Aufgaben

Kapitel 11.4 und 11.9

1. Bearbeiten Sie die Übungen 11.1 bis 11.32

11.1:

Das Label ist linksbündig und in der Höhe zentriert.

11.2:

http://docs.oracle.com/javase/1.4.2/docs/api/javax/swing/JFrame.html#JFrame%28java.lang.String%29

String für den Titel vom Fenster

11.3:

```
JButton button = new JButton("Ich bin ein Button");
contentPane.add(button);
```

11.4:

Das erste Label ist nun oben Links und das zweite Label jeweils in der Höhe zentriert.

```
JLabel label2 = new JLabel("neues Label");
contentPane.add(label2);
frame.pack();
```

11.5:

Das „File“ Menu kann geöffnet werden, sonst passiert nichts.

```
JMenuBar menubar = new JMenuBar();
frame.setJMenuBar(menubar);
JMenu fileMenu = new JMenu("File");
menubar.add(fileMenu);
JMenuItem openItem = new JMenuItem("Open");
fileMenu.add(openItem);
JMenuItem quitItem = new JMenuItem("Quit");
fileMenu.add(quitItem);
```

11.6:

```
private void makeMenuBar(JFrame frame)
{
    […]
    // Help Menu
    JMenu helpMenu = new JMenu("Help");
    menubar.add(helpMenu);
    // Item for HelpMenu
    JMenuItem aboutItem = new JMenuItem("About ImageViewer");
    helpMenu.add(aboutItem);
}
```

11.7:

```
public void actionPerformed(ActionEvent event)
{
    System.out.println("Item:" + event.getActionCommand());
}

private void makeMenuBar(JFrame frame)
{
    JMenuBar menubar = new JMenuBar();
    frame.setJMenuBar(menubar);
    // File Menu
    JMenu fileMenu = new JMenu("File");
    menubar.add(fileMenu);
    // Item for FileMenu
    JMenuItem openItem = new JMenuItem("Open");
    openItem.addActionListener(this);
    fileMenu.add(openItem);

    JMenuItem quitItem = new JMenuItem("Quit");
    quitItem.addActionListener(this);
    fileMenu.add(quitItem);

    // Help Menu
    JMenu helpMenu = new JMenu("Help");

    menubar.add(helpMenu);
    // Item for HelpMenu
    JMenuItem aboutItem = new JMenuItem("About ImageViewer");
```

```
        aboutItem.addActionListener(this);
        helpMenu.add(aboutItem);
    }
```

11.8:

```
        JMenuItem saveItem = new JMenuItem("Save");
        saveItem.addActionListener(this);
        fileMenu.add(saveItem);
```

11.9:

```
    private void openFile()
    {
        System.out.println("open...");
    }

    private void saveFile()
    {
        System.out.println("save...");
    }

    private void quit()
    {
        System.exit(0);
    }

    private void about()
    {
        System.out.println("about...");
    }

        // Item for HelpMenu
        JMenuItem aboutItem = new JMenuItem("About ImageViewer");
        aboutItem.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            { about(); }
        });
        helpMenu.add(aboutItem);

        […]
```

11.10:

Siehe 11.09

11.11:

Siehe 11.09

11.12:

ActionListener resp. actionPerformed ist alles innere Klasse implementiert.

11.13:

http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/System.html

```
private void quit()
    {
        System.exit(0);
    }
```

Das Programm wird mit ErrorCode 0 (alles OK) beendet.

11.14:

done

11.15:

Das Fenster (Frame) wird jeweils auf die Grösse des Bildes angepasst.
Falls man nach dem Öffnen des Bildes, das Fenster resized bleibt das Bild links oben.

11.16:

Funktioniert nicht, das ImagePanel, resp Label wird „überschrieben“ (überdeckt).

