

## Übungen zu Einführung in die Informatik: Programmierung und Software-Entwicklung: Lösungsvorschlag

### Aufgabe 12-1

### Ausnahmen und Ausnahmebehandlung

Präsenz

Gegeben sei eine Methode zum Umwandeln von Strings in Zeichenketten, die nur aus Großbuchstaben bestehen. Die Methode `printUpperCase(String s)` wandelt jeweils einen einzelnen String in Großbuchstaben um und gibt den umgewandelten String zurück.

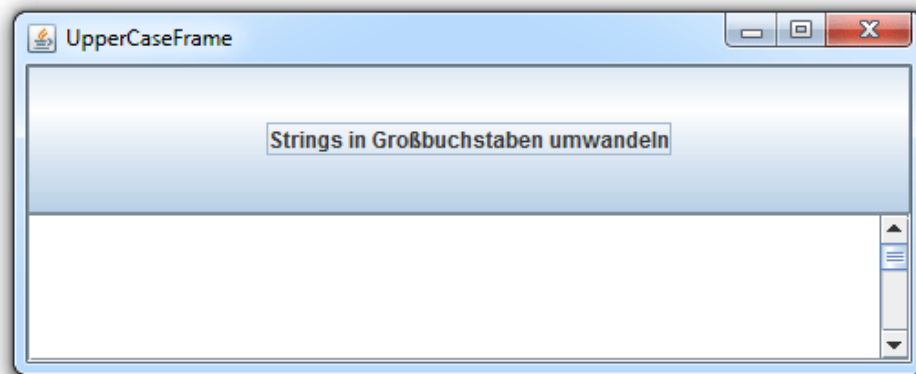
```
1 public String printUpperCase(String s)
2     throws NullPointerException, EmptyStringException {
3     if (s == null) {
4         throw new NullPointerException();
5     }
6     if (s.length() == 0) {
7         throw new EmptyStringException();
8     }
9
10    return s.toUpperCase();
11 }
```

Die Methode `printUpperCase` verwendet die folgenden Klassen `NullPointerException` und `EmptyStringException`:

```
1 public class NullPointerException extends Exception {
2 }
3
4 public class EmptyStringException extends Exception {
5 }
```

In dieser Aufgabe sollen Sie ein Programm mit einer grafischen Benutzeroberfläche implementieren, welches die Umwandlung eines Arrays von Strings testet. Die Anzahl der umzuwandelnden Strings `num` soll vom Benutzer frei wählbar sein und wie gewohnt durch einen modalen Dialog zu Beginn der Anwendung abgefragt werden.

- a) Die grafische Benutzeroberfläche soll wie folgt aussehen:



Es soll einen Button mit der oben angegebenen Aufschrift geben. Darunter soll der Ausgabebereich für die umgewandelten Strings bzw. für Meldungen über den Erfolg der Umwandlung platziert werden.

Da die Ausgabe länger sein kann, können Sie mit Hilfe der Klassen `ScrollPane` und `ScrollPaneConstants` der Swing-Bibliothek dem Ausgabebereich auf folgende Weise eine vertikale Scrollbar hinzufügen:

```
1 JScrollPane scrollPane = new JScrollPane(ausgabeBereich);
2 scrollPane.setHorizontalScrollBarPolicy(
3     ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
```

Schreiben Sie eine Klasse `UpperCaseFrame`, die die Hauptklasse dieser grafischen Benutzeroberfläche sein soll und das Fenster erzeugt. Um Ihr Programm ausführen zu können, schreiben Sie eine weitere Klasse `UpperCaseFrameMain`, die Sie wie gewohnt im gleichen Ordner wie Ihre Klasse `UpperCaseFrame` abspeichern.

### Lösung:

Bei der Implementierung der grafischen Benutzeroberfläche geht man wie folgt vor:

Deklarieren Sie eine Klasse `UpperCaseFrame`, die die Hauptklasse Ihrer grafischen Benutzeroberfläche wird und deshalb von der Klasse `JFrame` erbt. Ihre Klasse `UpperCaseFrame` soll zwei Attribute haben:

- ein Attribut vom Klassentyp  `JButton`  für den Button,
- ein Attribut vom Klassentyp  `JTextArea` , das als Ausgabebereich für die spätere Rückmeldung über das Ergebnis dient.

