2. Grundlagen und Basisklassen des Java-AWT

Vorbemerkungen:

- Im Rahmen des Umfangs der Vorlesung ist es unmöglich, alle AWT- und Swing-Klassen und Konzepte vollständig zu besprechen
  - AWT (JDK 1.3): 313 Klassen
  - Swing (JDK 1.3): 506 Klassen
- In diesem Kapitel wird eine Auswahl von AWT-Basiskomponenten, Hilfsklassen und AWT-Konzepten besprochen
- In einem späteren Kapitel wird analog Swing dargestellt
- Zu den besprochenen Klassen werden die wichtigsten Methoden eingeführt

Klassenhierarchie der Basiskomponenten

```
java.lang.Object
|
+-- java.awt.Component          allgemeine Eigenschaften
    |
    +-- java.awt.Container          kann Kinder besitzen
        |
        +-- java.awt.Window     Toplevel ohne Dekoration
        |   |
        |   +- java.awt.Frame   Toplevel mit Dekoration
        |
        +-- java.awt.Panel          Container mit Layout
            |
            +-- java.awt.Applet     Window für Browser
```

2.1.1 Class java.awt.Component

`java.lang.Object` → `java.awt.Component`

Beschreibung (JDK 1.3 Dokumentation):

A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user.

Bemerkungen:

- **`Component`** ist eine Meta-Klasse, die alle gemeinsamen und allgemein für graphische Komponenten wichtigen Eigenschaften beinhaltet
- Als abstrakte Klasse wird sie nicht instanziiert
- Die Kenntnis der wichtigsten Eigenschaften ist wichtig, da sie von allen Graphik-Klassen übernommen werden

2.1.1 Class java.awt.Component

Eigenschaften: Position x, y, Ausdehnung width, height

Set-Methoden:

```
void setSize ( int width, int height );
void setSize ( java.awt.Dimension d );
void setLocation ( int x, int y );
void setLocation ( java.awt.Point p );
void setBounds ( int x, int y, int width, int height );
void setBounds ( java.awt.Rectangle r );
```

2.1.1 Class java.awt.Component

Eigenschaften: Position x, y, Ausdehnung width, height

Get-Methoden:

```
java.awt.Dimension getSize();
java.awt.Rectangle getBounds();

int getX();          int getWidth();
int getY();          int getHeight();
```

Eigenschaft: Parent-Component

Get-Methode:

```
java.awt.Container getParent();
```

2.1.1 Class java.awt.Component

Eigenschaft: Cursor

Get-Methode:

```
java.awt.Cursor getCursor();
```

Set-Methode:

```
void setCursor ( java.awt.Cursor cursor );
```

Bemerkungen:

- Der Cursor wird automatisch umgeschaltet, wenn der Nutzer die Maus in den Bereich der Komponente bewegt und die Komponente die oberste sichtbare ist

2.1.1 Class java.awt.Component

Eigenschaft: Visibility

Set-Methoden:

```
void setVisible ( boolean b );
```

Beschreibung:

Determines whether this component should be visible when its parent is visible. Components are initially visible, with the exception of top level components such as **Frame** objects.

Get-Methoden:

```
boolean isVisible ( );
```

2.1.1 Class java.awt.Component

Eigenschaft: Enabling

Set-Methoden:

```
void setEnabled ( boolean b );
```

Beschreibung:

An enabled component can respond to user input and generate events. Components are enabled initially by default.

Get-Methoden:

```
boolean isEnabled ( );
```

2.1.1 Class java.awt.Component

Eigenschaft: Vordergrund- und Hintergrundfarbe

Set-Methoden:

```
void setBackground ( java.awt.Color c );
void setForeground ( java.awt.Color c );
```

Get-Methoden:

```
java.awt.Color getBackground();
java.awt.Color getForeground();
```

2.1.1 Class java.awt.Component

Eigenschaft: Font

Set-Methoden:

```
void setFont ( java.awt.Font f );
```

Get-Methoden:

```
java.awt.Font getFont();
```

2.1.1 Class java.awt.Component

Eigenschaft: Toolkit

Get-Methode:

```
java.awt.Toolkit getToolkit();
```

Gets the toolkit of this component. Note that the frame that contains a component controls which toolkit is used by that component. Therefore if the component is moved from one frame to another, the toolkit it uses may change.

