Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme

Prof. Dr. Reinhard von Hanxleden
Christoph Daniel Schulze, Steven Smyth

Institut für Informatik

3. Übung zur Vorlesung „Programmierung“

Third Installment: Statements
Abgabe am Sonntag, 20. November 2016 - 23:55

As a reminder, from the class homepage:

If your solution does not pass an automatic test, or if it does not contain the descriptive comments asked for, then it will not receive full credit. If this happens on a mandatory problem, this means that you risk failing that problem set.

If you have more than two failed problem sets, you will not be admitted to the final exam.

Aufgabe 1 - What Are The Odds? (Mandatory)

49 Punkte

Computers are often used to crack hard numerical problems. Since we're an introductory programming course, however, we'll start with slightly easier ones that will require you to use loops and conditionals.

Note: To satisfy the tests, the results computed by your programs should always be printed as the last line of output of your programs. No additional words, no punctuation; just the number. What you print before the last line is up to you.

(a) Write a program `DigitalRoot` that extends `ConsoleProgram`, asks the user for a non-negative number $n \geq 0$, and computes its *digital root* (if the number is negative, the program should simply output the string `Error` instead of a number). The digital root is computed by summing all the digits of the number, then summing up all the digits of the result and so on until there is only one digit left (which is the digital root). That single digit should be the output of your program. You can find a program that computes the sum of all the digits of a number in Chapter 4 in our book.

Note: The Wikipedia article on digital roots gives a formula that directly computes the digital root. For our intents and purposes, that's a bit boring. We actually want you to compute it with loops, using the code from Chapter 4.

An example trace could be:

```
Enter a number: 12345
6
```

(b) Computer scientists love the *Fibonacci sequence*, a sequence of numbers that follow a specific pattern. The first two numbers are 1; the next number in the sequence is the sum of the previous two. Thus, the sequence starts as follows: 1, 1, 2, 3, 5, 8, etc. Write a program `Fibonacci` that extends `ConsoleProgram`, asks the user for a number n, and outputs the n-th number in the Fibonacci sequence (that is, the output of your program for $n = 6$ should be 8). If the user enters a number smaller than 1, the program should simply output the string `Error` instead of a number.

An example trace could be:

```
Enter a number: 8
21
```

Aufgabe 2 - Sentinel Expressions (Mandatory) 49 Punkte

Celsius is a perfectly reasonable unit for measuring temperatures: 0°C is the freezing point of water, 100°C is the boiling point of water (under a certain set of conditions). However, certain countries still think that *Fahrenheit* is the better alternative. Which is why in this assignment you will write a program to convert from Fahrenheit to Celsius and back. The relationship between the two units is this:

$$C = \frac{5}{9}(F - 32)$$

Part (a) Write a program `TemperatureConverter` that extends `ConsoleProgram`. The program begins by asking the user whether to convert Celsius to Fahrenheit (in which case the user needs to enter a `1`) or Fahrenheit to Celcius (in which case the user needs to enter a `2`). The program then asks for the temperature and outputs the converted temperature. This is repeated until the *sentinel* aborts the loop. In our case, a temperature of **-1000** acts as a sentinel value. **Comment all *expressions* (except strings and method calls) in your program and name the *precedence* rules that apply.** Here is a sample run of what this should look like:

```
What do you want me to do?
(1) Convert Celsius to Fahrenheit
(2) Convert Fahrenheit to Celsius
Mode: 2
Temperature: 41
5.0
Temperature: 32
0.0
Temperature: -1000
Goodbye!
```

Hints on the tests:

- You don't need the first three lines, but if you do print them they must look exactly as above.
- The first value you read (the mode) should be an 'integer' and the following values (the temperatures) should be 'doubles'.
- When reading these values, the display should show the same texts as above.

Part (b) In the lecture you have seen how *context-free grammars* are defined and how Java specifies `DecimalNumeral`s. Now, create your own grammar G that specifies a *language* $L(G)$ for simple expressions. $L(G)$ should accept numeral terms formed by the operations *add* (+) and *multiply* (*). Make sure that the precedences are in correct order, meaning that multiplication precedes addition. It is sufficient to restrict the language to one-digit numerals. You can also use the `DecimalNumeral` rule from the slides. If you do so, you may assume that `DecimalNumeral` is specified by the grammar $G' = (V', \Sigma', R', S')$ according to the slide.

Part (c) Use your grammar from part (b) to construct a *derivation* for the expression `2 + 4 * 5 * 2`. Explain how the precedence is reflected in the corresponding *parse tree*.

