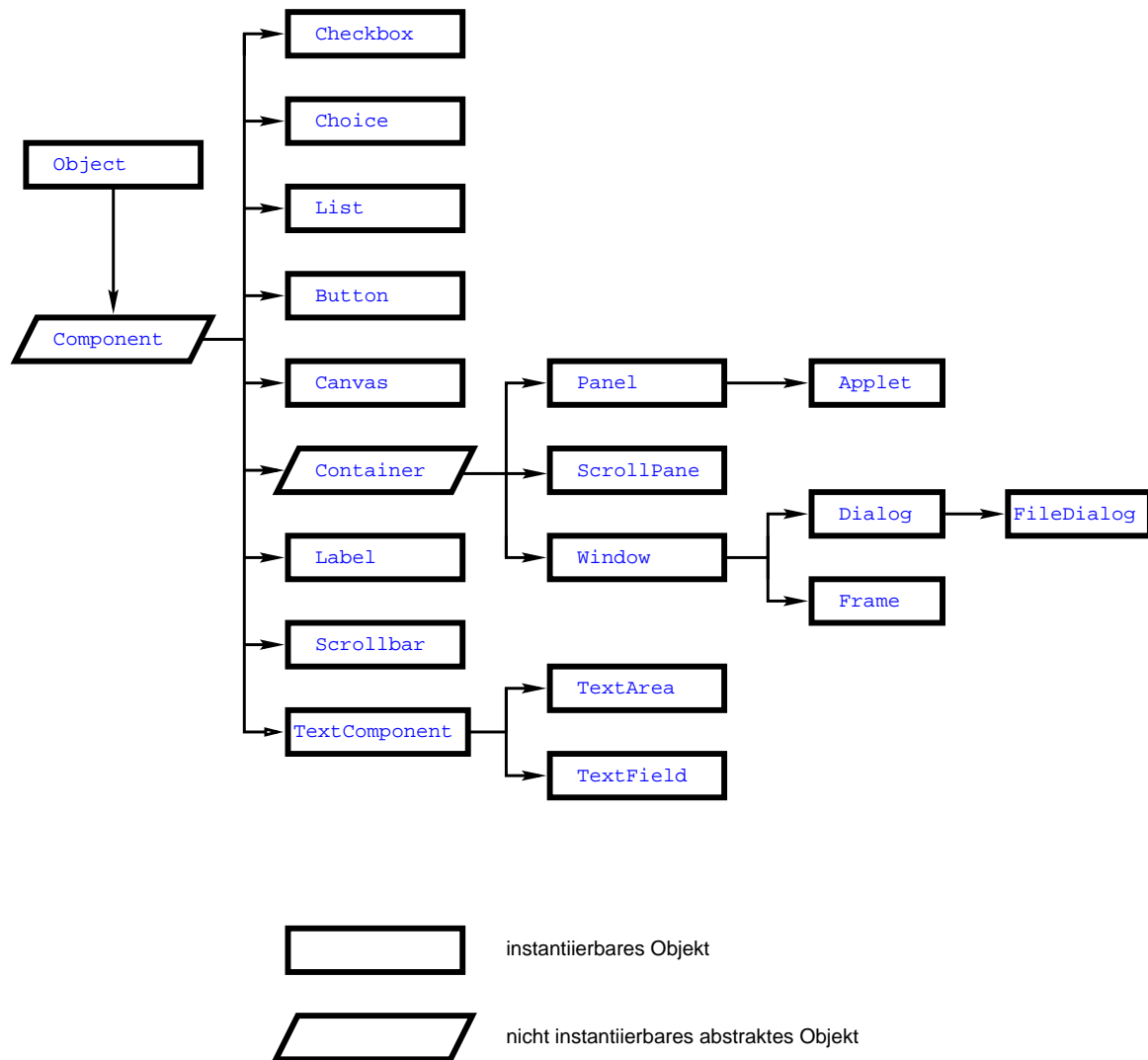


5. Das Java AWT (II)

Components und Container

AWT Hierarchie (II)



Wichtige Klassen des AWT (II)

Component: Die *Basisklasse*

Definiert *sehr viele* Methoden.

Kann nicht direkt instanziiert werden!

Canvas: Die einfachste *Zeichenerfläche* (Leinwand)

Besitzt im wesentlichen ein `Graphics`-Objekt.

Container: Ein *Behälter* für `Components`

Kann nicht direkt instanziiert werden!