11.17:

BorderLayout

11.18:

BorderLayout, GridLayout, FlowLayout

11.19:

BorderLayout & FlowLayout

11.20:

```
private void makeFrame()
{
    frame = new JFrame("ImageViewer");
    makeMenuBar(frame);

    Container contentPane = frame.getContentPane();
    contentPane.setLayout(new BorderLayout());

    JLabel filenameLabel = new JLabel();
    contentPane.add(filenameLabel, BorderLayout.NORTH);

    imagePanel = new ImagePanel();
    contentPane.add(imagePanel, BorderLayout.CENTER);

    JLabel statusLabel = new JLabel("Version 1.0");
    contentPane.add(statusLabel, BorderLayout.SOUTH);

    // building is done - arrange the components and show
    frame.pack();
    frame.setVisible(true);
}
```

11.21:

done

11.22:

```
        // create the Filter menu
        JMenu filterMenu = new JMenu("Filter");
        menubar.add(filterMenu);

        JMenuItem darkerItem = new JMenuItem("Darker");
            darkerItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_D, SHORTCUT_MASK));
            darkerItem.addActionListener(new ActionListener() {
                                public void actionPerformed(ActionEvent e) { makeDarker(); }
                            });
        filterMenu.add(darkerItem);

        JMenuItem lighterItem = new JMenuItem("Lighter");
            lighterItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_L, SHORTCUT_MASK));
            lighterItem.addActionListener(new ActionListener() {
                                public void actionPerformed(ActionEvent e) { makeLighter(); }
                            });
        filterMenu.add(lighterItem);

        JMenuItem thresholdItem = new JMenuItem("Threshold");
            thresholdItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_T, SHORTCUT_MASK));
            thresholdItem.addActionListener(new ActionListener() {
                                public void actionPerformed(ActionEvent e) { treshold(); }
                            });
        filterMenu.add(thresholdItem);


        // create the Quit menu
        JMenuItem quitItem = new JMenuItem("Quit");
            quitItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q, SHORTCUT_MASK));
            quitItem.addActionListener(new ActionListener() {
                                public void actionPerformed(ActionEvent e) { quit(); }
                            });
        fileMenu.add(quitItem);
```

11.23:

Das Frame (inkl Image) neu zeichnen.

11.24:

```
private void showStatus(String text)
{
    statusLabel.setText(text);
}
```

11.25:

Der Status „No image laoded.“ wird angezeigt im statusLabel.

11.26:

java.awt.Color

11.27:

```
public void lighter()
{
    int height = getHeight();
    int width = getWidth();
    for(int y = 0; y < height; y++) {
        for(int x = 0; x < width; x++) {
            setPixel(x, y, getPixel(x, y).brighter());
        }
    }
}
```

11.28:

```
public void threshold()
{
    int height = getHeight();
    int width = getWidth();
    for(int y = 0; y < height; y++) {
        for(int x = 0; x < width; x++) {
            Color pixel = getPixel(x, y);
            int brightness = (pixel.getRed() + pixel.getBlue() + pixel.getGreen()) / 3;
            if(brightness <= 85) {
                setPixel(x, y, Color.BLACK);
            }
            else if(brightness <= 170) {
                setPixel(x, y, Color.GRAY);
            }
            else {
                setPixel(x, y, Color.WHITE);
            }
        }
    }
}
```

11.29:

```
// create the Help menu
JMenuItem menu = new JMenu("Help");
menubar.add(menu);

JMenuItem aboutItem = new JMenuItem("About ImageViewer...");
aboutItem.addActionListener(new ActionListener()
{

    public void actionPerformed(ActionEvent e)
    {
        showAbout();
    }
});
menu.add(aboutItem);
```

11.30:

Siehe 11.29

11.31:

http://docs.oracle.com/javase/1.4.2/docs/api/javax/swing/JOptionPane.html#showMessageDialog%28java.awt.Component,%20java.lang.Object%29

Es gibt drei verschiedene Methoden.

Für unseren Zweck:

```
public static void showMessageDialog(Component parentComponent,
                                     Object message,
                                     String title,
                                     int messageType)
```

11.32:

```
    private void showAbout()
    {
        JOptionPane.showMessageDialog(frame,
            "ImageViewer\n" + "0.4",
            "About ImageViewer",
            JOptionPane.INFORMATION_MESSAGE);
    }
```

Handout PRG2_DAT5

2. Beantworten Sie die Kontrollfragen A

   Siehe PRG2_DAT5_KF.docx / PRG2_DAT5_KF.pdf

3. Wir nehmen an, es stehen die Datenstrukturen Stack, Queue, doppelt verkettete Liste, TreeSet und TreeList zu Auswahl. Wählen Sie für die folgenden Aufgabenstellungen eine passende Datenstruktur aus. Begründen Sie Ihre Wahl.

   a) Fortlaufend treffen Messdaten ein. Die einzelnen Messdaten müssen genau einmal bearbeitet werden. Es kann vorkommen, dass die eintreffenden Daten zu schnell für die Bearbeitung ankommen und somit zwischengespeichert werden müssen. Ihre Reihenfolge darf aber nicht vertauscht werden.

