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

4. Übung zur Vorlesung „Programmierung“

Come Fourth: Methods
Abgabe am Sonntag, 27. 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.

Remember to comment **all** your code appropriately.

Aufgabe 1 - (Mandatory) PowerSet

49 Punkte

The idea of this assignment is

- to refresh how (non-negative) integers are represented in the computer,
- to use logical bit operations (in particular &),
- to practice the usage of control structures,
- to practice defensive programming by sanitizing your input, and
- to review the concept of a power set.

Write a class `PowerSet` that extends `ConsoleProgram` and computes power sets. Recall that for a set S, the power set (*Potenzmenge*) of S is the set of all subsets of S. Your task is to write a program that, given some natural number N, computes the powerset of the set $\{0, ..., N\}$. Your program should work for any N between 0 and 10. It should start by asking the user for N; if N is outside of the desired range, the user should be asked again. As usual, the program must contain meaningful comments. Your solution must adhere to the format of this example trace:

```
This program prints the power set of the set of natural numbers 0 ... N.
Enter N (0 <= N <= 10): 1,5
Illegal numeric format
Enter N (0 <= N <= 10): -2
Enter N (0 <= N <= 10): 31
Enter N (0 <= N <= 10): 2
The powerset of { 0, 1, 2 } is
{ {  }, { 0 }, { 1 }, { 0, 1 }, { 2 }, { 0, 2 }, { 1, 2 }, { 0, 1, 2 } }
```

Hint:

- You may exploit the fact that integers are encoded as a sequence of bits, which can in turn be interpreted as sets. For example, the integer 13 is represented as 1101, where bits 3, 2 and 0 are set; hence 13 represents the set $\{0, 2, 3\}$.

As usual, you get points for this assignment only if your program passes the automatic test. To pass the automatic test, your output must adhere exactly to the format above, including the order of the elements of the power set.

Aufgabe 2 - All the (Mandatory) Pieces 49 Punkte

This is the second part of our Chess assignment. In Chess, each player begins the game with 16 pieces out of 6 types: 8 *pawns*, 2 *bishops*, 2 *knights*, 2 *rooks*, one *queen*, and one *king*. The pieces are colored *white* for player 1 or *black* for player 2. To get the chess terminology right: The columns of the chessboard are called *files* and the rows are called *ranks*.

Your task is to write a class `MethodicalChessboard`, which is capable to draw `GObject`s. Copy your chessboard source code from the last assignment to a new method `public void drawChessboard()` in your new class and, also, write two new methods.

First, write a method that enables you to color each square in a color of your choosing. Change the original code in your `drawChessboard` method to use `drawSquare`.

```
/**
 * Draws the square identified by {@code x} and {@code y}
 * in the color {@code color}.
 *
 * @param x
 *            file index
 * @param y
 *            rank index
 * @param color
 *            chosen color
 */
public void drawSquare(int x, int y, Color color) {
    // Implement this method
}
```

Second, write a method that draws a piece of a specific type on a square of the chessboard. Of course, you don't have to draw graphically sophisticated chess pieces manually. Use `GLabel`s and single letters for the chess pieces. Color them appropriately.

Chess symbols are also part of the unicode character set. Even so you may not have heard about unicode characters yet, you can use them to make your chessboard a little bit more shiny. (You don't have to do this, but think about the awesomeness you will gain!) You find the unicodes you need in the table below. Simply add them to your Strings, Java will do the rest for you (for example, the pawn is represented by the string `"\u2659"`).

The signature of the method should be:

```
/**
 * Draws a specific chess {@code piece} of a {@code player} at the position of a specific square
 * identified by {@code x} and {@code y}.
 * Pieces are: 0 pawn, 1 Knight, 2 Bishop, 3 Rook, 4 Queen, 5 King
 * The player playing white is identified by 0.
 * The player playing with the black pieces is identified by 1.
 *
 * @param x
 *            file index
 * @param y
 *            rank index
 * @param piece
```

```
     *            the piece
     * @param player
     *            player 1 or player 2
     */
    public void drawPiece(int x, int y, int piece, int player) {
        // Implement this method
    }
```