Panel: Objekt zum *Zusammenfassen grösserer Sinneinheiten* in einem Container.

`Applet` ist als Unterklasse von `Panel` definiert.

Window: einfaches *Fensterobjekt* (Dialog)

Wird als Popup-Window verwendet.

Frame: "richtiges" *Fensterobjekt*

Hat Titelbalken, kann vom User verkleinert und vergrössert werden, kann ein Menu haben . . .

Wichtige Klassen des AWT (III)

Button: Druckknopf

Checkbox: “Ankreuzfeld”

Choice: Auswahlmenu (Pull-down)

List: Auswahlmenu

Label: statischer Text

Scrollbar: Schieberegler

TextField: Texteingabe (einzeilig)

TextArea: Texteingabefeld (mit mehreren Zeilen)

Components

`Component`s sind der Grundbaustein einer graphischen Benutzeroberfläche.

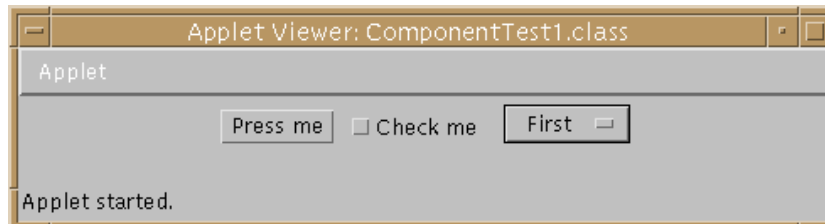
Merke: Alles, was man sieht, ist ein `Component`

`Component`s haben unzählige Methoden, welche wir zunächst alle nicht benötigen.

Merke: Im Zweifel ist die gesuchte Methode in `Component` definiert.

Schon bekannt: Die `paint(Graphics g)`-Methode:
Wird in Unterklassen überschrieben, um individuelle Graphiken zu zeichnen.

ComponentS: Button, Checkbox, Choice



```
import java.applet.*;
import java.awt.*;

public class ComponentTest1 extends Applet {

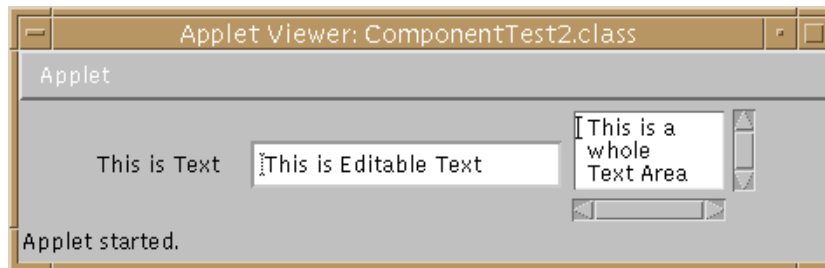
    private Button button;
    private Checkbox checkbox;
    private Choice choice;

    public void init() {
        button = new Button("Press me");
        this.add(button);

        checkbox = new Checkbox("Check me");
        this.add(checkbox);

        choice = new Choice();
        choice.addItem("First");
        choice.addItem("Second");
        choice.addItem("Third");
        this.add(choice);
    }
}
```

Components: Label, TextField, TextArea



```
import java.applet.*;
import java.awt.*;

public class ComponentTest2 extends Applet {

    private Label label;
    private TextField textField;
    private TextArea textArea;

    public void init() {
        label = new Label("This is Text");
        this.add(label);

        textField = new TextField("This is Editable Text",20);
        this.add(textField);

        textArea = new TextArea(" This is a\n whole\n Text Area",3,10);
        this.add(textArea);
    }
}
```

Peers

Für alle Bildschirmobjekte werden die vom Window-system bereitgestellten, nativen Objekte benutzt (sogenannte *Peers*).

Vorteile:

- garantiert es Plattform-Look-and-Feel
- effiziente Implementierung
- schnell anzupassen

Nachteile:

- garantiert es Plattform-Look-and-Feel
- "Protokollreibereien"
- hoher Systemressourcenverbrauch
- gemeinsamer Nenner

Lightweight Peers

Mit Java 1.1 wurden sogenannte Lightweight-Peers eingeführt.

Die Peers sind dabei vollständig in Java implementiert und direkte Unterklassen von `Component`.

Mit der Einführung der Java Foundation Classes gewinnen Lightweight-Peers an Bedeutung.

