COMPSCI 121: BRANCHES

SPRING 20

BYTE FROM COMPUTING HISTORY

Oak Ridge National Lab's Summit supercomputer became the world's most powerful in 2018. It occupies an area equivalent to two tennis courts and uses more than 27,000 powerful graphics processors. It enables programmer to use Artificial Intelligence (AI) to solve problems.

GOALS FOR TODAY'S CLASS

More nuts & bolts of programming!

- Logical Operators
- String and Character operations
- Switch statement
- String comparisons and character access

BOOLEAN DATA TYPE

Boolean variable may be set using true or false keywords.

```
boolean flag;
flag = true;
```

```
boolean evenFlag = (n % 2 == 0); // true if n is even
boolean positiveFlag = (x > 0); // true if x is positive

if (evenFlag) {
System.out.println("n was even when I checked it");
}

if (!evenFlag) {
System.out.println("n was odd when I checked it");
}
```

Don't have to write `if (evenFlag == true)`

LOGICAL OPERATORS

`&&` and

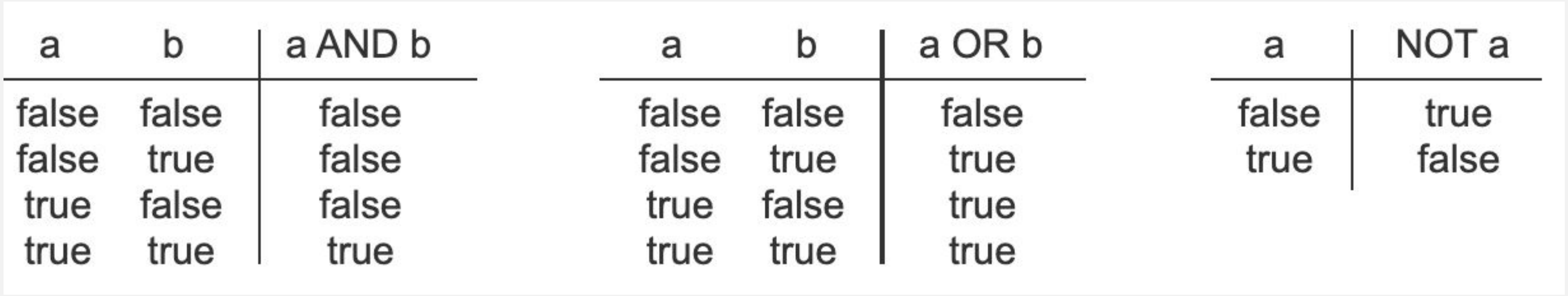
`||` or

`!` Not

1. `x > 0 && x < 10` is `true` (when x is greater than zero *AND* less than 10)
2. `isEven || n % 2 == 0` is `true` (if *EITHER* condition is true)
3. `!isEven` is `true` (if `isEven` is *NOT* true.)

BOOLEAN FUNCTIONS

Function	**In Java**
AND	&&
OR	||
NOT	!



a	b	a AND b
false	false	false
false	true	false
true	false	false
true	true	true

a	b	a OR b
false	false	false
false	true	true
true	false	true
true	true	true

a	NOT a
false	true
true	false

“Truth Tables” for functions AND, OR, NOT.

CLICKER QUESTION 1

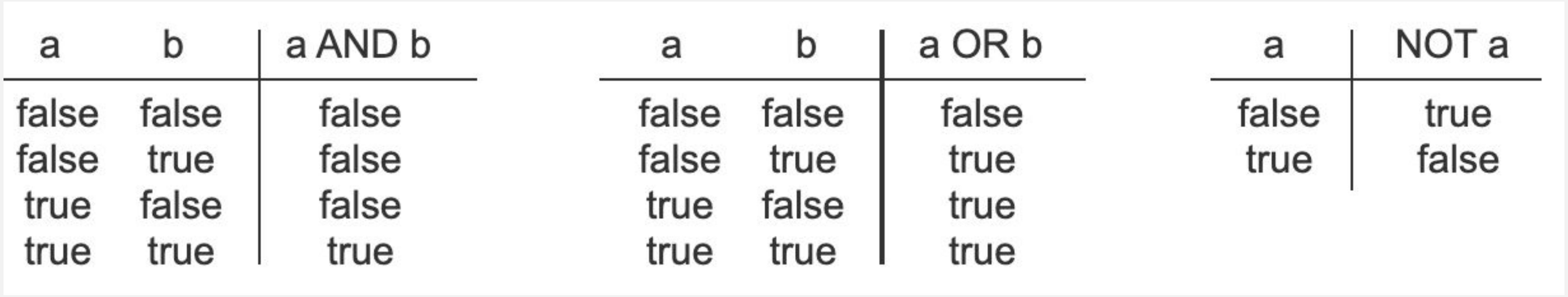
```
boolean var1 = true;
boolean var2 = false;
System.out.println(var2 && var1);
```