2.1.1 Class java.awt.Component

Methode: paint()

Beschreibung:

Paints this component. This method is called when the contents of the component should be painted in response to the component first being shown or damage needing repair. The clip rectangle in the Graphics parameter will be set to the area which needs to be painted.

```
void paint ( Graphics g );

Diese Methode wird von der Anwendung überladen, um die
Graphikausgabe durchzuführen.
```

2.1.2 Class java.awt.Container

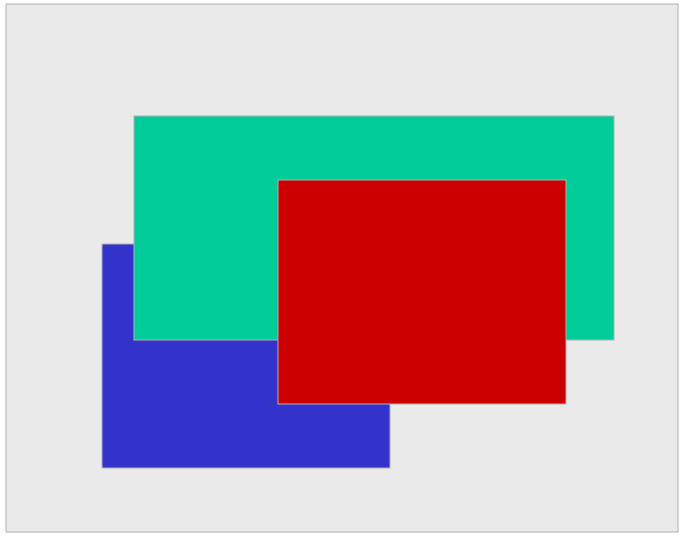
Component → Container

Beschreibung:

A generic Abstract Window Toolkit(AWT) container object is a component that can contain other AWT components. Components added to a container are tracked in a list. The order of the list will define the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list (and hence to the bottom of the stacking order).

Bemerkungen:

- **`Container`** ist eine Meta-Klasse, die nicht instaziiert wird; sie beinhaltet die Methoden, die notwendig sind, um andere Komponenten aufzunehmen
- Die Stacking-Order besagt, welche Kinder bei Überlappungen vorne sind



2.1.2 Class java.awt.Container

Get-Methoden:

```
int getComponentCount();
Component[] getComponents();
Component getComponent ( int n );
LayoutManager getLayoutManager();
```

LayoutManager: siehe Klasse java.awt.Panel

Set-Methoden:

```
void setLayoutManager ( LayoutManager layout );
```

Weitere Methoden:

```
Component add ( Component comp );
void remove ( Component comp );
```

Zentrale Methoden für die Hierarchie

2.1.3 Class java.awt.Frame

Component → Container → Window → Frame

Beschreibung:

A Frame is a top-level window with a title and a border.

Frames are capable of generating the following types of window events: WindowOpened, WindowClosing, WindowClosed, WindowIconified, WindowDeiconified, WindowActivated, WindowDeactivated.

Konstruktoren:

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

2.1.3 Class java.awt.Frame

Eigenschaften: Frame-Titel, Zustand (Iconifiziert), Resizable

Get-Methoden:

```
String getTitle();
int getState();
boolean isResizable();
```

Set-Methoden:

```
void setTitle ( String title );
void setState ( int state );
void setResizable ( boolean resizable );
```

2.1.3 Class java.awt.Frame

Öffentliche Klassen-Konstante für State:

```
static final int NORMAL
static final int ICONIFIED
```

Eigenschaft: Icon-Bild, (Laden mittels `java.awt.Toolkit`)

Get-Methoden:

```
java.awt.Image getIconImage();
```

Set-Methoden:

```
void setIconImage ( java.awt.Image image );
```

2.1.4 Class java.awt.Panel

Component → Container → Panel

Beschreibung:

Panel is the simplest container class. A panel provides space in which an application can attach any other component, including other panels.

Konstruktor:

```
Panel ( Frame owner )
Panel ( LayoutManager layout )
```

Bemerkungen:

Layout-Manager dienen dazu, die Geometrie der Kinder einer Container-Komponente zu verwalten, insbesondere, wenn sich der verfügbare Platz ändert (Beispiel: Nutzer ändert die Größe eines Toplevel-Windows). Die einzelnen Layout-Manager-Klassen werden zu einem späteren Zeitpunkt behandelt. Mit den Beispielen im nächsten Abschnitt wird auch der Layout-Manager **`BorderLayout`** eingeführt.

2.1.5 Class java.awt.Window

Component → Container → Window

Beschreibung:

A Window object is a top-level window with no borders and no menubar. A window must have either a frame, dialog, or another window defined as its owner when it's constructed. Windows are capable of generating the following window events: WindowOpened, WindowClosed.