Für eigene User-Interface-Komponenten mit spezieller Funktionalität bieten sich Lightweight-Peers an.

Achtung: Obwohl viele Methoden in `Component` definiert werden, müssen sie selbst implementiert werden (viele Methoden sind nicht explizit als `abstract` deklariert, sondern sind es nur implizit).

Container

- Objekte in denen mehrere Bildelemente ([Components](#)) zusammengefasst werden können
- Anordnung der Bildelemente wird durch das *Layout* des Containers bestimmt
- kann nicht selbst instantiiert werden
- Typische Container sind [Panel](#) und [Window](#)
- ermöglichen eine *hierarchische Gestaltung* der Benutzeroberfläche

Applets als Container

Da die Klasse `Applet` eine Unterklasse von `Panel` ist, ist jedes Applet selbst ein Container.

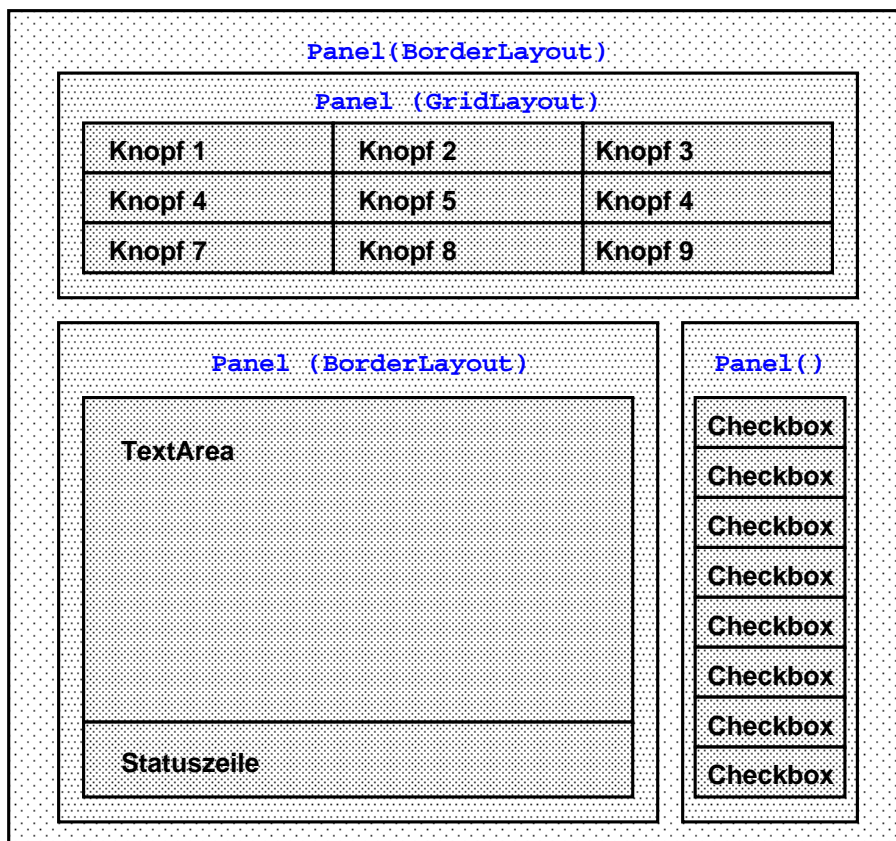
Man fügt Komponenten zu einem Container mit der `add`-Methode hinzu, mit `remove` entfernt man sie wieder:

Programmauszug:

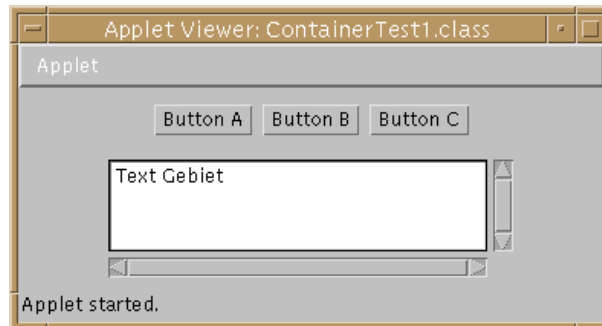
```
public class MyApplet extends Applet {
    // ...
    public void init() {
        Label l = new Label("Please press:");
        add(l);
        Button b = new Button("Press Me!");
        add(b);
    }
    //...
}
```

Fenstergestaltung mit Containern

Da `Container` eine Unterklasse von `Component` ist, kann ein `Container` selbst wieder zu einem anderen `Container` hinzugefügt werden. Damit können ganze Hierarchien von User-Interface-Elementen aufgebaut werden.