Schreiben Sie einen Konstruktor, der den `UpperCaseFrame` mit einem entsprechenden Titel und Größe (wir wählen hier 500x200 Pixel) initialisiert. In dem Konstruktor sollen weiterhin alle Attribute korrekt initialisiert werden. Das Layout des `ContentPane` des `UpperCaseFrames` wird auf ein `GridLayout` mit zwei Zeilen und einer Spalte initialisiert und der Button sowie der Ausgabebereich darauf platziert. Fügen Sie abschließend noch ein, dass das Programm ordnungsgemäß beendet wird, falls der `UpperCaseFrame` geschlossen wird. Benutzen Sie dazu die Methode `setDefaultCloseOperation`.

**Eine Implementierung der Klassen `UpperCaseFrame` und `UpperCaseFrameMain` für die gesamte Aufgabe finden Sie weiter unten.**

- b) Ergänzen Sie Ihre Klasse `UpperCaseFrame` um eine Ereignisbehandlung für den Button. Wird dieser Button gedrückt, soll der Benutzer zunächst in einer Methode `umwandelnStrings` mit Hilfe der Klasse `JOptionPane` nach der Anzahl `num` der umzuwandelnden Strings gefragt werden. Diese Methode soll sich auch darum kümmern, dass im Ausgabebereich Inhalte vorhergehender Eingaben gelöscht werden. Ein Test-Array `testArray` mit umzuwandelnden Strings soll in der Klasse als Konstante angelegt werden und den Wert `{null, "", "test"}` haben. Die Strings des Test-Arrays sollen durch Aufruf der Methode `printUpperCaseArray` umgewandelt werden. Diese Methode verwendet die vorhergenannte Methode `printUpperCase` und greift auf das Attribut `ausgabeBereich` für den Ausgabebereich in der grafischen Benutzeroberfläche zurück.

```
1 public void printUpperCaseForArray(String[] array, int num) {
2     for (int i = 0; i < num; i++) {
3         this.ausgabeBereich.append(this.printUpperCase(array[i]) + "\n");
4     }
5     this.ausgabeBereich.append("Fertig!\n");
6 }
```

Fügen Sie die Methoden in Ihre Implementierung der Klasse `UpperCaseFrame` ein und rufen Sie sie an geeigneter Stelle auf. Warum kommt es bei der gegenwärtigen Formulierung der

Methode zu einem Kompilierfehler? Fügen Sie in die Methode `printUpperCaseForArray` eine Ausnahmebehandlung ein, so dass das Programm fehlerfrei kompiliert wird und bei Ausführung mit Eingabe 3 folgender Text auf der Konsole erscheint:

```
Exception bei Versuch 1
Exception bei Versuch 2
TEST
Fertig!
```

### Lösung:

Bei der vorgegebenen Formulierung der beiden Methoden kommt es zu einem Kompilierfehler, weil es sich bei `NullPointerException` und `EmptyStringException` um *Checked Exceptions* handelt, die abgefangen werden müssen.

Implementierung der Methode `umwandelnStrings`:

```
1 private void umwandelnStrings() {
2     String einlesenAnzahlStrings =
3         JOptionPane.showInputDialog("Parameter \"num\" eingeben: ");
4     int num = Integer.parseInt(einlesenAnzahlStrings);
5
6     this.ausgabeBereich.setText("");
7     this.printUpperCaseForArray(testArray, num);
8 }
```

Einfügen eines `try`- und eines `catch`-Blocks in die Methode `printUpperCaseForArray`, sowie einer `append`-Anweisung für entsprechende Ausgaben:

```
1 public void printUpperCaseForArray(String[] array, int num) {
2     for (int i = 0; i < num; i++) {
3         try {
4             this.ausgabeBereich.append(this.printUpperCase(array[i]) + "\n");
5         } catch (Exception e) {
6             this.ausgabeBereich.append(
7                 "Exception bei Versuch " + (i + 1) + "\n");
8         }
9     }
10    this.ausgabeBereich.append("Fertig!\n");
11 }
```

Eine Implementierung der Klasse `UpperCaseFrame` für die gesamte Aufgabe finden Sie weiter unten. Die entsprechende Methode heißt hier `printUpperCaseForArrayB`.

