Kurzeinführung in die Programmierung graphischer Oberflächen mit Java Swing

Java bietet zwei Bibliotheken zur Programmierung graphischer Benutzeroberflächen, AWT und Swing. Swing wurde als Alternative zum älteren AWT entwickelt. Im Gegensatz zu AWT ist Swing unabhängig vom darunter liegenden Betriebssystem, was Vor -und Nachteile besitzt. Durch die Unabhängigkeit vom Betriebssystem benötigt Swing im allgemeinen mehr Arbeitsspeicher und CPU- Leistung, was derzeit allerdings kein Problem mehr darstellen dürfte. Mit Swing lassen sich aufwändigere Benutzeroberflächen mit weniger Entwicklungsaufwand als mit AWT realisieren. Demzufolge soll es in dieser kurzen Einführung in die Programmierung graphischer Benutzeroberflächen um Java-Swing gehen.

0. Arbeiten mit Swing

*import javax.swing.**
*import java.awt.** - ist auch bei der Programmierung mit Swing erforderlich, da Swing-Komponenten oft eine Erweiterung von AWT- Komponente darstellen und bestimmte Bestandteile, insbesondere die Klasse Container, AWT-Klassen sind. Klassen zur Erstellung von Fenstern werden in der Regel aus JFrame oder JDialog abgeleitet.
JDialog ermöglicht die Realisierung von modalen Dialogen. Eine einfache Instanz eines Fensters ist zunächst funktionslos. Funktionalität erlangt ein Fenster erst durch die Implementierung entsprechender Listener- Interfaces
(Bsp. WindowListener, ActionListener usw., dazu später mehr).

1. Erstellen eines Fensters

Ableitung einer Klasse von JFrame
class <Name> extends JFrame

1.1. Der Konstruktor

- Festlegung der Größe, der Sichtbarkeit und der Koordinaten (bzgl. des Bildschirms)
- Erschaffung eines Containers, in dem alle weiteren Fensterkomponenten Platz finden
- Initialisierung eventueller Fensterkomponenten inkl. der Registrierung ihrer Funktionalität (Eventhandling) und Platzierung der Komponenten im Container

Viele Komponenten werden als Instanzvariablen deklariert, da nicht nur der Konstruktor auf sie zugreift.

- Implementierung erwünschter Fensterfunktionen (insbesondere der Möglichkeit, ein Fenster zu schließen)
- Festlegung von Layouts mit Hilfe von entsprechenden LayoutManagern

Konstruktoren:

- *public JFrame( )*
- *public JFrame (String title)*

➔ Titel kann auch später mit der Methode *public void setTitle (String title)* zugefügt werden

1.2. Größe, Sichtbarkeit und Koordinaten

Methoden:

- *public void setSize(int width, int height)*
- *public void setLocation(int x, int y)*
  ➔die Koordinaten (0, 0) entsprechen der linken oberen Bildschirmecke
- *public void setVisible (boolean visible)*
  ➔für visible ➔ true, wird das Fenster sichtbar

1.3. Container

In Swing werden Komponenten nicht direkt auf einem Fenster platziert, sondern auf einem Container, der indirekt auch eine Komponente des Fensters ist. Dieser Container (ContentPane) beherbergt entsprechend alle sichtbaren Dialogelemente. Er ist der Rückgabetyp der Funktion *public Container getContentPane( ).* Komponenten werden in diesem Container mit Hilfe der Funktion *public add (Component comp)* platziert. Hier ist zu beachten, dass sich die Reihenfolge, in der die Komponenten im Container platziert werden, im Erscheinungsbild des Fensters niederschlägt.

1.4. Schließen des Fensters

Um ein Fenster schließen zu können, muss die entsprechende Funktionalität zunächst vereinbart werden. Dazu wird ein WindowListener registriert
public void addWindowListener(WindowListener l) . WindowListener ist ein Interface, das jegliche Veränderungen am Status eines Fensters steuert. In diesem speziellen Fall interessiert jedoch nur die Funktion *public void windowClosing(WindowEvent e)* . Da die übrigen Funktionen nicht unbedingt implementiert werden sollen, wird ein Objekt der Klasse WindowAdapter erzeugt und die Methode *windowClosing* überlagert.
Mit Hilf von *setVisible (false)* lässt sich das Fenster schließen, und durch *System.exit(...)* lässt sich das gesamte Programm beenden.