Panels



```
import java.applet.*;
import java.awt.*;

public class ContainerTest1 extends Applet {

    private Button a,b,c;
    private TextArea text;
    private Panel panel1,panel2;

    public void init() {
        panel1 = new Panel();
        a = new Button("Button A");
        b = new Button("Button B");
        c = new Button("Button C");
        panel1.add(a);
        panel1.add(b);
        panel1.add(c);

        panel2 = new Panel();
        text = new TextArea("Text Gebiet",4,30);
        panel2.add(text);

        this.add(panel1);
        this.add(panel2);
    }
}
```

Layouts

Der `LayoutManager`

- steuert das *graphische Erscheinungsbild* eines `Containers`
- jeder `Container` besitzt einen `LayoutManager`
- legt das Verhalten bei *resize-Operationen* fest
- ermöglicht plattformunabhängige Oberflächen-gestaltung
- vordefinierte `LayoutManager`:
`FlowLayout`, `BorderLayout`, `GridLayout`,
`CardLayout`, `GridbagLayout`

`LayoutManager` ist ein Interface. Man kann selbst neue Klassen schreiben, die `LayoutManager` implementieren, um eigene Layout-Regeln durchzusetzen.

Positionierung von Komponenten

Die Positionierung von einzelnen Komponenten liegt nicht in der Hand des Programmierers, sondern in der Hand des Layout-Managers. Dieser orientiert sich an Daten, die er von den einzelnen Komponenten erhält.

Alle `Components` implementieren die Methoden

- `public Dimension getMinimumSize()`
- `public Dimension getPreferredSize()`
- `public Dimension getMaximumSize()`

die der `LayoutManager` für die Bestimmung der Grösse und Position einzelner Komponenten benutzen kann (*aber nicht muss!*).

Daher macht es keinen Sinn, die Komponenten direkt über Aufrufe von `setLocation` oder `setSize` zu positionieren.

Setzen des **LayoutManager**

```
import java.awt.*;
import java.applet.*;

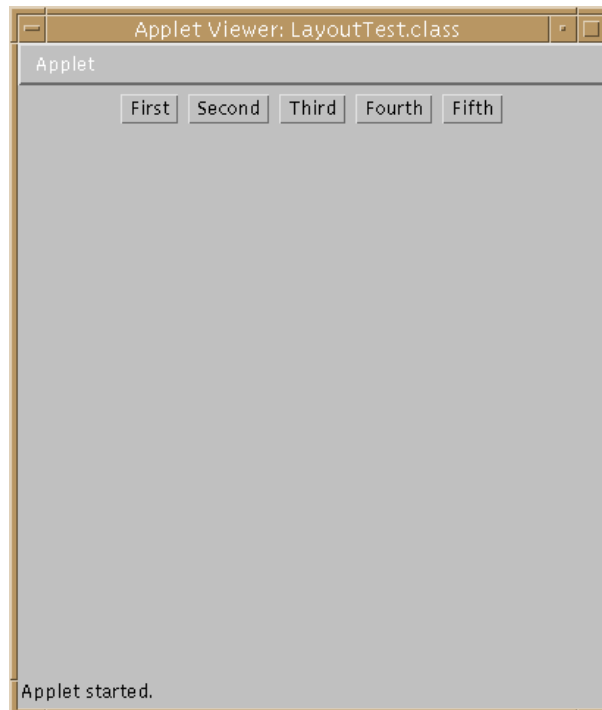
public class LayoutTest extends Applet {
    public void init() {
        setLayout(new FlowLayout());

        Button a = new Button("First");
        Button b = new Button("Second");
        Button c = new Button("Third");
        Button d = new Button("Fourth");
        Button e = new Button("Fifth");
        add(a);
        add(b);
        add(c);
        add(d);
        add(e);
    }
}
```

Es folgt eine Übersicht über Standard-Layouts.