- c) Ändern Sie jetzt den `catch`-Block in der Methode `printUpperCaseForArray` und fügen Sie gegebenenfalls weitere `catch`-Blöcke ein, so dass bei Ausführung des Programms folgender Text ausgegeben wird:

```
text[0] war null!
text[1] war leer!
TEST
Fertig!
```

### Lösung:

```
1 public void printUpperCaseForArray(String[] array, int num) {
2     for (int i = 0; i < num; i++) {
3         try {
4             this.ausgabeBereich.append(this.printUpperCase(array[i]) + "\n");
5         } catch (NullPointerException e) {
```

```

6         this.ausgabeBereich.append("text[" + i + "] war null!\n");
7     } catch (EmptyStringException e) {
8         this.ausgabeBereich.append("text[" + i + "] war leer!\n");
9     }
10 }
11 this.ausgabeBereich.append("Fertig!\n");
12 }

```

**Eine Implementierung der Klasse UpperCaseFrame für die gesamte Aufgabe finden Sie weiter unten. Die entsprechende Methode heißt hier printUpperCaseForArrayC.**

- d) Rufen Sie jetzt testweise das Programm mit dem Wert 4 für den Parameter `num` auf. Wenn das Programm jetzt ausgeführt wird, dann bricht es mit einer „ungefangenen“ Exception ab. Fügen Sie weitere `try`-, `catch`- oder `finally`-Blöcke ein, so dass dies nicht mehr passiert und stattdessen folgende Ausgabe erscheint:

```

text[0] war null!
Nach Versuch 1
text[1] war leer!
Nach Versuch 2
TEST
Nach Versuch 3
Fehler! Falscher Arrayzugriff!
Nach Versuch 4
Fertig!

```

Vermeiden Sie dabei möglichst Wiederholungen. Insbesondere soll nicht die Anweisung `append` mehrmals mit den gleichen übergebenen Parametern verwendet werden. Testen Sie Ihr Programm anschließend nochmals mit dem Wert 4 für den Parameter `num`.

#### **Lösung:**

```

1 public void printUpperCaseForArray(String[] array, int num) {
2     for (int i = 0; i < num; i++) {
3         try {
4             this.ausgabeBereich.append(this.printUpperCase(array[i]) + "\n");
5         } catch (NullPointerException e) {
6             this.ausgabeBereich.append("text[" + i + "] war null!\n");
7         } catch (EmptyStringException e) {
8             this.ausgabeBereich.append("text[" + i + "] war leer!\n");
9         } catch (ArrayIndexOutOfBoundsException e) {
10            this.ausgabeBereich.append("Fehler! Falscher Arrayzugriff!\n");
11        } finally {
12            this.ausgabeBereich.append("Nach Versuch " + (i + 1) + "\n");
13        }
14    }
15    this.ausgabeBereich.append("Fertig!\n");
16 }

```

**Eine Implementierung der Klasse UpperCaseFrame für die gesamte Aufgabe finden Sie weiter unten. Die entsprechende Methode heißt hier printUpperCaseForArrayD.**

- e) Wie könnte man die `RuntimeException` aus Teil d durch bessere Programmierung vermeiden?

#### **Lösung:**

Es muss in der Methode `umwandelnStrings` durch eine `if`-Abfrage sichergestellt werden,

dass beim Aufruf von `printUpperCaseForArray` vernünftige aktuelle Parameter verwendet werden.

```
1 private void umwandelnStrings() {
2     String einlesenAnzahlStrings =
3         JOptionPane.showInputDialog("Parameter \"num\" eingeben: ");
4     int num = Integer.parseInt(einlesenAnzahlStrings);
5
6     if ((num >= 0) && (num <= testArray.length)) {
7         this.ausgabeBereich.setText("");
8         this.printUpperCaseForArray(testArray, num);
9     } else {
10        this.ausgabeBereich.setText("Ungültige Anzahl " + num);
11    }
12 }
```