Konstruktor:

```
Window ( Frame owner )
```

2.1.6 Beispiele

Beispiel: Frame1
Demos: Frame1Demo1, Frame1Demo2

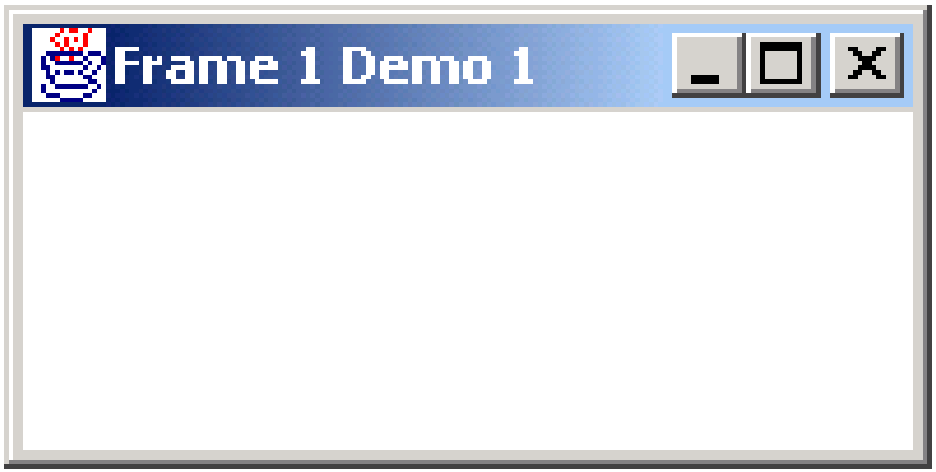
```
public class Frame1 extends Frame {
  public Frame1(String title) {
    setTitle(title + " - (c) by Snoopy Brown");
    }
  public Frame1(String title,int width,int height) {
    this(title);
    setSize(width,height);
    }
  public Frame1(String title,int width,int height,
                int x,int y) {
    this(title); setSize(width,height); setLocation(x,y);
    }
  }
```

Hinweis: Testen Sie setResizable() und setState()

2.1.6 Beispiele

Demo: Frame1Demo1

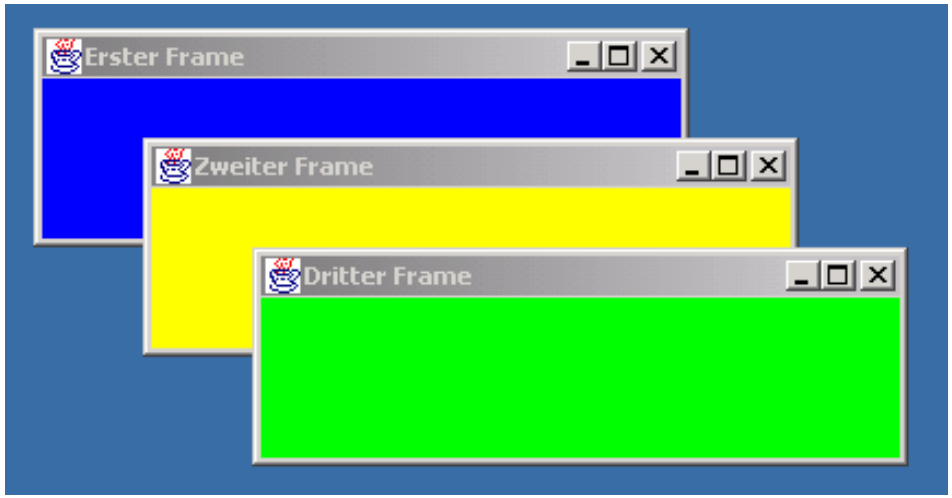
```
public class Frame1Demo1
{

  public static void main (String args[])
  {
  Frame1 frame = new Frame1("Frame 1 Demo 1",
                            200, 100, 400, 400);
  frame.setVisible(true);
  }

}
```



