JavaC - Zusammenfassung

Christina van der Geest

28. Februar 2019

Inhaltsverzeichnis

1 AWT Window

```
import java.awt.*;
@SuppressWarnings("serial")
public class MyAwtWindow extends Frame
{
    // Attribute:
    private int anzahl_ = 0;
    private Label label_;
    private Button incButton_;
    private Button decButton_;
    //-------------------------------------------------------------------------
    public int getAnzahl()
    {
        return anzahl_;
    }
    public void setAnzahl(int n)
    {
        anzahl_ = n;
    }
    //-------------------------------------------------------------------------
    public void setLabelText (String t)
    {
        label_.setText(t);
    }
    //-------------------------------------------------------------------------
    public MyAwtWindow()
    // Konstruktor
    {
        // *** Prolog:
        super ("Ihr Name");
        setSize(300, 200); // Breite, Hoehe

        // *** Ansicht (Darstellung) aufbauen:
        label_ = new Label("Anzahl = " + anzahl_, Label.CENTER);
        label_.setBackground(Color.GREEN);
        incButton_ = new Button("Inkrement");
        decButton_ = new Button("Dekrement");

        setLayout(new GridLayout(1, 3)); // 1 Zeile, 3 Spalten
        add(incButton_);
        add(label_);
        add(decButton_);

        // *** Event-Handling initialisieren:
        IncButtonListener ibl = new IncButtonListener( this );
        incButton_.addActionListener( ibl );
        // Alternative: mit anonymem Objekt -> Objekt ohne Namen
        // Direkte Uebergabe der Klasse an ein Objekt (hier ans Button-Objekt)
        // incButton_.addActionListener( new IncButtonListener(this) );
        decButton_.addActionListener( new DecButtonListener(this) );
        addWindowListener(new MyWindowListener() );

        // pack(); // Minimalgroesse einnehmen
        setVisible(true);
    }
    //-------------------------------------------------------------------------
    public static void main(String[] args)
    {
        MyAwtWindow mw1 = new MyAwtWindow();
        mw1.setLocation(10, 10);
        MyAwtWindow mw2 = new MyAwtWindow();
        mw2.setLocation(100, 150);
    }
}
```

1.1 Action & Window Listener in externer Klasse

1.1.1 MyAwtWindow

```
import java.awt.*;
@SuppressWarnings("serial")
public class MyAwtWindow extends Frame
{
    // Attribute:
    private int anzahl_ = 0;
    private Label label_;
    private Button incButton_;
    private Button decButton_;
    //-------------------------------------------------------------------------
    public int getAnzahl()
    {
        return anzahl_;
    }
    public void setAnzahl(int n)
    {
        anzahl_ = n;
    }
    //-------------------------------------------------------------------------
    public void setLabelText (String t)
    {
        label_.setText(t);
    }
    //-------------------------------------------------------------------------
    public MyAwtWindow()
    // Konstruktor
    {
        // *** Prolog:
        super ("Ihr Name");
        setSize(300, 200); // Breite, Hoehe

        // *** Ansicht (Darstellung) aufbauen:
        label_ = new Label("Anzahl = " + anzahl_, Label.CENTER);
        label_.setBackground(Color.GREEN);
        incButton_ = new Button("Inkrement");
        decButton_ = new Button("Dekrement");

        setLayout(new GridLayout(1, 3)); // 1 Zeile, 3 Spalten
        add(incButton_);
        add(label_);
        add(decButton_);

        // *** Event-Handling initialisieren:
        IncButtonListener ibl = new IncButtonListener( this );
        incButton_.addActionListener( ibl );
        // Alternative: mit anonymem Objekt -> Objekt ohne Namen
        // Direkte Uebergabe der Klasse an ein Objekt (hier ans Button-Objekt)
        // incButton_.addActionListener( new IncButtonListener(this) );
        decButton_.addActionListener( new DecButtonListener(this) );
        addWindowListener(new MyWindowListener() );

        // pack(); // Minimalgroesse einnehmen
        setVisible(true);
    }
    //-------------------------------------------------------------------------
    public static void main(String[] args)
    {
        MyAwtWindow mw1 = new MyAwtWindow();
        mw1.setLocation(10, 10);
        MyAwtWindow mw2 = new MyAwtWindow();
        mw2.setLocation(100, 150);
    }
}
```

1.1.2 DecButtonListener

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class DecButtonListener implements ActionListener
{
    private MyAwtWindow mw_;
    public DecButtonListener(MyAwtWindow mw)
    {
        mw_ = mw;
    }
    public void actionPerformed(ActionEvent e)
    {
        int n = mw_.getAnzahl();
        mw_.setAnzahl(n - 1);
        String t = (n % 2 == 0) ? "Anzahl = " : "Number = ";
        mw_.setLabelText(t + n);
    }
}
```

1.1.3 IncButtonListener

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class IncButtonListener implements ActionListener
{
    private MyAwtWindow mw;
    public IncButtonListener(MyAwtWindow mw)
    {
        this.mw = mw;
    }
    public void actionPerformed(ActionEvent e)
    {
        int n = mw.getAnzahl();
        n++;
        mw.setAnzahl(n);
        String t = (n % 2 == 0) ? "Anzahl = " : "Number = ";
        mw.setLabelText(t + n);
    }
}
```

1.1.4 MyWindowListener

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;


public class MyWindowListener extends WindowAdapter
{
    //-------------------------------------------------------------------------
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
    //-------------------------------------------------------------------------
}
```

2 Swing

2.1 Swing GUI Demo

```
// Das Layout der AWT-Bibliothek muss importiert werden
import java.awt.GridLayout;
import javax.swing.*;
@SuppressWarnings("serial")
public class GuiDemoWindow extends JFrame
{
    public GuiDemoWindow()
    // Konstruktor: "View" (Ansicht) aufbauen und initialisieren.
    {
        super("Demo Swing GUI");
        // Operation, um das GUI zu schliessen
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        // Layout
        getContentPane().setLayout(new GridLayout(1, 4)); // 1 Reihe, 4 Spalten
        // 1. Spalte: Temperaturanzeige
        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(3, 1)); // 3 Reihen, 1 Spalte
        p1.add(new JLabel("Temperatur", JLabel.CENTER));
        p1.add(new JLabel("27", JLabel.CENTER));
        JPanel p13 = new JPanel();
        p13.setLayout( new GridLayout(1, 2) ); // 1 Z. , 3 2 Sp.
        JRadioButton rb1 = new JRadioButton("Cels.");
        p13.add(rb1);
        JRadioButton rb2 = new JRadioButton("Fahr.");
        p13.add(rb2);
        ButtonGroup bg = new ButtonGroup();
        rb1.setSelected(true);
        p1.add(p13);
        bg.add(rb1);
        bg.add(rb2);
        getContentPane().add(p1);

        // 2. Spalte: Zutatenwahl
        JPanel p2 = new JPanel();
        p2.setLayout( new GridLayout(4, 1) ); // 4 Reihen, 1 Spalte
        p2.add(new JLabel("Zutaten:", JLabel.LEFT));
        p2.add(new JCheckBox("Kapern"));
        p2.add(new JCheckBox("Zwiebeln"));
        p2.add(new JCheckBox("Knoblauch"));
        getContentPane().add(p2);

        // 3. Spalte: Kommentarfeld
        JTextArea jta = new JTextArea("Kommentar:", 4, 10);
        JScrollPane jsp = new JScrollPane(jta,
            JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
            JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        getContentPane().add(jsp);

        // 4. Spalte: Buttons
        JPanel p4 = new JPanel();
        p4.setLayout(new GridLayout(2, 1)); // 2 Reihen, 1 Spalte
        p4.add(new JButton("Abschicken"));
        p4.add(new JButton("Reset"));
        getContentPane().add(p4);

        pack();
        setVisible(true);
    }
    public static void main(String[] args)
    {
        GuiDemoWindow app = new GuiDemoWindow();
        app.setLocation(30, 30);
    }
}
```

2.2 Temperaturumrechnung

```
import java.awt.GridLayout;
import javax.swing.*;
import java.awt.event.*;

public class UmrechnungWindow extends JFrame implements ActionListener
{
    // Achtung: Diese Variablen muessen hier stehen, damit man auch
    // ausserhalb des Konstruktors darauf zugreifen kann!
    private JRadioButton rb2;
    private JRadioButton rb1;
    private JTextArea jta;
    private JTextArea jka;
    private JButton button;

    public UmrechnungWindow()
    // Konstruktor
    {
        super("Temperaturumrechnung");
        super.setDefaultCloseOperation(EXIT_ON_CLOSE);
        getContentPane().setLayout(new GridLayout(3, 2, 6, 5)); //  3 Zeilen, 2 Spalten mit horizontalem ↩
            Abstand von 6 Pixels und vertikalem Abstand von 5 Pixels
        // 1. Zeile
        JPanel p1 = new JPanel();
        p1.add(new JLabel("Temperaturumrechnung", JLabel.LEFT));
        getContentPane().add(p1);

        // 2. Zeile
        JPanel p2 = new JPanel();
        p2.setLayout(new GridLayout(1,3));

        JPanel p3 = new JPanel();
        p3.setLayout(new GridLayout(2,1));
        p3.add(new JLabel("Eingabe", JLabel.LEFT));
        p3.add(new JLabel("Ergebnis", JLabel.LEFT));
        p2.add(p3);

        JPanel p4 = new JPanel();
        p4.setLayout(new GridLayout(2,1));
        jta = new JTextArea("25");
        p4.add(jta);
        jka = new JTextArea("77");
        p4.add(jka);
        p2.add(p4);

