**Secret code, part 2**

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.

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

String key = "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!*

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