Eine Implementierung der Klassen `UpperCaseFrameMain` und `UpperCaseFrame` für die gesamte Aufgabe finden Sie auch im ZIP-Archiv. Die entsprechende Methode heißt hier `umwandelnStringsE`.

```
1 /**
2  * Diese Klasse startet den UpperCaseFrame
3  *
4  * @author Annabelle Klarl
5  */
6 public class UpperCaseFrameMain {
7     /**
8      * Dieses Programmstueck startet das Programm.
9      *
10     * @param args
11     */
12     public static void main(String[] args) {
13         UpperCaseFrame uppercaseFrame = new UpperCaseFrame();
14         uppercaseFrame.setVisible(true);
15     }
16
17 }
```

```
1 import java.awt.Container;
2 import java.awt.GridLayout;
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
5
6 import javax.swing.JButton;
7 import javax.swing.JFrame;
8 import javax.swing.JOptionPane;
9 import javax.swing.JScrollPane;
10 import javax.swing.JTextArea;
11 import javax.swing.ScrollPaneConstants;
12
13 /**
14  * Diese Klasse repraesentiert das Hauptfenster der Praesenzaufgabe zum Umwandeln
15  * von Strings in Grossbuchstaben.
16  *
17  * @author Annabelle Klarl
18  */
19
20 public class UpperCaseFrame extends JFrame implements ActionListener {
21     private JButton anzahlstringsButton;
22
23     private JTextArea ausgabeBereich;
24
25     public static final String[] TESTARRAY = { null, "", "test" };
26 }
```

```

27  /**
28   * In diesem Programmstueck wird das Fenster erzeugt
29   */
30  public UpperCaseFrame() {
31      /* In der Kopfleiste des Fenster steht "UpperCaseFrame" */
32      this.setTitle("UpperCaseFrame");
33
34      /* Das Fenster ist 500 Pixel breit und 200 Pixel hoch. */
35      this.setSize(500, 200);
36
37      /* Hier werden alle Buttons erzeugt. */
38      this.anzahlstringsButton = new JButton(
39          "Strings in Grossbuchstaben umwandeln");
40
41      /* Hier wird der Ausgabe-Bereich erzeugt. */
42      this.ausgabeBereich = new JTextArea(60, 100);
43      JScrollPane scrollPane = new JScrollPane(this.ausgabeBereich);
44      scrollPane.setHorizontalScrollBarPolicy(
45          JScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
46
47      /*
48       * Der ContentPane ist der Ausschnitt des Fensters, auf dem Widgets d.h.
49       * Interaktionselemente (wie eine TextArea oder ein Button) platziert
50       * werden koennen.
51       */
52      Container contentPane = this.getContentPane();
53      contentPane.setLayout(new GridLayout(2, 1));
54      /* Hier wird der Button platziert. */
55      contentPane.add(this.anzahlstringsButton);
56      /* Hier wird der Ausgabebereich platziert. */
57      contentPane.add(scrollPane);
58
59      /*
60       * Hier wird der Frame als Listener fuer Knopfdruck-Ereignisse bei jedem
61       * der Buttons registriert.
62       */
63      this.anzahlstringsButton.addActionListener(this);
64
65      /*
66       * Wird das Fenster geschlossen (d.h. auf X gedruickt), wird mit Hilfe
67       * dieser Programmzeile auch unser Programm ordnungsgemaess beendet.
68       */
69      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
70  }
71
72  @Override
73  public void actionPerformed(ActionEvent e) {
74      // je nach Button entsprechende Methode ausfn
75      Object source = e.getSource();
76      if (source == this.anzahlstringsButton) {
77          this.umwandelnStringsE();
78      }
79  }
80
81  /**
82   * implementiert die Funktionalitaet des
83   * "Strings in Grossbuchstaben umwandeln"-Buttons
84   */
85  private void umwandelnStrings() {
86      String einlesenAnzahlStrings =
87          JOptionPane.showInputDialog("Parameter \"num\" eingeben: ");
88      int num = Integer.parseInt(einlesenAnzahlStrings);
89
90      this.ausgabeBereich.setText("");
91      this.printUpperCaseForArray(TESTARRAY, num);