Bemerkung: Eine Adapterklasse stellt zu einem entsprechenden Interface die Implementierung der Funktionen mit leeren Funktionsrümpfen zur Verfügung
➔ einzelne Funktionen können bei Bedarf überlagert werden.
Bsp.:

```
addWindowListener( new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
                System.exit(0);
        }
});
```

2. Die wichtigsten Swing-Komponenten

2.1. JComponent

JComponent ist ein Container, der andere Komponenten aufnehmen und einen Layout-Manager besitzen kann. Die Klasse JComponent bietet die Möglichkeit, Dialogelemente mit selbstdefiniertem Erscheinungsbild zu erzeugen. Um dies zu gewährleisten, wird die Funktion *protected void paintComponent(Graphics g)* überlagert. Die Klasse Graphics bietet Methoden zur Erzeugung von Linien, Füll – und Textelementen.
wichtige Methoden der Klasse Graphics:

- *drawChars(char[ ]data, int offset, int length, int x, int y)*
- *drawLine(int x1, int y1, int x2, int y2)*
- *drawOval(int x, int y, int width, int height)*
- *drawRect(int x, int y, int width, int height)*
- *drawString(String str, int x, int y)*
- *fillOval(int x, int y, int width, int height)*
- *fillRect(int x, int y, int width, int height)*
- *setColor(Color c)*

Die Klasse Color liefert eine Grundmenge von Farben (Konstanten der Klasse), zB. schwarz ➔ color.black

2.2. JPanel als „Unter“-Container

Die Klasse JPanel ist direkt von JComponent abgeleitet. Sie ist die wichtigste Basisklasse von GUI-Containern, die keine Hauptfenster sind, und besitzt standardmäßig ein FlowLayout ➔ LayoutManager. Ein JPanel kann weitere JPanels aufnehmen.

2.3. JLabel

JLabel ist ein Dialogelement zur Anzeige einer Beschriftung innerhalb eines Containers.
Konstruktor:

- *public JLabel (String text)*

wichtige Methoden:

- *public String getText( )*
- *public void setText(String text)*

2.4. JButton

Instanzen dieser Klasse erzeugen Schaltflächen, die wahlweise eine Beschriftung, ein Icon oder beides besitzen können
Konstruktoren:

- *public JButton (String text)*
- *public JButton (Icon Icon)*
- *public Jbutton (String text, Icon Icon)*

Um einem Button Funktionalität zu verleihen, muss für ihn ein ActionListener registriert werden. Dies wird mit Hilfe der Funktion

- *public void addActionListener (ActionListener l)* realisiert.

2.5. JTextField

JTextField stellt ein Textfeld zur Eingabe von Daten zur Verfügung.
Konstruktoren:

- *public JTextField (int columns)*
- *public JTextField (String Text)*
- *public JTextField (String Text, int columns)*

Die Angabe der Spaltenanzahl dient der Festlegung der Größe des Textfeldes, nicht der Begrenzung des einzugebenen Textes. Wird ein ActionListener registriert, so reagiert er auf das Drücken von **ENTER.**
Wichtige Methoden:

- *public String getText( )*
- *public void setText(String text)*
- *public String getText (int offs, int len)*➔ liefert Text einer bestimmten Länge, ausgehend von einer bestimmten Position aus
- *public String getSelectedText( )*

Anmerkung:
Die Funktion *getText( )* liefert immer einen String. Wenn man mit Hilfe eines Textfeldes Zahlen einlesen möchte, so kann die Funktion *public static int parseInt (String s)* der Klasse Integer verwendet werden, um den Übergabestring der Funktion *getText* in einen Integerwert zu überführen. Man sollte jedoch sicherstellen, dass der umzuwandelnde String ausschließlich aus Ziffern besteht.

2.6. JList

Sie dient der Darstellung einer Liste mit der Möglichkeit, ein oder mehrere Listenelemente auszuwählen. Eine JList kann beliebige Objekte aufnehmen.
Konstruktoren:

- *public JList ( )*
- *public JList (Object[ ] list)*
- *public JList (Vector list)*
- *public JList (ListModel list)*

Wird an den Konstruktor einer JList ein Array oder eine Instanz der Klasse Vector übergeben, so werden die entsprechenden Einträge in der JList dargestellt und es wird ein Listenmodel erzeugt. Die Möglichkeit zum Scrollen wird erreicht, indem man die JList in ein JScrollPane einbettet.
Bsp:
JList list = new JList( )
JScrollPane sp = new JScrollPane(list)
Die Instanz von JScrollPane wird dann auf dem jeweiligen Container platziert.

