**LK-Q1-I**

**2. Entwurf einer grafischen Benutzeroberfläche**

**Projekt Taschenrechner**

Die Programmierung von grafischen Benutzungsoberflächen ist kein verbindlicher inhaltlicher Schwerpunkt des Kernlehrplans Informatik. Dennoch sollten die Schülerinnen und Schüler in der Qualifikationsphase insbesondere im Leistungskurs schon aus Motivationsgründen in der Lage sein, ihre Java-Programme mit einer grafischen Benutzungsoberfläche zu versehen, ohne sich intensiv mit deren Programmierung auseinandersetzen zu müssen. Mithilfe der Java-Entwicklungsumgebung „Java-Editor“, die für Windows-Systeme kostenfrei genutzt werden kann ([www.javaeditor.de](http://www.javaeditor.de)), lassen sich interaktiv ansprechende Java-Swing-Oberflächen „zusammenklicken“ und durch Anpassung der Attribute (Properties) gestalten. Der Quelltext wird von der Entwicklungsumgebung automatisch erzeugt. Vom Programmierer müssen lediglich im Konstruktor die im Anwendungsprogramm benötigten Objekte erzeugt sowie die Aktionsmethoden implementiert werden, die nach Eintritt z. B. eines Klickereignisses automatisch nebenläufig aufgerufen werden.

Häufig ist der Übergang von der in der Einführungsphase verwendeten grafischen Programmierumgebung zu einer textbasierten Java-Programmierung mit Swing-Oberfläche nicht einfach. In diesem Unterrichtsvorhaben wird daher zunächst ein Programm mit einer GLOOP-Benutzungsoberfläche entwickelt und diese unter Beibehaltung der Datenmodell-Klasse durch eine Swing-Oberfläche ersetzt. Dabei wird den Schülerinnen und Schüler das Prinzip der Ereignissteuerung vermittelt.

**Projektphasen**

1. Aufgabenstellung

2. Entwicklung des Implementationsdiagramms für die GLOOP-Oberfläche

3. Implementation der GLOOP-Oberfläche

4. Entwurf der Datenmodellklasse

5. Implementation der Datenmodellklasse

6. Ereignissteuerung in GLOOP und Implementation der Aktionsmethoden

7. Entwicklung der Swing-Benutzungsoberfläche

8. Ereignissteuerung in Swing und Implementation der Aktionsmethoden

9. Vervollständigung und Test der Implementation

10. Reflexion des Projekts

**1. Aufgabenstellung**

Zu entwickeln ist ein Programm für einen einfachen Taschenrechner, der zwei Dezimalzahlen addieren, subtrahieren, multiplizieren und dividieren kann, ohne Beachtung der Regel „Punkt vor Strich“. Außerdem soll die Quadratwurzel einer Zahl berechnet und eine Prozentangabe in einen Dezimalbruch umgerechnet werden können.

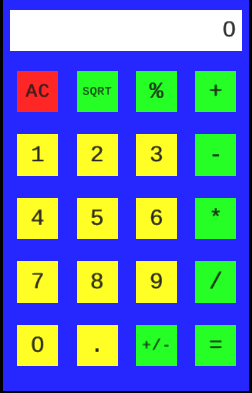
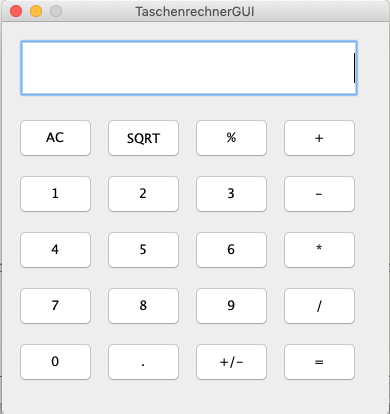
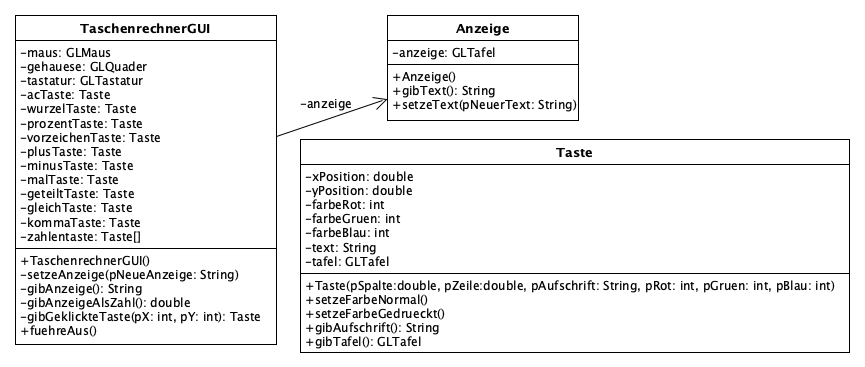
In der ersten Version soll die Benutzungsoberfläche mit GLOOP gestaltet werden, später wird die GLOOP-Oberfläche unter Beibehaltung der Implementation des Datenmodells durch eine Java-Swing-Oberfläche ersetzt.