        JPanel p5 = new JPanel();
        p5.setLayout(new GridLayout(2,1));
        JPanel p6 = new JPanel();
        p6.setLayout(new GridLayout(1,2));
        rb1 = new JRadioButton("C");
        p6.add(rb1);
        rb2 = new JRadioButton("F");
        p6.add(rb2);
        ButtonGroup bg = new ButtonGroup();
        bg.add(rb1);
        bg.add(rb2);
        p5.add(p6);
        button = new JButton("Rechne");
        p5.add(button);
        p2.add(p5);
        getContentPane().add(p2);

        button.addActionListener(this);
        pack();
        setVisible(true);

    }
    public static void main(String[] args)
    {
        UmrechnungWindow app = new UmrechnungWindow();
    }
    // Action Listener
    public void actionPerformed(ActionEvent e)
    {
        String text = jta.getText();
        double temp = Double.parseDouble(text);
        double result;
        if(rb1.isSelected())
        {   result = (temp*9.0/5.0) + 32;}
        else
        { result = (temp-32)*(5.0/9.0); }
        text = String.format("%.2f", result);

        jka.setText(text);
    }

}
```

2.3 Taschenrechner

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

@SuppressWarnings("serial")
public class Taschenrechner extends JFrame implements ActionListener
{
    // Lokale Variabeln deklarieren
    private JTextField txfAnzeige_;
    private JButton btnsNumber_[];   // Ziffern 0..9
    private JButton btnChangeSign_;  // Vorzeichen wechseln
    private JButton btnDecPoint_;    // Dezimalpunkt

    // Rechen-Operationen:
    private JButton btnAdd_, btnSub_, btnMul_, btnDiv_;
    private JButton btnResult_; // Resultat anzeigen
    private JButton btnClear_;  // Eingabe loeschen

    private double oldValue_;
    private char operator_ = '0';

    // Zustandsmaschine (Finite state machine - FSM) mit int-Werten:
    private int state_ = IDLE;
    private final static int IDLE = 0; // keine Eingabe
    private final static int ETXF = 1; // letzte Eingabe war im Textfeld
    private final static int EAOP = 2; // letzte Eingabe war arithm. Operator
    private final static int EEQU = 3; // letzte Eingabe war '='
    //-----------------------------------------------------------------

    // Elementklasse / Innere Klasse - Ein Objekt der Innere Klasse
    // gehoert zu einem Objekt der aeusseren Klasse
    // Elementklasse: Die Klassendefinition ist DIREKT, UNMITTELBAR
    // in einer anderen Klasse enthalten.
    private class ClearButtonListener implements ActionListener   // Element-Klasse
    {
        // Action-Listener fuer 'CLEAR'-Button (Eingabe loeschen).
        public void actionPerformed(ActionEvent e)
        {
            txfAnzeige_.setText("");
            operator_= '0'; // Wert in 'oldValue' ist ungueltig
            state_ = IDLE;
        }
    }
    // Taschenrechner-Konstruktor
    public Taschenrechner()
    {
        super("Einfacher Rechner");
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setSize(400, 300);  // Breite, Hoehe

        // *** GUI-Darstellung
        setLayout( new BorderLayout() );
        {   // Im "Norden - Das Eingabefeld:
            txfAnzeige_ = new JTextField (30);
            // Horizontal ausgerichtetes Eingabefeld
            txfAnzeige_.setHorizontalAlignment(JTextField.RIGHT);
            // Trick: Umrandung (Border) um txfAnzeige hinzufuegen:
            Border b = BorderFactory.createEmptyBorder(10, 10, 10, 10);
            txfAnzeige_.setBorder(b);
            // txfAnzeige im Norden zur ContentPane hinzufuegen:
            add(txfAnzeige_, BorderLayout.NORTH);
        }
        {   // Im Osten fuer arithmetische Operationen
            JPanel p = new JPanel();
            // GridLayout: 0 Zeilen (d.h. beliebig), 1 Spalte, mit je // 5 Px Abstand
            p.setLayout( new GridLayout(0, 1, 5, 5) );
            p.add( btnAdd_ = new JButton("+") );
            p.add( btnSub_ = new JButton("-") );
            p.add( btnMul_ = new JButton("*") );
            p.add( btnDiv_ = new JButton("/") );
            p.add( btnResult_ = new JButton("=") );
            // Trick: Umrandung (Border) um p hinzufuegen damit Abstand entsteht zu den anderen Elementen:
            Border b = BorderFactory.createEmptyBorder(20, 10, 20, 10);
            p.setBorder(b);
            // p im Osten zur ContentPane hinzufuegen:
            add(p, BorderLayout.EAST);
        }
        {   // Im "Center" Die Zahlen
            JPanel p = new JPanel();
            p.setLayout( new GridLayout(4, 3, 10, 10) ); // 4 Z., 3 Sp. je 10 Px Abstand
            // Array mit 10 Buttons mit den Zahlen
            btnsNumber_ = new JButton[10];
            // Buttons erzeugen:
            for (int i= 0; i < 10; i++)
            {
```

```
            btnsNumber_[i] = new JButton("" + i);
        }
        btnChangeSign_ = new JButton("CHS");  // Vorzeichen wechseln
        btnDecPoint_   = new JButton(".");    // Dezimalpunkt
        // Buttons zu p hinzufuegen:
        p.add(btnsNumber_[7]); p.add(btnsNumber_[8]); p.add(btnsNumber_[9]);
        p.add(btnsNumber_[4]); p.add(btnsNumber_[5]); p.add(btnsNumber_[6]);
        p.add(btnsNumber_[1]); p.add(btnsNumber_[2]); p.add(btnsNumber_[3]);
        p.add(btnsNumber_[0]); p.add(btnDecPoint_);   p.add(btnChangeSign_);
        // Trick: Umrandung (Border) um p hinzufuegen damit Abstand entsteht:
        Border b = BorderFactory.createEmptyBorder(20, 10, 20, 10);
        p.setBorder(b);
        // p im Center zur ContentPane hinzufuegen:
        add(p, BorderLayout.CENTER);
    }
    {   // Im "Sueden":
        btnClear_ = new JButton ("CLEAR"); // Eingabe loeschen
        add (btnClear_, BorderLayout.SOUTH);
    }
    //-------------------------------------------------------
    // *** Event-Handling initalisieren:
    // Event-Handler bei Ziffern-Buttons anmelden:
    for (int i= 0; i < 10; i++)
    {
        btnsNumber_[i].addActionListener(this);
    }
    // Die Aktionen des ClearButton werden an die Innere Klasse uebergeben
    btnClear_.addActionListener(new ClearButtonListener() );
    //
   // Lokale Klasse / Innere Klasse: wird innerhalb
   // des Konstruktors definiert
    // ChangeSign ButtonListener fuer Vorzeichenfehler
    class ChangeSignButtonListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            String t = txfAnzeige_.getText();
            if (t.indexOf('-') < 0)
                txfAnzeige_.setText('-' + t);
            else
                txfAnzeige_.setText( t.substring(1) );
            state_ = ETXF; // Eingabe fuer Textfeld
        }
    }
    btnChangeSign_.addActionListener(new ChangeSignButtonListener() );

    // Innere Klasse: Lokale Klasse DecPoint ButtonListener fuer Dezimalpunkt

    class DecPointButtonListener implements ActionListener
    {   // lokale Klasse
        public void actionPerformed(ActionEvent e)
        {   // Action-Listener fuer Decimalpunkt.
            String t = txfAnzeige_.getText();
            if (state_ == ETXF)
            {   // Vorherige Eingabe war in Textfeld.
                // Punkt anhaengen falls noch nicht vorhanden:
                if (t.indexOf('.') < 0) txfAnzeige_.setText(t + '.');
            }
            else
            {
                txfAnzeige_.setText("0.");
            }
            state_ = ETXF; // Eingabe war in Textfeld
        }
    }
    btnDecPoint_.addActionListener( new DecPointButtonListener() );


    class ResultButtonListener  implements ActionListener
    {   // Lokale Klasse - Innere Klasse: Die Klassendefinition ist in einem CODE Block { } enthalten.
        // Haeufig in einer Memberfunktion
        public void actionPerformed(ActionEvent e)
        {   // Action-Listener fuer '='-Button (Resultat anzeigen).
            if (operator_ == '0') return; // Operator fehlt
            if (state_ != ETXF) return;   // keine Eingabe in Textfeld
            evaluate('=');
            state_ = EEQU; // Eingabe war "= Button"
        }
    }
    btnResult_.addActionListener( new ResultButtonListener() );
    ActionListener alAdd = new ActionListener()

    // Anonyme Klasse: Die Klassendefinition ist ohne Klassenname in einem Ausdruck enthalten.
    // Dadurch wird gleichzeitig ein Objekt davon erzeugt.
    {
        public void actionPerformed(ActionEvent e)
        {   // Action-Listener fuer '+' (Addition).
            if (state_ == IDLE) return;
```

```
                if (state_ == ETXF)
                {   // vorherige Eingabe war in Textfeld.
                    evaluate('+');  // Ausdruck auswerten
                }
                else operator_='+';  // Operator sich merken
                state_ = EAOP; // Eingabe war arithm. Operator
            }
        };
        btnAdd_.addActionListener( alAdd );
        btnSub_.addActionListener( new ActionListener()
        //Anonyme Klasse: Funktion innerhalb von { } ohne Klassendefinition
        {   public void actionPerformed(ActionEvent e)
            {   // Action-Listener fuer '-' (Subtraktion).
                if (state_ == IDLE) return;
                if (state_ == ETXF)
                {   // vorherige Eingabe war in Textfeld.
                    evaluate('-');  // Ausdruck auswerten
                }
                else operator_='-';  // Operator sich merken
                state_ = EAOP; // Eingabe war arithm. Operator
            }
        });
        //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        btnMul_.addActionListener( new ActionListener()
        //Anonyme Klasse: Funktion innerhalb von { } ohne Klassendefinition
        {   // anonyme Klasse
            public void actionPerformed(ActionEvent e)
            {   // Action-Listener fuer '*' (Multiplikation).
                if (state_ == IDLE) return;
                if (state_ == ETXF)
                {   // vorherige Eingabe war in Textfeld.
                    evaluate('*');  // Ausdruck auswerten
                }
                else operator_='*';  // Operator sich merken
                state_ = EAOP; // Eingabe war arithm. Operator
            }
        });
        //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        btnDiv_.addActionListener( new ActionListener()
        //Anonyme Klasse: Funktion innerhalb von { } ohne Klassendefinition
        {
            public void actionPerformed(ActionEvent e)
            {   // Action-Listener fuer '/' (Division).
                if (state_ == IDLE) return;
                if (state_ == ETXF)
                {   // vorherige Eingabe war in Textfeld.
                    evaluate('/');  // Ausdruck auswerten
                }
                else operator_='/';  // Operator sich merken
                state_ = EAOP; // Eingabe war arithm. Operator
            }
        });
        // *** Epilog:
        this.setVisible(true);
    }
    // Action-Listener fuer die Ziffern-Buttons.
    public void actionPerformed(ActionEvent e)
    {
        JButton btn = (JButton)e.getSource();
        String s = btn.getText();
        // char ch = btn.getText().charAt(0); // 1. Zeichen
//      // ch muss Ziffer '0'..'9' sein.
//      int n = ch - '0';
        String t1 = (state_ != ETXF) ? "" : txfAnzeige_.getText();
        txfAnzeige_.setText(t1 + s);
        state_ = ETXF; // Eingabe war in Textfeld
    }
    //-------------------------------------------------------------------------
    private void evaluate(char op)
    {
    //-------------------------------------------------------------------------
        Nicht relevant
    //-------------------------------------------------------------------------
    }
    //=========================================================================
    public static void main(String[] args)
    {
        Taschenrechner mainWindow = new Taschenrechner();
        mainWindow.setLocation(20,  20);
    }
    //-------------------------------------------------------------------------
}
```