A. 0
B. 1
C. true
D. false
E. error

CLICKER QUESTION 1 ANSWER

```
boolean var1 = true;
boolean var2 = false;
System.out.println(var2
 && var1);
```

A. 0
B. 1
C. true
D. [✓ false]
E. error



a	b	a AND b
false	false	false
false	true	false
true	false	false
true	true	true

a	b	a OR b
false	false	false
false	true	true
true	false	true
true	true	true

a	NOT a
false	true
true	false

TERNARY OPERATOR - CONDITIONAL EXPRESSIONS

```
if (condition) {
  myVar = expr1;
}
else {
  myVar = expr2;
}
```

```
myVar = (condition) ? expr1 : expr2
```

exprWhenTrue :

condition ?

```
int x = 2;
(x == 2) ? 5 : 9 * x
5
```

exprWhenFalse.

NOTE: () around first expression

```
int x = 10;
(x == 2) ? 5 : 9 * x

90
```

SHORT CIRCUIT EVALUATION

Only evaluates the *second* operand if necessary.

`true || anything` is always **true**

`false && anything` is always **false**

`!(A && B)` is the same as `!A || !B`

`!(A || B)` is the same as `!A && !B`

De Morgan's laws

`!(x < 5 && y == 3)` is the same as `x >= 5 || y != 3`

If I don't want the case where `x` is less than 5 AND `y` is 3, then I need `x` to be greater than OR equal to 5, or I need `y` to be anything but 3.

CLICKER QUESTION 2

```
((x > 2) || (y < 4)) && (z == 10)
```

Given `int x = 4, y = 1, z = 10` which comparisons are evaluated?

A. `Error`
B. `(x>2), (y<4) and (z==10)`
C. `(x>2) and (z==10)`
D. `(x>2) and (y <4)`

CLICKER QUESTION 2 ANSWERS

```
((x > 2) || (y < 4)) && (z == 10)
```

Given `int x = 4, y = 1, z = 10` which comparisons are evaluated?