FlowLayout

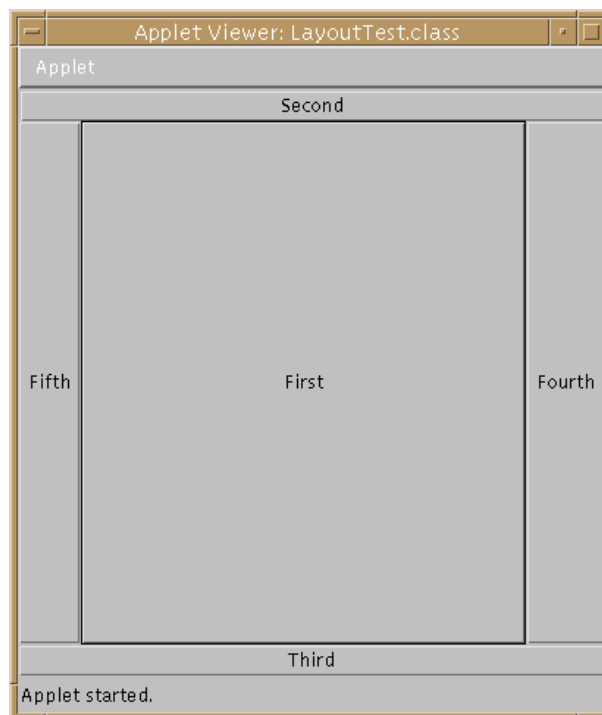
`FlowLayout` ist der `DefaultLayoutManager` für viele `Container`. Die enthaltenen Komponenten werden auf die Grösse gesetzt, die sie über `getPreferredSize()` zurückliefern. Dann werden sie zeilenweise angeordnet (vergleichbar mit word-wrap).



BorderLayout

```
setLayout(new BorderLayout());
```

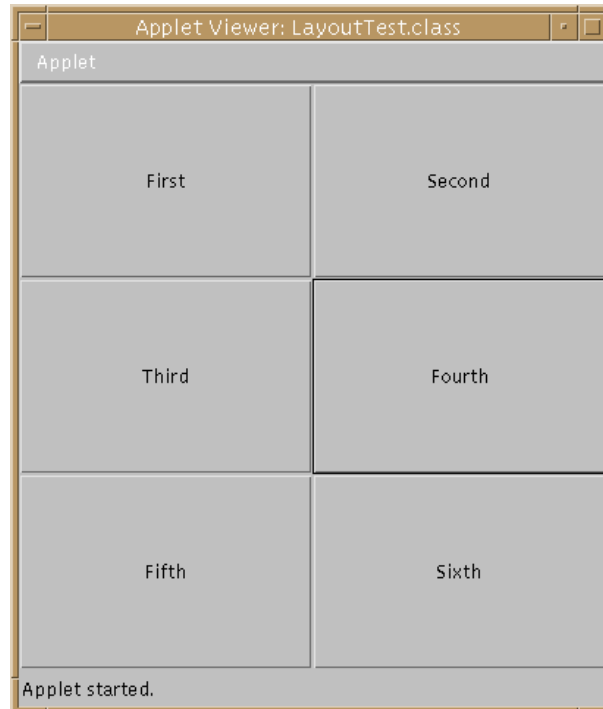
Der `BorderLayout`-Manager teilt den `Container` in fünf Regionen auf: `North`, `East`, `West`, `South` und `Center`. Jede dieser Regionen kann genau eine Komponente enthalten (die aber wieder ein `Container` sein kann). Um auf eine der Regionen zuzugreifen, übergibt man ihren Namen an `add` als erstes Argument.



GridLayout

```
setLayout(new GridLayout(3,2));
```

Der `GridLayout`-Manager teilt den vorhandenen Platz gleichmässig auf. Die `add`-Methode weist den Komponenten nacheinander von links nach rechts und oben nach unten je eine Zelle zu.



Layout-Bemerkungen

- Jede Komponente kann nur einmal verwendet werden. Fügt man eine Komponente zu mehreren Containern hinzu, so wird sie aus allen bis auf den letzten wieder gelöscht. Dies verhindert zirkuläre Abhängigkeiten.
- Da für die Bestimmung des Layouts alle mittels `add` hinzugefügten Komponenten existieren müssen, wird die Berechnung solange hinausgezögert, bis der Container selbst gemalt wird. Mit der Methode `validate()` erzwingt man, dass das Layout neu berechnet wird.
- Man kann explizit positionieren, wenn man den `LayoutManager` eines Containers auf `null` setzt. *Dadurch verliert man Plattformunabhängigkeit!*

