

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

Lösungsvorschlag

Aufgabe 13-1

Ballonwettbewerb

Präsenz

In einer Stadt werden mehrere tausend Ballons mit Absenderpostkarten losgelassen. Bis zu einem gewissen Stichtag kommt ein Teil der Postkarten mit Angabe des Fundortes zurück. Sieger ist, wessen Ballon am weitesten geflogen ist (in km), wobei bei gleicher Entfernung mehrere Sieger möglich sind.

- a) Schreiben Sie eine Klasse `Postkarte`, die eine Absenderpostkarte wie oben beschrieben repräsentiert. Die Klasse soll Attribute für den Namen des Teilnehmers und die zurückgelegte Entfernung (als Ganzzahl vom Typ `int`) haben. Fügen Sie einen Konstruktor zur Initialisierung der Attribute sowie "Getter" für beide Attribute hinzu.

Lösung:

Eine Implementierung der Klasse `Postkarte` finden Sie auch im ZIP-Archiv.

```
1 /**
2  * Diese Klasse repräsentiert eine Postkarte bestehend aus dem Teilnehmer,
3  * der diese Postkarte anfaenglich losgeschickt hat, und der Entfernung in km
4  * (wobei hier nur ganzzahlige Entfernungen erlaubt sind),
5  * die diese Postkarte zurueckgelegt hat.
6  *
7  * @author Annabelle Klarl
8  */
9 public class Postkarte {
10
11     private String teilnehmer;
12     private int entfernung;
13
14     /**
15      * Konstruktor
16      *
17      * @param teilnehmer
18      * @param entfernung zurueckgelegte Entfernung in km als Ganzzahl
19      */
20     public Postkarte(String teilnehmer, int entfernung) {
21         this.teilnehmer = teilnehmer;
22         this.entfernung = entfernung;
23     }
24
25     /**
26      * Diese Methode gibt den Namen des Teilnehmers zurueck,
27      * der diese Postkarte anfaenglich abgeschickt hat.
28      *
29      * @return der Name des Teilnehmers
30      */
31     public String getTeilnehmer() {
32         return this.teilnehmer;
33     }
34
35     /**
```

```

36     * Diese Methode gibt die Entfernung zurueck,
37     * die die Postkarte zurueckgelegt hat.
38     *
39     * @return die zurueckgelegte Entfernung
40     */
41     public int getEntfernung() {
42         return this.entfernung;
43     }
44 }

```

- b) Implementieren Sie nun eine verkettete Liste zur Speicherung von Postkarten nach dem Beispiel in der Vorlesung. Nennen Sie die Klasse, die Ihre verkettete Liste darstellt, `MeinePostkartenListe` und die Klasse für Elemente in dieser Liste `MeinPostkartenListenelement`.

Lösung:

Eine Implementierung der Klassen `MeinePostkartenListe` und `MeinPostkartenListenelement` finden Sie auch im ZIP-Archiv.

```

1  /**
2   * Diese Klasse implementiert eine verkettete Liste, die Postkarten speichert.
3   * Dazu werden als Elemente Objekte vom Typ
4   * {@link MeinPostkartenListenelement} benutzt.
5   *
6   * @author Annabelle Klarl
7   */
8  public class MeinePostkartenListe {
9
10     private MeinPostkartenListenelement first;
11
12     /** Konstruktor: erstellt eine leere Liste */
13     public MeinePostkartenListe() {
14         this.first = null;
15     }
16
17     /**
18      * Fuegt die gegebene Postkarte an erster Stelle in die Liste ein.
19      * Alle bisherigen Listen-Elemente werden um eins nach hinten gerueckt.
20      *
21      * @param postkarte
22      */
23     public void addFirst(Postkarte postkarte) {
24         this.first = new MeinPostkartenListenelement(postkarte, this.first);
25     }
26
27     /**
28      * Bestimmt die Anzahl der Elemente in der Liste
29      *
30      * @return die Anzahl der Elemente in der Liste
31      */
32     public int size() {
33         int size = 0;
34         MeinPostkartenListenelement element = this.first;
35         while (element != null) {
36             element = element.getNext();
37             size++;
38         }
39         return size;
40     }
41
42     /**
43      * Gibt das Element an Position i in der Liste zurueck
44      * (ohne es aus der Liste zu loeschen!).

