Software Development I – Exercises **Winter Term 2012/2013**

Übungen zu Softwareentwicklung 1 **Assignment 10**

Name: ______________	Teaching Assistant: ______________
Student ID (Matr.Nr.): ______________	Points (max. 24): ______________
Group: ______________	Deadline: Fri, Jan. 25th, 2013 12:00 noon
Instructor: ______________	Editing time: ______________

Question 1: Find Your Way **10 points**

Write a program that allows searching for a passage through a maze. The maze is a character matrix where the symbol '_' indicates a field where you may travel and the symbol "#" indicates that this field is blocked. (You can use a static maze in your solution, e.g. **`static final char`** `[][] maze = {...}`.)
You shall start off your journey from a certain starting point assuming that the starting point lays to the west of the final destination. You may move through the maze either by stepping to the east (right), to the north (up) or to the south (down) (i.e., three recursive calls!), naturally assuming that you may not move outside the boundaries of your maze.
If you found your passage across the maze from the start to the destination, indicate the path traversed in the maze with '+' symbols.

An example of a dialog with your program can be found here:

```
The Maze
--------------
|________#___|
|________##__|
|________#___|
|________#_#_|
|___#____#_#_|
|______#___#_|
|______#_____|
|________###_|
|_____#______|
--------------
Find a passage from start [x]: 0
Find a passage from start [y]: 0
To destination [x]: 8
To destination [y]: 11

A passage from [0,0] to [8,11] could be found!
The passage through the maze is:
-------------
|++++++++#___|
|_______+##__|
|_______+#___|
|_______+#_#_|
|___#___+#_#_|
|______#+++#_|
|______#__+++|
|________###+|
|_____#_____+|
|-------------
```

Provide a recursively working function **`findPassage(Position start, Position destination)`** to return whether a passage exists from start to destination through different mazes you define.
Declare and use a **`Position`** class to manage coordinates in the maze. Beside constructors and setter/getter methods, this class should offer (at least) the following functions:

```
class Position {
    private int x;
    private int y;

    public Position up () { }          // Returns the next position north to the current position
    public Position down() { }         // Returns the next position south to the current position
    public Position right() { }        // Returns the next position east to the current position

    public boolean outOfBounds() { } // Checks if the current position is outside the maze
    public boolean blocked()  { }    // Checks if the current position is blocked
}
```

Write the program in Java. Provide all the material requested below at the very bottom ("Requested material…")

Question 2: My Little Pattern Matcher — 14 points

Write a program, which allows to search for a certain pattern in a given text string. The following rules shall be applied for special characters:

- **?** … one arbitrary character, e.g., "a" or "9"
- **.** … one single text character (a-z, A-Z), e.g. "a", "A"
- **#** … one single number character, e.g. "1", "9"
- ***** … a series of arbitrary characters, minimal 0 characters, e.g., "", "a", "abc"
- **+** … a series of arbitrary characters, at least one, e.g., "a", "ab"
- **$** … a series of number characters, at least one, e.g., "1", "12"

Any other character shall be a match as is (cf. examples below).

Provide a recursively working function **`isMatching(String text, String pattern)`** to return whether the given text matches the given pattern.

Implement the pattern matcher on relying on character comparison only without using predefined Java-functionality for pattern matching.

Examples

a) with non-special characters

- `isMatching ("abc", "abc");    => true`
- `isMatching ("abc", "abcd");   => false`
- `isMatching ("abcd", "abc");   => false`

b) with special characters

- `isMatching ("abc", "*");      => true`
- `isMatching ("abc", "+");      => true`
- `isMatching ("", "+");         => false`
- `isMatching ("123", "*#");     => true`
- `isMatching ("123", "+#");     => false`
- `isMatching ("123", "$");      => true`
- `isMatching ("123", ".$");     => false`
- `isMatching ("123.", "$.");    => true`
- `isMatching ("!Hugo12344!", ".Hugo###$?!");   => true`

Write the program in Java. Provide all the material requested below at the very bottom ("Requested material…")

Requested material for all programming problems:

- **For each exercise, hand in the following:**
  a) **The idea for your solution written in text form**
  b) **Source code (Java classes including English(!) comments)**
  c) **Test plan for analyzing <u>boundary values</u> (e.g., minimal temperature allowed, maximum number of input, etc.) and <u>exceptional cases</u> (e.g., textual input when a number is required, etc.). State the expected behavior of the program for each input and make sure there is no “undefined” behavior leading to runtime exceptions. List all your test cases in a table (test case #, description, user input, expected (return) values).**
  d) **The output of your java program for all test cases in your test plan**
- Pay attention to using adequate and reasonable data types for your implementation, check the user input carefully and print out meaningful error messages.