

Praktische Informatik 1

Abstract Window Toolkit

Thomas Röfer

- Desktop-Metapher
- AWT und Swing
- Applets und Anwendungen
- Rahmen und Ereignisbehandlung
- Grafik und Schriften
- Menüs und Komponenten
- Container und Layouts
- Musterlösung

Grafik

- In allen Komponenten kann gezeichnet werden, indem man *paint(Graphics)* überschreibt
- Aber es kann zu Konflikten mit den Zeichenroutinen der Komponenten selbst kommen
- Allgemeine Zeichenfläche
 - *class Canvas*
 - In Swing gibt es kein *Canvas*, stattdessen z.B. *(J)Panel*
- Zeichenschnittstelle
 - *interface Graphics*
 - Wird an *paint* übergeben
 - Arbeitet zustandsbasiert
 - Speichert den aktuellen Zustand des Zeichenwerkzeugs (Farbe, Schriftart usw.)

Grafik – Beispiel

- Erweiterung der Implementierung von *BoulderDash* in eine AWT-Anwendung
- Nutzung der Musterlösungsimplementierung
- Genutzte Schnittstelle der Musterlösung
 - *public void step(int dir)* – Ausführen eines Schritts
 - *protected char[][] field* – Das Feld
 - *protected void print()* – Ausgabe des Spielfeldes
 - *protected void printMessage(String message)* – Ausgabe des Grunds für das Spielende
- Ansatz: Leite lokale Klasse *Wrapper* von *BoulderDash* ab und überschreibe einige Methoden und füge neue für die graphische Ausgabe hinzu
 - Wrapper hat Zugriff auf die Attribute und Methoden der Hauptfensterklasse *BoulderDashGUI*

Grafik – Beispiel

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;

class BoulderDashGUI extends Frame
    implements KeyListener, WindowListener {
    class Wrapper extends BoulderDash {
        private int IMAGE_SIZE = 64;
        private Image[] images = new Image[96];

        public Wrapper() {
            images[' '] = Toolkit.getDefaultToolkit().getImage("Empty.png");
            images['M'] = Toolkit.getDefaultToolkit().getImage("Wall.png");
            images['E'] = Toolkit.getDefaultToolkit().getImage("Sand.png");
            images['A'] = Toolkit.getDefaultToolkit().getImage("Exit.png");
            images['S'] = Toolkit.getDefaultToolkit().getImage("Player.png");
            images['O'] = Toolkit.getDefaultToolkit().getImage("Stone.png");
            images['U'] = Toolkit.getDefaultToolkit().getImage("Monster.png");
            setPreferredSize(new Dimension(field[0].length * IMAGE_SIZE,
                field.length * IMAGE_SIZE));
        }
    }
}
```

Grafik – Beispiel

```
protected void print() {} // unterdrücke Konsolenausgabe

void paint(Graphics g) {
    for (int y = 0; y < field.length; ++y) {
        for (int x = 0; x < field[0].length; ++x) {
            g.drawImage(images[field[y][x]], x * IMAGE_SIZE, y * IMAGE_SIZE,
                IMAGE_SIZE, IMAGE_SIZE, BoulderDashGUI.this);
        }
    }
}
```


Grafik – Beispiel

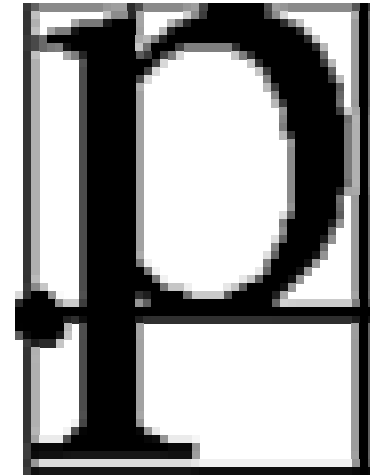
```
Wrapper game;
```