2.4 Lambda Rechner

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

import javax.swing.*;
import javax.swing.border.*;

@SuppressWarnings("serial")
public class TaschenrechnerLambda extends JFrame implements ActionListener
{
    //-------------------------------------------------------------------------
    private JTextField txfAnzeige_;
    private JButton btnsNumber_[];   // Ziffern 0..9
    private JButton btnChangeSign_;  // Vorzeichen wechseln
    private JButton btnDecPoint_;    // Dezimalpunkt
    // Rechen-Operationen:
    private JButton btnAdd_, btnSub_, btnMul_, btnDiv_;
    private JButton btnResult_; // Resultat anzeigen
    private JButton btnClear_;  // Eingabe loeschen

    private double oldValue_;
    private char operator_ = '0';

    // Zustandsmaschine (Finite state machine - FSM) mit int-Werten:
    private int state_ = IDLE;
    private final static int IDLE = 0; // keine Eingabe
    private final static int ETXF = 1; // letzte Eingabe war im Textfeld
    private final static int EAOP = 2; // letzte Eingabe war arithm. Operator
    private final static int EEQU = 3; // letzte Eingabe war '='
    //-------------------------------------------------------------------------
    public TaschenrechnerLambda()
    {
        // *** Prolog:
        super("Einfacher Rechner");
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setSize(400, 300);  // Breite, Hoehe

        //---------------------------------------------------
        // GUI - Teil wie bei Taschenrechner
        //---------------------------------------------------

        // *** Event-Handling initalisieren:

        // Event-Handler bei Ziffern-Buttons anmelden:
        for (int i= 0; i < 10; i++)
        {
            btnsNumber_[i].addActionListener(this);
        }
        btnClear_.addActionListener
        (   // Lambda-Ausdruck:
            event ->
            {   // Action-Listener fuer 'CLEAR'-Button (Eingabe loeschen).
                txfAnzeige_.setText("");
                operator_= '0'; // Wert in 'oldValue' ist ungueltig
                state_ = IDLE;
            }
        );
        btnChangeSign_.addActionListener
        (   // Lambda-Ausdruck:
            event ->
            {   // Action-Listener fuer 'CHS' (Vorzeichen wechseln).
                String t = txfAnzeige_.getText();
                if (t.indexOf('-') < 0)
                    txfAnzeige_.setText('-' + t);
                else
                    txfAnzeige_.setText( t.substring(1) );
                state_ = ETXF; // Eingabe fuer Textfeld
            }
        );
        btnDecPoint_.addActionListener
        (   // Lambda-Ausdruck:
            event ->
            {   // Action-Listener fuer Decimalpunkt.
                String t = txfAnzeige_.getText();
                if (state_ == ETXF)
                {   // Vorherige Eingabe war in Textfeld.
                    // Punkt anhaengen falls noch nicht vorhanden:
                    if (t.indexOf('.') < 0) txfAnzeige_.setText(t + '.');
                }
                else
                {
                    txfAnzeige_.setText("0.");
                }
                state_ = ETXF; // Eingabe war in Textfeld
            }
        );
        btnResult_.addActionListener
```

```
        (   // Lambda-Ausdruck:
            event ->
            {   // Action-Listener fuer '='-Button (Resultat anzeigen).
                if (operator_ == '0') return; // Operator fehlt
                if (state_ != ETXF) return;   // keine Eingabe in Textfeld
                evaluate('=');
                state_ = EEQU; // Eingabe war "= Button"
            }
        );
        btnAdd_.addActionListener
        (   // Lambda-Ausdruck:
            event ->
            {   // Action-Listener fuer '+' (Addition).
                if (state_ == IDLE) return;
                if (state_ == ETXF)
                {   // vorherige Eingabe war in Textfeld.
                    evaluate('+');  // Ausdruck auswerten
                }
                else operator_='+';  // Operator sich merken
                state_ = EAOP; // Eingabe war arithm. Operator
            }
        );
        btnSub_.addActionListener
        (   // Lambda-Ausdruck:
            event ->
            {   // Action-Listener fuer '-' (Subtraktion).
                if (state_ == IDLE) return;
                if (state_ == ETXF)
                {   // vorherige Eingabe war in Textfeld.
                    evaluate('-');  // Ausdruck auswerten
                }
                else operator_='-';  // Operator sich merken
                state_ = EAOP; // Eingabe war arithm. Operator
            }
        );
        btnMul_.addActionListener
        (   // Lambda-Ausdruck:
            event ->
            {   // Action-Listener fuer '*' (Multiplikation).
                if (state_ == IDLE) return;
                if (state_ == ETXF)
                {   // vorherige Eingabe war in Textfeld.
                    evaluate('*');  // Ausdruck auswerten
                }
                else operator_='*';  // Operator sich merken
                state_ = EAOP; // Eingabe war arithm. Operator
            }
        );
        btnDiv_.addActionListener
        (   // Lambda-Ausdruck:
            event ->
            {   // Action-Listener fuer '/' (Division).
                if (state_ == IDLE) return;
                if (state_ == ETXF)
                {   // vorherige Eingabe war in Textfeld.
                    evaluate('/');  // Ausdruck auswerten
                }
                else operator_='/';  // Operator sich merken
                state_ = EAOP; // Eingabe war arithm. Operator
            }
        );
        this.setVisible(true);
    }
    //-----------------------------------------------------------------------
    public void actionPerformed(ActionEvent e)
    // Action-Listener fuer die Ziffern-Buttons.
    {
        JButton btn = (JButton)e.getSource();
        String s = btn.getText();
        // char ch = btn.getText().charAt(0); // 1. Zeichen
//      // ch muss Ziffer '0'..'9' sein.
//      int n = ch - '0';
        String t1 = (state_ != ETXF) ? "" : txfAnzeige_.getText();
        txfAnzeige_.setText(t1 + s);
        state_ = ETXF; // Eingabe war in Textfeld
    }
    //-----------------------------------------------------------------------
    private void evaluate(char op) {
    //-------------------------------------------------------
    // Funktion Evaluate - wie bei Taschenrechner
    //-------------------------------------------------------
    }
    //=======================================================================
    public static void main(String[] args)
    {
        TaschenrechnerLambda mainWindow = new TaschenrechnerLambda();
        mainWindow.setLocation(20,  20);
    }}
```

2.5 IncDec mit Grafik Button

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

import javax.swing.*;
//=============================================================================
@SuppressWarnings("serial")
public class IncDecWindow extends JFrame
{
    //-------------------------------------------------------------------------
    // Attribute:
    private int anzahl_ = 0;
    private JLabel lblTop_;
    private JLabel lblBottom_;
    private JButton incButton_;
    private JButton decButton_;
    //-------------------------------------------------------------------------
    public IncDecWindow()
    // Konstruktor
    {
        // **** Prolog:
        super ("Inc - Dec mit grafischen Buttons");
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        //---------------------------------------------------------------------
        // **** Ansicht (Darstellung) aufbauen:
        lblTop_ = new JLabel("Anzahl = " + anzahl_, JLabel.CENTER);
        lblTop_.setBackground(Color.GREEN); lblTop_.setOpaque(true);
        lblBottom_ = new JLabel("Number  = " + anzahl_, JLabel.CENTER);
        lblBottom_.setBackground(Color.GREEN); lblBottom_.setOpaque(true);
        incButton_ = new GrafikButton("Inc", "Inkrement");
        decButton_ = new GrafikButton("Dec", "Dekrement");

        setLayout( new BorderLayout() );

        add(lblTop_, BorderLayout.NORTH);
        add(lblBottom_, BorderLayout.SOUTH);
        JPanel p = new JPanel();
        p.setLayout( new GridLayout(1, 2) );
        add(p, BorderLayout.CENTER);
        p.add(incButton_);
        p.add(decButton_);

        //---------------------------------------------------------------------
        // **** Event-Handling initialisieren:
        incButton_.addActionListener( new ActionListener()
        {   // Action-Listener fuer Inc-Button (anonyme Klasse)
            public void actionPerformed(ActionEvent e)
            {
                anzahl_++;
                lblTop_.setText("Anzahl = " + anzahl_);
                lblBottom_.setText("Number = " + anzahl_);
            }
        });
        //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        decButton_.addActionListener( new ActionListener()
        {   // Action-Listener fuer Dec-Button (anonyme Klassen)
            public void actionPerformed(ActionEvent e)
            {
                anzahl_--;
                lblTop_.setText("Anzahl = " + anzahl_);
                lblBottom_.setText("Number = " + anzahl_);
            }
        });
        //---------------------------------------------------------------------

        // **** Epilog:
        pack(); // Minimalgroesse einnehmen
        setVisible(true);
    }
    //-------------------------------------------------------------------------
    public static void main(String[] args)
    {
        IncDecWindow mw1 = new IncDecWindow();
        mw1.setLocation(10,  10);
//      IncDecWindow mw2 = new IncDecWindow();
//      mw2.setLocation(100, 150);
    }
    //-------------------------------------------------------------------------
}
//=============================================================================
```

```java
import java.awt.*;
import javax.swing.JButton;

@SuppressWarnings("serial")
public class GrafikButton extends JButton {

    private String textKlein_;
    private String textMitte_;

    public GrafikButton(String textKlein, String textMitte)
    {
        setPreferredSize(new Dimension(200, 150)); // Breite, Hoehe
        this.setBackground(Color.YELLOW);
        this.textKlein_ = textKlein;
        this.textMitte_ = textMitte;
    }

    public void paintComponent (Graphics g)
    {
        super.paintComponent(g);
        int w= this.getWidth(); int h= this.getHeight();
        g.setColor(Color.blue);
        g.fillOval(10, 10, w-20, h-20 ); // linke, obere Ecke, Breite, Hoehe
        g.setColor(Color.red);
        g.drawString(textKlein_, 20, 20); // linke, untere Ecke von 'textKlein'
        Font font = new Font ("Serif", Font.ITALIC, h/5);
        g.setFont(font);
        FontMetrics fm = g.getFontMetrics();
        int tw= fm.stringWidth(textMitte_);
        g.drawString( textMitte_, (w-tw)/2, h/2);
    }
}
```