   Queue

   Die Reihenfolge ist wichtig und die Messdaten müssen zwischengespeichert werden.

   b) Die wichtigste Eigenschaft eines Programms ist die schnelle Abfrage der gespeicherten Zahlen. Die Zahlen sind ganzzahlig und kommen höchstens einmal vor. Beim Einlesen ist die Reihenfolge der Zahlen zufällig.

   TreeSet

   Schnelle Abfrage: die Daten müssen sortiert sein, höchstens einmal: keine doppelte Werte

4. Neben den vorgestellten Datenstrukturen gibt es in Java noch einige weitere Klassen, siehe http://java.sun.com/javase/6/docs/api/java/util/Collection.html.

a) Informieren Sie sich kurz über die wichtigsten Methoden und Eigenschaften von vier Klassen die unter "All Known Implementing Classes" zu finden sind. Wählen Sie dabei die Klassen aus, bei denen Ihnen der Name interessant erscheint.

z.B.: PriorityQueue: http://docs.oracle.com/javase/6/docs/api/java/util/PriorityQueue.html

add() Inserts the specified element into this priority queue.

remove() Removes a single instance of the specified element from this queue, if it is present.

clear() Removes all of the elements from this priority queue.

size() Returns the number of elements in this collection.

contains() Returns `true` if this queue contains the specified element.

offer() Inserts the specified element into this priority queue.

peek() Retrieves, but does not remove, the head of this queue, or returns `null` if this queue is empty.

pool() Retrieves and removes the head of this queue, or returns `null` if this queue is empty.

b) Gehen Sie zur Dokumentation von `PriorityQueue()` und bearbeiten Sie folgende Punkte:

1. Beschreiben Sie die Klasse in eigenen Worten.

Eine Prioritätswarteschlange ist eine Datenstruktur, die vergleichbar wie ein SortedSet die Elemente sortiert. Wenn etwa Nachrichten in ein System kommen, sind diese in der Regel nicht alle gleich wichtig, sondern unterscheiden sich in ihrer Priorität. Elemente höherer Priorität sollen zuerst verarbeitet werden. Das Fundament bildet eine Queue mit einem modifizierten FIFO-Prinzip, das deswegen nicht nur pur FIFO ist (»Wer zuerst kommt, mahlt zuerst«), weil Elemente, die schon am Anfang stehen, von später kommenden Elementen mit höherer Priorität verdrängt werden können.

PriorityQueue ist eine Implementierung einer solchen Prioritätswarteschlange. Damit die Priorisierung möglich ist, müssen die Elemente eine natürliche Sortierung besitzen (String oder Wrapper-Objekte haben diese zum Beispiel), oder ein Comparator muss angegeben werden. PriorityQueue verbietet das null-Element, und Elemente dürfen durchaus doppelt vorkommen (wie eine Liste); daher kann auch SortedSet nicht die Aufgabe einer Prioritätswarteschlange übernehmen, da eine Menge Elemente nur einmal enthalten kann.

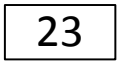
Jeweils das Element mit dem kleinsten Wert (Comparator) hat die höchste Priorität!

2. Gegeben ist folgender Ablauf:

```
offer(23);
offer(51);
System.out.println(poll());
offer(5);
offer(11);
System.out.println(poll());
System.out.println(poll());
System.out.println(poll());
```

Welchen Rückgabewert gibt die Methode `poll()` jeweils aus, wenn beim Einfügen nach aufsteigendem Zahlenwert geordnet wird? Zeichnen Sie jeweils den aktuellen Aufbau der Queue auf.

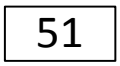
offer(23);



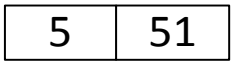
offer(51);

System.out.println(poll());

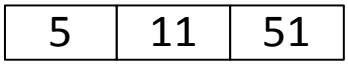
Ausgabe: 23



offer(5);

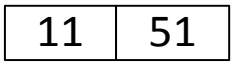


offer(11);



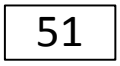
System.out.println(poll());

Ausgabe: 5



System.out.println(poll());

Ausgabe 11



System.out.println(poll());

Ausgabe 51

Queue ist nun leer