```
BoulderDashGUI() {  
    super("Boulder Dash");  
    game = new Wrapper();  
    addKeyListener(this);  
    addWindowListener(this);  
    pack();  
    setVisible(true);  
}  
  
public void paint(Graphics g) {  
    game.paint(g);  
}
```

```
public void keyPressed(KeyEvent e) {  
    switch (e.getKeyCode()) {  
        case KeyEvent.VK_UP:  
            game.step(0);  
            break;  
        case KeyEvent.VK_LEFT:  
            game.step(1);  
            break;  
        case KeyEvent.VK_DOWN:  
            game.step(2);  
            break;  
        case KeyEvent.VK_RIGHT:  
            game.step(3);  
            break;  
    }  
    repaint();  
}  
  
public void keyReleased(KeyEvent e) {}  
public void keyTyped(KeyEvent e) {}  
:
```

Schriften

- Begriffe
 - Grundlinie, Oberlänge, Unterlänge, Höhe
 - Proportionalschrift, Schrittweite
 - Unterschneidung, Ligaturen
 - Serifen
- Auswahl einer Schrift (*Font*)
 - Name, z.B. „SansSerif“
 - Stil, (ver-odert) z.B. *Font.BOLD* | *Font.ITALIC*
 - Größe in Punkten (1pt = 1/72 Zoll), z.B.
 - `g.setFont(new Font("SansSerif", Font.PLAIN, 16));`
- Informationen über eine Schrift (*FontMetrics*)
 - `getAscent()`, `getDescent()`, `getHeight()`
 - `stringWidth(String)`



Schrift – Beispiel

```
class Wrapper extends BoulderDash {  
    :  
    private Font font = new Font("SansSerif", Font.BOLD, IMAGE_SIZE);  
    private String message = null;  
    :  
    protected void printMessage(String message) {  
        this.message = message;  
    }  
  
    public void step(int dir) {  
        if (message == null) {  
            super.step(dir);  
        }  
    }  
}
```

```
void paint(Graphics g) {  
    :  
    if (message != null) {  
        g.setFont(font);  
        int width = g.getFontMetrics()  
            .stringWidth(message);  
        int x = (field[0].length * IMAGE_SIZE - width) / 2;  
        int y = (field.length - 1) * IMAGE_SIZE / 2;  
        g.setColor(Color.BLACK);  
        g.fillRect(x, y, width, IMAGE_SIZE);  
        g.setColor(Color.WHITE);  
        g.drawString(message, x, y + IMAGE_SIZE -  
            g.getFontMetrics().getMaxDescent());  
    }  
}
```


Menüs

- *MenuBar*
 - Die Menüleiste
 - Es können *Menus* und *MenuItems* hinzugefügt werden
 - Es gibt speziellen Eintrag für das Hilfe-Menü
 - Wird auf manchen Plattformen rechtsbündig dargestellt
 - Ist in *JMenuBar* nicht implementiert!
- *Menu*
 - Container für Menüeinträge und (Unter)Menüs
- *MenuItem*
 - Ein Menüeintrag
 - Es kann ein *ActionListener* zugeordnet werden
 - Diesem wird ein *ActionEvent* übergeben, dessen Methode *getActionCommand()* den Titel des gewählten Menüeintrags enthält

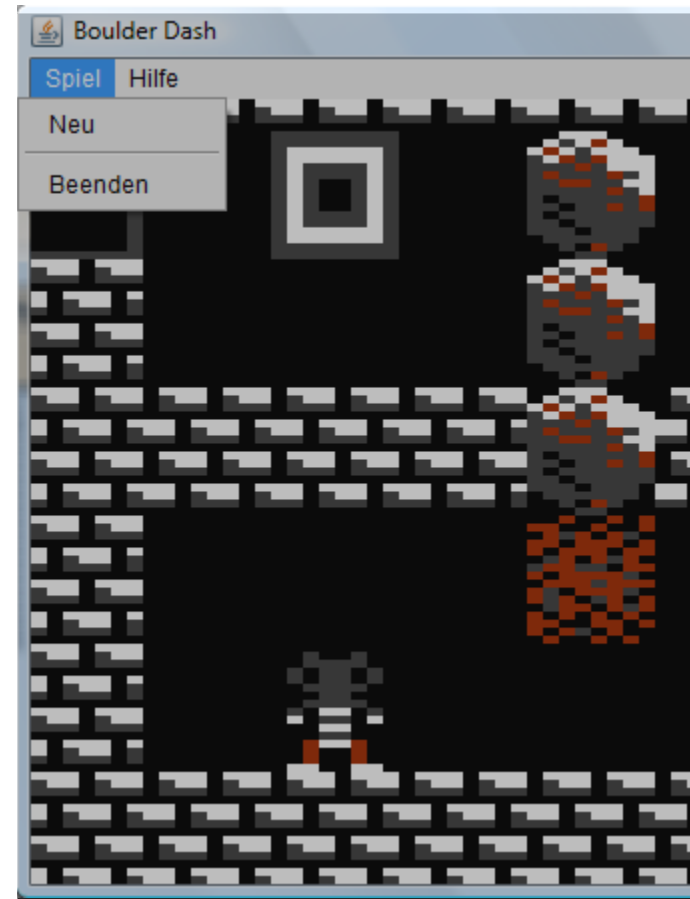


Menüs – Beispiel