3 Theorie

3.1 String & StringBuffer

Ein Objekt von String ist unveränderbar (immutable), d.h. es kann nach seiner Erstellung nicht mehr angepasst werden. Das macht es einfach um verschiedene Referenzen darauf zeigen zu lassen. Man muss sich keine Sorgen machen, dass eine Referenz das Original verändert. String bietet eigene Funktionen wie length, charAt, equals, equalsIgnoreCase, compareTo. Merke: 'String + Irgendetwas (int, double etc)' ergibt einen neuen String und 'Irgendetwas + String' ergibt ebenfalls einen String.
Ein Objekt von StringBuffer kann mit den Funktionen, welche die Klasse StringBuffer bereitstellt einfach verändert werden (in der Länge und bezüglich des Inhalts). Die Buffer-Capacity bezeichnet die mögliche Buffergrösse, welche ohne Verlängerung möglich wäre. Die Buffer-Length gibt hingegen die aktuelle Anzahl enthaltener Zeichen an.

```
// Beispiel 1: Erstellen eines neuen Strings
String t = "Hallo ";
final int N = 10;
StringBuffer buf= new StringBuffer( N * t.length() );
for (int i= 0; i < N; i++)
{   buf.append(t); }
String s = buf.toString();

// Beispiel 2: Schlechte Nutzung von String
String s = new String();
String t = new String( "Hallo " );  // unsinnig: besser Sting t = "Hallo"
for (int i= 0; i < 10; i++)
{   s = s + t;  }   // Ineffizient, da String unveraenderbar ist, muss bei jedem Durchgang ein neues String-↩
    Objekt erzeugt werden.
```

```
public class StringPerformance {

    public static String appendTestString(String s, int n)
    // Append Test with String
    {
        long startTime = System.currentTimeMillis();
        String tmp = "";
        for (int i= 0; i < n; i++)
        {   tmp = tmp + s;}
        long endTime = System.currentTimeMillis();
        System.out.printf("mit 'String'        = %1$5d ms.\n",
                endTime - startTime);
        return tmp;
    }
    public static String appendTestStringBuffer(String s, int n)
    // Append Test with StringBuffer:
    {
        long startTime = System.currentTimeMillis();
        StringBuffer tmp = new StringBuffer (s.length() * n);
        for (int i= 0; i < n; i++)
        {   tmp.append(s);}
        String res = tmp.toString();
        long endTime = System.currentTimeMillis();
        System.out.printf("mit 'StringBuffer'  = %1$5d ms.\n",
                endTime - startTime);
        return res;
    }
    public static String appendTestStringBuilder(String s, int n)
    // Append Test with StringBuilder
    {
        long startTime = System.currentTimeMillis();
        StringBuilder tmp = new StringBuilder (s.length() * n);
        for (int i= 0; i < n; i++)
        {   tmp.append(s);}
        String res = tmp.toString();
        long endTime = System.currentTimeMillis();
        System.out.printf("mit 'StringBuilder' = %1$5d ms.\n",
                endTime - startTime);
        return res;
    }
    public static void main(String[] args)
    {
        appendTestString        ("HalloHallo",   10000);
        appendTestString        ("HalloHallo",   10000);
        appendTestStringBuffer  ("HalloHallo",  100000);
        appendTestStringBuffer  ("HalloHallo",  100000);
        appendTestStringBuilder ("HalloHallo", 1000000);
        appendTestStringBuilder ("HalloHallo", 1000000);
        appendTestStringBuffer  ("HalloHallo",  100000);
        appendTestStringBuffer  ("HalloHallo",  100000);
        appendTestStringBuilder ("HalloHallo", 1000000);
        appendTestStringBuilder ("HalloHallo", 1000000);
        appendTestString        ("HalloHallo",   10000);
        appendTestString        ("HalloHallo",   10000);
    }}
```

3.2 Ein- und Ausgabe

Standard-Datenströme (engl. ßtandard streams") sind vordefinierte Kommunikations-Kanäle zwischen einem Computerprogramm und seiner Umgebung (Tastatur, Bildschirm). Die drei Standard-Datenströme **stdin (Standardeingabe), stdout (Standardausgabe) und stderr (Standardfehlerausgabe)** werden seit den 1970er Jahren praktisch von allen Betriebssystemen und Programmiersprachen unterstützt (Fortran bereits 1950) und für die Ein-/Ausgabe verwendet. Sie werden in der Regel bei allen Nicht-Gui-Programmen verwendet. Also bei Programmen, die in einem Terminalfenster ausgeführt werden und Eingabedaten von der Tastatur beziehen und Ausgabedaten in das Terminalfenster (Bildschirm) schreiben.Die drei Standarddatenströme können umgelenkt werden – das heisst, Eingabedaten aus einer Datei beziehen (statt von der Tastatur) oder Ausgabedaten in eine Datei schreiben (statt ins Terminalfenster). Das Programm merkt normalerweise nichts von einer solchen Umlenkung.
In Java werden die drei Standard-Datenströme durch folgende drei (statische) Objekte der Klasse System repräsentiert:

- System.out repräsentiert die Standardausgabe – stdout .
  Entspricht dem Terminalfenster bzw. der Console in Eclipse (wenn nicht umgelenkt). .println() wird darauf angewendet. Das heisst, dorthin (auf System.out) wird der Text "Hallo Welt"geschrieben. println() steht für "print line"das heisst, die Ausgabe wird automatisch durch einen Zeilenvorschub abgeschlossen. Möchte man keinen Zeilenvorschub am Ende, kann print() statt println() verwendet werden.
  Seit Java 5 gibt es zusätzlich zu println() und print() die Methode printf(), welche analog der bekannten C-Funktion fprintf() mit einem Formatstring arbeitet der Platzhalter beinhalten kann.s

- System.err repräsentiert die Standardfehlerausgabe – stderr.
  Entspricht dem Terminalfenster bzw. der Console in Eclipse (wenn nicht umgelenkt).