A. `Error`

B. `(x>2), (y<4) and (z==10)`

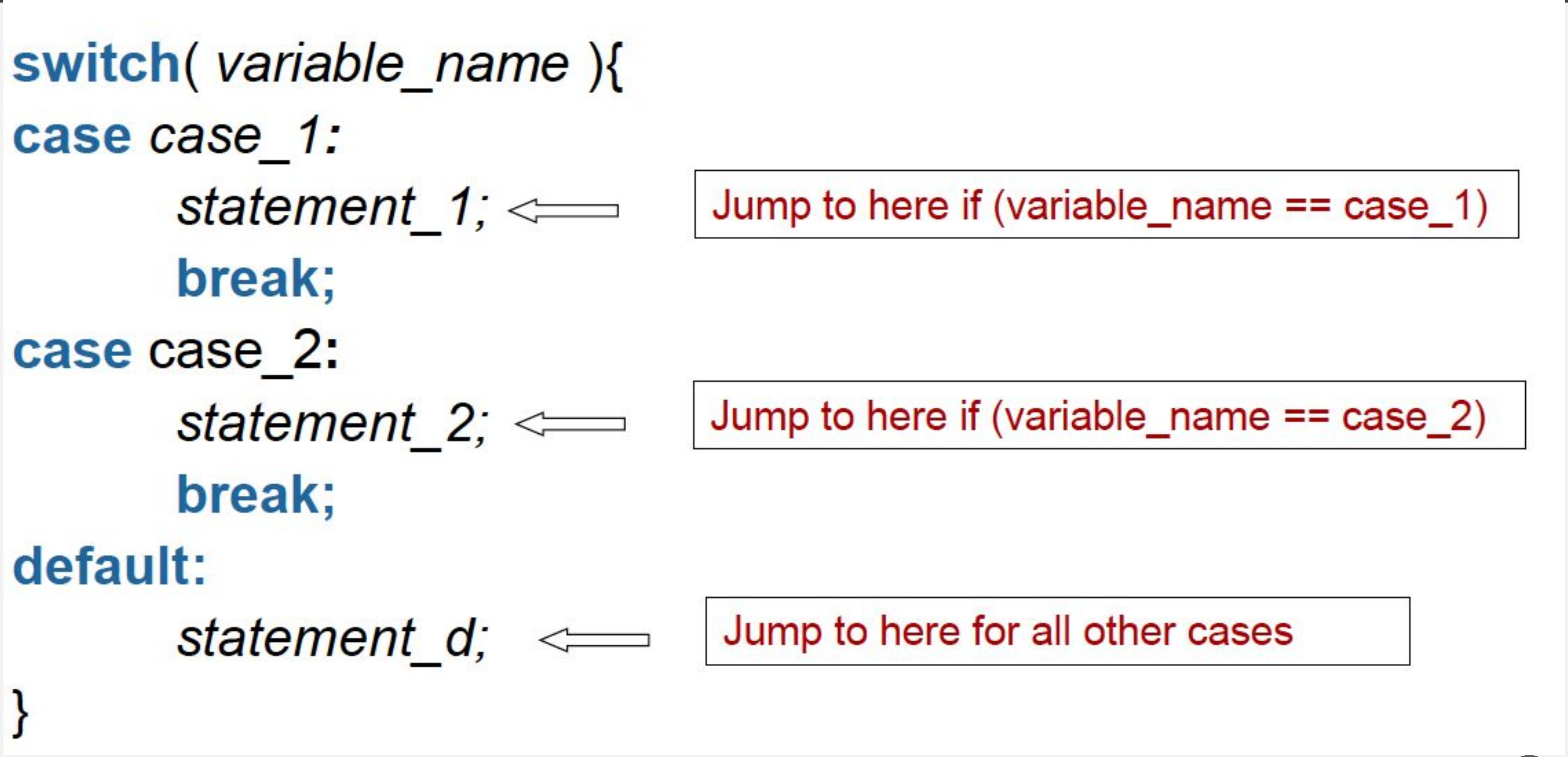
C. `(x>2) and (z==10)`

D. `(x>2) and (y <4)`

`(4 > 2)` is true so OR operator evaluates to true.

`(z == 10)` is evaluated to determine final result.

USING SWITCH STATEMENTS INSTEAD OF MULTIPLE IF STATEMENTS



```
switch( variable_name ){
case case_1:
        statement_1;
        break;
case case_2:
        statement_2;
        break;
default:
        statement_d;
}
```

Jump to here if (variable_name == case_1)

Jump to here if (variable_name == case_2)

Jump to here for all other cases

SWITCH WITH BREAK: EXAMPLE

```
switch(grade) {
       case 'A' :
                 System.out.println("Excellent!");
             break;
        case 'B' :
        case 'C' :
                   System.out.println("Well done");
                 break;
        case 'D' :
                   System.out.println("You passed");
        case 'F' :
                   System.out.println("Better try again");
                   break;
        default :
                   System.out.println("Invalid grade");}
System.out.println("Your grade is " + grade);
```

Break statement

End Switch

SWITCH STATEMENT- KEEP IN MIND

- Duplicate case values are not allowed.
- The value for a case must be the same data type as the variable in the switch.
- The value for a case must be a constant or a literal. Variables are not allowed.
- The break statement is used inside the switch to terminate a statement sequence.
- The break statement is optional. If omitted, execution will continue on into the next case.
- The default statement is optional, and it must appear at the end of the switch.

CLICKER QUESTION 3

```
String island = "Corfu";
switch(island) {
case "Corfu":
    System.out.println("User wants to visit Corfu");
case "Crete":
    System.out.println("User wants to visit Crete");break;
case "Santorini":
    System.out.println("User wants to visit Santorini");break;
default:
    System.out.println("Unknown Island");}
```

What is the output?