2.6.1. Listenelemente selektieren

Bei der Verwendung einer JList sollte man zunächst überlegen, wieviel Elemente gleichzeitig ausgewählt werden dürfen: Einzelselektion ⇔ Mehrfacheslektion. Mit Hilfe der Funktion *public void setSelectionMode (int selectionMode)* kann dies festgelegt werden.
Die Klasse ListSelectionModel liefert dafür drei Konstanten:

- SINGLE_SELECTION ➔ höchstens ein Element
- SINGLE_INTERVAL_SELECTION ➔ mehrere zusammenhängende Elemente
- MULTIPLE_INTERVAL_SELECTION➔ willkürliche Auswahl von Elementen

Bsp.

```
jlist = new JList( );
jlist.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

Wichtige Methoden:

- *public int getSelectedIndex()*
- *public int[] getSelectedIndices()*
- *public Object getSelectedValue()*

2.6.2. Listeninhalt ändern

Soll der Inhalt eine JList dynamisch veränderbar sein, so muss das Listenmodel das Interface ListModel implementieren. Eine mögliche Implementierungsvariante für diesen Zweck ist die Klasse DefaultListModel.
Wichtige Methoden:

- *pubic void clear ( )*
- *pubic void addElement (Object obj)*
- *pubic void removeElementAt(int index)*
- *public int size ( )*
- *public Object elementAt (int index)*

Bsp.

```
DefaultListModel dlist = new DefaultListModel( );
JList jlist = new JList( dlist );
```

3. Layout-Manager

Sie regeln die Platzierung von Dialogelementen, dadurch müssen keine Koordinaten angegeben werden. Einem Fenster oder einem Panel wird ein Layout über den Aufruf der Funktion *public void setLayout(LayoutManager manager)* zugefügt. Es gibt im wesentlichen fünf LayoutManager, die auf unterschiedliche Weise die Anordnung der Komponenten vornehmen.

- FlowLayout
- GridLayout
- BorderLayout
- CardLayout
- GridBagLayout

Gemeinsamkeit (BorderLayout ausgenommen): Die Anordnung von Komponenten richtet sich nach der Reihenfolge deren Auftretens. Um komplexe Layout realisieren zu können, besteht die Möglichkeit, Layout zu schachteln

Auf das Card –und das GridBagLayout wird an dieser Stelle nicht weiter eingegangen.

3.1. FlowLayout

Die Elemente werden einfach hintereinander (Reihenfolge beachten) platziert. Ist in einer Zeile kein Platz mehr, wird in der nächsten fortgefahren.
Zuordnung durch:

- *setLayout(new FlowLayout ( ))*

Konstruktoren:

- *public FlowLayout( )*
- *public FlowLayout(int align)*
- *public FlowLayout(int align, int hgap, int vgap)*

align regelt die Ausrichtung ➔ drei Konstanten

- FlowLayout.CENTER - zentriert (voreingestellt)
- FlowLayout.LEFT - linksbündig
- FlowLayout.RIGHT -rechtsbündig

hgap horizontaler Abstand zwischen Komponenten
vgap vertikaler Abstand (5 entspricht der Voreinstellung)

3.2. GridLayout

Dieser Layout-Manager ordnet die Komponenten in ein rechteckiges Gitter ein. Vorab muss Spalten – und Zeilenanzahl festgelegt werden.
Konstruktoren:

- *public GridLayout(int rows, int columns)*

oder

- *public GridLayout(int rows, int columns, int hgap, int vgap)*

Elemente werden der Reihe nach in das Gitter eingefügt, wobei oben links begonnen wird. Ist die erste Zeile voll, geht es in der zweiten wiederum links weiter.
➔ im GridLayout wird der Zeilenumbruch kontrollierbar.
Achtung: die Größe der Komponenten richtet sich nach der Größe der Spalten. Verändert man die Fenstergröße, so ändern sich entsprechend die Dimensionen des Gitters und somit auch der enthaltenen Komponenten.

3.3. BorderLayout

Der Bildschirm wird in fünf Bereiche aufgeteilt. Diese Bereiche sind die vier Randbereiche des Fensters, denen die geographischen Himmelsrichtungen zugeordnet werden und das dadurch eingeschlossene Zentrum. Hier ist nicht die Reihenfolge der Komponenten bestimmend, sondern die Angabe der gewünschten Position.
Konstruktor:

- *public BorderLayout( )*

Die fünf Bereiche sind mit „Center“, „North“, „East“, „West“ und „South“ bezeichnet. Beim Einfügen einer Komponente muss also zusätzlich der Bereich in Form eines Strings angegeben werden ➔ *public void add (String name, Component Comp)*
Bsp.: *add(“North“, new Jpanel())*

Schachtelung von Layouts

Um Layouts zu schachteln, erzeugt man einfach mehrere Panels, verleiht jedem dieser Panels das gewünschte Layout und schachtelt dann die Panels ineinander.

4. Einfache Menüs

Menüs werden direkt auf dem Fenster platziert

- Menüleiste

Instanzierung durch die Klasse *JMenuBar* ➔ *public JMenuBar( )*
Platzieren der Menuleiste durch *setJMenuBar(JMenuBar mb)* der Klasse *JFrame*

- Menüs

Instanzen der Klasse *Menu* ➔ public JMenu(String label) werden durch *public void add(Menu m)* der Menuleiste zugefügt

- Menüeinträge

Sind die elementaren Bestandteile eines Menüs
Instanzierung der Klasse JMenuItem ➔ public JMenuItem(String label)
Sie werden einem Menü durch *public void add(MenuItem mi)* zugefügt.
Es besteht die Möglichkeit Menüs zu schachteln, da JMenu von JMenuItem abgeleitet ist (dürfte auf den ersten Blick etwas kurios erscheinen). Um einen Menueintrag mit Funktionalität zu versehen, muss für den entspr. Eintrag ein Actionlistener registriert werden ➔ *public void addActionListener(ActionListner l)*

5. ActionEvents und ihr Handling

➔ *import java.awt.event.**
ActionEvents werden von den Komponenten (insofern es vorher vereinbart wurde) JMenuItem, JButton und JTextField gesendet:

- JMenuItem beim Auswahl des Menupunktes
- JButton bei seiner Betätigung
- JTextField beim Drücken von Enter

Ein möglicher Empfänger dieser ActionEvents muss das Interface ActionListener implementieren. Es besitzt lediglich die Funktion
public void actionPerformed(ActionEvent e).
Die Klasse ActionEvent stellt zwei sehr hilfreiche Methoden bereit

- *public Object getSource( )*

➔Ermittlung welches Objekt das Event gesendet hat

- *public String getActionCommand( )*

➔ liefert bei einem JButton oder einem JMemuItem die entsprechende Beschriftung
Mit Hilfe der beiden Methoden wird die Unterscheidung möglicher Ereignisquellen realisierbar. Die Reaktion auf solche ActionEvents wird innerhalb von actionPerformed vom Entwickler des Programms vereinbart.
Bsp.:

