
- *<String object name>*.**`substring(x,y);`**
  - returns the string beginning at index x and ending just before index y
- *<String object name>*.**`startsWith(str2);`**
  - returns true if the String begins with str2
- *<String object name>*.**`length();`**
  - returns the number of characters in the String object

Positions in a `String`

- Positions within a String start at 0, not 1
  - A position is referred to as an *index*

- The '`J`' in "`Java is fun.`" is at index 0

The twelve characters in the string "`Java is fun.`" have indices 0 through 11. The index of each character is shown above it.



0	1	2	3	4	5	6	7	8	9	10	11
J	a	v	a		i	s		f	u	n	.

`String` methods

Method name	Description
`indexOf(`**str**`)`	index where the start of the given string appears in this string (-1 if not found)
`length()`	number of characters in this string
`substring(`**index1**`,` **index2**`)` or `substring(`**index1**`)`	the characters in this string from *index1* (inclusive) to *index2* (exclusive); if *index2* is omitted, grabs till end of string
`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters
`charAt(`**index**`)`	returns the character at the given index

There are more… see the web for a complete list

Modifying Strings

- The methods that appear to modify a string (`substring`, `toLowerCase`, `toUpperCase`, etc.) actually create a new string and return it.

```
String s = "honest abe";
s.toUpperCase();              // does not change s
out.println(s);               // output: honest abe
```

- If you want to modify the variable, you must reassign it to store the result of the method call:

```
String s = "honest abe";
s = s.toUpperCase();
out.println(s);               // output: HONEST ABE
```

Comparing two Strings

```
String a = "may";
String c = "May";

if (a == c) {
   out.println("a == c");
}
```

Never compare objects using **==**
This will compile, but the results may not be what you are really asking for

Comparing two Strings

```
String a = "may";
String c = "May";

if (a == c) {
   out.println("a == c");
}

if (a.equals(c)) {
   out.println("a.equals(c)");
}
```

Always use the `equals` method when comparing two objects

Java Basics:
Type char and
The Character Class

Type `char`

- `char` : A primitive type representing single characters
  - Each character inside a `String` is stored as a `char` value
  - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as `'a'` or `'4'` or `'\n'` or `'\''`
  - It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';
out.println(letter);                  // S
```

The `charAt` method

- The individual characters in a `String` can be accessed using the `charAt` method.

```
String food = "cookie";
char firstLetter = food.charAt(0);    // 'c'
out.println(firstLetter + " is for " + food);
out.println("That's good enough for me!");
```

`char` vs. `int`

- All `char` values are assigned numbers internally by the computer, called *ASCII/UTF-16* values
  - Examples:

`'A'` is 65,	`'B'` is 66,	`' '` is 32
`'a'` is 97,	`'b'` is 98,	`'*'` is 42

  - Conveniently, the alphabet is in order (`'b' < 'j'`)

`char` vs. `int`

- Occasionally we want to convert a `char` to/from `int`
  - Adding a char and an int, returns an int: `'a' + 7 // 104`
  - To convert back to a char, use a cast: `(char)('a' + 7) // 'h'`
  - Always use the character literal rather than its ASCII/UTF-16 value

```
if (ch == 97) //<--BAD STYLE!!
```

Instead use: `if (ch == 'a')  //<--GOOD STYLE!!`

`char` vs. `String`

- `"h"` is a `String`
  `'h'` is a `char`   (the two behave very differently)

- `String` is an object; it contains methods