- System.in repräsentiert die Standardeingabe – stdin.
  Sie entspricht der Tastatur (wenn nicht umgelenkt). Dabei werden die via Tastatur eingegebenen Zeichen vom Betriebssystem zwischengespeichert. Erst nach der Eingabe von ¡Ret¿ (Betätigen der Return-Taste) wird das wartende Java-Programm äufgewecktünd kann nun die Daten aus dem Zwischenspeicher lesen und verarbeiten.

```
import java.io.*;
public class TheOldWay
{
public static void main(String[] args) throws Exception
{
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("Bitte Name eingeben: ");
// Einlesen in String:
    String name = br.readLine();
// Ausgeben:
    System.out.println("Hallo " + name);}
}
```

```
import java.io.*;

public class TextEinAusgabe
{
   public static void main(String[] args) throws Exception
   {
      BufferedReader cin;  // Console Input
      cin = new BufferedReader (new InputStreamReader (System.in));
      System.out.print("Bitte Text eingeben: ");
      // Einlesen in String:
      String textZeile = cin.readLine();
      // In Grossbuchstaben wandeln:
      textZeile = textZeile.toUpperCase();
      // Ausgeben:
      System.out.println("Zeile gewandelt= " + textZeile);
      System.out.println("Anzahl Zeichen = " + textZeile.length());
   }
}
```

```
import java.util.Scanner;
public class TheNewWay
{
public static void main(String[] args)
{
    Scanner sin = new Scanner(System.in);
    System.out.print("Bitte Name eingeben: ");
// Einlesen in String:
    String name = sin.next();
// Ausgeben:
    System.out.println("Hallo " + name);}
}
```

```
import java.util.Scanner; // neu ab Java 5

public class DemoScanner {
    public static void main(String[] args)
    {
        Scanner sin = new Scanner(System.in);

        System.out.print("Bitte eine ganze Zahl eingeben: ");
        while (! sin.hasNextInt() )
        {   // Das Eingegebene kann nicht als 'int' interpretiert werden.
            String s = sin.nextLine();  // eingebene Zeile einlesen
            System.out.println("Fehler: '" + s  + "' ist keine ganze Zahl!");
            System.out.print("Eine ganze Zahl bitte:  ");
        }
        // Eingegebene Ziffern einlesen und in 'int' umwandeln:
        int x = sin.nextInt();
        // Rest der eingegebenen Zeile einlesen bzw. konsumieren:
        String s2 = sin.nextLine();

        System.out.println("x= " + x);
        System.out.println("Anzahl nachfolgende Zeichen = " + s2.length() );
}}
```

3.3 Packages & Import

Packages sind Sammlungen von zusammengehörigen Klassen und bilden ein zusätzliches Strukturierungsmittel für Klassen. Pakete werden auf Datei-Verzeichnisse abgebildet, Klassen auf .java-Dateien. Eine **Compilation Unit** entspricht einer .java-Datei. Sie kann mehrere Klassendefinitionen enthalten, aber **nur eine public Klasse** (Klassenname / Dateiname). Ein Paket definiert den Namensraum und legt die Sichtbarkeit fest. Es kann aus mehreren Compilation Units bestehen. Pakete können zur Kapselung verwendet werden, da (nicht-public) Klassennamen nur im gleichen Paket sichtbar sind. Compilation Units (.java-Dateien) können nicht zur Kapselung verwendet werden, da sie keine Sichtbarkeitsgrenze aufweisen.
Regeln:
- Package-Anweisung muss erste Anweisung sein.
- Es ist höchstens eine Package-Anweisung pro Datei zulässig.
- Die .java-Datei muss sich in einer Verzeichnisstruktur befinden, welche dem Package-Namen entspricht. - Wenn die Package-Anweisung fehlt, gehören die Klassen zum sogenannten Default-Package.
- Merke: Jede Klasse gehört zu einem Paket

Wichtige Packages:

- java.lang System, String, Integer, Character, Object, Math, ...
- java.io File, InputStream, OutputStream, Reader, Writer, ...
- java.util Vector, Stack, ArrayList, BitSet, HashTable, Random, ...
- java.awt Button, CheckBox, Frame, Color, ...

Eine import-Anweisung dient dazu die Namen von Klassen aus einem Package zu importieren. Danach können die Klassennamen direkt, d.h. ohne Package-Namen verwendet werden.

```
\\ Beispiel Import-Anweisung
import java.util.Vector;
... Klassenbeschreibung, etc.
Vector obj = new Vector ();  \\ ohne Quantifizierung
java.util.Date d = new java.util.Date(); \\ mit Quantifizierung
```

3.4 Klassen

Bei Java gibt es keine Trennung zwischen Deklaration und Definition (Header, C-File). Die Implementierung ist immer innerhalb einer Klasse. Zudem gibt es keine Initialisierungslisten. Die Übergabe von Parametern erfolgt immer by value. Bei Referenzen erfolgt eine ßhallow copyünd keine "deep Copy". Alle Attribute haben einen definierten Startwert (falls keiner gegeben - Initialisierung mit null oder 0.00).
Ein Objekt ist eine Variable deren Datentyp durch eine Klasse (Referenzdatentypen) definiert ist.

- abstract: Von dieser Klasse kann kein Objekt instanziert werden. Nur diese Klassen können abstrakte Methoden enthalten. Dabei handelt es sich um Methoden, welche keine Implementierung besitzen.
- final: Von dieser Klasse kann nicht weiter abgeleitet werden.
- public: Klasse ist öffentlich und überall verwendbar

- ohne public: Klasse ist nur im gleichen Paket verwendbar
- protect: Methoden und Attribute sind von Unterklassen und anderen Klassen im gleichen Package verwendbar
- finalize(): hat in Java praktisch keine Bedeutung und sollte vermieden werden. Sie entsprechen **nicht** den Destruktoren von C++.
- static:
  - Statische Klassenmethoden: Ihr Aufruf erfolgt mit dem Klassennamen. Sie ersetzen die globalen, freien Funktionen, aus C++ welches es in Java nicht gibt (z.B. java.lang.Math).
  Bsp: double z = Math.sqrt(x);
  - Statische Klassenattribute: Zugriff erfolgt über den Klassennamen. Sie existieren genau ein Mal in der Klasse und werden oft zur Definition von Konstanten verwendet.
  Bsp: final static int SIZE = 100;
  - Statischer Block: Dieser Teil wird ausgeführt, wenn die Klasse geladen wird (static initializer).
  Bsp.: class Ufo { static { */ Initialisierung */ }}
  - import static java.xx.bbb.*/yy : Importiert alle (oder nur eine) statische Methoden und Attributen der Klasse bbb. Nun können alle statischen Methoden und Attribute direkt - also ohne Klassenname verwendet werden.
  Bsp.: import static java.lang.Math.*;
  double z = PI * sqrt(sin(x) + cos(y));
- synchronized: Methode kann nicht unterbrochen werden
- final: Bezeichnen Konstanten und werden damit "read only"

3.4.1 Vergleiche

Vergleiche mit == und != beziehen sich auf die Referenzen also die Adressen und nicht auf den Inhalt des Referenzobjekts.
Person p1= new Person("Meier");
Person p2= new Person("Meier");
if (p1 == p2) System.out.println("gleich");
else System.out.println(üngleich");
Ausgabe: ungleich, denn die Adresse von p1 und p2 sind nicht dieselben.
Vergleiche mit .equals() erfolgt ein inhaltlicher Vergleich. if(p1.equals(p2)) wäre besser. Für eigene Klassen muss die Methode equals() überschrieben werden.
Die Methode int hashCode() aus der java.lang.Object-Bibilothek kann ebenfalls für Vergleich verwendet werden. Dabei wird ein int-Wert aus den Attributen des Objektes berechnet. Damit kann effizient bestimmt werden, ob 2 Objekte sicher verschieden sind, denn dann sind die Hash-Codes ungleich. Es kann jedoch nicht mit Sicherheit bestimmt werden, ob die Objekte gleich sind. Das muss dann noch mit equals geprüft werden. Achtung: Immer wenn equals überschrieben wird, ist auch die Funktion hashCode() zu überschreiben!

3.5 Vererbung

In Java gibt es keine Mehrfachvererbung bei Klassen. Es kann nur von einer Klasse geerbt (A extends B) werden, aber eine Klasse kann mehrere Interfaces (A implements C, D, E) haben. Die abgeleitete Klasse erweitert die Basisklasse. Es wird zuerst der Konstruktor der Basisklasse aufgerufen (falls keiner vorhanden wird eine Default-Konstruktor erzeugt) und dann jener der abgeleiteten Klasse. Dem Konstruktor der Basisklasse kann mit super (xxx) auch ein Parameter übergeben werden. Die abgeleitete Klasse erbt die Methoden der Basisklasse, kann diese überschreiben und auch neue Methoden hinzufügen. Konstruktoren haben keine Initialisierungslisten.

- this: ist vom Typ des aktuellen Objektes und kann verwendet werden, wenn der übergebene Parameter gleich heisst, wie das Attribut der Klasse.
  Bsp.: public class Person { // private string name; // public Person(String name) // { this. name = name } } //
- super: ist vom Typ der unmittelbar darüberliegenden Oberklasse
  super.super ist illegal

Eine Referenz von einer Oberklasse darf auf eine Objekt einer davon abgeleiteten Klasse zeigen. Beim Aufruf der Methode wird in Java gemäss dem "late Binding"die Klasse der referenzierten Klasse verwendet.
Bsp.: Person person = new Student("Meier", 1298);
pers. print(); // Das print() der Klasse Student wird aufgerufen.