2.1.6 Beispiele

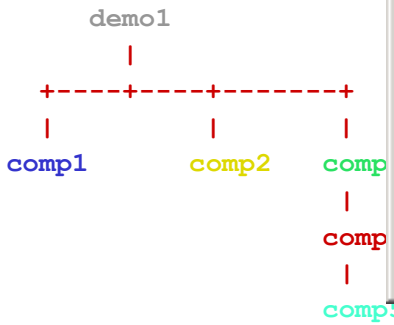
Demo: Frame1Demo2

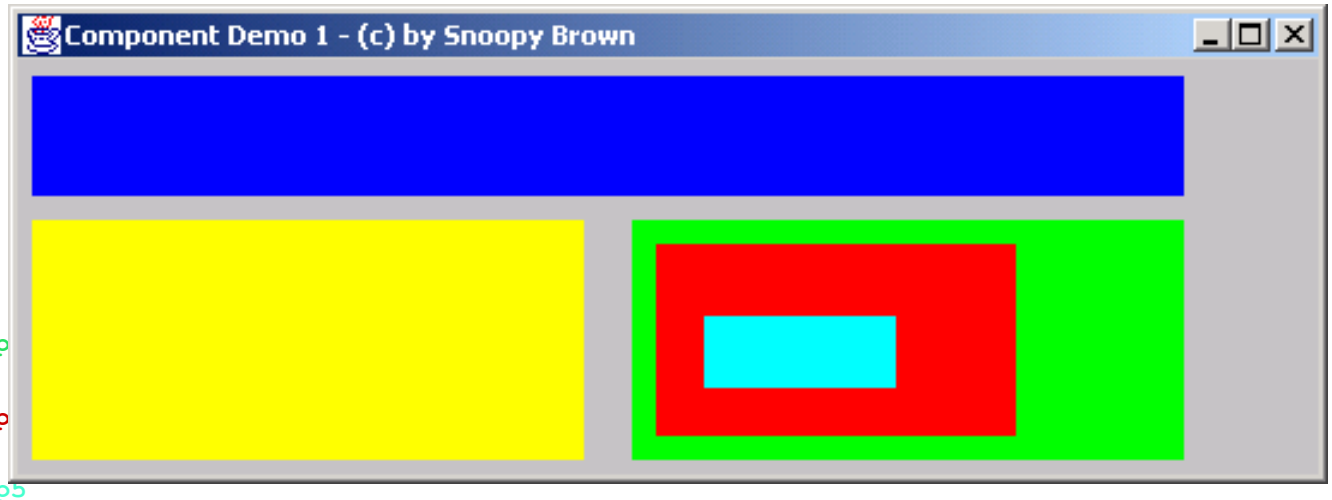
```
public class Frame1Demo2 {
  public static void main(String[] args) {
    Frame1 f1 = new Frame1("Erster Frame",300,100,100,100);
    f1.setBackground(Color.blue);
    Frame1 f2 = new Frame1("Zweiter Frame",300,100,150,150);
    f2.setBackground(Color.yellow);
    Frame1 f3 = new Frame1("Dritter Frame",300,100,200,200);
    f3.setBackground(Color.green);
    f1.setVisible(true);
    f2.setVisible(true);
    f3.setVisible(true);
    }
}
```



2.1.6 Beispiele

Beispiel: Component1
Demo: Component1Demo1





2.1.6 Beispiele

Beispiel: Component1
Demo: Component1Demo1

```
public class ComponentDemo1 extends Frame1
  {
  private Panel comp1, comp2, comp3, comp4, comp5;

  public Component1(String title) {
    super(title);
    setSize(550,200);
    setBackground(Color.lightGray);
    setLayout(null);       // Ausschalten des Layout-Managers

    comp1 = new Panel();          // Erzeugen des ersten Kinds
    comp1.setLocation(10,30);
    comp1.setSize(480,50);
    comp1.setBackground(Color.blue);
    add(comp1);                              // comp1 ins Frame
```

2.1.6 Beispiele

Beispiel: (Fortsetzung)

```
comp2 = new Panel();
comp2.setLocation(10,90);
comp2.setSize(230,100);
comp2.setBackground(Color.yellow);
add(comp2);                                  // comp2 ins Frame

comp3 = new Panel();
comp3.setLocation(260,90);
comp3.setSize(230,100);
comp3.setBackground(Color.green);
comp3.setLayout(null);
add(comp3);                                  // comp3 ins Frame
```

Beispiel: (Fortsetzung)

```
    comp4 = new Panel();
    comp4.setLocation(10,10);
    comp4.setSize(150,80);
    comp4.setBackground(Color.red);
    comp4.setLayout(null);
    comp3.add(comp4);                          // comp4 in comp3

    comp5 = new Panel();
    comp5.setLocation(20,30);
    comp5.setSize(80,30);
    comp5.setBackground(Color.cyan);
    comp4.add(comp5);                          // comp5 in comp4
    }
  } // Ende des Konstruktors
```

2.1.6 Beispiele

Beispiel: Window1
Demo: Window1Demo1

```
public class Window1 extends java.awt.Window {
  private Frame myframe;

  public Window1 ( Frame frame, int x, int y,
                   int width, int height,
                   Color color, Cursor cursor ) {
    super ( frame );
    setLocation ( x, y );
    setSize ( width, height );
    setBackground ( color );
    setCursor ( cursor );
    }
}
```