Abbildung 2: Taschenrechner mit Java-Swing-Oberfläche

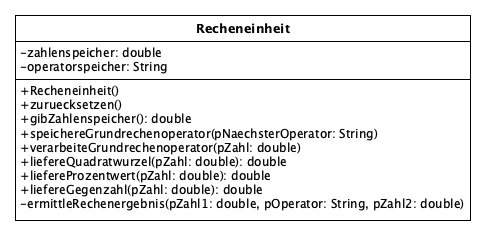
Abbildung 1: Taschenrechner mit GLOOP-Oberfläche

**2. Implementationsdiagramm für die Benutzungsoberfläche des GLOOP-Taschenrechners**



**4. Entwurf der Datenmodellklasse**

4.1 Implementationsdiagramm der Klasse Recheneinheit



4.2 Dokumentation der Klasse Recheneinheit

Ein Objekt der Klasse Recheneinheit stellt grundlegende Rechenoperationen zur Verfügung. Unter anderem können Verkettungen von Grundrechenoperationen berechnet werden. Dazu speichert das Objekt einen Zahlenwert als Zwischenergebnis und einen noch nicht ausgewerteten Grundrechenoperator. Sobald der zweite Operand für den gespeicherten Grundrechenoperator übergeben wird, erfolgt eine Auswertung des gespeicherten Operators, dessen Resultat als neues Zwischenergebnis gespeichert wird.

**public Recheneinheit()**

Initialisiert ein neues Objekt der Klasse Recheneinheit. Im Speicher ist eine 0 und es gibt keinen nicht ausgewerteten Operator.

**public void zuruecksetzen()**

Setzt das Objekt in den Zustand bei der Initialisierung zurück, d.h. im Speicher ist eine 0 und es gibt keinen nicht ausgewerteten Operator.

**public double gibZahlspeicher()**

Liefert den aktuell gespeicherten Zahlenwert zurück, d. h., das Zwischenergebnis der bis jetzt durchgeführten Grundrechenoperationen.

**public void speichereGrundrechenoperator(String pNaechsterOperator)**

Speichert den Grundrechenoperator pNaechsterOperator zur späteren Anwendung.

**public void verarbeiteGrundrechenoperator(double pZahl)**

Die Methode wendet den gespeicherten Grundrechenoperator auf den im Objekt gespeicherten Zahlenwert als ersten Operanden und den in pZahl übergebenen Wert als zweiten Operanden an. Das Ergebnis wird als neuer Zahlenwert gespeichert.Gibt es keinen gespeicherten Grundrechenoperator, wird pZahl als neuer Zahlenwert gespeichert.

**public double liefereQuadratwurzel(double pZahl)**

Liefert die Quadratwurzel von pZahl zurück.

**public double liefereProzentwert(double pZahl)**

Liefert ein Hundertstel von pZahl zurück.

**public double liefereGegenzahl(double pZahl)**

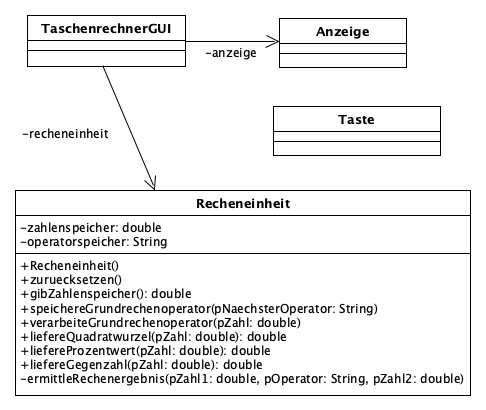
Liefert die Gegenzahl von pZahl zurueck.