```
class BoulderDashGUI extends Frame
    implements KeyListener, WindowListener, ActionListener {
    :
    BoulderDashGUI() {
        :
        setMenuBar(createMenu());
        :
    }
    :
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Neu")) {
            game = new Wrapper();
            repaint();
        } else if (e.getActionCommand().equals("Beenden")) {
            windowClosing(null);
        } else if (e.getActionCommand().equals("Über Boulder Dash...")) {
        }
    }
}
```

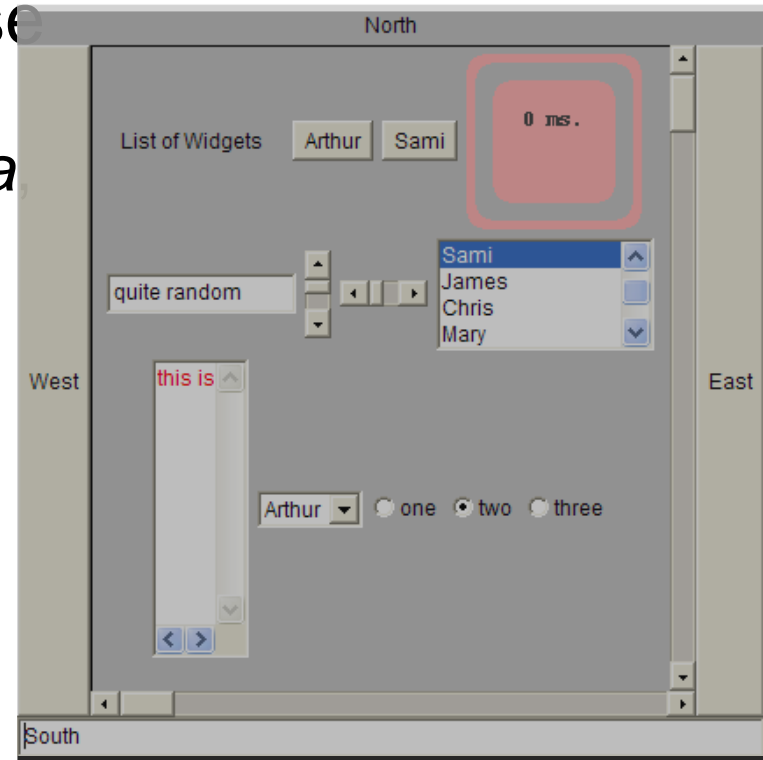
Menüs – Beispiel

