PRG (ETS de Ingeniería Informática) - Academic year 2019-2020

Lab practice 2. Using recursion for solving problems.

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Contents

1 Context and previous work

In this practice, two problems for manipulating objects of the class `String` are proposed to be solved. The corresponding methods and test classes will be designed to ensure that the solutions to the problems are correct.

It is recommended to previously study section 10.6 *Recursividad con objetos de tipo* ***`String`*** of the third edition of the book for this subject[1] and to be sure you understood some of the examples as it is the problem for counting the number of occurrences of char `'a'` in a `String s`.

Activity 1: Creating the package `pract2` in BlueJ

Open the *BlueJ* project of the subject (`prg`) and create a new package named `pract2`. Add to the package the file `PRGString.java` that you have previously downloaded from the folder `Resources / Laboratory / Practice 2` of the site PoliformaT for PRG. The class `PRGString` is a class of utilities that includes the methods that solve the problem of counting the number of `'a'`s in a `String s` (section 10.6 of the book) and the methods (to be completed) that solve the problems that are posed to you next.

[1] If you have the second, then section 11.6.

2 Problem A. *Prefix*

Given two `Strings` `a` and `b`, potentially empty, it says that `a` is *prefix* of `b` when all the characters of `a` are consecutive at the beginning from `b` in the same original order.

The consequence of the above definition is that the *empty string* is prefix of any other, even if that other one was also empty. Note, on the other hand, that one string can not be a prefix of another if the first is longer than the second.

Activity 2: method `isPrefix(String, String)`

Write the code for the recursive version of method `isPrefix(String, String)` to check if a String is prefix of another one. In order to do it properly, you have to:

- define trivial cases and general cases of the recursion, and
- define the solution of the problem in all possible cases. The profile of the method should be exactly the following one:

```
public static boolean isPrefix(String a, String b)
```

- Put the appropriate comments in the method, describing the parameters, the type of the result and the preconditions if it is the case.
- Check the code follow the style by using *Checkstyle* of *BlueJ* and correct it.

Activity 3: validation of method `isPrefix(String, String)`

Write a program class `TestIsPrefix` that executes the method with different data to check whether there are execution errors and that the returned result in each case is the correct one.

The data to be tested must reflect the different situations that may occur in the execution of the method, such as, for example: (a) that both strings are empty, (b) that only one of them is empty, (c) that the first string is longer than the second one, (d) that the second string is prefix of the second one, etc. The following table shows different cases with specific instances and the expected result for each case.

Case	`a`	`b`	**Result**
`a` and `b` are empty	`""`	`""`	`true`
`a` empty	`""`	`"recursion"`	`true`
`b` empty	`"recursion"`	`""`	`false`
`a` longer than `b`	`"recursion"`	`"rec"`	`false`
`a` and `b` equally long and `a` is prefix of `b`	`"recursion"`	`"recursion"`	`true`
`a` and `b` equally long and `a` is not a prefix of `b`	`"123456789"`	`"recursion"`	`false`
`a` shorter than `b` and `a` is prefix of `b`	`"rec"`	`"recursion"`	`true`
`a` shorter than `b` and `a` is not prefix of `b`:			
- fails the first character	`"pecur"`	`"recursion"`	`false`
- fails the last character	`"recurso"`	`"recursion"`	`false`
- fails any intermediate character	`"remursi"`	`"recursion"`	`false`

The class `TestIsPrefix` should include a method with the following profile:

```
private static void testIsPrefix(String a, String b)
```

for showing on screen both strings used for the test, and the result returned by the method `isPrefix(String, String)` and the expected result. For obtaining the correct expected result you can use the method `startsWith(String)` from the `String` class.

The method `main()` should invoke the method `testIsPrefix(String, String)` for each test case. You can define an array of objects of the class `String` for storing the different instances of the problem to be checked.

3 Problem B. *Contained*

Given two `String`s `a` and `b`, which can be empty, `a` is *contained* in `b` when all the symbols of `a` appear in the same consecutive order in any place within `b`. In other words, that `a` is prefix of `b` or of any of the possible substrings of `b`.

Obviously, as in the case of `isPrefix (String, String)`, you can see that the *empty string* is contained in any other, even if the other one was also empty. Also, a string can not be contained within another one if the first is longer than the second.

Activity 4: method `isSubstring(String, String)`

The recurrence relation for method `isSubstring(String)` should be defined **in terms of using** `isPrefix(String, String)`, in order to check if a string is substring of another one. Steps to be done:

- Describe the trivial cases and the general cases of the recursion by defining the solution to the problem in each case. The profile of the method should be the following one:

  ```
  public static boolean isSubstring(String a, String b)
  ```

  Take into account that for invoking the method `isPrefix(String, String)` there are no positional parameters, i.e., no indexes are used.
- Write the appropriate comments in the method, describing the parameters, the result, and if it is the case, the precondition.
- Check the code follows the style by using the *Checkstyle* option of *BlueJ* and correct your code if it is the case.

Activity 5: validation of method `isSubstring(String, String)`

Write a program class `TestIsSubstring` that allows to execute the method with different data to verify that there are no execution errors and that the returned result in each case is correct. As before, you should identify the different situations that can occur, proving that, in all of them, the method works properly. In this case, you can compare the result with the method `contains(String)` of the class `String`.

4 Evaluation

This practice is part of the first block of practices of the subject that will be evaluated in the first part of it. The value of said block is 50% with respect to the total of the practices. The percentage value of the practices in the subject is 25% of the final grade.