PRG (E.T.S. d'Enginyeria Informàtica)
Academic year 2014-2015
Lab practise 2 – Problem solving by means of recursion

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Contents

1 Context and previous work

This lab practise is the second one in relation with the unit 1 *Recursion.* Here, you will find some problems you have to solve by using the recursive design. Moreover, you should prepare the test cases needed to exhaustively check that the solutions you provide are correct.

In order to take the maximum profit during the lab session, you should do a comprehensive reading of this guide. We suggest you trying to understand all the posed problems. Please, be aware that each problem requires a previous analysis that must be done before attending the lab session. It is recommended that you read the available methods of the class `String`.

2 An example of recursive design on `String` objects

Given a string `s` we need to count how many times the char `‘a’` appears within `s`. Thinking recursively we obtain:

- Trivial case: `String s` is empty, so the number of `‘a’` is zero.
- General case: `String s` is not empty, so:
  - if the first char in `s` is `‘a’`, then the total amount of `‘a’` in `s` is 1 plus the amount of `‘a’` contained in the substring of `s` without the first char.
  - if the first char in `s` is not `‘a’`, then the total amount of `‘a’` in `s` is equal to the amount of `‘a’` contained in the substring of `s` without the first char.

A possible implementation of the previous recursive analysis is the following recursive method:

```
/**
 * Returns the amount of ‘a’ in the String received as input parameter.
 * @param s. String object for counting the appearances of char ’a’.
 * @return int.
 */
public static int countA( String s ) {
    // Trivial case: Empty string
    if ( s.length() == 0 ) return 0;
    // General case: Non empty string.
    else if ( s.charAt(0) == ’a’ ) return 1 + countA( s.substring(1) );
    else return countA( s.substring(1) );
}
```

In order to compare different approaches, this problem can be solved in a way similar to the solutions provided for some problems stated in the classroom, i.e., by using an additional parameter as index for delimiting the portion of the string that remains to be processed. The case analysis is like the previous one, but you can see in the implementation how the new parameter `pos` should be used:

```
/**
 * Returns the amount of ‘a’ in the String received as input parameter.
 * @param s. String object for counting the appearances of char ’a’.
 * @param pos. Index within s where the substring starts.
 * @return int.
 * PRECONDITION: pos >= 0
 */
public static int countA( String s, int pos ) {
    // Trivial case: Empty string
    if ( pos >= s.length() ) return 0;
    // General case: Non empty string.
    else if ( s.charAt(pos) == ’a’ ) return 1 + countA( s, pos+1 );
    else return countA( s, pos+1 );
}
```

It can be performed one more analysis, very similar to the previous ones and with a very similar solution, but considering the last char in the string instead of the first one. This implies that the substring must be obtained removing the last position.

The following method is a possible implementation of this idea. The case analysis is in the comments inside the source code.

```
/**
 * Returns the amount of ‘a’ in the String received as input parameter.
 * @param s. String object for counting the appearances of char ’a’.
 * @return int.
 */
public static int countA( String s ) {
    // Trivial case: Empty String
    if ( s.length() == 0 ) return 0;
    // General case: Non empty string.
    else if ( s.charAt( s.length()-1 ) == ’a’ )
        return 1 + countA( s.substring( 0, s.length()-1 ) );
    else
        return countA( s.substring( 0, s.length()-1 ) );
}
```

Remind that `s.substring(i,j)` is an `String` object representing the substring of `s` formed by the chars between positions `i` and `j-1`, `s.charAt(j)` doesn’t belong to the substring.

3 Lab work. Environment

Students should create a specific BlueJ project for this lab practise and create a class named `PRGString` for including the methods to be implemented for solving the following problems.

4 Problem A. *Prefix*

Given two `String` objects `a` and `b`, that can be empty, `a` is a prefix of `b` if all the chars in `a` are in the same order at the beginning of `b`.