Name	`piece`	`player`	Shortcut	Unicode	Character
White Pawn	0	0	(white) P	\u2659	
White Knight	1	0	(white) N	\u2658	
White Bishop	2	0	(white) B	\u2657	
White Rook	3	0	(white) R	\u2656	
White Queen	4	0	(white) Q	\u2655	
White King	5	0	(white) K	\u2654	
Black Pawn	0	1	(black) P	\u265F	
Black Knight	1	1	(black) N	\u265E	
Black Bishop	2	1	(black) B	\u265D	
Black Rook	3	1	(black) R	\u265C	
Black Queen	4	1	(black) Q	\u265B	
Black King	5	1	(black) K	\u265A	

Third, write your `run()` method such that a chessboard similar to one of the following is drawn, meaning that the coloring of the board and the position and colors of the pieces shoud be similar. Of course, the pieces in your version may look different. Also, you may scale the board as you see fit and use a border if you want to.

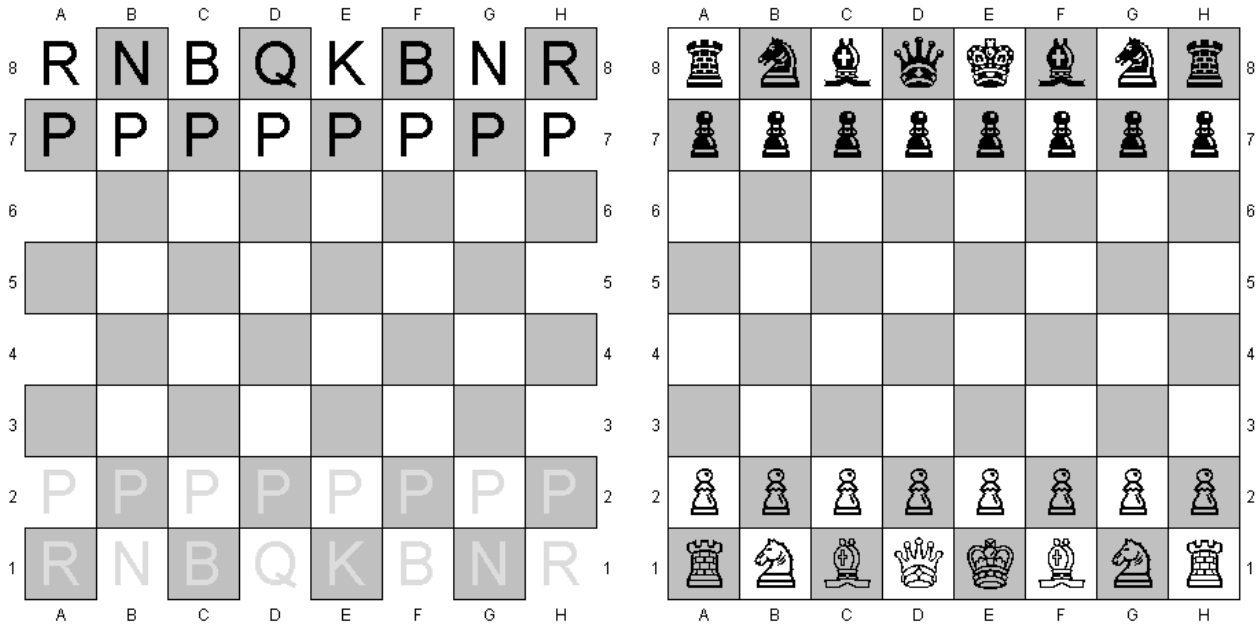

Abbildung 1:

Finally, in addition to the usual comments, add comments to your program that explain and relate the following terms to your code:

- Method

- Parameter
- Type
- Argument
- Scope
- Control statement

Hints:

- Use `Color.LIGHT_GRAY` for the dark squares.
- As computer scientists counting almost always starts at 0. So, even if the chessboard contains squares from **a8** to **h1** externally (meaning graphically visible here), your progam should count from **00** to **77** internally.
- Java Strings support unicode directly. Simply use the codes above in your Strings, for example `label.setLabel("\u2654");`.
- If you use the unicode character set, consider the different character symbols for white and black pieces. You should not color the labels differently in this case.
- If you prefer a border around the chessboard or differntly/explicitly colored white squares, you are free to do so. The test should ignore this.
- We used `SansSerif-44` as font for the pieces in the right example.