**private double ermittleRechenergebnis(double pZahl1,**

**String pOperator, double pZahl2)**

Hilfsmethode: Ermittelt die Auswertung des Operators pOperator bei Anwendung auf pZahl1 und pZahl2 und liefert sie als Rückgabewert.

4.3 Implementationsdiagramm des GLOOP Taschenrechners



**5. Implementation der Klasse Recheneinheit**

|  |  |
| --- | --- |
| 1 | public class Recheneinheit { |
| 2 | private double zahlspeicher; |
| 8 | private String operatorspeicher; |
|  |  |
| 9 | public Recheneinheit() { |
| 10 | zuruecksetzen(); |
| 11 | } |
|  |  |
| 12 | public void zuruecksetzen(){ |
| 13 | zahlspeicher = 0; |
| 14 | operatorspeicher = ""; |
| 15 | } |
|  |  |
| 16 | public double gibZahlspeicher() { |
| 17 | return zahlspeicher; |
| 18 | } |
|  |  |
| 19 | public void speichereGrundrechenoperator(String pNaechsterOperator) { |
| 20 | operatorspeicher = pNaechsterOperator; |
| 21 | } |

|  |  |
| --- | --- |
|  |  |
| 22 | public void verarbeiteGrundrechenoperator(double pZahl) { |
| 23 | if (operatorspeicher.equals("")) { |
| 24 | zahlspeicher = pZahl; |
| 25 | } else { |
| 26 | zahlspeicher = ermittleRechenergebnis(zahlspeicher,  operatorspeicher, pZahl); |
| 27 | operatorspeicher = ""; |
| 28 | } |
| 29 | } |
|  |  |
| 30 | public double liefereQuadratwurzel(double pZahl) { |
| 31 | return Math.sqrt(pZahl); |
| 32 | } |
|  |  |
| 33 | public double liefereProzentwert(double pZahl) { |
| 34 | return pZahl / 100; |
| 35 | } |
|  |  |
| 36 | public double liefereGegenzahl(double pZahl) { |
| 37 | return -pZahl; |
| 38 | } |
|  |  |
| 39 | private double ermittleRechenergebnis(double pZahl1,  String pOperator, double pZahl2) { |
| 40 | double ergebnis = 0; |
| 41 | if (pOperator.equals("+")) { |
| 42 | ergebnis = pZahl1 + pZahl2; |
| 43 | } else if(pOperator.equals("-")) { |
| 44 | ergebnis = pZahl1 - pZahl2; |
| 45 | } else if (pOperator.equals("\*")) { |
| 46 | ergebnis = pZahl1 \* pZahl2; |
| 47 | } else if (pOperator.equals("/")) { |
| 48 | ergebnis = pZahl1 / pZahl2; |
| 49 | } |
| 60 | return ergebnis; |
| 51 | } |
|  |  |
| 52 | } |

**6. Ereignissteuerung in GLOOP und Implementation der Aktionsmethoden**

6.1 Ereignissteuerung in GLOOP

Eingaben über die Tastatur oder die Maus in ein laufendes Java-Programm sind Ereignisse, auf die das Programm durch Aufruf einer spezifischen Methode oder Ausführen einer Anweisungsfolge reagieren muss, wenn das Ereignis eintritt. Solche Methoden werden im Folgenden **Aktionsmethoden** genannt. Da ein GLOOP-Programm nicht automatisch eine Nachricht sendet, wenn die Maus betätigt oder eine Taste gedrückt wurde, müssen diese Ereignisse in einer Schleife, Ereignisschleife genannt, kontinuierlich abgefragt werden.

Im Taschenrechnerprogramm ist die Ereignisschleife in der Methode fuehreAus der Klasse TaschenrechnerGUI enthalten.