```
String s = "h";
s = s.toUpperCase();           // "H"
int len = s.length();          //  1
char first = s.charAt(0);      // 'H'
```

- `char` is primitive like `int`; you can't call methods on it

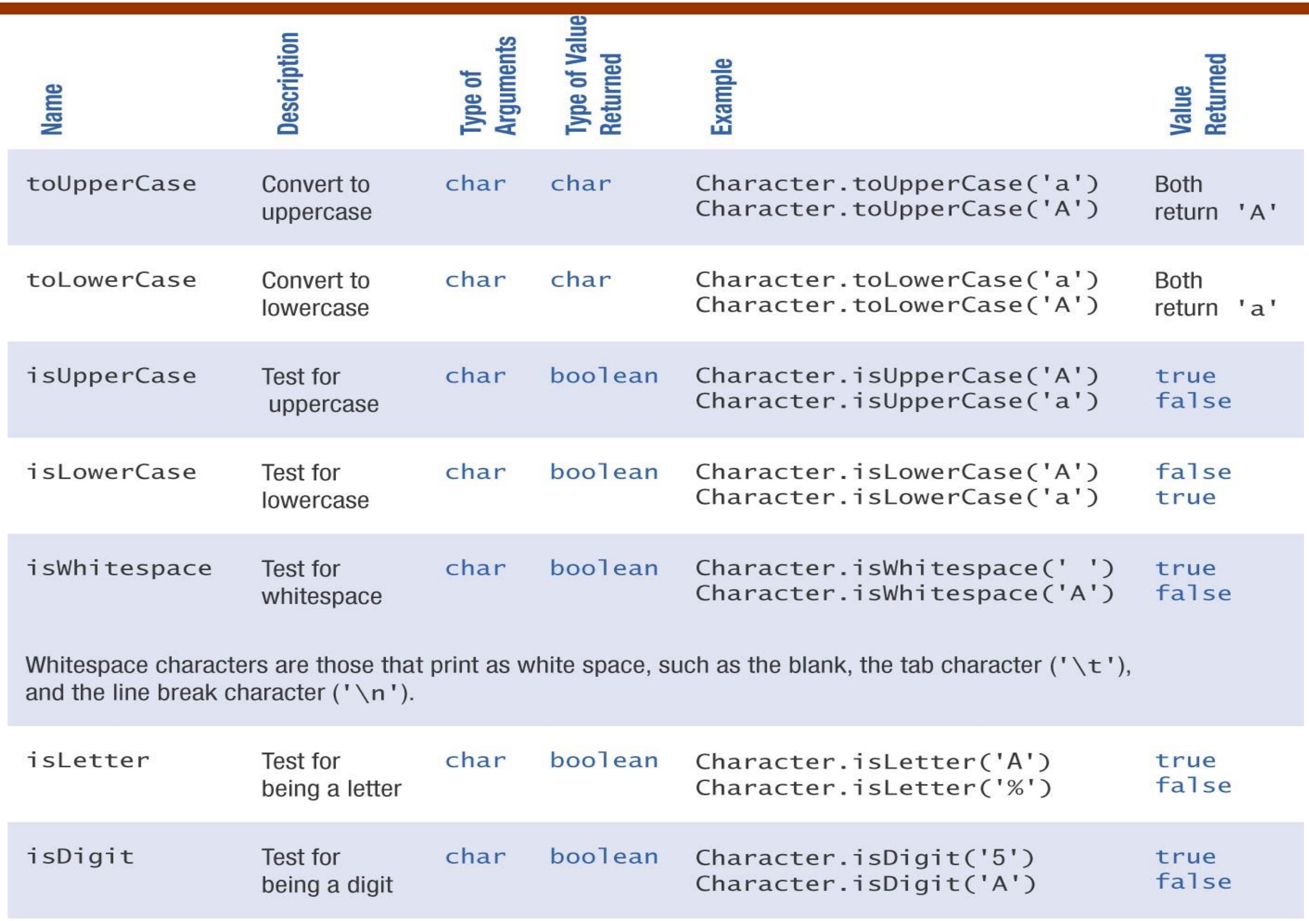
```
char c = 'h';
c = c.toUpperCase();     // ERROR: "cannot be dereferenced"
```

`Character` wrapper class

- Just as the `Math` class provided us with many useful mathematical functions, the `Character` class provides many character-related functions
- These are all static methods
  - Called on the `Character` class
- They take a single character as a parameter
  - Already saw earlier that we cannot call methods directly on data of type `char`, since `char` is a primitive type
- Example:

```
Character.isLetter(ch)    //ch is var of type char
```

Static Methods in Class `Character`



Name	Description	Type of Arguments	Type of Value Returned	Example	Value Returned
`toUpperCase`	Convert to uppercase	`char`	`char`	`Character.toUpperCase('a')` `Character.toUpperCase('A')`	Both return `'A'`
`toLowerCase`	Convert to lowercase	`char`	`char`	`Character.toLowerCase('a')` `Character.toLowerCase('A')`	Both return `'a'`
`isUpperCase`	Test for uppercase	`char`	`boolean`	`Character.isUpperCase('A')` `Character.isUpperCase('a')`	`true` `false`
`isLowerCase`	Test for lowercase	`char`	`boolean`	`Character.isLowerCase('A')` `Character.isLowerCase('a')`	`false` `true`
`isWhitespace`	Test for whitespace	`char`	`boolean`	`Character.isWhitespace(' ')` `Character.isWhitespace('A')`	`true` `false`
Whitespace characters are those that print as white space, such as the blank, the tab character (`'\t'`), and the line break character (`'\n'`).					
`isLetter`	Test for being a letter	`char`	`boolean`	`Character.isLetter('A')` `Character.isLetter('%')`	`true` `false`
`isDigit`	Test for being a digit	`char`	`boolean`	`Character.isDigit('5')` `Character.isDigit('A')`	`true` `false`