Aufgabe 3 - Another Brick in the Pyramid Wall

1 Punkt

Your assignment is to write a class `MethodicalPyramid` that extends `GraphicsProgram`. The program is supposed to draw a pyramid with rectangles, as such:

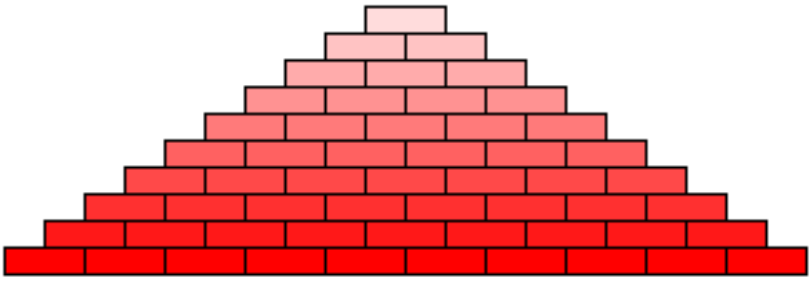

Abbildung 2:

Before drawing the pyramid, it should ask the user for the number of bricks in the bottom layer. Also, as you can see in the picture, each layer of bricks should have a different colour. The bottom layer uses plain red (`new Color(255, 0, 0)`), while the top layer uses a very bright red (`new Color(255, 220, 220)`). The layers in between should smoothly interpolate from plain red to very bright red.

Write your class in a way such that important parts of its functionality are placed in methods. Understanding why we want to use methods is an important step; so, while you're solving this assignment, don't just write the methods we want you to write, but also understand why *these* methods instead of another set of methods.

First, write a method that returns the color to be used for a given layer out of a given number of total layers:

```
/**
 * Returns the color to be used for bricks in the given layer.
 *
 * @param layerIndex
 *            index of the layer whose color to return. {@code 0} is the
```

```
 *            bottom layer, {@code numberOfLayers - 1} is the top layer.
 * @param numberOfLayers
 *            the number of layers in the pyramid.
 * @return the color to be used for the given layer, or {@code null}if
 *         {@code layerIndex} is invalid.
 */
public Color layerColor(int layerIndex, int numberOfLayers) {
    // Implement this method
}
```

Second, write a method that draws a given layer of the pyramid:

```
/**
 * Draws the given layer with bricks filled with the given color. If
 * {@code layerIndex} is invalid, no bricks at all should be drawn.
 *
 * @param layerIndex
 *            index of the layer to draw. {@code 0} is the bottom layer,
 *            {@code numberOfLayers - 1} is the top layer.
 * @param numberOfLayers
 *            the number of layers in the pyramid.
 * @param layerColor
 *            color the layer's bricks should be filled with.
 */
public void drawLayer(int layerIndex, int numberOfLayers, Color layerColor) {
    // Implement this method
}
```

Third, use the `run()` method such that it uses the two new methods to draw the pyramid. In particular, the `run()` method should not do anything itself to calculate layer colors, and should not instantiate any `GRect` objects or call the `add(...)` method.

The automatic tests will only call your new methods to test your implementation, not your `run()` method. It is thus very important to specify them exactly as shown above.

Zusatzaufgabe 4 - Integration

1 Punkt

To calculate the area bounded by the graph of a function $f(x) : \mathbb{R} \to \mathbb{R}$ and the x axis, a common way is to calculate the integral of f. Another method is to approximate the area by placing rectangles of a given width between the graph and the x axis. The narrower the rectangles, the better the approximation. The concept looks something like this:

As you can see, there are always two rectangles for each slice of the x axis. The first, the *min rect*, underestimates the area of the function. The second, the *max rect*, overestimates the area. To approximate the area in the slice, we use the sum of the area of both rectangles and divide by 2.

