// to use the scanner  
**import** java.util.Scanner;  
  
// the main methode for the encryption to run  
**public class** stringencryption {  
  
 // the main codes for the assignment  
 **public static void** main(String[] args) {  
  
 // to use the scanner  
 Scanner scanner = **new** Scanner(System.*in*);  
  
 // this loop: is for the continue statement within the inner loop so that it doesn't confuse which loop to follow  
 loop:  
 // the loop for the encryption (but actually, it just encrypts one thing despite the repeats)  
 **while** (**true**) {  
  
 // interactive platform messages  
 System.*out*.println("\n ------------------------------------- " +  
 "\n Please enter your message to be encrypted (strings only) :)");  
  
 // user can input string/ strings  
 String message = scanner.nextLine();  
  
 // same logic as the one bellow (added this after the "innerloop")  
  
 verify:  
 **while** (**true**) {  
 **if** (*isString*(message)) {  
 **break** verify;  
 } **else** {  
 System.*out*.println("Please enter STRINGS only, thank you.");  
 **continue** loop;  
 }  
  
 }  
  
 // not only can the user verify, I can too -that the algorithm got all the strings  
 System.*out*.println("\n ----------------------------------- \n" +  
 "Please confirm that you want to encrypt: " + message + "\n" +  
 " >> Y/N <<");  
  
 // inner loop for the yes and no or other answers  
 **while** (**true**) {  
 // user can input Y/y N/n for the following if statements  
 String yesorno = scanner.next().toLowerCase();  
 scanner.nextLine();  
  
 /\*  
 if the String the user wants to encrypt is correct, it will be encrypted  
 by pulling from the reverse input methode in the private static methode  
 Here is also where the computer reiterate the original message and creates the encrypted message  
 Then, the code will come to an end with break;  
 \*/  
 **if** (yesorno.equals("y")) {  
 String encryptedMessage = *reverseString*(message);  
 System.*out*.println("\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n" +  
 "Your encrypted message: " + encryptedMessage);  
 System.*out*.println("........................ \n this was your original message:" +  
 "\n" + message);  
 **break**;  
 }  
 /\* if the user does not want to encrypt the message re-outputted,  
 the code will repeat from the "loop:" aka the outer-loop \*/  
 **else if** (yesorno.equals("n")) {  
 **continue** loop;  
 }  
 // any other input will be rendered false and so will continue in the inner loop  
 **else** {  
 System.*out*.println("I do not understand. Please let me know 'Y' or 'N'");  
 **continue**;  
 }  
 }  
  
 scanner.close();  
 }  
 }  
  
 /\*  
 String is the return and input type:  
 input is the parameter that the method accepts --> for the string's "input" that I want to reverse  
 by using for, from the input.length, from the last string to the first (i=0)  
 then by using the "reverse function" I can re-generate the input from the last to the first character  
 \*/  
  
 /\*  
 StringBuilder is commonly used to efficiently construct strings that may be:  
 1) appending  
 2) inserting  
 3) deleting  
 4) or replacing characters...etc.  
 In this case it would be replacing characters in the reversed order  
  
 It explains that StringBuffer serves a similar purpose;  
 but Builder is commonly more preferred because it is not synchronized,  
 meaning - not designed to handle multiple threads (downside is not handling threads "safely")  
 \*/  
  
 **private static** String reverseString(String input) {  
 StringBuilder reversed = **new** StringBuilder();  
 **for** (**int** i = input.length() - 1; i>=0; i--){  
 reversed.append(input.charAt(i));  
 }  
 **return** reversed.toString();  
 }  
  
 // make sure that the inputs are string and strings only + AND THAT SPACES ARE OK  
 **public static boolean** isString(String input) {  
// String [] words = input.split(" +");  
//  
// for (String word : words) {  
// if (!word.matches("^[a-zA-Z]+$")) {  
// return false;  
// }  
// }  
  
// return true;  
 **return** input.matches("^[a-zA-Z ]+$");  
 }  
}