6.2 Implementation der Methode fuehreAus

|  |  |
| --- | --- |
| 1 | public void fuehreAus(){ |
| 2 | while (!tastatur.esc()){ |
| 3 | if (maus.linksklick()){ |
|  | //Die Taste, auf dem die Maus gedrückt wurde, wird ermittelt |
| 4 | Taste taste = gibGeklickteTaste(maus.gibX(), maus.gibY()); |
| 5 | if (taste! = null) { |
|  | //Die Taste wird für 100 Millisekunden grau |
| 6 | taste.setzeFarbeGedrueckt(); |
| 7 | Sys.warte(100); |
| 8 | taste.setzeFarbeNormal(); |
| 9 | String aufschrift = taste.gibAufschrift(); |
| 10 | switch (aufschrift) { |
| 21 | case "+": |
| 22 | case "-": |
| 23 | case "\*": |
| 24 | case "/": aktionGrundrechenartTaste(aufschrift);break; |
| 25 | case "=": aktionGleichTaste("");break; |
| 26 | case ".": aktionKommaTaste(); break; |
| 27 | case "+/-": aktionVorzeichenTaste();break; |
| 28 | case "%": aktionProzentTaste();break; |
| 29 | case "SQRT": aktionWurzelTaste();break; |
| 30 | case "AC": aktionAllClearTaste();break; |
| 31 | default : aktionZahlTaste(aufschrift); |
| 32 | } |
| 33 | } |
| 34 | } |
| 35 | Sys.warte(); |
| 36 | } |
| 37 | Sys.beenden(); |
| 38 | } |

Die Ereignisschleife wie auch das Programm werden erst nach Betätigen der Escape-Taste beendet.

In den Zeilen 10 bis 31 werden in Abhängigkeit der geklickten Taste Aktionsmethoden aufgerufen. Es werden acht verschiedene Aktionsmethoden benötigt. Die Methode

aktionZahlTaste wird aufgerufen, wenn eine Zifferntaste geklickt wurde. Aus den Namen der anderen Aktionsmethoden wird deutlich, nach welchem Tastaturereignis sie aufgerufen werden.

Die folgende Tabelle zeigt, welche Anweisungen nach Klicken der verschiedenen Tasten von den Aktionsmethoden ausgeführt werden müssen.

6.3 Implementation der Aktionsmethoden

|  |  |
| --- | --- |
| Taste | Aktion |
| 0,1,...9 | Wenn im Anzeigeobjekt 0 oder 0.0 steht oder  die letzte geklickte Taste keine Zifferntaste ist  schreibe die Ziffer ins Anzeigeobjekt  sonst  verlängere die Zahl im Anzeigeobjekt um die Ziffer |
| . | Wenn noch kein Komma eingegeben wurde  verlängere die Zahl im Anzeigeobjekt um das Komma  sonst  tue nichts |
| +, - \*, / | rufe die Methode verarbeiteGrundrechenoperator der Recheneinheit mit der Zahl des Anzeigeobjektes als Parameter auf und zeige das Ergebnis im Anzeigeobjekt  speichere den Operator in der Recheneinheit |
| = | rufe die Methode verarbeiteGrundrechenoperator der Recheneinheit mit der Zahl des Anzeigeobjektes als Parameter auf und zeige das Ergebnis im Anzeigeobjekt  Lösche den Operator in der Recheneinheit |
| SQRT, +/-, % | rufe die zugehörige Methode (liefereQuadratwurzel, liefereGegenzahl, liefereProzentwert) der Recheneinheit mit der Tastenaufschrift und der Zahl des Anzeigeobjektes als Parameter auf und zeige das Ergebnis im Anzeigeobjekt |
| AC | rufe die Methode zuruecksetzen der Recheneinheit auf und schreibe 0 in das Anzeigeobjekt |

Um die Eingabe einer Ziffer oder eines Kommas richtig bearbeiten zu können, wird die Klasse TaschenrechnerGUI um die beiden Attribute neueZahl und komma erweitert. Beide Attribute werden mit false initialisiert. komma wird auf true gesetzt, sobald ein Komma eingegeben wurde und wieder auf false, wenn eine neue Zahl eingegeben wurde. neueZahl erhält den Wert false nach jeder Zifferntaste oder einem Komma, bei allen anderen Tasten den Wert true. Über die Tabelle hinaus wird in der Aktionsmethode aktionZahlTaste die Anzahl der Zeichen im Anzeigeobjekt auf 14 beschränkt. Die folgende Implementation einiger Aktionsmethoden ergeben sich direkt aus der Tabelle.

