

**Secret Code**

Strings utilize many different aspects of programming: calling methods (dot notation), passing in parameters, indexing, return types, and more. We haven't looked in-depth at them for quite a while – let's review. **For all problems, you may need to review the documentation online or on our website if you don't remember how to use a particular method (e.g. how to use the String class'** indexOf() **method).**

Before you begin writing code, check your understanding:

1. What would "computer science".substring(1) return?
2. What would "computer science".substring(6, 10).indexOf("r") return?
3. What would "computer science".substring(10).substring(2, 4) return?
4. If you are interested in moving to Eclipse, a pro-level IDE with many great features, check the powerpoints. Later, once you have a handle on it, don't forget to check the "More Eclipse tips and tricks" powerpoint for lots of nice time-saving features.
5. Make a class called StringMethods – this class has no instance variables and should have a no-parameter (default) constructor. All the exercises (methods) below will go in this class. Class names always begin with an upper-case letter!
6. Make a class called Runner with a public static void main(String[] args) method. This class should be used to test all the methods in StringMethods.
7. In the Runner class, inside the main() method, create an object of type StringMethods (e.g. StringMethods test = new StringMethods()). Use this object to call the methods of the StringMethods class (e.g. test.unusualHello("Ron")).
   1. The main method is the 'start button' that Java looks for, to begin execution. Make sure to review the main method powerpoint, even if you're not using Eclipse.
8. Complete the method public void unusualHello(String name), that will print a greeting and "you dummy!" added to the supplied nameparameter.

unusualHello("Bob") >>> Hello Bob, you dummy!

1. Complete the method public String stringRipper(String str), that will return a String containing only the first and last letters of str.

stringRipper("something") >>> "sg"

1. Complete the method String mirrorEnds(String s) that will return the "mirror image" string at the beginning and end of s. In other words, zero or more characters at the very beginning of the given string, and at the very end of the string in reverse order (possibly overlapping).

mirrorEnds("abXYZba") >>> "ab"

mirrorEnds("abcd") >>> ""

1. (Riddle) You are given two ropes and a lighter. Each of the ropes has the following property: *if you light one end of the rope on fire, it will take exactly one hour to burn all the way to the other end.* However, the ropes don't burn at a uniform rate. In other words, half the rope may burn in the first five minutes, and then the other half would take 55 minutes.

The rate at which the two ropes burn is not necessarily the same, so the second rope will always take an hour to burn from one end to the other, but may do it at some varying rate, which is not necessarily the same as the one for the first rope. **How would you accurately measure exactly 45 minutes?**

1. Complete the method public boolean hasSub(String str, String sub), that will return true if sub can be found in str (case sensitive) WITHOUT using the contains() method.

hasSub("Computer science", "sci") >>> true

1. Complete the method public boolean evenFooBar(String str), that will return true if "foo" and "bar" appear the same number of times in the String parameter str.

evenFooBar("foofoobarbarfoo") >>> false

evenFooBar("foobarbarfoo") >>> true

1. Complete the method public String microwaveTime(String str), that takes a String of numbers representing time and converts it to "microwave time" – adding a colon and returning a String in the format **minutes:seconds**. A zero should be added before Strings with two or less numbers. Strings with less than 1 or more than 4 numbers should return "invalid input".

microwaveTime("12") >>> "0:12"

microwaveTime("123") >>> "1:23"

microwaveTime("4500") >>> "45:00"

1. Complete the method public String reverseWords(String str), that will return a String where the order of each word in str has been reversed, word by word. The String class' split() method may be useful for this.

reverseWords("hello world") >>> "world hello"

reverseWords("the sky is blue") >>> "blue is sky the"

1. Complete the method public int sumString(String str), that will "scan" str and sum all the (whitespace separated) integers that can be found in the String. Use a Scanner object to tokenize and parse the input (check the "More on Scanners" powerpoint, in the AP folder).

sumString("Hi 5 there are 2 or 3 numbers in this String") >>> 10

**Secret code**

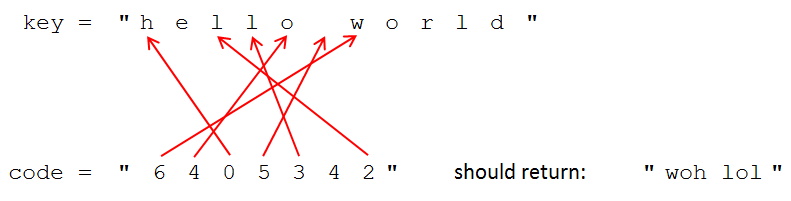
Make a new class called SecretCode, with the following:

**Instance variables**

* String key - a String "key phrase" that contains letters, which will be used in decoding
* String code - a String containing a series of integers which represent the indexes in key

**Methods**

* A two-parameter constructor to initialize key and code
* A method public String decode() that will return a "decoded" message. The message has been encoded in the following manner: each integer in code is an ***index*** pointing to a location in key.To decode the message, simply look up each integer in key and output the corresponding character. Example, if key = "hello world" and code = "6 4 0 5 3 4 2":



To test your method, construct a new SecretCode object in the Runner class with the following inputs:

String somePhrase = "six perfect quality black jewels amazed the governor"

String code = "35 10 10 33 9 24 3 17 41 8 3 20 51 16 38 44 47 32 33 10 19 38 35 28 49"

If your method is written correctly, it should return a *covert message*. You must use a Scanner object to parse the String (rather than split() and Integer.parseInt())! Scanner objects will help on this and future labs.

**(Advanced) Longest substring**

*Problems in red with an* ***(Advanced)*** *tag are not required to get 100 points but are interesting problems of a challenging nature. Give them a shot if you have time and like a challenge!*

Given a String str, find the ***length*** of the longest substring without repeating characters.

"abcabcbb" >>> 3 //"abc"

"bbbbb" >>> 1 //"b"

"pwwkew" >>> 3 //"wke"

**(Advanced)** **Has pairs**

Complete the method: public boolean hasPairs(String s, char c), that returns true if, for every occurrence of character c, there is another c to its right or left. This problem is slightly harder than it appears and will take some creative problem solving.

hasPairs("aaccaa", 'a') >>> true

hasPairs("aacaa", 'c') >>> false

hasPairs("aaccddadd", 'a') >>> false