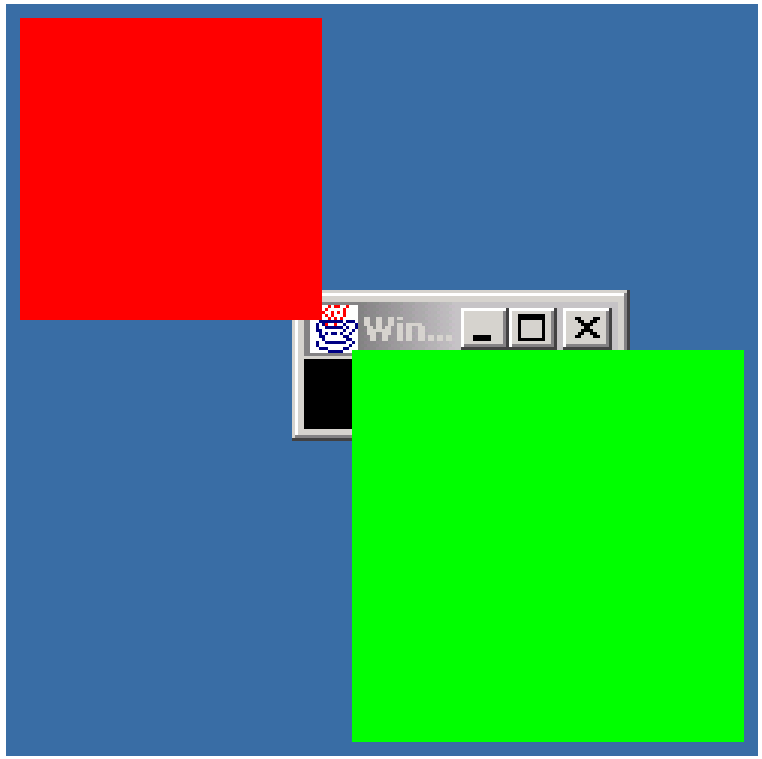
2.1.6 Beispiele

Demo: Window1Demo1

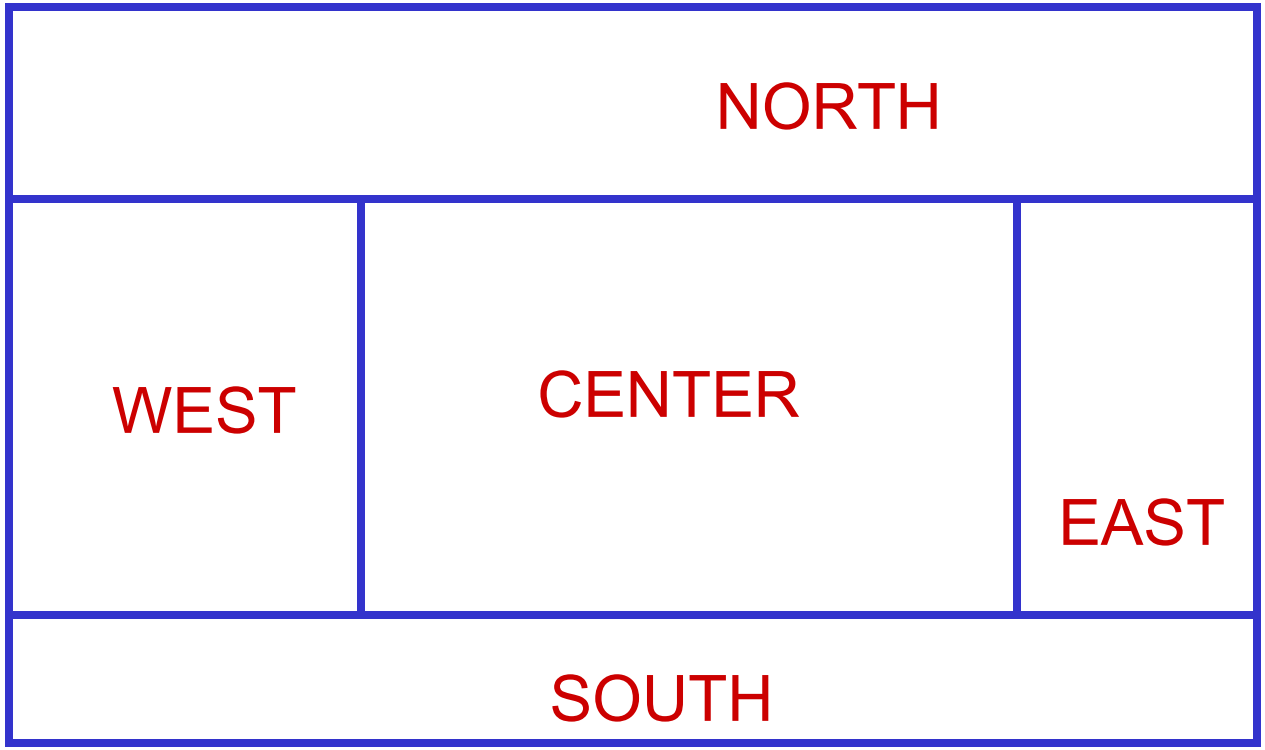
```
public class Window1Demo1 extends Frame1 {

  public Window1Demo1(String title,int width,int height,
                      int x,int y, Color c) {
    super(title,width, height, x, y);
    setBackground(c);

    Window1 window1 = new Window1 ( this,10,10,100,100,
      Color.red, new Cursor (Cursor.MOVE_CURSOR) );
    window1.setVisible(true);
    Window1 window2 = new Window1 ( this,120,120,130,130,
      Color.green, new Cursor (Cursor.HAND_CURSOR) );
    window2.setVisible(true);
  }
```

