Object-Oriented

Internally, the library class `java.lang.String` stores an array of characters. It also provides a variety of useful methods for comparing, manipulating, and searching text in general.

Manager:		Recorder:	
Presenter:		Reflector:	

Content Learning Objectives

After completing this activity, students should be able to:

- Describe how characters and strings are represented in memory.
- Explain the difference of calling static versus non-static methods.
- Recognize common mistakes when working with the String API.

Process Skill Goals

During the activity, students should make progress toward:

- Working with all team members to reach consensus on hard questions. (Teamwork)

Facilitation Notes

This activity takes an "inside look" at the `String` class to introduce object-oriented programming concepts. Report out on #4 and #5, and help students make connections to previous activities about arrays and memory diagrams. Note that when drawing strings in the future, students will not need to use this level of detail.

#10 introduces the concept of `this` as an implicit argument. Spend some time reporting out in the middle of Model 2 to make sure students understand the object itself is passed. Point out that methods from the `String` class make it easier to manipulate character arrays.

Model 3 should be a quick review of errors students have previously seen at some point in the course. When reporting out, have teams discuss why these errors occur in the context of what they have learned in Model 1 and Model 2.

Key questions: #6, #12, #20

Source files: *StrCmp.java*

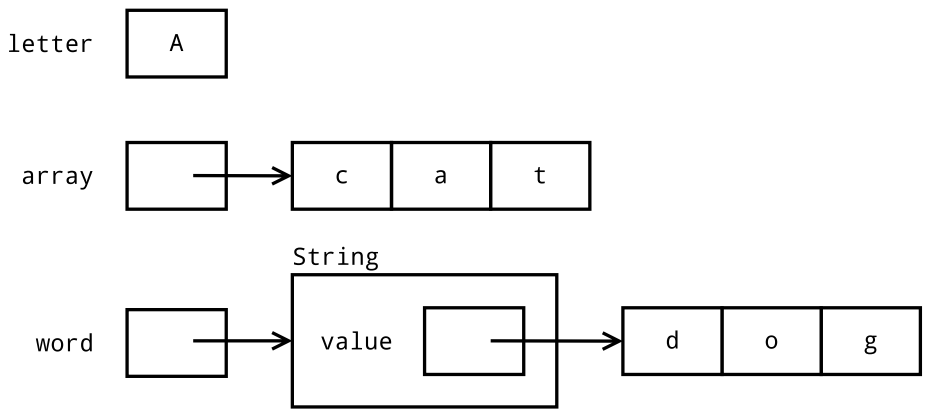
Model 1 Character Arrays

The primitive type `char` is used to store a single character, which can be a letter, a number, or a symbol. In contrast, the reference type `String` ***encapsulates*** an array of characters.

```
char letter;
letter = 'A';

char[] array;
array = new char[]
        {'c', 'a', 't'};

String word;
word = "dog";
```



Questions (15 min)

Start time:

1. How is the syntax of character literals (like `'A'`) and string literals (like `"dog"`) different?

Character literals use single quotes, and strings use double quotes.

2. What is the index of `'d'` in the string above? What is the index of `'g'`?

The index of `'d'` is 0, the index of `'g'` is 2.

In general, the last character is at `length - 1`.

3. What is the *value* of a `char` variable (i.e., stored in the variable's memory)? What is the *value* of an array variable? What is the *value* of a `String` variable?

Since `char`s are primitive, the value of a `char` variable is the character itself. Arrays and strings are reference types, so their variables contain memory locations.

4. Based on the diagram, what does it mean for an object to encapsulate data? How do you access the `char[]` inside of a `String` object?

The data is stored inside the class and accessed via methods. `String` (the class) makes it more convenient to deal with character arrays.

5. Draw a memory diagram for the given code. (List the name of each variable next to a box containing its value.)

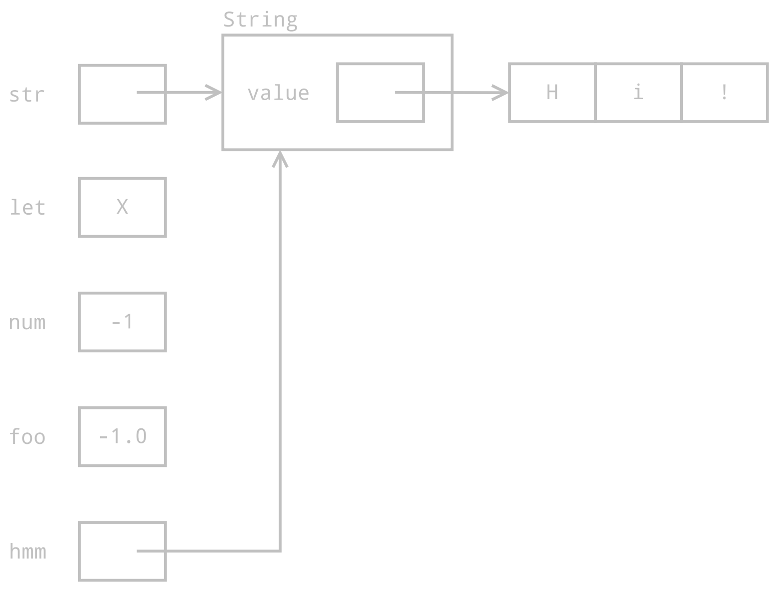
```
String str;
str = "Hi!";

char let;
let = 'X';

int num;
num = -1;

double foo;
foo = num;

String hmm;
hmm = str;
```



6. Recall that the == operator compares the *value* of two variables. What does it mean for two `char` variables to be ==? What does it mean for two `String` variables to be ==?