A. `User wants to visit Corfu`
   `User wants to visit Crete`

B. `User wants to visit Crete`

C. `Unknown Island`

D. `User wants to visit Corfu`

E. `Error`

CLICKER QUESTION 3 ANSWER

```
 String island = "Corfu";
switch(island) {
 case "Corfu":
   System.out.println("User wants to visit Corfu");
 case "Crete":
   System.out.println("User wants to visit Crete");break;
 case "Santorini":
   System.out.println("User wants to visit Santorini");break;
 default:
   System.out.println("Unknown Island");}
```

A. `User wants to visit Corfu`
   `User wants to visit Crete`

B. `User wants to visit Crete`

C. `Unknown Island`

D. `User wants to visit Corfu`

E. `Error`

2 statements printed as missing `break` after first case.

STRINGS: REVIEW

- A String is made up of a sequence of characters.
- Each character has an index number- it's position in the sequence.
- Example:

W	o	n	d	e	r	f	u	l	.
0	1	2	3	4	5	6	7	8	9

1. The index starts at 0, *not* 1.
2. The String class charAt method returns a char given an index number.

char	**charAt**(int index) Returns the char value at the specified index.

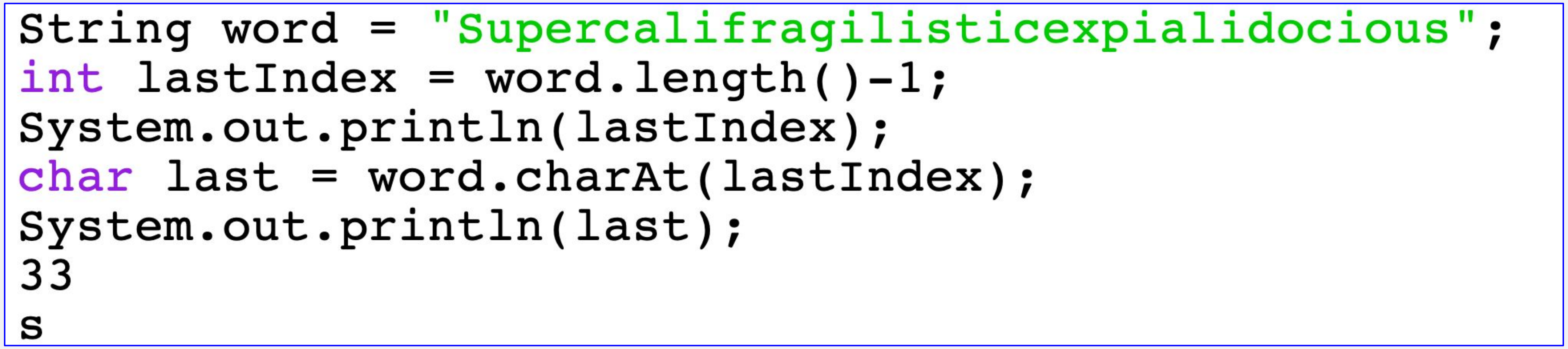
```
String str = "Wonderful.";
char c1 = str.charAt(0);
char c2 = str.charAt(5);
```

c1 is `W`
c2 is `r`

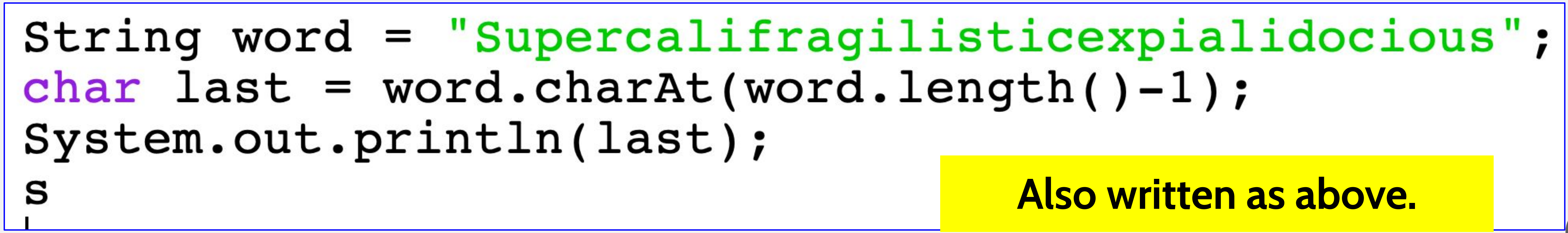
WORKING WITH STRINGS

PROBLEM: Find the last character in the string "`Supercalifragilisticexpialidocious`".