```

```

45  *
46  * @param i
47  * @return das Listen-Element an Position i
48  */
49  public Postkarte get(int i) {
50      MeinPostkartenListenelement element = this.first;
51      while (element != null && i > 0) {
52          element = element.getNext();
53          i--;
54      }
55      if (element == null) {
56          throw new IndexOutOfBoundsException();
57      }
58      return element.getPostkarte();
59  }
60  }

```

```

1  /**
2   * Diese Klasse repraesentiert ein Element in einer
3   * {@link MeinePostkartenListe}.
4   * Sie verweist auf die gespeicherte Postkarte fuer dieses Element
5   * und verweist ebenso auf das naechste Element vom Typ
6   * {@link MeinPostkartenListenelement} in der Liste.
7   *
8   * @author Annabelle Klarl
9   */
10 public class MeinPostkartenListenelement {
11
12     private Postkarte postkarte;
13     private MeinPostkartenListenelement next;
14
15     /**
16      * Konstruktor: erstellt ein neues Element mit einem Verweis
17      * auf die entsprechende Postkarte
18      *
19      * @param postkarte
20      */
21     public MeinPostkartenListenelement(Postkarte postkarte) {
22         this.postkarte = postkarte;
23         this.next = null;
24     }
25
26     /**
27      * Konstruktor: erstellt ein neues Element mit einem Verweis
28      * auf die entsprechende Postkarte und
29      * erstellt einen Verweis auf das naechste Listen-Element.
30      *
31      * @param postkarte
32      * @param next
33      */
34     public MeinPostkartenListenelement(
35         Postkarte postkarte, MeinPostkartenListenelement next) {
36         this.postkarte = postkarte;
37         this.next = next;
38     }
39
40     /**
41      * Diese Methode gibt die zu diesem Listen-Element gehoerende Postkarte
42      * zurueck.
43      *
44      * @return
45      */
46     public Postkarte getPostkarte() {
47         return this.postkarte;
48     }

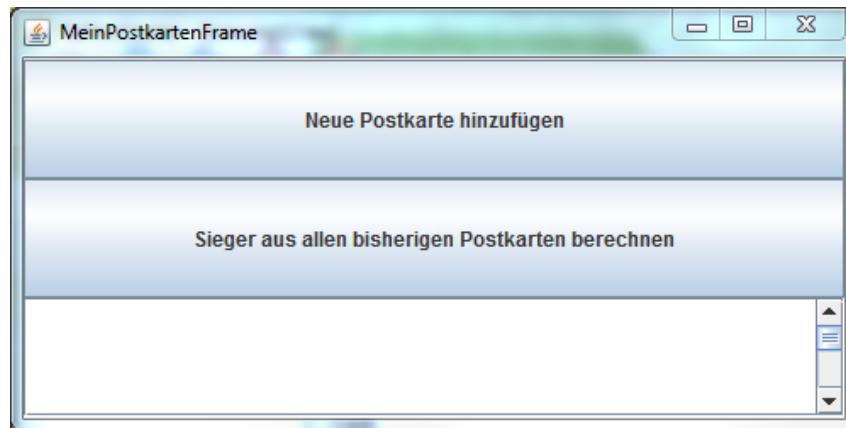
```

```

49
50  /**
51   * Diese Methode gibt das naechste Listen-Element zurueck.
52   *
53   * @return
54   */
55  public MeinPostkartenListenelement getNext() {
56      return this.next;
57  }
58 }

```

- c) Die grafische Benutzeroberfläche für die Verwaltung des Ballonwettbewerbs soll wie folgt aussehen:



Es soll zwei Buttons mit der oben genannten Aufschrift geben. Darunter soll ein Ausgabebereich platziert werden.

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

Erweitern Sie Ihre Klasse `MeinPostkartenFrame` um eine Ereignisbehandlung für die beiden Buttons. Wird der Button “Neue Postkarte hinzufügen” gedrückt, soll der Benutzer zunächst mit Hilfe der Klasse `JOptionPane` nach dem Namen des Teilnehmers und der zurückgelegten Entfernung gefragt werden. Anschließend soll aus diesen Angaben ein Objekt vom Typ `Postkarte` erstellt werden und in einer Liste vom Typ `MeinePostkartenListe` gespeichert werden. Darüber soll der Benutzer im Ausgabebereich informiert werden.

Wird der Button “Sieger aus allen bisherigen Postkarten berechnen” gedrückt, sollen in der Liste aller bisher eingegebenen Postkarten alle Postkarten gesucht werden, die die weiteste Entfernung erreicht haben. Der Benutzer soll anschließend darüber informiert werden, welche Postkarten die weiteste Entfernung erreicht haben. Beachten Sie, dass dies auch mehrere Postkarten sein können.

Hinweis: In Ihrer Klasse `MeinPostkartenFrame` werden Sie ein Attribut brauchen, um die Liste von Postkarten speichern zu können (Modell der GUI).