Two `char` variables are == if they have the same character. In contrast, two `String` variables are == if they refer to the same `String` object.

7. How could you determine whether two character arrays have the same contents? In other words, how does the `String.equals` method work internally?

Using a loop, you compare each character in the first array to the corresponding character in the second array. If any two characters don't match, or the arrays have different lengths, then they are not equal.

Model 2 Non-Static Methods

Method	Returns	Description
`charAt(int)`	`char`	Returns the char value at the specified index of `this` string.
`indexOf(String)`	`int`	Returns the index within `this` string of the first occurrence of the specified substring.
`length()`	`int`	Returns the length of `this` string.
`substring(int, int)`	`String`	Returns a new string that is a substring of `this` string (from `beginIndex` to `endIndex - 1`).
`toUpperCase()`	`String`	Returns a copy of `this` string with all the characters converted to upper case.

Each method listed above is non-`static`. That is, they have an ***implicit parameter*** named `this` that is passed automatically. (Note: There are many other `String` methods not listed above.)

Questions (15 min) Start time:

8. If the variable `str` references the string `"hello world"`, then what is the return value of the following method calls?

a) `str.charAt(8)` `'r'`

b) `str.indexOf("wo")` 6

c) `str.length()` 11

d) `str.substring(4, 7)` `"o w"`

e) `str.toUpperCase()` `"HELLO WORLD"`

9. Explain what precedes the dot operator (`.`) in the expressions above. What does it have to do with the keyword `this` in Model 2?

The variable referencing the string precedes the dot operator. It is passed as the implicit `this` parameter to the non-`static` method.

10. How many arguments does each method call in #8 have? (Hint: None of them have zero; don't forget to count the implicit argument.)

a) 2

b) 2

c) 1

d) 3

e) 1

To call a `static` method, you write *ClassName.methodName*, for example: `Math.abs(-5)`

To call a non-`static` method, you write *objectName.methodName*, for example: `str.charAt(8)`

Methods can be designed either way. Most `String` methods are non-`static`, because that makes the code easier to read.

Static (`str` passed explicitly)	**Non-Static (`str` passed implicitly)**
`String.charAt(str, 8) // wrong`	`str.charAt(8)`

11. Label each method below as `static` or non-`static` (based on how it is called).

a) `color.indexOf("RED")` non-static

b) `String.format("%3d", x)` static

c) `title.substring(0, 10)` non-static

d) `String.valueOf(3.14)` static

e) `name.charAt(0)` non-static

12. Consider the following statement and compiler error. Why is it impossible to call `charAt` this way? How would you correct the error?

```
char c = String.charAt(0);
```

Error: non-static method charAt(int) cannot be referenced from a static context

`charAt` is a non-static method; you can't call it using the `String` class. You need to call it using an object like `name`. Otherwise, the compiler won't know which string you want.

13. For each method in #11, what object is referenced by `this`? (Write N/A if `this` is not passed to the method.)

a) color

b) N/A

c) title

d) N/A

e) name

Model 3 Common Mistakes

Use *JShell* to run each of the code snippets. For Program B, do not provide any user input (just press Enter). Record the output of the last statement in the space below the table.

Program A	**Program B**

`String greeting = "hello world";`
`greeting.toUpperCase();`
`System.out.println(greeting);`
	`Scanner in = new Scanner(System.in);`
	`String line = in.nextLine();`
	`char letter = line.charAt(1);`

```
hello world
```

```
StringIndexOutOfBoundsException
```

Questions (15 min)

Start time:

14. What is the logic error you see when you run Program A?

The `"hello world"` greeting is still lowercase.

15. In Program A, what is returned by the string method? What happens to the return value?

A new string is returned, but its value is not saved anywhere.

16. Describe two different ways you can fix the logic error in #15.

```
greeting = greeting.toUpperCase();    // or just combine the two lines
System.out.println(greeting);         System.out.println(greeting.toUpperCase());
```

17. In what cases will Program B throw an exception? What is the error message displayed?

If the user enters a blank line, then there will be no character at position 1. It will show the out of bounds exception and display the stack trace.

18. Describe two different ways you can fix the run-time error in #17.

1. Check if the string is long enough: `if (line.length() >= 2)`
2. Keep asking for input until valid: `do { ... } while (line.length() < 2);`

19. To compare strings, you must use either the `String.equals` or `String.compareTo` method. Predict the output of the following code.

```
String name1 = new String("Mark");
String name2 = new String("Mark");

// compare name1 and name2
if (name1 == name2) {
    System.out.println("name1 and name2 are identical");
} else {
    System.out.println("name1 and name2 are NOT identical");
}
```

```
    name1 and name2 are NOT identical
```

```
// compare "Mark" and "Mark"
if (name1.equals(name2)) {
    System.out.println("name1 and name2 are equal");
} else {
    System.out.println("name1 and name2 are NOT equal");
}
```

```
    name1 and name2 are equal
```

20. What is the difference between ***identical*** and ***equal*** in the previous question?

"Identical" means the variables refer to the same `String` object. "Equal" means the strings have the same characters, in the `Arrays.equals` sense.