|  |  |
| --- | --- |
| 1 | public void aktionZahlTaste(String pTastenzeichen){ |
| 2 | String anzeigeText = gibAnzeige(); |
| 3 | if (anzeigeText.equals("0") || neueZahl){ |
| 4 | anzeigeText = pTastenzeichen; |
| 5 | neueZahl = false; |
| 6 | } else if (anzeigeText.length() < 14 ) { |
| 7 | anzeigeText = anzeigeText+pTastenzeichen; |
| 8 | } |
| 9 | setzeAnzeige(anzeigeText); |
| 10 | } |
|  |  |
| 11 | public void aktionKommaTaste() { |
| 12 | String anzeigeText = gibAnzeige(); |
| 13 | if (neueZahl) { |
| 14 | setzeAnzeige("0."); |
| 15 | neueZahl=false; |
| 16 | komma = true; |
| 17 | } else if (!komma) { |
| 18 | setzeAnzeige(anzeigeText + "."); |
| 19 | komma = true; |
| 20 | } |
| 21 | } |

|  |  |
| --- | --- |
|  |  |
| 22 | private void aktionGrundrechenartTaste(String pOperator){ |
| 23 | recheneinheit.verarbeiteGrundrechenoperator(gibAnzeigeAlsZahl()); |
| 24 | setzeAnzeige("" + recheneinheit.gibZahlspeicher()); |
| 25 | recheneinheit.speichereGrundrechenoperator(pOperator); |
| 26 | neueZahl = true; |
| 27 | komma = false; |
| 28 | } |
|  |  |
| 29 | public void aktionAllClearTaste(){ |
| 30 | recheneinheit.zuruecksetzen(); |
| 31 | setzeAnzeige("0"); |
| 32 | neueZahl = true; |
| 33 | komma = false; |
| 34 | } |

**Aufgabe**

Implementieren Sie die fehlenden Aktionsmethoden aktionGleichTaste,

aktionWurzeltaste, aktionProzentTaste und aktionVorzeichenTaste.

**7. Entwicklung der Swing-Benutzungsoberfläche**

Ein einfaches Werkzeug für Windows-Rechner zur Entwicklung von Java-Programmen einschließlich der interaktiven Erstellung von Java-Swing-Benutzungsoberflächen ist der frei verfügbare Java-Editor, der sich unter der URL <http://javaeditor.org/doku.php?id=de:java-editor> herunterladen lässt. Dort ist auch eine ausführliche Bedienungsanleitung für den Java-Editor zu finden.

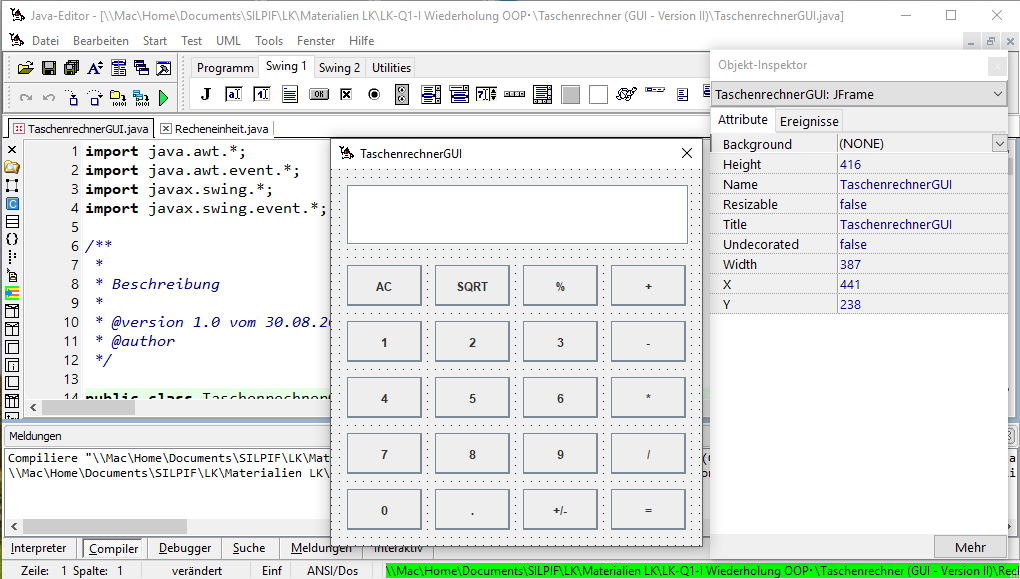
Um den Taschenrechner mit der in Abbildung 2 dargestellten Java-Swing-Oberfläche zu implementieren, sollte zunächst ein neuer Ordner mit Namen TRSwing erstellt werden. Nach Start des Java-Editors sollte ein neues JFrame-Objekt erstellt und unter dem Namen TaschenrechnerGUI in diesem Ordner gespeichert werden. Die Oberfläche kann interaktiv „zusammengeklickt“ werden. Die zugehörige Java-Quelltext-Datei wird vom Java-Editor parallel automatisch generiert. Für den Taschenrechner müssen 12 JButton-Objekte sowie ein JTextField-Objekt entsprechend der Abbildung 3 in das JFrame-Objekt gezogen werden.

