Loops and Strings:

Recall that String is a class that represents text. A String object can not be changed once created.

Some useful methods and operators:

+: creates a new String that is the result of concatentating two Strings

length(): returns the number of characters in the String

charAt(5): returns the 6th character of the String. (In Java, the first character is at index 0.)

So,

String s = "Hello"

s.length() => returns 5

s.charAt(1) => returns 'e'

"".length() => returns 0

"Hello".charAt(0) => returns 'H'

"".charAt(0) => Error

"Hello".charAt("Hello".length()) => Error because it asks for the index 5 character in a string of only 5 characters.

"Hello there".charAt(5) => returns ' '

("hello" + "there").length() => returns 10

Example 1: Write a method that determines if a String is a palindrome (the same forward as backwards). For example "radar" is a palindrome.

The trick is to first think how YOU would do it. What are the different steps you need?

- You read the string backwards and compare with how it is read forwards.

- That means you first compare the first character with the last, then the 2nd with the 2nd to last, and so on.

So, we need one index to keep track of which character from the front we are looking at:

The test we need is:

if (s.charAt(index) == s.charAt(s.length() - 1 - index)) // remember that we are indexing from 0

index = index + 1; // then keep going

else

return false; // it can't be a palindrome

This is awkwardly structured because it is not immediately obvious that index is always incremented. Since the index increment is part of the loop, it should occur outside the if statement.

However, that will leave us with a "do nothing" statement. It is cleaner to reorganize the code to avoid these.

if (s.charAt(index) != s.charAt(s.length() - 1 - index)) // remember that we are indexing from 0

return false; // it can't be a palindrome

index = index + 1;

Now, when do we stop? When index reaches the middle: that is, when index >= s.length() / 2

public static boolean isPalindrome(String s) {

int index = 0;

while (index < s.length() / 2) {

if (s.charAt(index) != s.charAt(s.length() - 1 - index)) // remember that we are indexing from 0

return false;

index = index + 1;

} // if we reach the end of the loop, index == s.length() / 2 and we never encountered a mismatched pair

return true; // thus we must have a palindrome.

}

Here is the same thing, but as a for loop:

public static boolean isPalindrome(String s) {

for (int index = 0; index < s.length() / 2; index = index + 1) {

if (s.charAt(index) != s.charAt(s.length() - 1 - index)) // remember that we are indexing from 0

return false; // it can't be a palindrome

}

return true;

}

Example 2:

Write a method that capitalizes every lower case letter in a string.

How would we do it?

Look at each character one at a time. If it is lower case, capitalize it. If it is not lower case, keep it.

Note that we can not change the String so we must create a new String that is the same as the original, but with capital letters.

That means we need to build a string. We may be tempted to use the + operator:

String result = "";

for (....) {

....

result = result + c;

....

}

return result;

However, this is not a good solution. The + operator creates a new String each time. If we are dealing with a very long String, such as if we want to capitalize a DNA code of millions of characters,

we will be creating a LOT of unnecessary Strings and using up our memory.

Java provides a StringBuilder class to create Strings.

StringBuilder has all the same methods as String (charAt, length, etc) plus several others.

Once useful one is append that adds new characters (or other values) to the end of the string being created.

How to create an empty StringBuilder? Just like you create any other initial instance: StringBuilder result = new StringBuilder();

What do we return at the end? We can't return result because it is not a String. But Object has a method that returns a String representation, and every class inherits it from Object.

result.toString();

Finally, how to we test if letters are lower case, and how to make them upper case?

Remember that char is a primitive type, and a char is just a number. Let us treat them as numbers. The only problem is that applying a primitive arithmetic operator will result in an int type

so we will need to remember to typecast back to char.

public static String capitalize(String s) {

StringBuilder builder = new StringBuilder();

for (int index = 0; index < s.length(); index = index + 1) {

char c = s.charAt(index);

if (s.charAt(index) >= 'a' && s.charAt(index) <= 'z')

c = (char)(s.charAt(index) - 'a' + 'A');

builder.append(c);

}

return builder.toString();

}

}