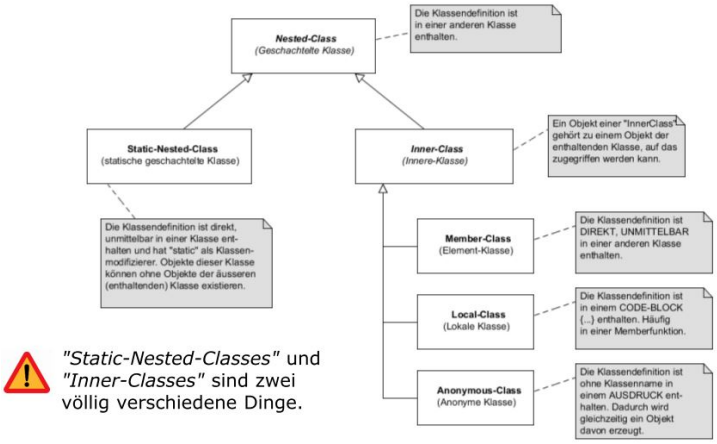
Up- und Downcasting: Ein Objekt einer abgeleiteten Klasse Sparkonto kann an ein Objekt der Oberklasse Konto übergeben werden. Ob eine Oberklasse an ein Objekt der abgeleiteten Klasse übergeben bzw. dazu umgewandelt werden kann, muss mit **instanceof** zuerst geprüft werden.

```
if (konto instanceof Sparkonto)
{ sparkonto = (Sparkonto) konto }
```

Instance Initializer: Code-Block, der nach dem Konstruktor der Oberklasse und vor dem Konstruktor der Klasse bei der Erzeugung eines Objekts aufgerufen wird. Sie werden bei anonymen Klassen verwendet, welche keine Konstruktoren haben.

```
{
numbers = new int[100];
for (int i = 0; i < 100; i++)
numbers[i] = i;
}
```

3.5.1 Klassenverschachtelung



Die **Element-Klasse** wird innerhalb der äusseren Klasse aber nicht innerhalb des Konstruktors definiert.
Die **lokale Klasse** wird innerhalb des Konstruktors definiert.
Anonyme Klasse wird innerhalb eines Funktionsaufrufs innerhalb des Konstruktors definiert.

3.6 Interfaces

Ein Java-Interface ist eine Menge von Methodendeklarationen ohne Implementierung (**abstract**) und konstanten Werten. Dabei sind alle Methoden automatisch public und abstract und alle Werte public und static final.

```
interface Collectable
{  int MAXELEMENTS = 1000;
    void add (Object o);
    void delete (Object o);
    Object find (Object o); }
```

Eine Klasse kann ein Interface implementieren (**implements**) und erbt dadurch alle Methoden. Da diese jedoch abstract sind, muss sie **alle, ausser die Default** Methoden überschreiben und implementieren. Referenzen von Interfaces können wie abstrakte Klassen verwendet werden. Eine Klasse kann mehrere Interfaces implementieren.
Interfaces können von anderen Interfaces erben mit **extends** und das ganze wird Schnittstellenvererbung genannt.

Änderungen bei Interfaces ab Java 8
- Default-Methoden bei Interfaces dienen zur Erweiterung von bestehenden Interfaces, ohne dass die Klassen, welche bereits diese Interfaces implementieren geändert werden müssen. Die default-Implementierung wird dann verwendet, wenn eine Klasse die entsprechende Methode des Interfaces nicht selbst implementiert.
- Statische Interface Methoden zusätzliche Methoden innerhalb eines Interfaces, welche aber in den Klassen die diese Interfaces implementieren, nicht überschrieben werden können.
- Funktionale Interfaces (nur neuer Begriff) sind Interfaces welche genau eine abstrakte Methode enthalten. Sie sind wichtig im Zusammenhang mit sogenannten Lambda-Ausdrücken. Das Package java.util.functions enthält 40 funktionale Interfaces. In der Praxis werden neu dort Lambda-Ausdrücke eingesetzt, wo früher i.a.R. anonyme Klassen verwendet wurden.

```
public interface MyData
{  // Default-Interface-Methode
    default void print(String str)
    {   if (!isNull(str))
        System.out.println("MyData Print::" + str); }
   // Statische Interface Methode
    static boolean isNull(String str)
    {   System.out.println("Interface Null Check");
        if ( str == null ) return true;
        if ( str.equals("") ) return true;
        return false;   } }
```

3.7 Lambdas

Lambda-Ausdrücke ermögliche eine besonders kompakte Implementierung von funktionalen Schnittstellen (Interfaces), d.h. Interfaces mit genau einer abstrakten Methode. Dies werden häufig anstelle von anonymen Klassen verwenden. Sie erlauben ebenfalls einen direkten Zugriff auf die Variablen der Umgebung (!).

```
// Anonyme Klasse (1) vs. Lamda-Ausdruck (2)
(1) incButton.addActionListener
(    new ActionListener()
{   public void actionPerformed(ActionEvent event) {
     label.setText("Anz = " + ++anzahl); } } );
(2) decButton.addActionListener (
     event -> label.setText("Anz = " + --anzahl) );
```

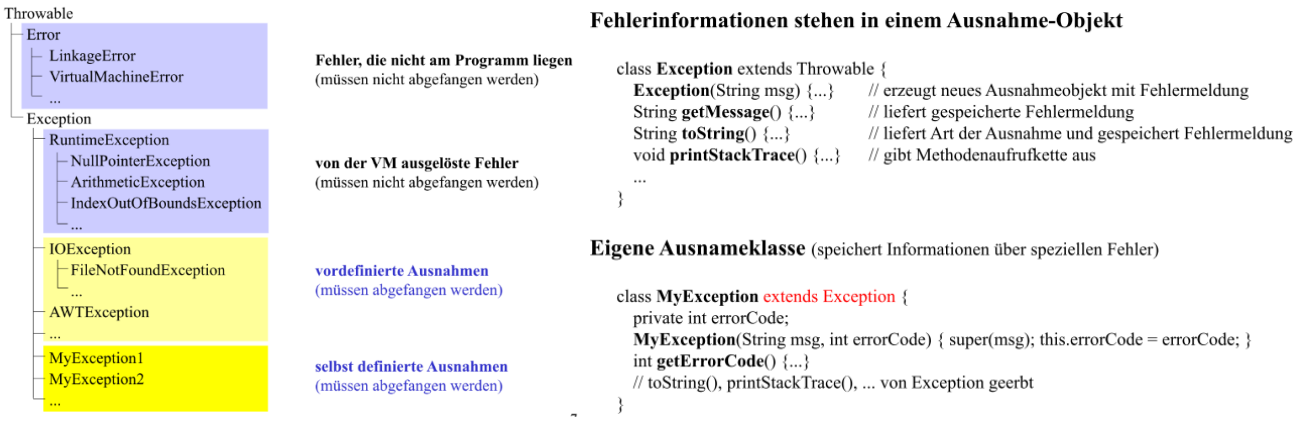
Sowohl mit einem Lambda-Ausdruck als auch einer anonymen Klasse kann auf kann auf die Umgebung zugegriffen werden. Der Code mit dem Lambda-Ausdruck ist jedoch kompakter und übersichtlicher. Man muss da aber das zugrunde liegende Interface kennen. Ein Lambda-Ausdruck ist daran erkennbar, dass er einen Pfeil-Operator enthält. Innerhalb eines Lambda-Ausdrucks kann .super und .this verwendet werden.

Ein Lambda-Ausdruck besteht aus::
- Interface-Namen (optional): Name des zugrunde liegenden Interfaces, eingeschlossen in runden Klammer.
- Parameterliste: Liste von durch Komma getrennten formalen Parametern, eingeschlossen in runden Klammern. Die Datentypen bei den Parametern können dabei weggelassen werden.
- Pfeil-Operators →
- Body: aus einem einzelnen Ausdruck (ohne ";") oder aus einem Anweisungsblock, das heisst, mehrere Anweisungen eingeschlossen in " ".

3.8 Exception Handling

Beim Exception Handling in Java wird beim Auftreten eines Fehlers im geschützten Block die Ausführung dessen abgebrochen, der Fehlerbehandlungscode ausgeführt und dann der geschützt Block weitergeführt (wie Interupts). Dadurch kann der fehlerfreie Fall von den Fehlerfällen sauber getrennt werden. Zudem kann man keinen Fehler vergessen, da der Compiler prüft, ob zu jeder möglichen Ausnahme ein Behandlung existiert. Achtung: Die speziellen Exception-Typen müssen vor den allgemeienn Exception-Typen liegen. Am Ende wird oftmals ein finally eingefügt, damit gewisse Abschlussarbeiten wie z.B. das Schliessen eines Dokuments auch im Fehlerfall ausgeführt wird.

```
Beispiel:
try {
     p(...);
     q(...);
     r(..);
     In.open("myfile.txt");
} catch(Exception1 e){
     error(...)}
} catch(Exception2 e){
     error(...)}
}finally \{
     In.close() }}
// finally sorgt dafuer, dass in jedem Fall das Dok geschlossen wird.
```



public static void main(String[] args) **throws Exception** wird im Funktionskopf angegeben, wenn auf die Standard Exceptions der in der Funktion enthaltenen Methoden zurückgegriffen werden. Das heisst, dass nicht alle Exceptions im Programm selbst behandelt werden sollen ("Quick and Dirty Programming).

3.9 GUI

3.9.1 AWT - Abstract Window Toolkit

AWT verwendet die GUIs des zugrundeliegenden Betriebssystems. Deshalb sind die Plattform-Eigenheiten nicht völlig eliminierbar. Es hat eine beschränkte Funktionalität, da es dem kleinsten gemeinsamen Nenner der verschiedenen Plattformen entspricht.

3.9.2 Swing

Die GUI-Elemente werden in Java ërstellt". Dadurch ist das "Look and Feelänpassbar (standardmössig L&F des Betriebssystems). Es kann zur Laufzeit umgeschaltete werden. Zudem ist der Funktionsumfang sehr gross. Es baut auf Teilen von AWT auf, ersetzt jedoch auch gewisse Teile. Intern wird die MVC-Architektur verwendet.

3.9.3 Event Handling

Ein Framework ist ein Programmgerüst, in das vom Programmierer das Anwendungsprogramm eingebettet wird. Aus dem Framework heraus werden Funktionen des Anwendungsprogramms aufgerufen (Hollywood-Prinzip). Je nach Ereignisart werden verschiedene Event-Typen unterscchieden. Diese werden unterteilt in High-Level-Events, Intermediate- Level-Events und Low-Level-Events.