Abbildung 3: Java-Editor

Abbildung 3: Java-Editor

Die Aufschrift der JButton-Objekte wird in der Zeile Text des Objektinspektors eingegeben. Die Namen werden in der Zeile Name des Objektinspektors festgelegt und sollten mit dem Buchstaben b beginnen, um deutlich zu machen, dass es sich um ein JButton-Objekt handelt. Im Beispielprogramm wurden die Bezeichnungen (b0, ... b9 für die Zahlentasten sowie bKomma, bPlus, bMinus, bMal, bGeteilt, bGleich, bAC, bWurzel, bProzent, bVorzeichen für die anderen Tasten festgelegt). Der Text des JTextfield-Objektes wird auf „0“ gesetzt und der Name auf „tfAnzeige“.

Wenn die automatisch generierte Java-Klasse TaschenrechnerGUI mit dem Java-Editor compiliert und anschließend die Main-Methode aufgerufen wird, erscheint das Java-Swing-Fenster der Abbildung 2 auf dem Bildschirm, ohne dass etwas passiert, wenn ein Button geklickt wird.

**8. Ereignissteuerung in Swing und Implementation der Aktionsmethoden**

Wenn man sich den Quelltext der automatisch erstellten Klasse TaschenrechnerGUI anschaut, entdeckt man für **jeden** JButton eine Methode der folgenden Art.

public void bVorzeichen\_ActionPerformed(ActionEvent evt) {

// TODO hier Quelltext einfügen

} // end of bVorzeichen\_ActionPerformed

Diese Aktions-Methoden werden immer dann aufgerufen, wenn der zugehörige Button geklickt wurde. Bei ereignisgesteuerten Java-Programmen läuft im Hintergrund eine erst bei Schließen des Programmfensters endende Schleife, die permanent überprüft, ob ein Ereignis, das vorher durch Hinzufügen eines sogenannten ActionListener angemeldet wurde, ausgelöst wurde. Tritt das Ereignis ein, wird das laufende Programm unterbrochen und diese Aktionsmethode aufgerufen. Bei der Implementation des Taschenrechnerprogramms mit GLOOP musste die Ereignisschleife eigens implementiert werden, bei der Java-Swing-Oberfläche läuft diese automatisch im Hintergrund ab, weil die Entwicklungsumgebung jedem JButton einen ActionListener hinzufügt. Für den JButton bVorzeichen geschieht das beispielsweise mit folgender Syntax:

bVorzeichen.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent evt) {

bVorzeichen\_ActionPerformed(evt);

}

});

An die Aktionsmethoden übergeben wird ein Parameter vom Typ ActionEvent,der verschiedene Informationen zu dem Ereignis – bei Mausereignissen beispielsweise die Position des Mausklicks – liefert, die aber beim Taschenrechnerprojekt keine Rolle spielen.

Um das Taschenrechnerprogramm mit Java-Swing-Benutzungsoberfläche zu vervollständigen, müssen diese Aktionsmethoden analog zu den Implementationen des GLOOP-Taschenrechners implementiert werden. In der GLOOP-Implementation wurde für alle Zifferntasten dieselbe Aktions-Methode mit der Ziffer als Parameter aufgerufen. Dies wäre auch bei der Java-Swing-Implementation möglich gewesen. Da dann aber erhebliche Änderungen am automatisch erzeugten Quelltext für die Benutzungsoberfläche erforderlich wären, sollte bei Einführung der Gestaltung von Benutzungsoberflächen mit Java-Swing zunächst darauf verzichtet werden. Dasselbe gilt für die Aktionsmethoden für einen bzw. zwei Operanden. Die Implementation der Aktionsmethoden entspricht bis auf geringe Unterschiede der GLOOP-Implementation.