Ohne **LayoutManager**

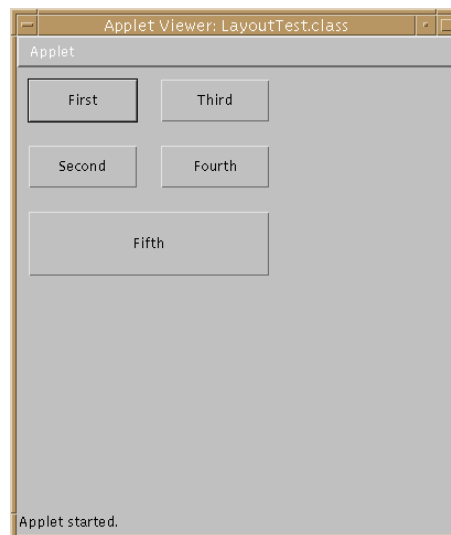
```
public class LayoutTest extends Applet {
    public void init() {

        setLayout(null);

        Button a = new Button("First");
        Button b = new Button("Second");
        Button c = new Button("Third");
        Button d = new Button("Fourth");
        Button e = new Button("Fifth");

        a.setBounds(10,10,100,40);
        b.setBounds(10,70,100,40);
        c.setBounds(130,10,100,40);
        d.setBounds(130,70,100,40);
        e.setBounds(10,130,220,60);

        add(a); add(b); add(c); add(d); add(e);
    }
}
```



Window

Ein `Window`-Objekt ist ein spezieller Container, dessen Peer ein eigenes Fenster ohne Rahmen ist. Der Default-Layout-Manager ist `BorderLayout`.

Konstruktor: `public Window(Frame parent)`

Der Konstruktor für `Window` verlangt als Übergabeparameter ein `Frame`-Objekt. *Merkregel: Ein Dialog-Popup gehört zu einem "vernünftigen" Fenster.*

Woher bekommt man dieses `Frame`-Objekt?

Im Falle von Applets existiert ein Top-Level-Window, zum Beispiel das Appletviewer-Fenster oder das Fenster des Netscape Navigator. Dieses erhält man über wiederholte `getParent()`-Aufrufe aus `Component`. *Achtung:* Dieser Ansatz ist nicht portabel und nicht zuverlässig! Zudem sind eigenständige Windows bei Applets selten sinnvoll.

Im Fall von Applications sollte man sowieso schon ein `Frame`-Objekt haben.

Wichtig: Windows sind per default nicht sichtbar — man muss `show` aufrufen.

Frame (I)

Ein `Frame` ist ein “richtiges” Window, mit sämtlichen Dekorationen, einer Titelleiste, usw.

Konstruktoren:

```
public Frame()
```

erzeugt ein (noch) nicht sichtbares Window. Der Default-Layout-Manager ist `BorderLayout`.

```
public Frame(String title)
```

erzeugt ein (noch) nicht sichtbares Window mit Titel `title`.

Frame (II)

Wichtige Methoden:

- `String getTitle()` und `void setTitle()`
Zugriff auf die Titelleiste
- `Image getIconImage()` und `void setIconImage(Image im)`
wird nicht von allen Plattformen unterstützt
- `MenuBar getMenuBar()` und `void setMenuBar(MenuBar mb)`
Zugriff auf das Window-Menu
- `boolean isResizable()` und `void setResizable(boolean rs)`
Muss gesetzt werden, bevor der Peer erzeugt wird
- `int getCursorType()` und `void setCursor(int cursorType)`
wird ab Java 1.1 durch die Methoden `set-` und `getCursor` von `Component` ersetzt — siehe unten!

Frame (III)

Einige Methoden aus `Window` und `Component`

- `public void show()` und `public void hide()`
Windows sind per default unsichtbar!
- `public boolean isShowing()`
ist true, wenn das Fenster sichtbar ist
- `public void toFront()` und `public void toBack()`
bringen das Fenster in den Vordergrund oder Hintergrund
- `public void dispose()`
entfernt das Window *vollständig* (inklusive Peer)
- `public void pack()`
bringt Subkomponenten auf ihre `preferredSize` und passt die Grösse des Windows an.

Sowohl `pack` als auch `show` sorgen für die Erzeugung des Peers des Windows.