```

```

92     }
93
94     /**
95      * implementiert die Funktionalitaet des
96      * "Strings in Grossbuchstaben umwandeln"-Buttons
97      */
98     private void umwandelnStringsE() {
99         String einlesenAnzahlStrings =
100             JOptionPane.showInputDialog("Parameter \"num\" eingeben: ");
101         int num = Integer.parseInt(einlesenAnzahlStrings);
102
103         if ((num >= 0) && (num <= TESTARRAY.length)) {
104             this.ausgabeBereich.setText("");
105             this.printUpperCaseForArray(TESTARRAY, num);
106         } else {
107             this.ausgabeBereich.setText("Unguelteige Anzahl " + num);
108         }
109     }
110
111     /**
112      * wandelt einen String in Grossbuchstaben um
113      *
114      * @param s
115      *
116      * @return
117      */
118     public String printUpperCase(String s)
119         throws NullPointerException, EmptyStringException {
120         if (s == null) {
121             throw new NullPointerException();
122         }
123         if (s.length() == 0) {
124             throw new EmptyStringException();
125         }
126
127         return s.toUpperCase();
128     }
129
130     public void printUpperCaseForArray(String[] array, int num) {
131         printUpperCaseForArrayD(array, num);
132     }
133
134     /**
135      * wandelt ein Array von Strings in Grossbuchstaben um
136      *
137      * @param array
138      * @param num
139      *
140      * @return
141      */
142     public void printUpperCaseForArrayB(String[] array, int num) {
143         for (int i = 0; i < num; i++) {
144             try {
145                 this.ausgabeBereich.append(
146                     this.printUpperCase(array[i]) + "\n");
147             } catch (Exception e) {
148                 this.ausgabeBereich.append(
149                     "Exception bei Versuch " + (i + 1) + "\n");
150             }
151         }
152         this.ausgabeBereich.append("Fertig!\n");
153     }
154
155     /**
156      * wandelt ein Array von Strings in Grossbuchstaben um

```

```

157      *
158      * @param array
159      * @param num
160      *
161      * @return
162      */
163      public void printUpperCaseForArrayC(String[] array, int num) {
164          for (int i = 0; i < num; i++) {
165              try {
166                  this.ausgabeBereich.append(
167                      this.printUpperCase(array[i]) + "\n");
168              } catch (NullPointerException e) {
169                  this.ausgabeBereich.append("text[" + i + "] war null!\n");
170              } catch (EmptyStringException e) {
171                  this.ausgabeBereich.append("text[" + i + "] war leer!\n");
172              }
173          }
174          this.ausgabeBereich.append("Fertig!\n");
175      }
176
177      /**
178       * wandelt ein Array von Strings in Grossbuchstaben um
179       *
180       * @param array
181       * @param num
182       *
183       * @return
184       */
185      public void printUpperCaseForArrayD(String[] array, int num) {
186          for (int i = 0; i < num; i++) {
187              try {
188                  this.ausgabeBereich.append(
189                      this.printUpperCase(array[i]) + "\n");
190              } catch (NullPointerException e) {
191                  this.ausgabeBereich.append("text[" + i + "] war null!\n");
192              } catch (EmptyStringException e) {
193                  this.ausgabeBereich.append("text[" + i + "] war leer!\n");
194              } catch (ArrayIndexOutOfBoundsException e) {
195                  this.ausgabeBereich.append("Fehler! Falscher Arrayzugriff!\n");
196              } finally {
197                  this.ausgabeBereich.append("Nach Versuch " + (i + 1) + "\n");
198              }
199          }
200          this.ausgabeBereich.append("Fertig!\n");
201      }
202  }