Als Beispiele sind im folgenden Listing die Aktionsmethoden für die Tasten b1, bKomma, bPlus , bSQRT und bAC angegeben.

|  |  |
| --- | --- |
| 1 | public void b1\_ActionPerformed(ActionEvent evt) { |
| 2 | String anzeigeText = tfAnzeige.getText(); |
| 3 | if (anzeigeText.equals("0") || anzeigeText.equals("0.0") ||  neueZahl){ |
| 4 | tfAnzeige.setText("" + "*1*"); |
| 5 | neueZahl = false; |
| 6 | } else { |
| 7 | tfAnzeige.setText(anzeigeText + "*1*"); |
| 8 | } |
| 9 | } |
|  |  |
| 10 | public void bKomma\_ActionPerformed(ActionEvent evt) { |
| 11 | String anzeigeText = tfAnzeige.getText(); |
| 12 | if (neueZahl) { |
| 13 | tfAnzeige.setText("0."); |
| 14 | neueZahl=false; |
| 15 | komma = true; |
| 16 | } else if (!komma) { |
| 17 | tfAnzeige.setText(anzeigeText+"."); |
| 18 | komma = true; |
| 19 | } |
| 20 | } |
|  |  |
| 21 | public void bPlus\_ActionPerformed(ActionEvent evt) { |
| 22 | double zahl = gibAnzeigeAlsZahl(); |
| 23 | recheneinheit.verarbeiteGrundrechenoperator(zahl); |
| 24 | tfAnzeige.setText(""+recheneinheit.gibZahlspeicher()); |
| 25 | recheneinheit.speichereGrundrechenoperator("+"); |
| 26 | neueZahl = true; |
| 27 | komma=false; |
| 28 | } |
|  |  |
| 29 | public void bSQRT\_ActionPerformed(ActionEvent evt) { |
| 30 | double zahl = gibAnzeigeAlsZahl(); |
| 31 | double ergebnis = recheneinheit.liefereQuadratwurzel(zahl); |
| 32 | tfAnzeige.setText("" + ergebnis); |
| 33 | neueZahl = true; |
| 34 | komma=false; |
| 35 | } |
|  |  |
| 36 | public void bAC\_ActionPerformed(ActionEvent evt) { |
| 37 | recheneinheit.zuruecksetzen(); |
| 37 | tfAnzeige.setText("0"); |
| 39 | } |

**Erläuterungen:**

Zeilen 1-9: In der Methode muss die Länge der Anzeige nicht begrenzt werden, weil der Inhalt des JTextfield-Objektes automatisch horizontal gescrollt wird.

Zeile 2: Die Methode getText liefert den Inhalt eines JTextField-Objektes

Zeilen 4 und 7: Bei der Implementation der Aktionsmethoden für die anderen Zifferntasten, muss die rot und kursiv geschriebene Ziffer durch die Aufschrift der entsprechenden Taste ersetzt werden.

Zeile 4: Die Methode setText setzt den Text eines JTextfield-Objektes auf den als Parameter übergebenen Wert.

Zeile 26: Bei der Implementation der Aktionsmethoden für die minus-, mal-, geteilt- und gleich-Taste, muss das Zeichen für den Operator entsprechend ersetzt werden.

Zeile 31: Bei der Implementation der Aktionsmethoden für die Vorzeichen- und Prozent-Taste, muss die entsprechende Methode der Recheneinheit aufgerufen werden

**9. Vervollständigung der Implementation und Test des Programms**

Die Klasse Recheneinheit wird unverändert aus der GLOOP-Implementation übernommen und muss in den Ordner TRSwing der Java-Swing-Implementation kopiert werden.

Die Attributliste der Klasse TaschenrechnerGUI muss um die drei Attribute recheneinheit, fertig und komma ergänzt werden.

...

private JButton bVorzeichen = new JButton();

// Ende Attribute

private Recheneinheit recheneinheit;

private boolean neueZahl=false;

private boolean komma=false;

Am Ende des Konstruktors der Klasse TaschenrechnerGUI wird das Objekt recheneinheit instanziiert.

...

setVisible(true);

recheneinheit = new Recheneinheit();

} // end of public TaschenrechnerGUI

Mit der vollständigen Implementation kann das Taschenrechnerprogramm getestet werden.

**10. Reflexion des Projekts**

In der Reflexionsphase sollten die beiden Implementationen verglichen werden. Insbesondere ist die Unabhängigkeit des Datenmodells von der Benutzungsoberfläche noch einmal hervorzuheben.