```
public class myWindow extends JFrame implements ActionListner {
        public myWindow {
        …
        }//Konstruktor
        …
        public void actionPerformed(ActionEvent e) {
                if(e.getActionCommand.equals(…) {
                …
                }//if
        }//actionPerformed
}//myWindow
```

6. Die Klasse JDialog – modale Dialoge

Diese Klasse bietet die Möglichkeit, Dialogfenster zu erschaffen. Der strukturelle Aufbau entspricht dem der Klasse JFrame ➔ Platzierung von Komponenten erfolgt auf einem ContentPane. Wichtige Methoden entsprechen denen von JFrame. Die Klasse bietet die Möglichkeit modale Fenster zu erzeugen.

➔ ein modales Fenster hält für den Zeitraum seiner Existenz den übrigen Programmablauf an.

Konstruktoren:

- *public JDialog (JFrame owner)*
- *public JDialog (JFrame owner, boolean modal)*
- *public JDialog (JFrame owner, String title)*
- *public JDialog (JFrame owner, String title ,boolean modal)*

Erzeugung eines modalen Dialogs:

- Übergabe von **true** an den Konstruktor der abgeleiteten Klasse
  new JDialog(owner, true)
- Aufruf der Funktion *public void setModal(boolean modal)*

Da eine Dialogfenster in der Regel von einem anderen Fenster erzeugt wird, wird durch die Angabe von *owner* eine logische Zugehörigkeit sicher gestellt ➔ es wird festgesetzt zu welchem Fenster das Dialogfenster gehört. Ein Dialogfenster kann auch zu einem anderen Dialogfenster gehören. ➔ *public JDialog (JDialog owner)* ist auch möglich.
Bei Dialogfenstern soll deren Größe oftmals unveränderlich sein. Dies kann erreicht werden, indem man die Funktion
public void setResizable(boolean resizable) aufruft und **false** an diese übergibt.

Literatur:

GoTo Java 2 - Handbuch der Java-Programmierung 2. Auflage
Guido Krüger, Addison-Wesley