SOLUTION: Find the character at length of string -1.



```
String word = "Supercalifragilisticexpialidocious";
int lastIndex = word.length()-1;
System.out.println(lastIndex);
char last = word.charAt(lastIndex);
System.out.println(last);
33
s
```



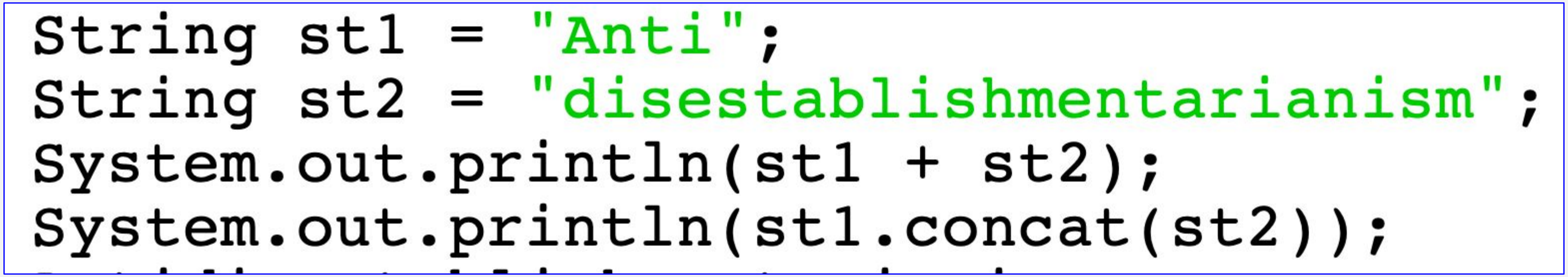
```
String word = "Supercalifragilisticexpialidocious";
char last = word.charAt(word.length()-1);
System.out.println(last);
s
```

Also written as above.

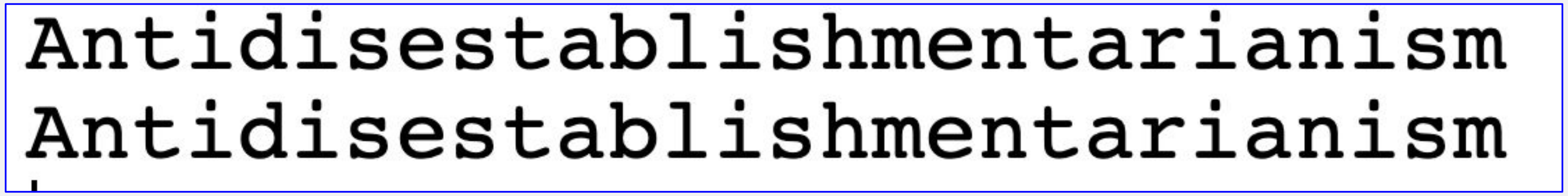
WORKING WITH STRINGS

PROBLEM: How to join 2 Strings (`st1` and `st2`)

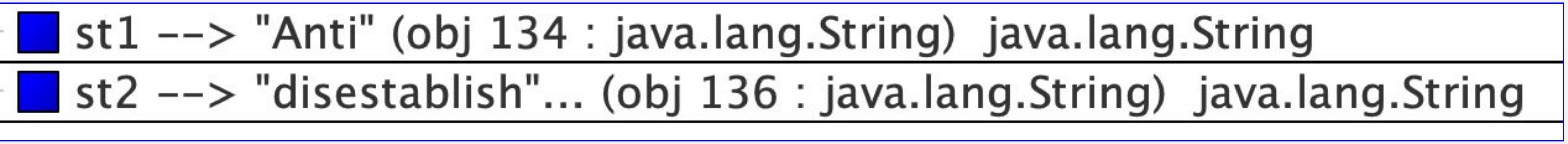
SOLUTION: Use the + operator or use `st1.concat(st2)` to return a new string that appends `st2` to `st1`.



```java
String st1 = "Anti";
String st2 = "disestablishmentarianism";
System.out.println(st1 + st2);
System.out.println(st1.concat(st2));
```