```

## Aufgabe 12-2

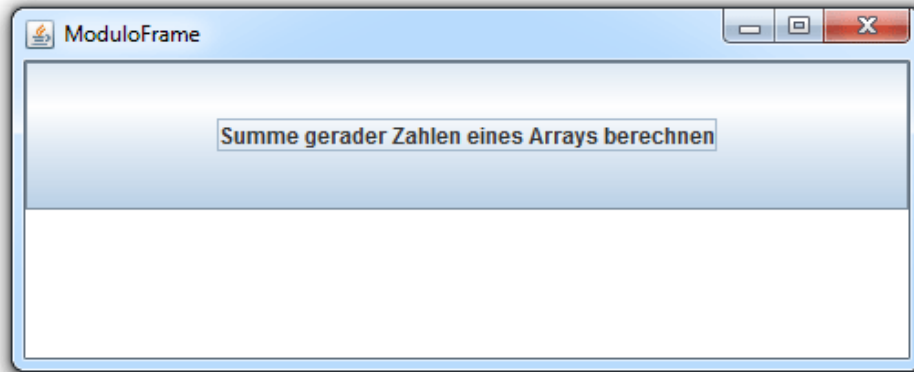
## Ausnahmen und Ausnahmebehandlung

## Hausaufgabe

Der Modulo-Operator % kann in Java dazu verwendet werden, den Rest einer ganzzahligen Division zu berechnen. Es gilt also z.B.  $4 \% 2 == 0$  und  $5 \% 2 == 1$ .

In dieser Aufgabe sollen Sie ein Programm mit einer grafischen Benutzeroberfläche implementieren, welches die Summe aller geraden Zahlen (Zahlen, die ohne Rest durch 2 teilbar sind) in einem `int`-Array ermittelt. Das `int`-Array soll durch einen modalen Dialog zu Beginn der Anwendung abgefragt und vom Benutzer eingegeben werden.

- a) Die grafische Benutzeroberfläche soll wie folgt aussehen:



Es soll einen Button mit der oben angegebenen Aufschrift geben. Darunter soll der Ausgabebereich für die berechnete Summe der geraden Zahlen platziert werden.

Schreiben Sie eine Klasse `ModuloFrame`, die die Hauptklasse dieser grafischen Benutzeroberfläche sein soll und das Fenster erzeugt. Um Ihr Programm ausführen zu können, schreiben Sie eine weitere Klasse `ModuloFrameMain`, die Sie wie gewohnt im gleichen Ordner wie Ihre Klasse `ModuloFrame` abspeichern.

### Lösung:

Bei der Implementierung der grafischen Benutzeroberfläche geht man wie folgt vor:

Deklarieren Sie eine Klasse `ModuloFrame`, die die Hauptklasse Ihrer grafischen Benutzeroberfläche wird und deshalb von der Klasse `JFrame` erbt. Ihre Klasse `ModuloFrame` soll zwei Attribute haben:

- ein Attribut vom Klassentyp `JButton` für den Button,
- ein Attribut vom Klassentyp `JTextArea`, das als Ausgabebereich für die spätere Rückmeldung über das Ergebnis dient.

Schreiben Sie einen Konstruktor, der den `ModuloFrame` mit einem entsprechenden Titel und Größe (wir wählen hier 500x200 Pixel) initialisiert. In dem Konstruktor sollen weiterhin alle Attribute korrekt initialisiert werden. Das Layout des `ContentPane` des `ModuloFrames` wird auf ein `GridLayout` mit zwei Zeilen und einer Spalte initialisiert und der Button sowie der Ausgabebereich darauf platziert. Fügen Sie abschließend noch ein, dass das Programm ordnungsgemäß beendet wird, falls der `ModuloFrame` geschlossen wird. Benutzen Sie dazu die Methode `setDefaultCloseOperation`.

**Eine Implementierung der Klassen `ModuloFrame` und `ModuloFrameMain` für die gesamte Aufgabe finden Sie weiter unten.**

- b) Deklarieren Sie nun eine Methode `summeGeraderZahlen` innerhalb der Klasse `ModuloFrame`, welche die Summe aller geraden Zahlen in einem `int`-Array ermittelt. Das Array soll dabei als Parameter der Methode übergeben werden und die berechnete Summe soll den Rückgabewert der Methode bilden. Falls in der Reihung eine negative Zahl vorkommt, soll eine benutzerdefinierte und gecheckte Ausnahme `NegativeElementException` ausgelöst und die Ausführung der Methode `summeGeraderZahlen` dabei abgebrochen werden. Berücksichtigen Sie dies auch bei der Deklaration des Methodenkopfs dieser Methode. Um eine Ausnahme `NegativeElementException` verwenden zu können, müssen Sie zudem erst eine Klasse `NegativeElementException` deklarieren, die eine *Checked Exception* realisiert. Eine Fehlermeldung muss nicht in das Fehlerobjekt integriert werden.