Varianten zur Implementierung des Event Handlings Variante 1: Listener in der Hauptklasse
die Hauptklasse implementiert das Listener-Interface.
z.B. public class MyFrame extends JFrame implements ActionListener
Nachteil: es gibt nur einen Listener auch bei mehreren Buttons.
Variante 2: Listener als externe Klassen
für jeden benötigten Listener wird eine eigene Klasse erstellt.
z.B. IncButtonListener implements ActionListener ...
DecButtonListener implements ActionListener ...
Nachteil: zusätzliche Klassen, welche eine Konstruktor benötigen.
Variante 3: Listener als innere (anonyme) Klassen
Bsp: decButton.addActionListener (new ActionListener() ...);
Vorteile: Zugriff zur Hauptklasse möglich, pro Button eigener Listener.
Nachteil: etwas unübersichtlich.
Variante 4: Listener als Lambda-Ausdrücke (seit Java 8)
Für jeden benötigten Listener wird ein Lamda-Ausdruck erstellt. Dabei kann der Funktionskopf ("void actionPerformed(ActionEvent e)"weggelassen werden.
Bsp: event -¿ label.setText(Änz = "+ –anzahl)
Vorteile: Zugriff zur Hauptklasse möglich, pro Button eigener Listener, kompakt.
,

3.9.4 Layout Manager

Beim **GridLayout** wird die Anzahl Zeilen und Spalten im Konstruktor angeben. Die Reihenfolge von add() bestimmt die Anordnung. Bei einem 2-dimensionalen Grid gehts zuerst hinuntern und dann nach rechts. Alle Zellen gleich gross.
Beim **FlowLayout** sind die Zellen so breit wie der Text darin.
Beim **BorderLayout** müssen nicht alle Zellen gleich gross sein. Die Zelle im Zentrum wird maximiert und jene an den Rändern minimiert. Bei add() muss eine Position mit den Himmelsrichtungen angegeben werden.
Beim **CardLayout** werden verschiedene Layout kombiniert.

3.10 AWT- & Swing Komponenten

Container:	Container / JContainer	Window / JWindow
	Panel / JPanel	Frame / JFrame
	Applet / JApplet	Dialog / JDialog
	ScrollPane / JScrollPane	FileDialog / JFileChooser
LayoutManager:	BorderLayout	GridBagLayout
	CardLayout	GridLayout
	FlowLayout	BoxLayout (mit Swing)
Component:	Container / JContainer	ScrollBar / JScrollBar
	Button / JButton	TextComponent / JTextComponent
	Canvas	TextArea / JTextArea
	CheckBox / JCheckBox	TextField / JTextField
	Choice / JComboBox	- / JTable
	Label / JLabel	- / JTree
	List / JList	

3.11 Others

Bezüglich Binding entsprechen die abstrakten Klassen von Java den virtuellen Klassen von C++.
Der Garbage Collector kann explizit mit dem Befehl **ja va.lang.System.gc()** resp. **java.lang.Runtime.gc()** aufgerufen werden.
Arrays werden wie alle Objekte bei Java auf dem Heap mit new erzeugt und als Referenz zurückgegeben. Deshalb gehören sie im Gegensatz zu C++ zu den Referenzdatentypen.

Java vs. C++
(+) Java hat ein integriertes Speichermanagement mit dem Garbage Collector, was bei C++ selbst gemacht werden muss.
(+) Java ist plattformunabhängig.
(+) Java verfügt über umfassende Klassenbibliotheken (auch für die GUI Programmierung).
(–) Java kann einfacher Reverse Engineered werden.
(–) C++ ist schneller.
(–) Für die Verwendung von Java benötigt man eine Java Virtual Machine mit einer JRE (Java Runtime Environment) auf dem ausführenden Gerät.

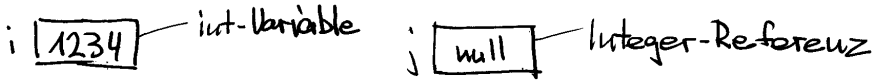
4 Papierübungen

```
import java.awt.*;
import javax.swing.*;
public class GuiLayoutTest extends JFrame
{
   public GuiLayoutTest()
   //Konstruktor: "View" (Ansicht) aufbauen und initialisieren.
   {
      super("---Digestif---");
      this.setDefaultCloseOperation(EXIT_ON_CLOSE);

      JPanel panel = new JPanel();
      getContentPane().setLayout( new GridLayout(2,1) );
      //2 Zeilen, 1 Spalte
      panel.setLayout(new BorderLayout() );
      getContentPane().add( panel );
      getContentPane().add(new JButton("Grappa") );
      panel.add (new JButton ("Weinbergpfirsichbrand") , BorderLayout.WEST);
      panel.add (new JButton ("Enzian") , BorderLayout.EAST);
      panel.add (new JButton ("Marillen") , BorderLayout.SOUTH);
      pack();   //Minimalgroesse einnehmen
      setVisible(true);
   }
   //--------------------------------------------------------------------------
   public static void main(String[] args)
   {
      GuiLayoutTest app = new GuiLayoutTest();
   }}
```

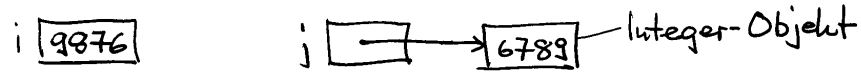
Aufgabe 1: Einfache Datentypen vs. Referenz-Datentypen – Lösung

a)

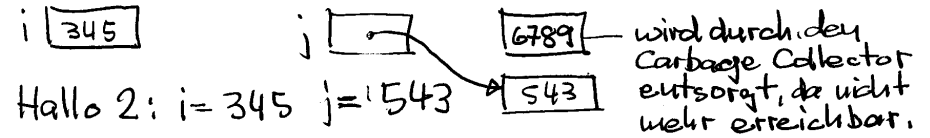


Hallo! i= 1234

b)



Hallo 1: i= 9876 j= 6789

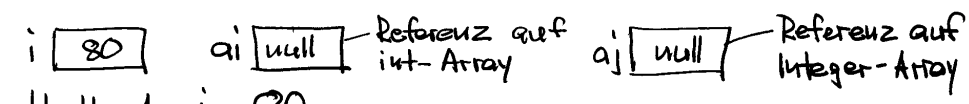


Hallo 2: i= 345 j= 543

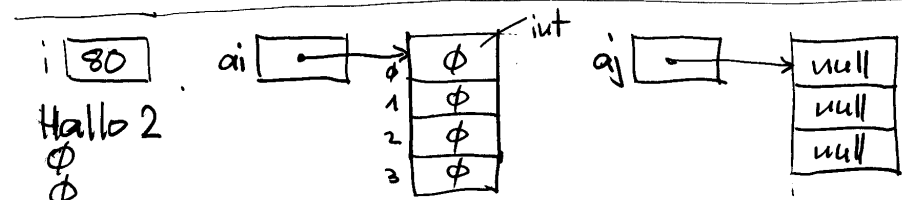
c)

Hallo: i+j= 808 — Stringverkettung, nicht Addition!

d)

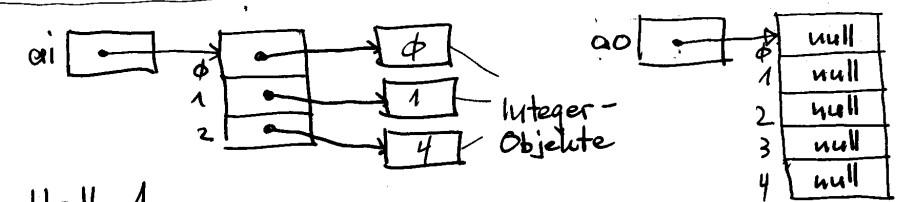


Hallo 1: i= 80

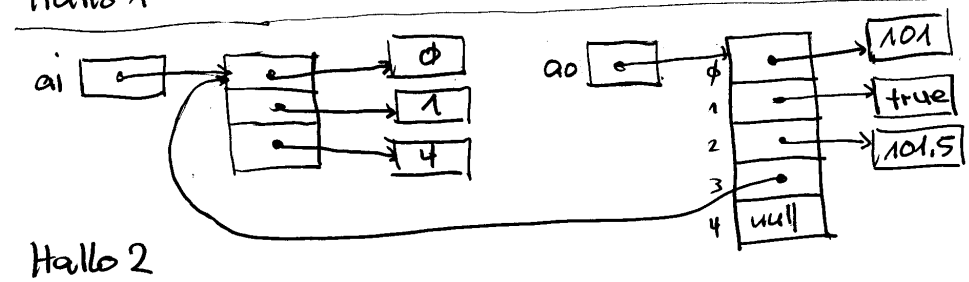


Hallo 2
0
0
0
0

e)



Hallo 1



Hallo 2

Aufgabe 5: Packages (Programmorganisation) – **Lösung**

a) Weshalb wird für die Anweisung `System.out.println("Hallo");` keine `import`-Anweisung benötigt?

Die Klasse "System" gehört zum Package "java.lang". Dieses Package wird automatisch, auch ohne import-Anweisung, importiert.

b) Was muss geändert werden, wenn das Programm von Übung 2, Aufgabe 1 ("TextEinAusgabe") ohne `import`-Anweisung auskommen soll?

An Stelle von "BufferedReader" ist jeweils "java.io.BufferedReader" zu schreiben. Entsprechend bei "InputStreamReader" (neu: "java.io.InputStreamReader") und "IOException" (→ "java.io.IOException").

c) Wie lautet die `import`-Anweisung, um in der Klasse "August" (aus dem Package "aaa") die Klasse 'Xaver' aus dem Default-Package zu verwenden?