Write a program `FunctionArea` that is a subclass of `ConsoleProgram` and that consists of the following methods:

```
/**
 * The function whose area to calculate.
 *
 * @param x
 *            the x coordinate.
 * @return the value of f at the x coordinate.
 */
public double f(double x) {
```

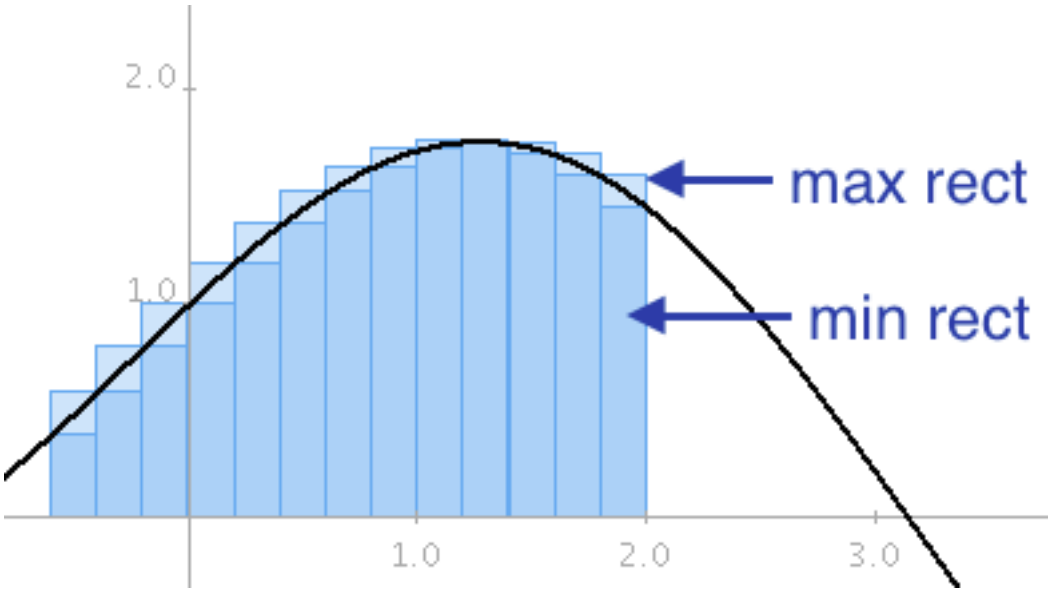

Abbildung 3:

```
    // This is the function whose area we want to calculate. Hardcoding
    // it here is a bit unfortunate; we can improve the design once we
    // know more about classes and interfaces
    return Math.sin(x) + Math.cos(0.5 * x);
}

/**
 * Calculates the height of a rectangle at the given x coordinate and with
 * the given width. The height of the rectangle is the function value at its
 * left and right side that is nearer to the x axis. If the function value
 * is negative, the height is negative.
 *
 * @param x
 *            the rectangle's x coordinate (left boundary).
 * @param width
 *            the rectangle's width.
 * @return the rectangle's height, which may actually be negative.
 */
public double minRectHeight(double x, double width) {
    // Implement this method
}

/**
 * Calculates the height of a rectangle at the given x coordinate and with
 * the given width. The height of the rectangle is the function value at its
 * left and right side that is further from the x axis. If the function value
 * is negative, the height is negative.
 *
 * @param x
 *            the rectangle's x coordinate (left boundary).
 * @param width
 *            the rectangle's width.
```

```
     * @return the rectangle's height, which may actually be negative.
     */
    public double maxRectHeight(double x, double width) {
        // Implement this method
    }

    /**
     * Approximates the area enclosed by the x axis, {@link #f(double)}, and two
     * vertical lines at {@code left} and {@code right}. The approximation
     * divides the x axis section into different parts of width
     * {@code rectWidth} (the rightmost part may have to be smaller to keep it
     * from extending beyond the right boundary). For each part, the function
     * computes the min and max rectangle and uses the min rectangle's area plus
     * half the difference of the two rectangle areas as the approximate area
     * for that part.
     *
     * @param left
     *            left boundary.
     * @param right
     *            right boundary.
     * @param rectWidth
     *            width of the rectangles used to approximate the area.
     * @return the approximate area. If the left boundary is right of the right
     *         boundary, the result is 0.
     */
    public double approxFunctionArea(double left, double right, double rectWidth) {
        // Implement this method
    }
```

Write a `run(...)` method to test `approxFunctionArea(...)` by asking the user for the parameters to call the method with. The exact way you implement `run(...)` is not of much interest to us, since the automatic tests call the other methods directly instead of trying to parse console output.

If you want to test your `approxFunctionArea(...)` method, here's a table of inputs and the expected results:

`left`	`right`	`rectWidth`	Result
-1.0	0.0	0.2	0.4999
-1.0	0.0	0.3	0.5009
1.0	4.0	1.0	1.9345
1.0	4.0	0.2	2.0490