**Lösung:**

Deklaration der Klasse `NegativeElementException`:

```
1 public class NegativeElementException extends Exception {  
2 }
```

Deklaration der Methode `summeGeraderZahlen`:

```
1 public int summeGeraderZahlen(int[] arr)  
2     throws NegativeElementException {  
3     int sum = 0;  
4  
5     for (int i = 0; i < arr.length; i++) {  
6         if (arr[i] < 0) {  
7             throw new NegativeElementException();  
8         }  
9         if (arr[i] % 2 == 0) {  
10            sum = sum + arr[i];  
11        }  
12    }  
13    return sum;  
14 }
```

Eine Implementierung der Klasse `ModuloFrame` für die gesamte Aufgabe finden Sie weiter unten.

- c) Ergänzen Sie die Klasse `ModuloFrame` nun um eine Ereignisbehandlung für den Button. Wird dieser Button gedrückt, soll der Benutzer zunächst in einer Methode `summeBerechnen` mit Hilfe der Klasse `JOptionPane` nach dem `int`-Array gefragt werden, das als Eingabe für die Berechnung dient. Die Methode `summeBerechnen` muss sich daher ebenfalls um die Konvertierung des eingelesenen Strings in ein `int`-Array kümmern. Im ZIP-Archiv finden Sie die Klasse `Konverter.java`, mit der Sie in der Methode `konvertiereZuIntArray` einen Komma-separierten String in ein `int`-Array konvertieren können (d.h. die Eingabe `1,2,3` wird konvertiert in ein Array `[1,2,3]`). Aus dem durch die Konvertierung erhaltenen `int`-Array soll mithilfe der statischen Methode `summeGeraderZahlen` aus Teilaufgabe b) die Summe der geraden Zahlen berechnet und diese anschließend im Ausgabebereich des `ModuloFrames` ausgegeben werden. Fügen Sie `try`-, und `catch`- Blöcke in die Methode `summeBerechnen` ein, so dass, falls die Methode `summeGeraderZahlen` eine `NegativeElementException` auslöst, die Meldung „Fehler!“ ausgegeben wird.

**Lösung:**

Deklaration der Methode `summeBerechnen`:

```
1 public void summeBerechnen() {  
2     String einlesenArray =  
3         JOptionPane.showInputDialog("Array eingeben: ");  
4     int[] arr = Konverter.konvertiereZuIntArray(einlesenArray);  
5  
6     int sum = 0;  
7     try {  
8         sum = summeGeraderZahlen(arr);  
9         this.ausgabeBereich.setText("Summe = " + sum);  
10    } catch (NegativeElementException e) {  
11        this.ausgabeBereich.setText("Fehler!");  
12    }  
13 }
```

Eine Implementierung der Klassen `ModuloFrameMain` und `ModuloFrame` finden Sie auch im ZIP-Archiv.

```

1  /**
2   * Diese Klasse startet den ModuloFrame
3   *
4   * @author Annabelle Klarl
5   */
6  public class ModuloFrameMain {
7      /**
8       * Dieses Programmstueck startet das Programm.
9       *
10      * @param args
11      */
12     public static void main(String[] args) {
13         ModuloFrame moduloFrame = new ModuloFrame();
14         moduloFrame.setVisible(true);
15     }
16
17 }