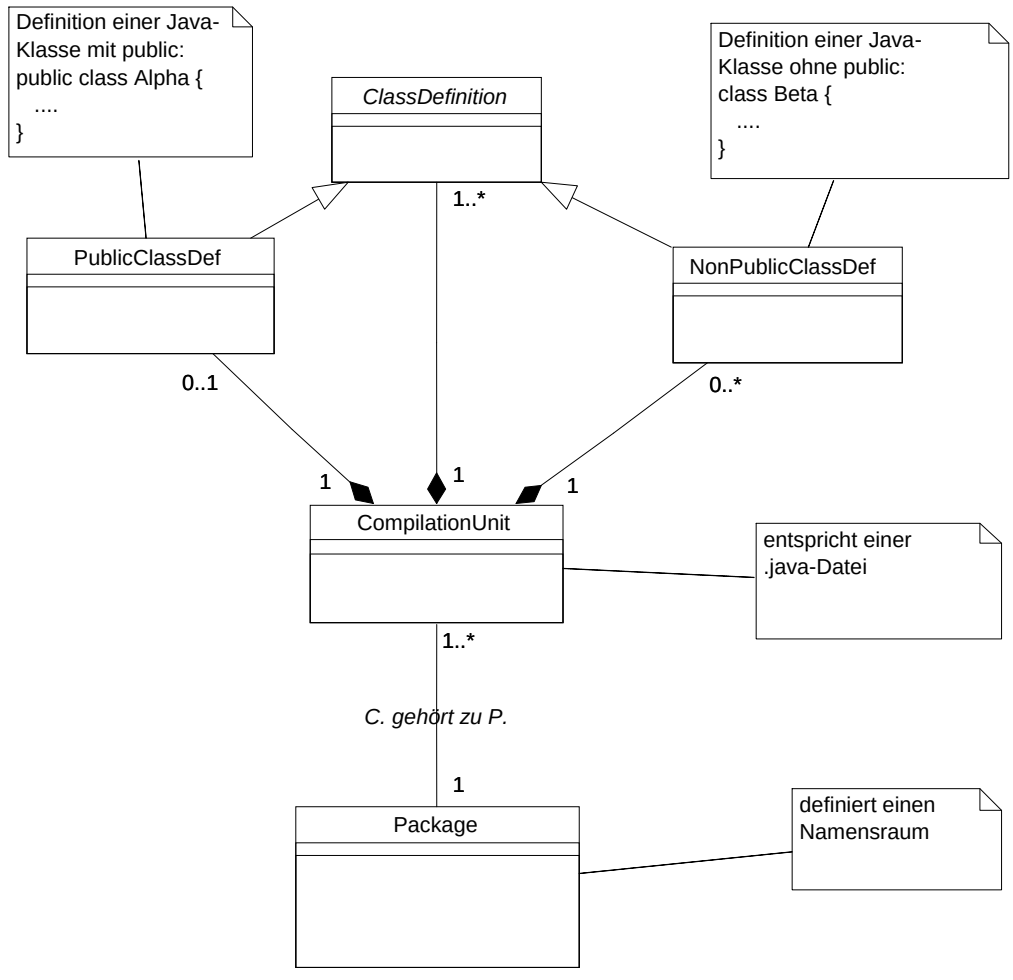
Eine solche import-Anweisung gibt es nicht. Es ist nicht möglich Klassen im Default-Package in Klassen zu verwenden, die nicht im selben Default-Package definiert sind!

d) Was kann über die Filestruktur (Verzeichnisse) ausgesagt werden, wenn die Datei "Julius.java" die Anweisung "package romeo.julia;" enthält?

Die Datei "Julius.java" ist im Unterverzeichnis "julia" untergebracht, und dieses Unterverzeichnis ist Teil des Verzeichnisses "romeo" (Pfad: "romeo/julia/Julius.java").

Aufgabe 7: Packages (Programmorganisation) – **Lösung**

Gegeben ist das folgende UML-Klassendiagramm, welches die für die Programmorganisation bei Java wichtigen Begriffe darstellt.



Aufgabe 1: Verständnisfragen – **Lösung**

a) Erklären Sie, was man bei der Klasse "StringBuffer" unter den Begriffen "Size" und "Capacity" versteht.

"Size" = aktuelle Anzahl Zeichen in einem StringBuffer-Objekt.

"Capacity" = mögliche Anzahl Zeichen, die ein StringBuffer-Objekt total enthalten kann, ohne dass eine Reorganisation des Speichers erfolgt.

b) Erklären Sie, warum es bei der Klasse "String" nicht sinnvoll ist von "Capacity" zu sprechen.

Weil ein String-Objekt seine Grösse nicht (nie) ändern kann.

c) Erklären Sie, 1. was man unter dem Begriff "immutable" versteht und 2. warum diese Eigenschaft so wichtig ist.

"immutable" = unveränderbar. String-Objekte sind inhaltlich "immutable" (unveränderbar). Das heisst, der Text, den sie beinhalten, kann nicht geändert werden.

Vorteil: Referenzen darauf können in einem Programm beliebig "herumgeschoben", dupliziert etc. werden, man kann trotzdem sicher sein, dass der Inhalt eines solchen Objektes nicht verändert wird oder wurde.

d) Gegeben ist die folgende Klasse "Person" (Fragment):

```
public class Person
{  private String name;
   public Person (String name)
   {  this.name = new String(name);
   }
   // ...
}
```

Erklären Sie, warum der obige Konstruktor unsinnig ist, obwohl das Programm gleichwohl korrekt funktioniert.

'`this.name = name;`' *genügt; es braucht keine String-Objekt-Kopie (deep-copy) da das String-Objekt '*`name`*' ja immutable ist.*

e) Erklären Sie, was - im Vergleich - die Vor- und Nachteile der Klassen "String" und "StringBuffer" sind.

String:
- Vorteile: einfach in der Anwendung (z.B. +-Operator), immutable;
- Nachteil: unter Umständen werden temporär String-Objekte erzeugt, die nachher vom GC wieder entsorgt werden müssen.

StringBuffer:
- Vorteile: vollständige Kontrolle über Speicherverwendung (Capacity);
- Nachteile: nicht immutable, etwas komplizierter in der Anwendung.

Aufgabe 0: Verständnisfragen – **Lösung**

a) Wie wird eine Exception erzeugt und "ausgelöst"?

Indem ein Exception-Objekt erzeugt wird (z.B. mit `new MyException("XYZ nicht gefunden")` *und dieses mit* `throw` *geworfen wird. (throw ist ein Schlüsselwort).*

b) Woran sind Funktionen erkennbar, die Exceptions *erzeugen* ?

Exceptions sind Objekte von Exception-Klassen. Die werden wie alle anderen Objekte auch mit new erzeugt. Siehe Teilaufgabe a).

c) Woran sind Funktionen erkennbar, die Exceptions *weiterleiten,* aber sie nicht selber erzeugen?

Am Ende ihres Funktionskopfes steht " throws" gefolgt von der (den) Exception(s) (Klassennamen). Innerhalb der Funktion kommt das Schlüsselwort "throw" nicht vor.

d) Was ist der Unterschied bei der Behandlung ("catch") zwischen "checked Exceptions" und "Runtime Exceptions"?

"Checked Exceptions" müssen irgendwo im Programm behandelt werden (oder zumindest an die JVM weitergeleitet werden), "RunTime Exceptions" hingegen nicht.

e) Von welcher Klasse werden normalerweise *eigene* Exception-Klassen abgeleitet?

Von der Klasse "java.lang.Exception".

Aufgabe 3: Pi berechnen mit N-Eck - **fakultativ**

Sourcecode der Lösung

Siehe Verzeichnis "EclipseWsp" Projektverzeichnis "lueb09_A3_NEckPi".

Formeln

$$S_{2N} = \sqrt{2 - \sqrt{4 - S_N{}^2}} \qquad (1)$$

$$nPi = \frac{U}{2} = \frac{N}{2} \cdot S_N \qquad (2)$$

Bemerkungen zur Lösung:

Bei der obigen Lösung wird bei jedem Iterationsschritt zusätzlich noch der relative Fehler von nPi im Vergleich zum Pi von "java.lang.Math" ausgegeben. Damit stellt man fest, dass nach etwa 17 Iterationsschritten die Genauigkeit des berechnetes Näherungswertes für Pi (= nPi) nicht mehr wie erwartet zu, sondern abnimmt. Schlussendlich resultiert für nPi der Wert 0 (!).

Dieses Phänomen rührt daher, dass gemäss der obigen Formel (1) der Wert der inneren Wurzel bei grossen Werten von N gegen 4 tendiert (Grenzwert), sodass dann in der äusseren Wurzel drin 2 – 2 gerechnet wird, wodurch ein Wert von 0 für die neue Seitenlänge resultiert. Mit Formel (2) entsteht dann der beobachtete Wert 0 für nPi.

Dieses Phänomen kann dadurch vermieden werden, indem Formel (1) wie folgt algebraisch umgeformt wird:

$$S_{2N} = \sqrt{2 - \sqrt{4 - S_N{}^2}} \cdot \frac{\sqrt{2 + \sqrt{4 - S_N{}^2}}}{\sqrt{2 + \sqrt{4 - S_N{}^2}}} = \frac{S_N}{\sqrt{2 + \sqrt{4 - S_N{}^2}}} \qquad (3)$$

Mit Formel (3) ergibt sich dann ein Grenzwert für grosse N von $S_{2N} = \frac{S_N}{2}$.

Das heisst, für grosse Werte von N halbiert sich die Seitenlänge, wenn man die Anzahl der Ecken verdoppelt.