Aufgabe 3 - Chessbored? 1 Punkt

This is the first of a set of assignments during which you will build a fully functional implementation of Chess.

In this first assignment, you will start by drawing the playing field. Use appropriate control statements to draw a chessboard including the row and column designators around the board (the letters and numbers). It should look similar to the following example.

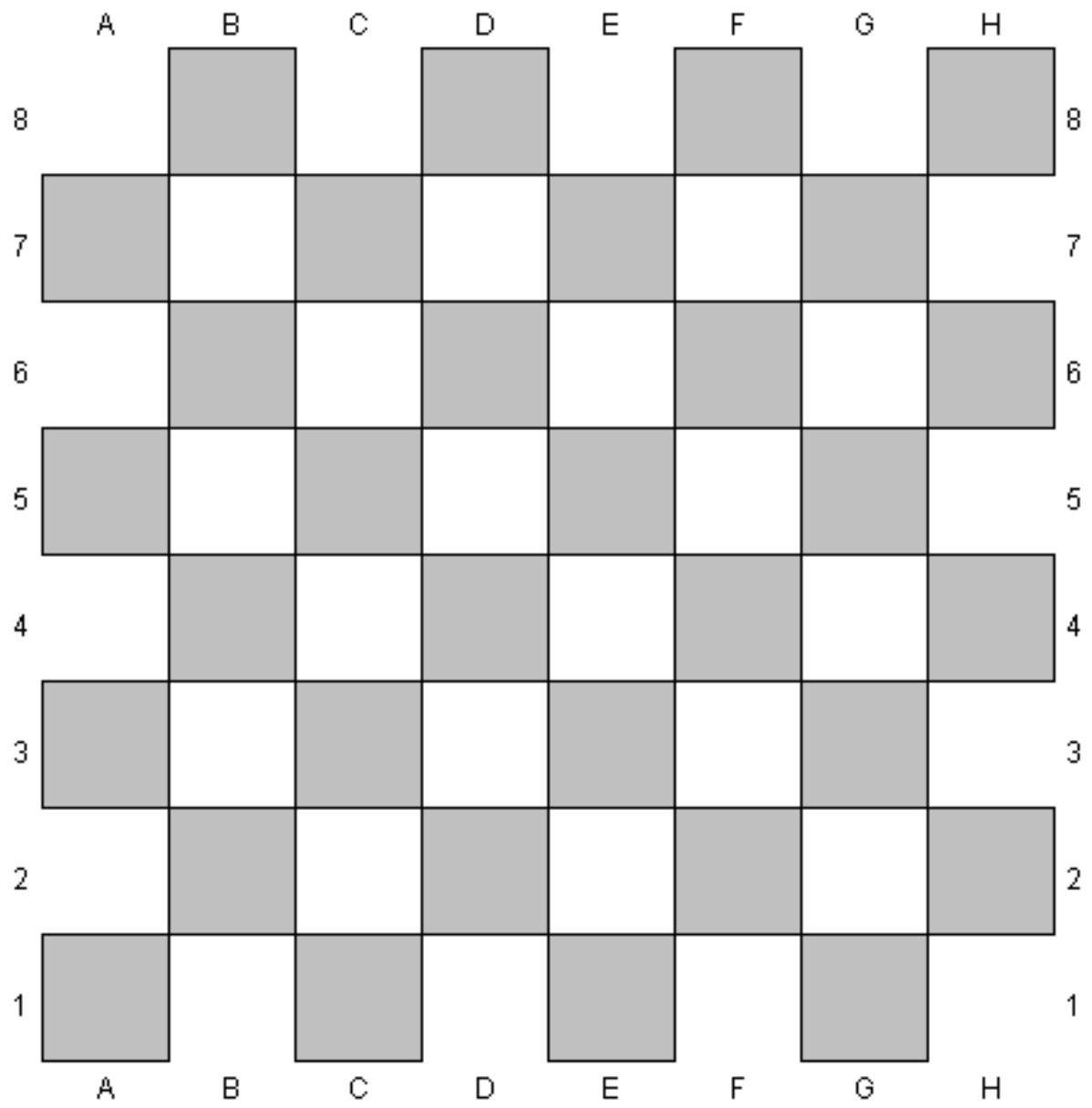

Abbildung 1:

Name your class `Chessboard` and extend `GraphicsProgram`. Make sure that the designators are aligned properly.

Hints on the tests:

- Use `GRect`s to draw the board pieces and `GLabel`s for the row and column designators.
- Use `Color.LIGHT_GRAY` for the dark board pieces.