Ein Beispiel

Dieses Programm erzeugt ein Fenster mit einem Button:

```
import java.awt.*;

public class MyApplication extends Object {

    public static void main(Strings argv[]) {
        Frame top = new Frame("My Application");
        Button b = new Button("Push!");
        top.add("Center",b);
        top.pack();
        top.show();
    }
}
```

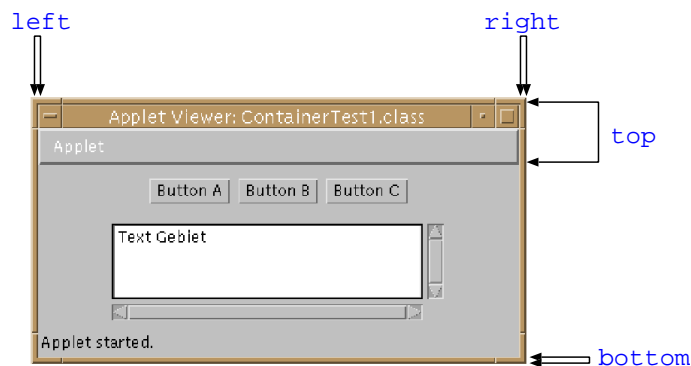
Problem: Das Fenster reagiert nicht auf "Close" — es bleibt, solange die JVM läuft!

Dazu: Eventhandling!

Insets

Damit das Layout nicht mit den Dekorationen ins Gehege kommt, fragt der `LayoutManager` den `Container` nach seinen `Insets` (Einschüben).

`Insets`-Objekte ähneln `Rectangle`-Objekten, haben aber viel weniger Funktionalität. Ihre wichtigste Eigenschaft sind vier `public int`-Variablen, deren Bedeutung das folgende Bild erklärt:



Man erhält die `Insets` eines `Containers` über die `getInsets()`-Methode, die man auch überschreiben muss, wenn das Layout einen anderen Rand beachten soll (aber **Achtung**: Nicht kleiner machen!)

Cursor

Jedes `Component` kann einen anderen Mauszeiger erhalten. Leider erlaubt es das AWT derzeit nicht, eigene Mauszeiger zu erzeugen, aber man kann aus einer Vielzahl von Standard-Zeigern wählen:

```
Cursor.DEFAULT_CURSOR      Cursor.N_RESIZE_CURSOR
Cursor.CROSSHAIR_CURSOR    Cursor.S_RESIZE_CURSOR
Cursor.TEXT_CURSOR         Cursor.E_RESIZE_CURSOR
Cursor.WAIT_CURSOR         Cursor.W_RESIZE_CURSOR
Cursor.HAND_CURSOR         Cursor.NE_RESIZE_CURSOR
Cursor.MOVE_CURSOR         Cursor.NW_RESIZE_CURSOR
                           Cursor.SE_RESIZE_CURSOR
                           Cursor.SW_RESIZE_CURSOR
```

Man erhält ein `Cursor`-Object durch Aufruf der statischen Methode

```
static public Cursor getPredefinedCursor(int type)
```

mit einer der oben genannten Variablen als `type`.

Beispiel:

```
Button b = new Button("WAIT NOW!");
b.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
```

Menus

Es gibt noch die eine weitverzweigte Hierarchie von [Menus](#), [MenuItem](#)s, [PopupMenu](#)s.

Diese können nahezu beliebig geschachtelt werden.

In Java 1.0 war die einzige Möglichkeit, ein Menu darzustellen, einen Menubalken an ein eigenes Fenster zu heften.

In Java 1.1 gibt es komponentenspezifische Popup-Menus!

JavaBeans

- Die portable Version von ActiveX/OCX/...
- JavaBeans folgen einem genau spezifizierten Namensschema: *properties* werden über *get-/set*-Paare adressiert
- das komplette AWT besteht aus Beans
- Beans können unsichtbar sein
- Es gibt Tools zur Einbindung von Beans als ActiveX-Controls (*bridges*)
- Trennung von Bildschirmdarstellung und Funktionalität zwingend notwendig; nicht nur deswegen, aber auch daher das neue Event-Modell!

Ausblick — Swing

- Swing ist der Codename für die AWT-Erweiterungen in den *Java Foundation Classes*, die derzeit von Javasoft (und Netscape) entwickelt werden
- Swing unterstützt JavaBeans
- Swing ist *lightweight*
- Swing hat *pluggable look-and-feel*
- Swing ist schöner!
- Das API ist stabil (Version 1.0)
- Swing ist Bestandteil des JDK-1.2