By definition, the empty string is considered as prefix of any other string. The empty string is a prefix of the empty string. A longer string will never be a prefix of a shorter one.

4.1 Starting activities

Students should define a recursive method `isPrefix()` for checking if one string is prefix of another one, establishing properly the trivial and general cases, and describing the solution for both. The header of the method must be:

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

Please, put attention in the fact that there are no additional parameters for indexing within the strings.

4.2 Lab activities

1. Write the method `isPrefix()` taking into consideration your previous analysis.
2. Put comments in the method, describe their parameters and the returning data type, and specify preconditions if they are necessary.
3. Check the proper operation of your method. You can do the following steps:
   - Write a list with all the possible combinations that can appear during the execution of the method, such as: both strings are empty, only one of both is empty, the first one is longer than the second one, the contrary, the first one is prefix of the second one, or not, etc.
   - Check if your method operates well in every anticipated situation by using BlueJ. As an additional checking, you can compare the behaviour of your method with the behaviour of the method `startsWith( String )` of the `String` class.

5 Problem B. *Substring*

Given two `String` objects `a` and `b`, that can be empty, it can be said that `a` is a substring of `b` if all the chars in `a` are contained in `b` at any position and in the same order they appear within `a`. If `a` is contained in `b` from the first position (index equal to zero) then `a` is also a prefix of `b`. If it is contained at any other position it can be said that `a` is the prefix of some of the possible substrings of `b`.

As in the previous problem, `isPrefix()`, the empty string is also a substring of any string. The empty string is a substring of the empty string. A longer string cannot be a substring of a shorter one.

5.1 Starting activities

Students should recursively define the method `isSubstring()` by using the previously defined method `isPrefix()`, enumerate the trivial and general cases for the recursion, and define the solution for each case. The header of the method must be:

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

As in the previous problem you can realise that there are no additional parameters for specifying positions inside the strings.

5.2 Lab activities

1. Write the method `isSubstring()` from your analysis.
2. Put comments and describe the parameters and the return value, and specify preconditions if necessary.
3. Check if your method runs correctly by enumerating all the possible combinations and testing each one. Additionally you can compare the behaviour of your method with the behaviour of the method `contains(String)` defined in the `String` class.

6 Problem C. *Palindromes*

A string is palindrome if it can be equally read in both senses, from left to right and *vice versa*. Some examples from Spanish:

```
Eva usaba rimel y le miraba suave.
Se dice detrás:  a casar te decides.
La ruta nos aportó otro paso natural.
Dábale arroz a la zorra el abad.
```

Please, realise how white spaces, punctuation symbols and accents are no relevant for determining whether a sentence is palindrome. The difference of upper and lower case letters are no relevant either. In order to simplify the problem you can consider only the letters for checking if a string is palindrome.

6.1 Lab activities

Students should design a Java recursive method for checking whether the sentence contained in a `String` object is palindrome. The header of the method must be:

```
public static boolean isPalindrome( String a )
```

For solving this problem it is recommended to follow the same steps done for the previous problems, that in summary are:

- Defining recursively the solution to the problem.
- Writing comments in the solution.
- Checking if the obtained method runs correctly for all the possible instances of input data.

Note: The method must no perform a preprocessing of the string, i.e., it must do all the necessary operations just in a unique traversal through the string.

For simplicity, consider only the alphabetic characters as the relevant ones for testing whether a sentence is palindrome. These are the letters from `'a'` to `'z'` and `'ñ'`, independently if they are in upper or lower case. The remaining characters can be considered as white spaces, i.e., must be ignored. It is easy to check in Java if a `char` is a letter by means of the static method `Character.isLetter( char )`.

By the other hand, you can consider that initially the sentences do not contain accentuated letters. Once your method run well then improve it for accepting accentuated letters.

7 Evaluation

This lab practise is part of the first block of PRG to be evaluated in the first partial exam. The weight of this block is the 40% of the lab practise grade. And the lab practise grade is the 20% of the PRG final grade.