```

```

1  import java.awt.Container;
2  import java.awt.GridLayout;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5
6  import javax.swing.JButton;
7  import javax.swing.JFrame;
8  import javax.swing.JOptionPane;
9  import javax.swing.JTextArea;
10
11 /**
12  * Diese Klasse repraesentiert das Hauptfenster der Hausaufgabe zur Berechnung
13  * der Summe aller geraden Zahlen mittels Modulo.
14  *
15  * @author Annabelle Klarl
16  *
17  */
18 public class ModuloFrame extends JFrame implements ActionListener {
19     private JButton geradezahlenButton;
20
21     private JTextArea ausgabeBereich;
22
23     /**
24      * In diesem Programmstueck wird das Fenster erzeugt
25      */
26     public ModuloFrame() {
27         /* In der Kopfleiste des Fensters steht "ModuloFrame" */
28         this.setTitle("ModuloFrame");
29
30         /* Das Fenster ist 500 Pixel breit und 200 Pixel hoch. */
31         this.setSize(500, 200);
32
33         /* Hier werden alle Buttons erzeugt. */
34         this.geradezahlenButton = new JButton(
35             "Summe gerader Zahlen eines Arrays berechnen");
36
37         /* Hier wird der Ausgabe-Bereich erzeugt. */
38         this.ausgabeBereich = new JTextArea(10, 100);
39
40         /*
41          * Der ContentPane ist der Ausschnitt des Fensters, auf dem Widgets d.h.
42          * Interaktionselemente (wie eine TextArea oder ein Button) platziert
43          * werden koennen.
44          */
45         Container contentPane = this.getContentPane();
46         contentPane.setLayout(new GridLayout(2, 1));

```

```

47      /* Hier wird der Button platziert. */
48      contentPane.add(this.geradezahlenButton);
49      /* Hier wird der Ausgabebereich platziert. */
50      contentPane.add(this.ausgabeBereich);
51
52      /*
53       * Hier wird der ModuloFrame als Listener fuer Knopfdruck-Ereignisse bei
54       * jedem der Buttons registriert.
55       */
56      this.geradezahlenButton.addActionListener(this);
57
58      /*
59       * Wird das Fenster geschlossen (d.h. auf X gedrueckt), wird mit Hilfe
60       * dieser Programmzeile auch unser Programm ordnungsgemaess beendet.
61       */
62      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
63  }
64
65  @Override
66  public void actionPerformed(ActionEvent e) {
67      // je nach Button entsprechende Methode ausfuehren
68      Object source = e.getSource();
69      if (source == this.geradezahlenButton) {
70          this.summeBerechnen();
71      }
72  }
73
74  /**
75   * implementiert die Funktionalitaet des
76   * "Summe gerader Zahlen berechnen"-Buttons
77    */
78  public void summeBerechnen() {
79      String einlesenArray = JOptionPane.showInputDialog("Array eingeben: ");
80      int[] arr = Konverter.konvertiereZuIntArray(einlesenArray);
81
82      int sum = 0;
83      try {
84          sum = this.summeGeraderZahlen(arr);
85          this.ausgabeBereich.setText("Summe = " + sum);
86      } catch (NegativeElementException e) {
87          this.ausgabeBereich.setText("Fehler!");
88      }
89  }
90
91
92  /**
93   * berechnet die Summe aller geraden Zahlen
94   *
95   * @param arr
96   *
97   * @return Summe der geraden Zahlen
98    */
99  public int summeGeraderZahlen(int[] arr) throws NegativeElementException {
100      int sum = 0;
101
102      for (int i = 0; i < arr.length; i++) {
103          if (arr[i] < 0) {
104              throw new NegativeElementException();
105          }
106          if (arr[i] % 2 == 0) {
107              sum = sum + arr[i];
108          }
109      }
110      return sum;
111  }

```

**Aufgabe 12-3****Klausurvorbereitung 6 und 9 ECTS****Hausaufgabe**

Überlegen und formulieren Sie Fragen zu Themen aus der Vorlesung, bei denen Sie noch Probleme haben und senden Sie diese per Mail an Philipp Wendler ([philipp.wendler@lmu.de](mailto:philipp.wendler@lmu.de)). Die am meisten gestellten Fragen werden in der Zentralübung am 07.02.2018 behandelt.

*Besprechung der Präsenzaufgaben in den Übungen vom 26.01.2018 und 29.01.2017. Abgabe der Hausaufgaben bis Mittwoch, 31.01.2018, 14:00 Uhr über UniworX (siehe Folien der ersten Zentralübung). Erstellen Sie zu jeder Aufgabe Klassen, die die Namen tragen, die in der Aufgabe gefordert sind. Geben Sie nur die entsprechenden **.java**-Dateien ab. Wir benötigen **nicht** Ihre **.class**-Dateien.*