2.1.6 Beispiele

```
(Fortsetzung)

public static void main (String args[])  {
  Window1Demo1 demo = new Window1Demo1("Window1Demo1",
    50,50, 100, 100, Color.black );
  demo.setCursor ( new Cursor(Cursor.CROSSHAIR_CURSOR) );
  demo.setVisible(true);
  }

}
```



Layout-Management der Klasse java.awt.BorderLayout



2.1.6 Beispiele

Beispiel: BorderLayoutDemo1

```
public class BorderLayoutDemo1 extends Frame {

  private Panel north, south, east, west, center;

  public BorderLayoutDemo1(String title) {

    setTitle(title);

    north = new Panel();
    add ( north, BorderLayout.NORTH );
    north.setBackground ( Color.black );

    south = new Panel();
    add ( south, BorderLayout.SOUTH );
    south.setBackground ( Color.red );
```

Beispiel: (Fortsetzung)

```
    east = new Panel();
    add ( east, BorderLayout.EAST );
    east.setBackground ( Color.green );

    west = new Panel();
    add ( west, BorderLayout.WEST );
    west.setBackground ( Color.blue );

    center = new Panel();
    add ( center, BorderLayout.CENTER );
    center.setBackground ( Color.cyan );

    }

... main(): siehe Beispielprogramm
```

2.1.6 Beispiele

Beispiel: `BorderLayoutDemo1`

Auswahl einiger wichtigster Hilfsklassen

- Geometrieklassen
  - Point
  - Rectangle
  - Dimension
- Color, SystemColor
- Cursor
- Font
- Toolkit

2.2.1 Class java.awt.Point

Beschreibung:

A point representing a location in (x, y) coordinate space, specified in integer precision.

Öffentliche Instanzvariablen:

```
public int x
public int y
```

2.2.1 Class java.awt.Point

Konstruktoren:

```
Point()
Point ( int x, int y )
Point ( Point p )
```

Set-Methoden:

```
void setLocation ( int x, int y );
void setLocation ( Point p );

void translate ( int dx, int dy );
```

2.2.2 Class java.awt.Rectangle

Beschreibung:

A Rectangle specifies an area in a coordinate space that is enclosed by the Rectangle object's top-left point (*x*, *y*) in the coordinate space, its width, and its height.
A Rectangle object's width and height are public fields. The constructors that create a Rectangle, and the methods that can modify one, do not prevent setting a negative value for width or height.

2.2.2 Class java.awt.Rectangle

Öffentliche Instanzvariablen:

```
public int width
public int height
public int x
public int y
```

Konstruktoren:

```
Rectangle()
Rectangle ( Dimension d )
Rectangle ( int width, int height )
Rectangle ( int x, int y, int width, int height )
Rectangle ( Point p, Dimension d )
Rectangle ( Rectangle r )
```

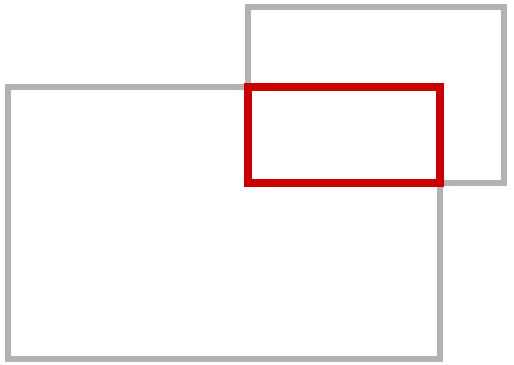
2.2.2 Class java.awt.Rectangle

Get-Methoden:

```
java.awt.Point getLocation();
java.awt.Dimension getSize();
```