```
private MenuBar createMenu() {  
    MenuBar mb = new MenuBar();  
    Menu m = new Menu("Spiel");  
    MenuItem mi = new MenuItem("Neu");  
    mi.addActionListener(this);  
    m.add(mi);  
    m.addSeparator();  
    mi = new MenuItem("Beenden");  
    mi.addActionListener(this);  
    m.add(mi);  
    mb.add(m);  
    m = new Menu("Hilfe");  
    mi = new MenuItem("Über Boulder Dash...");  
    mi.addActionListener(this);  
    m.add(mi);  
    mb.setHelpMenu(m);  
    return mb;  
}
```



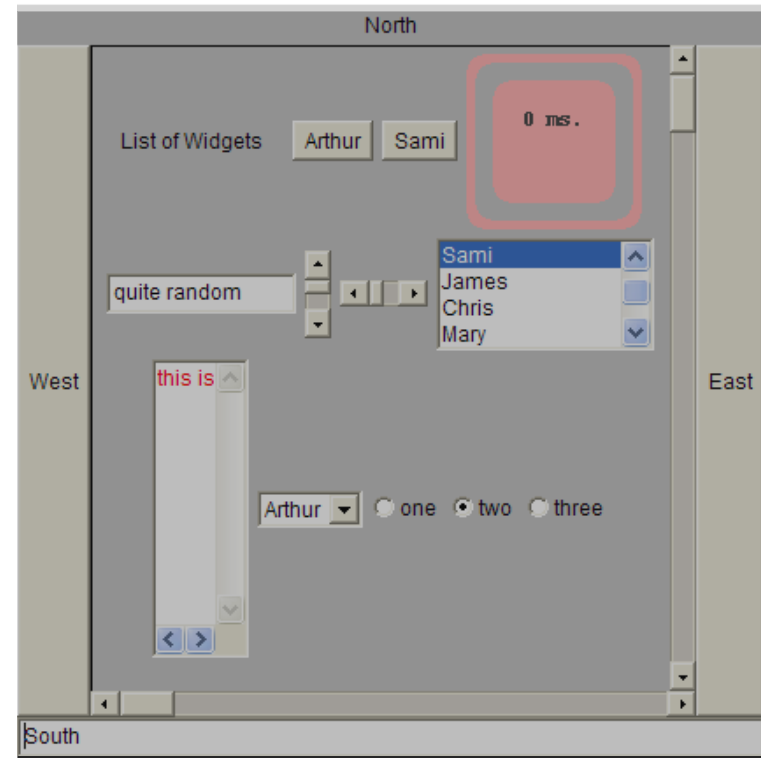
Komponenten

- *Component* ist die Basisklasse aller interaktiven Elemente
 - *Label, List, Scrollbar, TextArea, TextField, Choice, Button, ...*
 - Alle Container
- Methoden, z.B.
 - *set/getBackground(Color)*
 - *set/getFont(Font)*
 - *setVisible()*
 - *set/getLocation (int, int)*
 - *set/get[Preferred]Size(int, int)*
 - *paint(Graphics), update(Graphics), repaint()*



Container und Layouts

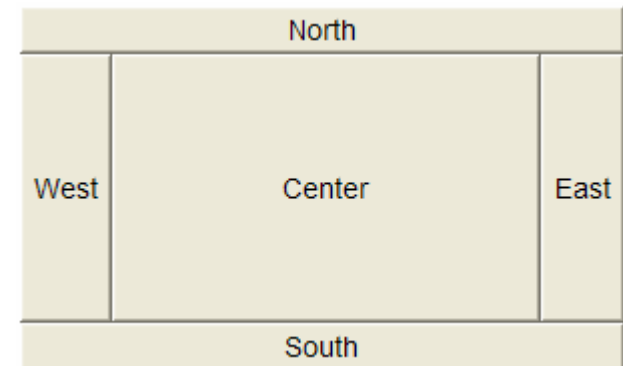
- **Container**
 - Rechteckiger Bereich, der andere Komponenten enthalten kann
 - Ihm ist ein *LayoutManager* zugeordnet
 - *Panel*, *Window*, *ScrollPane*
- **Window**
 - Unterstützt *pack()*, um seine Größe so anzupassen, dass alle Komponenten ihre *preferredSize* erhalten
- **LayoutManager**
 - Erlauben die Anordnung von Komponenten nach vorgegebenen Kriterien
 - Notwendig für Plattformunabhängigkeit
 - *BorderLayout*, *CardLayout*, *FlowLayout*, *GridLayout*,



Container und Layouts

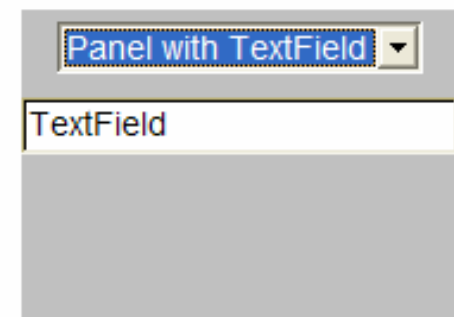
- *BorderLayout*

- Standard-*LayoutManager* für alle Fenster, z.B. Rahmen und Dialoge
- Speichert Komponenten in fünf Bereichen
 - *North, South, East, West und Center*
- Der überschüssige Raum wird dem mittleren Bereich zugeordnet



- *CardLayout*

- Speichert mehrere Komponenten, die nacheinander angezeigt werden sollen, d.h. man kann zwischen verschiedenen Komponenten



Container – Beispiel

```
public Wrapper() {  
    :  
    gameArea.setPreferredSize(new Dimension(  
        field[0].length * IMAGE_SIZE, field.length * IMAGE_SIZE));  
}  
    :  
Canvas gameArea;  
  