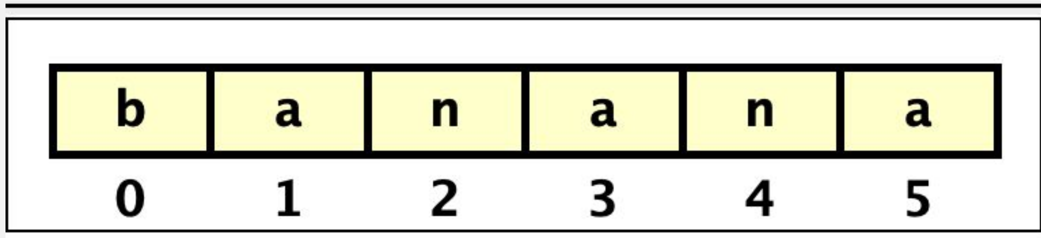
```
Antidisestablishmentarianism
Antidisestablishmentarianism
```



```
🟦 st1 --> "Anti" (obj 134 : java.lang.String)  java.lang.String
🟦 st2 --> "disestablish"... (obj 136 : java.lang.String)  java.lang.String
```

INDEXOF METHOD

```
String fruit = "banana";
int index = fruit.indexOf('a');
System.out.println(index);
index = fruit.indexOf('a', 2);
System.out.println(index);
1
3
```



Where in the string to start looking.

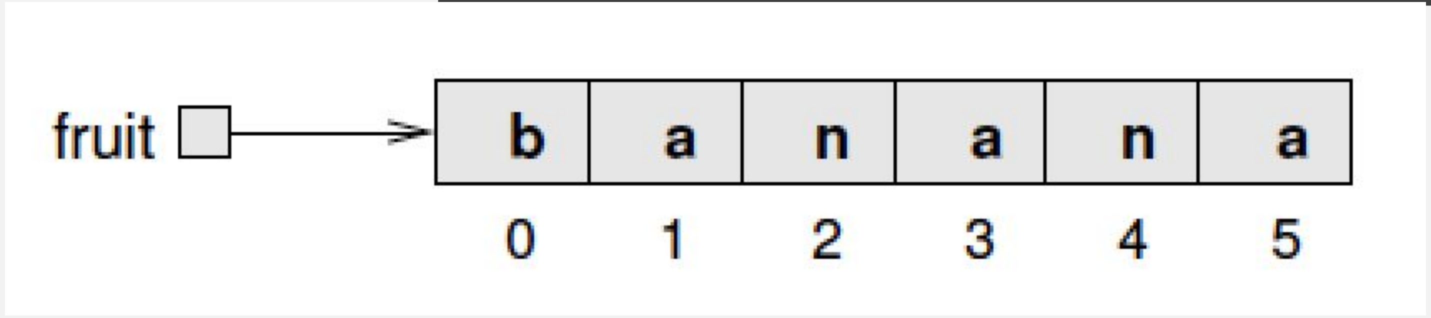
```
String fruit = "watermelon";
char c = 'w';

int index = fruit.indexOf(c);
return (index > -1);
```

true

Also has `lastIndexOf(item)`

SUBSTRING OPERATIONS



```
String fruit = "banana";
System.out.println(fruit.substring(0, 3));
System.out.println(fruit.substring(2, 5));
System.out.println(fruit.substring(6, 6));
ban
nan
```

***substring*(startIndex, endIndex)** returns substring starting at **startIndex** and ending at **endIndex - 1.** Length of the substring is given by **endIndex - startIndex.**

What does `fruit.substring(4, 6)` return ?

CLICKER QUESTION 5

```
 String str = “I ordered the large coffee, not the small tea.”;
 String subStr = str.substring(10, str.length());
```

What is the value of `subStr`?

A. `the large coffee, not the small tea`

B. `the large coffee, not the small tea.`

C. `Error, index out of bounds.`

D. `he large coffee, not the small tea.`

CLICKER QUESTION 5 ANSWER

```
 String str = "I ordered the large coffee, not the small tea.";
 String subStr = str.substring(10, str.length());
```

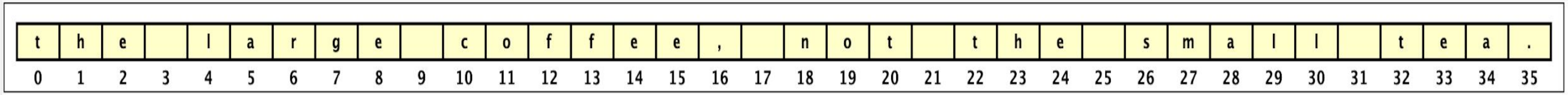
What is the value of `subStr`?

