Übersichtsblatt:
GUI-Grundlagen in Java

Definition (informell):

Unter einem GUI versteht man eine graphische Benutzeroberfläche für Programme, wie man sie z.B. aus Microsoft Windows oder neueren Linux Distributionen kennt.

Aufbau:

Ein GUI ist grundsätzlich aus drei Arten von Objekten aufgebaut.

Top-Level-Objekte:

Dies sind Objekte in denen keine anderen Objekte enthalten sind. Sie bilden mit anderen Worten eigene Fenster. Beispiele hierfür sind JFrame und JDialog.

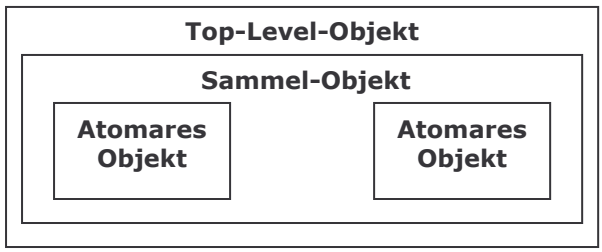
Sammel-Objekte:

Dies sind Objekte, die i. d. R. selbst gar nicht sichtbar sind. Sie dienen ausschließlich dazu andere Objekte zu gruppieren. Beispiel hierfür ist JPanel.

Atomare Objekte:

Dies sind alle Objekte die wir im Fenster sehen und benutzen können, und die keine weiteren Objekte enthalten. Beispiele hierfür sind JButton oder Jlabel.

Schematischer Aufbau:



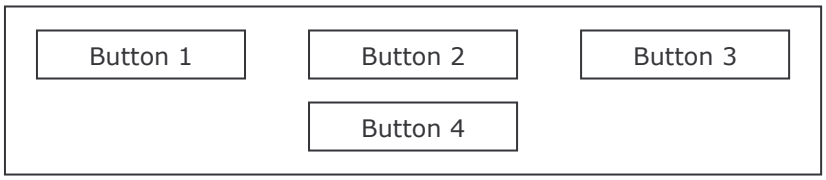
Layouts:

Um die Elemente in einem Fenster anzuordnen stellt Java eine ganze Reihe von Layout-Managern zur Verfügung. Einige davon werden hier nun vorgestellt.

Flow-Layout:

Befehl: `JPanel jp.setLayout(new FlowLayout());`

Hier werden alle Elemente nebeneinander angeordnet. Wenn das Fenster allerdings zu klein ist um dies zu ermöglichen, so werden die Elemente, die nicht mehr in die Zeile Passen, in die nächste Zeile gepackt. Beiden Zeilen werden zentriert dargestellt.



Grid-Layout:

Befehl: `JPanel jp.setLayout(new GridLayout(Zeilen,Spalten));`

Hierbei wird eine Anordnung der Elemente in einem Raster aus Zeilen und Spalten konstanter Größe erreicht. Ausschlaggebend für die Größe der Objekte ist die Größe des größten Objekts.

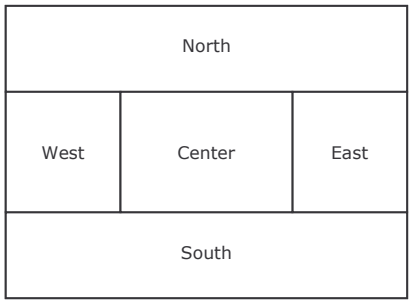
Button 1	Button 1
Button 1	Button 1

Übersichtsblatt:
GUI-Grundlagen in Java

Border-Layout:

Befehl: `JPanel jp.setLayout(new BorderLayout());`

Ein Border-Layout hat immer 5 Bereiche mit den Namen: North, South, West, East und Center. Diese Bereiche sind wie folg angeordnet:



Will man nun ein Objekt zu einem dieser Bereiche hinzufügen, so muss man im add-Befehl angeben wohin dieses Objekt gepackt werden soll. Der add-Befehl sieht dann also folgendermaßen aus:
`jp.add(Objektname,BorderLayout.Berech);`

Event-Handling:

Damit ein GUI nicht nur schön aussieht, sondern auch etwas kann, nämlich auf Bentutzereingaben reagieren benötigt man das so genannte Event-Handling. Ein Event ist in diesem Zusammenhang eine Aktion die vom Benutzer ausgeführt wird. Beispiele hierfür sind das Minimieren oder schließen eines Fenster, das drücken einer Maustaste oder das klicken auf einen Button. Um auf solche Events reagieren zu können, muss man dem Programm beibringen auf sie zu achten. Hierfür bietet Java so genannte Listener an (man kann sie sich als Objekte Vorstellen, die für nix anderes gut sind als darauf zu achten ob ein bestimmtes Event geschieht). Einer von ihnen ist der ActionListener. Dieser steht in Java als Interface zur Verfügung und muss dementsprechend erst in einer Klasse implementiert werden. Wobei hier die Methode actionPerformed() im Mittelpunkt steht. Um dies zu tun gibt es drei Möglichkeiten:

1. Man schreibt für jedes Event das im Programm auftreten kann, d.h. für jeden Knopf der vielleicht gedrückt wird, eine eigene Klasse, die das Interface ActionListener implementiert und in einer eigenen Datei abgelegt wird.

   → Man muss sich für jedes Event einen Namen ausdenken
   → Gigantisches Dateichaos

2. Man schreibt für jedes Event eine interne Klasse

   → Man muss sich immer noch für jedes Event einen Namen ausdenken
   → Die Fensterklasse wird Gigantisch groß und somit unübersichtlich und schwer wartbar.

3. Man teilt jedem Button direkt einen eigenen ActionListener zu und tut dies unter Nutzung einer anonymen Klasse.
   Befehl:

```
JButton button1 = new JButton();
button1.addActionListener(){
        public void actionPerformed(ActionEvent ae){
                //mach was wenn der Knopf gedrückt wird
        }
}
```

Anmerkung:

Anonyme und interne Klassen werden vom Java Compiler in eigenständige Klassen übersetzt. Tatsächlich läuft also immer die Variante 1 ab, auch dann wenn man sich für die weit aus komfortablere Variante 3 entschieden hat. Für den Entwickler ist dies allerdings unerheblich.