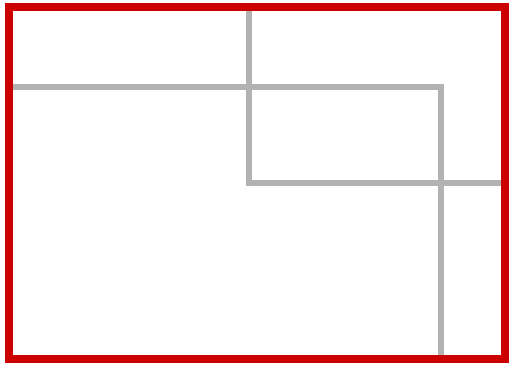
Set-Methoden:

```
void setLocation ( Point p );
void setLocation ( int x, int y );
void setSize ( Dimension d );
void setSize ( int width, int height );
void grow ( int w, int h );
void translate ( int dx, int dy );
```



Weitere Methoden:

```
Rectangle intersection ( Rectangle r );
Rectangle union ( Rectangle r );
```



2.2.3 Class java.awt.Dimension

Beschreibung:

The Dimension class encapsulates the width and height of a component (in integer precision) in a single object. (...) Normally the values of width and height are non-negative integers. The constructors that allow you to create a dimension do not prevent you from setting a negative value for these properties. If the value of width or height is negative, the behavior of some methods defined by other objects is undefined.

2.2.3 Class java.awt.Dimension

Konstruktoren:

```
Dimension()
Dimension ( Dimension d )
Dimension ( int width, int height )
```

Öffentliche Instanzvariablen:

```
public int width
public int height
```

Set-Methoden:

```
void setSize ( int width, int height );
```

2.2.4 Class java.awt.Color

Beschreibung:

A class to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace. Every color has an implicit alpha value of 1.0 or an explicit one provided in the constructor.

Konstruktoren:

```
Color ( r, g, b );      } int
Color ( r, g, b, a );   } 0 ... 255

Color ( r, g, b );      } float
Color ( r, g, b, a );   } 0 ... 1.0
```

```
a: alpha-Wert
  =0 transparent
  =1 deckend
```

2.2.4 Class java.awt.Color

Öffentliche Klassenkonstante:

```
static final Color black        static final Color blue
static final Color cyan         static final Color
darkGray
static final Color gray         static final Color green
static final Color lightGray    static final Color magenta
static final Color orange       static final Color pink
static final Color red          static final Color white
static final Color yellow
```

Get-Methoden:

```
int getRed();      int getGreen();      0 ... 255
int getBlue();     int getAlpha();
```

Weitere Methoden:

```
Color brighter();
```

Beschreibung:

This method applies an arbitrary scale factor to each of the three RGB components of this Color to create a brighter version of this Color. Although brighter and **darker** are inverse operations, the results of a series of invocations of these two methods might be inconsistent because of rounding errors.

```
Color darker();
```

2.2.5 Class java.awt.SystemColor

`Color → SystemColor`

Beschreibung:

A class to encapsulate symbolic colors representing the color of GUI objects on a system. For systems which support the dynamic update of the system colors (when the user changes the colors) the actual RGB values of these symbolic colors will also change dynamically.

2.2.5 Class java.awt.SystemColor

Öffentliche Klassenkonstante:

```
static final SystemColor activeCaption
static final SystemColor acticveCaptionBorder
static final SystemColor activeCaptionText

static final SystemColor inactiveCaption
static final SystemColor inacticveCaptionBorder
static final SystemColor inactiveCaptionText

static final SystemColor control
static final SystemColor controlDkShadow
static final SystemColor controlHighlight
static final SystemColor controlShadow
static final SystemColor controlText
```

2.2.5 Class java.awt.SystemColor

Öffentliche Klassenkonstante:

```
static final SystemColor info
static final SystemColor menu
static final SystemColor menuText
static final SystemColor scrollbar

static final SystemColor text
static final SystemColor textHighlight
static final SystemColor textHighlightText
static final SystemColor textInactiveText
static final SystemColor textText

static final SystemColor window
static final SystemColor windowBorder
static final SystemColor windowText
```

2.2.6 Class java.awt.Cursor

Beschreibung:

A class to encapsulate the bitmap representation of the mouse cursor.