BoulderDashGUI() {  
    :  
    gameArea = new Canvas() {  
        public void paint(Graphics g) {  
            game.paint(g);  
        }  
    };  
    add(gameArea);  
    gameArea.addKeyListener(this);  
    :  
    public void paint(Graphics g) {  
        gameArea.repaint();  
    }  
}
```

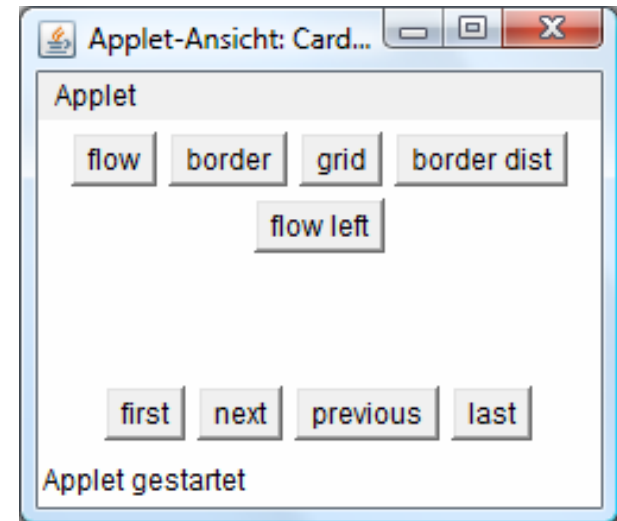
Container und Layouts

- *FlowLayout*

- Standard-*LayoutManager* für Panels
- Kann beliebig viele Komponenten enthalten, die an den Fenstergrenzen umgebrochen werden
- Unterschiedliche Ausrichtungen
 - *FlowLayout.LEFT*,
FlowLayout.CENTER,
FlowLayout.RIGHT

- *GridLayout*

- Ordnet Komponenten in einem Gitter an
- Alle Komponenten haben die gleiche Größe



Button 1	Button 2
Button 3	Button 4
Button 5	

GridLayout – Beispiel

```
class AboutBox extends Dialog implements WindowListener, ActionListener {
    AboutBox(Frame owner) {
        super(owner, "Über Boulder Dash", true);
        addWindowListener(this);

        setLayout(new GridLayout(3,1));
        add(new Label("Boulder Dash", Label.CENTER));
        add(new Label("Version 1.0", Label.CENTER));
        Button b = new Button("OK");
        b.addActionListener(this);
        add(b);

        Rectangle r = owner.getBounds();
        setBounds(r.x + r.width / 2 - 100, r.y + r.height / 2 - 60, 200, 120);
        setVisible(true);
    }

    public void windowClosing(WindowEvent e) {
        dispose();
    }
}
```

```
        :
        new AboutBox(this);
        :
```

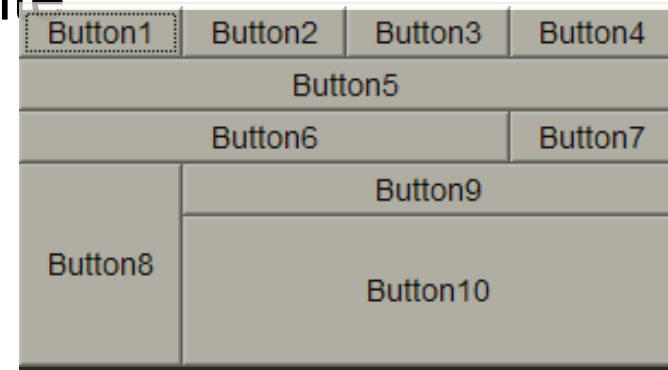
Container und Layouts

- *GridBagLayout*

- Kompliziertester, aber auch mächtigster *LayoutManager*
- Erlaubt die Zuordnung sog. Einschränkungen (*Constraints*) zu jeder Komponente

- *GridBagConstraints*

- Legt die Einschränkungen für Komponenten in *GridBagLayout* fest, z.B.
- Relative Breite und Höhe
 - *gridwidth, gridheight*
 - Anzahl Zellen, *GridBagConstraints.RELATIVE, GridBagConstraints.REMAINDER*
- Gewichtungen
 - *weightx, weighty*



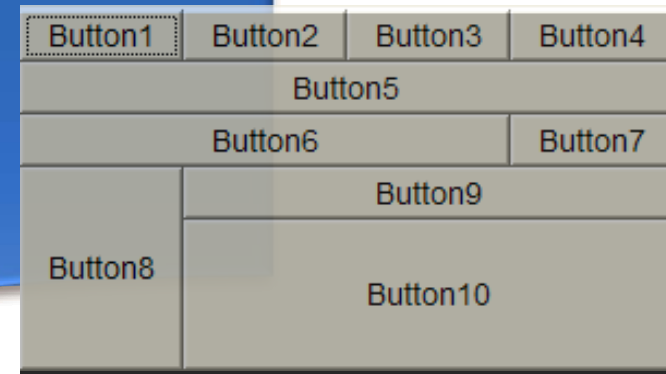
GridBagLayout – Beispiel

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

public class GridBagApplet extends Applet {
    private void makebutton(String name, GridBagConstraints c) {
        Button button = new Button(name);
        ((GridBagLayout) getLayout()).setConstraints(button, c);
        add(button);
    }

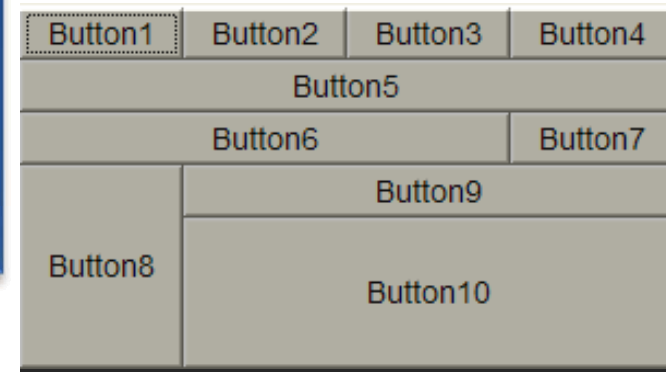
    public void init() {
        setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        makebutton("Button1", c);
        makebutton("Button2", c);
        makebutton("Button3", c);

        c.gridwidth = GridBagConstraints.REMAINDER;
        makebutton("Button4", c);
    }
}
```