Zusatzaufgabe 4 - Golden Rectangles

1 Punkt

(Remember that this is a more difficult task aimed at students that already have Java experience. You are not expected to solve it, but ypu are invited to try.)

To quote Wikipedia: "In mathematics, two quantities are in the *golden ratio* if their ratio is the same as the ratio of their sum to the larger of the two quantities."

That is, if we have $a, b \in \mathbb{R}$ with $a > b > 0$, the following equation must hold:

$$\frac{a+b}{a} = \frac{a}{b}$$

This ratio is usually called φ, and working out these equations yields $\varphi \approx 1,618$. We can extend this definition to *golden rectangles*: a golden rectangle is a rectangle where the ratio between the longer side and the shorter side is φ.

Write a program `GoldenRectangles` that draws a number of, well, golden rectangles. More specifically, the program should ask the user for the width (long side) of the first rectangle, and for the number n of rectangles to draw. It should then work out the length of the short side and start drawing the requested number of golden rectangles, all contained in the first, largest rectangle. For $width = 400$ and $n = 8$, it should look like this:

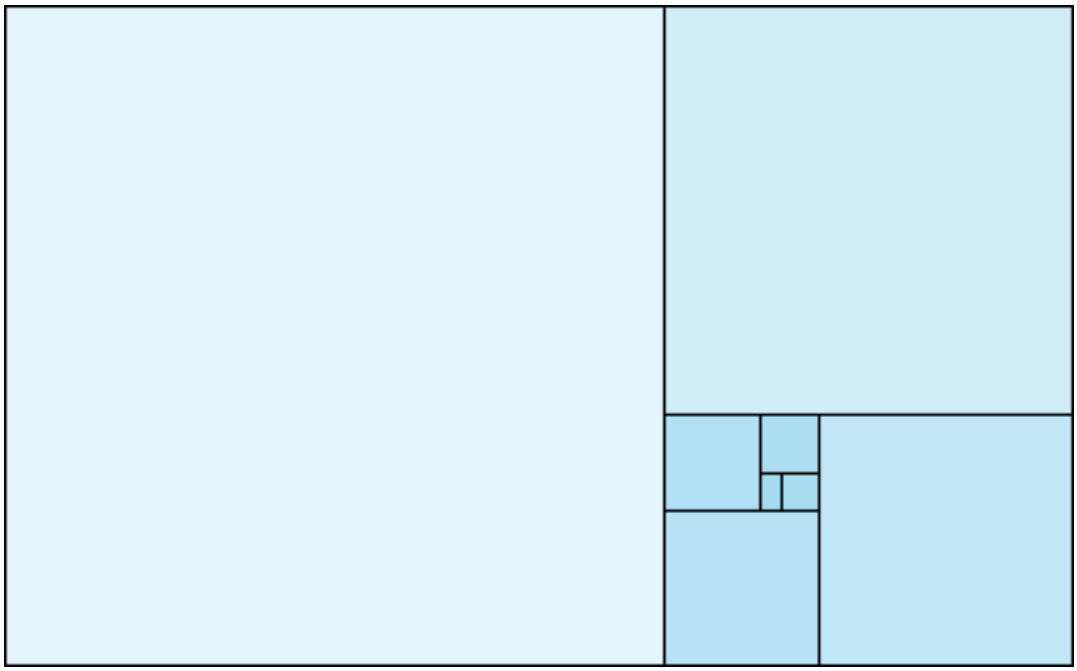

Abbildung 2:

To make it a bit clearer, here is what your program should draw for $n = 1$ to $n = 4$:

That is, each rectangle should be drawn on top of its predecessor.

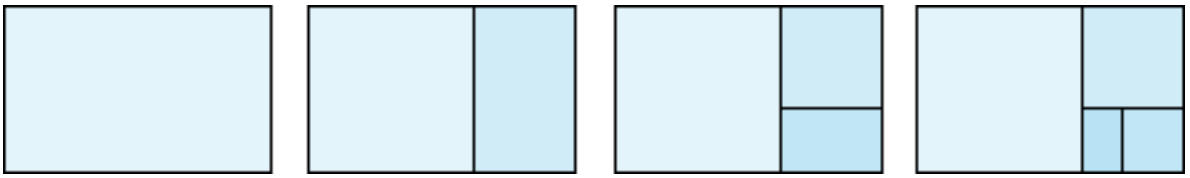

Abbildung 3:

Hints on the tests:

- Be sure to create the rectangles the same way as in the example shown above (use ‘GRect’ and nothing else)
- Don’t be discouraged by the test results, rounding errors might occur from time to time