Konstruktoren:

```
Cursor ( type )
```

```
Type ist eine Konstante aus der Cursor-Klasse.
```

```
Bsp.:
```

```
Cursor cur = new Cursor ( Cursor.CROSSHAIR_CURSOR );
```

2.2.6 Class java.awt.Cursor

Öffentliche Klassenkonstante:

```
static final int CROSSHAIR_CURSOR
static final int CUSTOM_CURSOR
static final int DEFAULT_CURSOR
static final int E_RESIZE_CURSOR
static final int HAND_CURSOR
static final int MOVE_CURSOR
static final int N_RESIZE_CURSOR
static final int S_RESIZE_CURSOR
static final int SE_RESIZE_CURSOR
static final int SW_RESIZE_CURSOR
static final int TEXT_CURSOR
static final int W_RESIZE_CURSOR
static final int WAIT_CURSOR
```

2.2.7 Class java.awt.Font

Fonts

- Typeface oder **Font** (Zeichensatz): beschreibt das Gesamtdesign einer zusammen gehörigen Menge von Zeichen
- Beispiele: Times, Helvetica, Arial, Courier
- Unterscheidung: **Serif Fonts** und **Sans Serif Fonts**

Arial

m

Times

m

2.2.7 Class java.awt.Font

Character Ein Symbol, das ein Zeichen repräsentiert

g „klein-g“

€ „Euro“

Glyph Darstellung eines Zeichen in einem bestimmten Font

	Arial	Times	Courier
„klein-g“	g	g	g
„Euro“	€	€	€

Fonts

- Unterscheidung: **Proportional-Fonts** und **Monospaced Fonts**

Times MIW

Courier MIW

2.2.7 Class java.awt.Font

Fonts

- Unterscheidung: **Bitmap-Fonts** und **Outline-Fonts** (skalierbare Fonts)

Bitmap-Fonts

- liegen als Raster-Zeichensätze in den jeweiligen Größen und Stilen vor
- werden einfach aus dem Fontfile geladen und das Muster des Zeichens wird in den Frame Buffer kopiert
- sind nicht skalierbar, d.h. wenn eine Fontgröße nicht existiert, existiert sie nicht
- brauchen mehr Speicher (Laden des gleichen Fonts in mehreren Größen)
- sind für die jeweilige Fontgröße optimierbar

2.2.7 Class java.awt.Font

Outline-Fonts

- werden geometrisch durch Linien und Kurvensegmente beschrieben (i.d.R. durch Splines)
- sind daher vollständig skalierbar
- brauchen weniger Speicher, da ein Font nur einmal geladen wird
- brauchen mehr Rechenzeit, da sie zur Laufzeit gerastert werden müssen
- sind u.U. für bestimmte Fontgrößen optisch weniger optimal als Bitmap-Fonts

2.2.7 Class java.awt.Font

Font-Styles

- Plain — Plain Font
- Bold — **Bold Font**
- Italic — *Italic Font*
- Bold+Italic — ***Bold Italic Font***

Öffentliche Klassenkonstante:

```
static final int PLAIN
static final int BOLD
static final int ITALIC
```

2.2.7 Class java.awt.Font

Konstruktoren:

```
Font ( String name, int size, int style )
```

name - the font name. This can be a **logical font name** or a **font face** name. A logical name must be either: Dialog, DialogInput, Monospaced, Serif, SansSerif, or Symbol. If name is null, the name of the new Font is set to Default.

style - the style constant for the Font The style argument is an integer bitmask that may be PLAIN, or a bitwise union of BOLD and/or ITALIC (for example, ITALIC or BOLD|ITALIC). Any other bits set in the style parameter are ignored. If the style argument does not conform to one of the expected integer bitmasks then the style is set to PLAIN.

size - the **point size** of the Font **(Typographische Punkte)**

2.2.8 Class java.awt.Toolkit

Beschreibung:

This class is the abstract superclass of all actual implementations of the Abstract Window Toolkit. Subclasses of Toolkit are used to bind the various components to particular native toolkit implementations.

Methoden

```
void beep();
int getScreenResolution();
java.awt.Dimension getScreenSize();
```

Methoden

```
java.awt.Image getImage ( URL url );

java.awt.Image getImage ( String filename );
```

Returns an image which gets pixel data from the specified file, whose format can be either GIF, JPEG or PNG. The underlying toolkit attempts to resolve multiple requests with the same filename to the same returned Image. Since the mechanism required to facilitate this sharing of Image objects may continue to hold onto images that are no longer of use for an indefinate period of time, developers are encouraged to implement their own caching of images by using the createImage variant wherever available.

Methoden

```
java.awt.Image createImage ( URL url );

java.awt.Image createImage ( String filename );
```

Returns an image which gets pixel data from the specified file. The returned Image is a new object which will not be shared with any other caller of this method or its getImage variant.