GridBagLayout – Beispiel

```
c.weightx = 0.0;  
makebutton("Button5", c);  
  
c.gridwidth = GridBagConstraints.RELATIVE;  
makebutton("Button6", c);  
c.gridwidth = GridBagConstraints.REMAINDER;  
makebutton("Button7", c);  
  
c.gridwidth = 1;  
c.gridheight = 2;  
c.weighty = 1.0;  
makebutton("Button8", c);  
  
c.weighty = 0.0;  
c.gridwidth = GridBagConstraints.REMAINDER;  
c.gridheight = 1;  
makebutton("Button9", c);  
makebutton("Button10", c);  
  
setSize(300, 150);  
}
```



Musterlösung zu Übungsblatt 10

- Aufgabe 1
 - Teilweise dürftige Fehlerbehandlung
 - Doppelter Code
- Aufgabe 2
 - Jede Produktion durch eine Methode implementieren
 - Rekursiver Abstieg in den Ausdruck
 - kein „Klammern zählen“
 - „a | b“ ist nicht „[a] [b]“
 - Teilweise nicht alle Fehlerfälle abgefangen

Praktische Informatik I WS 2007/08

Übungsblatt 10

Musterlösung

Aufgabe 1 Warteschlangen unbeschränkt stapeln (40%)

In dem auf der PI-1 Webseite bereitgestellten Paket uebung10.zip sind Implementierungen für einen beschränkten Stapel und eine beschränkte Warteschlange enthalten. Ersetzt beide durch unbeschränkte Varianten, die auf einfach verketteten Listen basieren. Die Schnittstelle soll dabei erhalten bleiben, d.h. die Methoden push, pop, top und empty sollen ihre Signaturen behalten.

```
1  /**
2   * Die Klasse stellt ein Element einer einfach verketteten Liste
3   * dar. Sie wird von StackAndQueueBase, Stack und Queue verwendet.
4   */
5  class Element {
6      /** Der in diesem Listenelement gespeicherte Wert. */
7      int value;
8
9      /**
10       * Eine Referenz auf das nächste Element in der Liste. null
11       * steht dabei für das Ende der Liste.
12       */
13      Element next;
14
15      /**
16       * Konstruktor zum Erzeugen eines neuen Listenelements.
17       * @param value Der zu speichernde Wert.
18       * @param next Eine Referenz auf das nächste Element in der
19       * Liste. null steht dabei für das Ende der
20       * Liste.
21       */
22      Element(int value, Element next) {
23          this.value = value;
24          this.next = next;
25      }
26  }
27
28  /**
29   * Die Klasse realisiert alle Methoden, die einem unbeschränkten
30   * Stapel und einer unbeschränkten Warteschlange gemeinsam sind.
31   * Stapel bzw. Warteschlange werden als einfache Verkettung von
32   * Instanzen der Klasse Element realisiert.
33   */
34  class StackAndQueueBase {
35      /** Der Anfang der Liste. Elemente werden immer von hier entnommen. */
```