A. `the large coffee, not the small tea`

B. `the large coffee, not the small tea.`

C. `Error, index out of bounds.`

D. `he large coffee, not the small tea.`



t	h	e		l	a	r	g	e		c	o	f	f	e	e	,		n	o	t		t	h	e		s	m	a	l	l		t	e	a	.
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

COMPARING STRINGS

```
String name1 = "Alan Turing";
 String name2 = "Ada Lovelace";
 int diff = name1.compareTo(name2);
 if(diff == 0)
  System.out.println ("The names are the same.");
 if(diff < 0)
  System.out.println ("name1 comes before name2.");
 if(diff > 0)
  System.out.println ("name2 comes before name1.");
```

Return value from `compareTo` is difference between first characters in the strings that differf.

If the strings are equal, their difference is zero.

If the first string (the one on which the method is invoked) comes first in the alphabet, the difference is negative.

Otherwise, the difference is positive.

This is printed: **name2 comes before name1.**

CLICKER QUESTION 6

```
String a = "cow", b = "snake", c = "walrus";
```

```
a compareTo(b) ->
b compareTo(c) ->
c compareTo(a) ->
c compareTo(c) ->
```

A. All negative values
B. All positive values
C. Negative, negative, positive, zero
D. Positive, negative, positive, zero

CLICKER QUESTION 6 ANSWERS

```
String a = "cow", b = "snake", c = "walrus";
```

```
a compareTo(b) -> negative value
b compareTo(c) -> negative value
c compareTo(a) -> positive value
c compareTo(c) -> zero
```

A. All negative values
B. All positive values
C. Negative, negative, positive, zero
D. Positive, negative, positive, zero

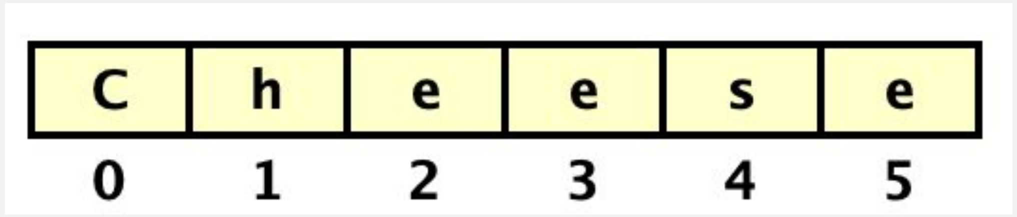
CHARACTER CLASS STATIC METHODS

```
Character.isDigit
Character.isLetter
Character.isLetterOrDigit
Character.isLowerCase
Character.isUpperCase
Character.isWhitespace
Character.toLowerCase
Character.toUpperCase
```

- Must use `Character.methodName`
- Static methods do not need objects to be created.
- All methods return values.

CHARACTER CLASS: EXAMPLE

Each character in a String has an index number.



```
String str = “Cheese”;
char char1 = str.charAt(1);
h
char char2 = Character.toUpperCase(char1);
H
```

CLICKER QUESTION 7

Consider this string:

```
String str = “Hi 357 times!!”;
```

Which one of the following returns true?

```
A.   Character.isWhitespace(str.charAt(6));
B.   Character.isDigit(str.charAt(6));
C.   Character.isLetter(str.charAt(5));
D.   Character.isLowerCase(str.charAt(0));
```

CLICKER QUESTION 7 ANSWER

H	i		3	5	7		t	i	m	e	s	!	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13

```
String str = “Hi 357 times!!”;

Which one of the following returns True?

A.  Character.isWhitespace(str.charAt(6));True
B.  Character.isDigit(str.charAt(6)); False
C.  Character.isLetter(str.charAt(5)); False
D.  Character.isLowerCase(str.charAt(0)); False
```

TO-DO LIST:

- Check your iClicker grades in Moodle.
- Complete zyBook chapter 5 exercises.
- Communicate with us using the Moodle private forum or Piazza.
- Remember to start early on projects :-)