Lösung:

Bei der Implementierung der GUI geht man wie gewohnt vor. Da die Ausgabe der siegreichen Postkarten allerdings relativ lang 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(this.ausgabeBereich);
2 scrollPane.setHorizontalScrollBarPolicy(
3     ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

```

Vergessen Sie außerdem nicht Ihrer Klasse `MeinPostkartenFrame` ein Attribut vom Typ `MeinePostkartenListe` hinzuzufügen, das alle eingetragenen Postkarten speichern soll und bei der Benutzung des Buttons “Neue Postkarte hinzufügen” um ein Element erweitert wird.

Um alle Postkarten mit der größten Entfernung zu finden, geht man wie folgt vor:

1. Lege eine neue Liste zur Speicherung aller Sieger-Postkarten an.
2. Deklariere eine lokale Variable `groessteEntfernung`, die die maximale Entfernung einer Postkarte speichern soll, und initialisiere sie mit -1.
3. Durchlaufe die Liste von gespeicherten Postkarten und wiederhole für jedes Element:
 - Falls die Postkarte weiter geflogen ist als die aktuell größte Entfernung, lösche die bisherige Siegerliste. Füge die aktuelle Postkarte in die nun leere Siegerliste ein und speichere die Entfernung dieser Postkarte als nun aktuell größte Entfernung in der lokalen Variablen `groessteEntfernung`.
 - Falls die Postkarte genauso weit geflogen ist wie die aktuell größte Entfernung, füge die aktuelle Postkarte in die Siegerliste ein.
4. Gebe die Siegerliste aus.

Lösung:

Eine Implementierung der Klassen `MeinPostkartenFrameMain` und `MeinPostkartenFrame` finden Sie auch im ZIP-Archiv.

```

1 /**
2  * Diese Klasse startet den MeinPostkartenFrame
3  *
4  * @author Annabelle Klarl
5  */
6 public class MeinPostkartenFrameMain {
7
8     /**
9      * Dieses Programmstueck startet das Programm.
10     *
11     * @param args
12     */
13     public static void main(String[] args) {
14         MeinPostkartenFrame frame = new MeinPostkartenFrame();
15         frame.setVisible(true);
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 import javax.swing.JButton;
6 import javax.swing.JFrame;
7 import javax.swing.JOptionPane;
8 import javax.swing.JScrollPane;
9 import javax.swing.JTextArea;
10 import javax.swing.ScrollPaneConstants;
11
12 /**
13  * Diese Klasse repraesentiert das Hauptfenster der Postkarten-Praesenzaufgabe,
14  * wobei eine eigene verkettete Liste verwendet wird.
15  *
16  * @author Annabelle Klarl

```

```

17  */
18  public class MeinPostkartenFrame extends JFrame implements ActionListener {
19      private JButton postkarteHinzufuegenButton;
20      private JButton siegerBerechnenButton;
21
22      private JTextArea ausgabeBereich;
23
24      private MeinePostkartenListe postkartenListe;
25
26      /** In diesem Programmstueck wird das Fenster erzeugt */
27      public MeinPostkartenFrame() {
28          /* Verbindung zum Modell der GUI */
29          this.postkartenListe = new MeinePostkartenListe();
30
31          /* In der Kopfleiste des Fenster steht "MeinPostkartenFrame" */
32          this.setTitle("MeinPostkartenFrame");
33
34          /* Das Fenster ist 500 Pixel breit und 250 Pixel hoch. */
35          this.setSize(500, 250);
36
37          /* Hier werden alle Buttons erzeugt. */
38          this.postkarteHinzufuegenButton =
39              new JButton("Neue Postkarte hinzufuegen");
40          this.siegerBerechnenButton =
41              new JButton("Sieger aus allen bisherigen Postkarten berechnen");
42
43          /* Hier wird der Ausgabe-Bereich erzeugt. */
44          this.ausgabeBereich = new JTextArea(10, 100);
45          JScrollPane scrollPane = new JScrollPane(this.ausgabeBereich);
46          scrollPane.setHorizontalScrollBarPolicy(
47              ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
48
49          /*
50           * Der ContentPane ist der Ausschnitt des Fensters, auf dem Widgets d.h.
51           * Interaktionselemente (wie eine TextArea oder ein Button) platziert
52           * werden koennen.
53           */
54          Container contentPane = this.getContentPane();
55          contentPane.setLayout(new GridLayout(3, 1));
56          /* Hier wird der Button platziert. */
57          contentPane.add(this.postkarteHinzufuegenButton);
58          contentPane.add(this.siegerBerechnenButton);
59          /* Hier wird der Ausgabebereich platziert. */
60          contentPane.add(scrollPane);
61
62          /*
63           * Hier wird der Frame als Listener fuer Knopfdruck-Ereignisse bei
64           * jedem der Buttons registriert.
65           */
66          this.postkarteHinzufuegenButton.addActionListener(this);
67          this.siegerBerechnenButton.addActionListener(this);
68
69          /*
70           * Wird das Fenster geschlossen (d.h. auf X gedrueckt), wird mit Hilfe
71           * dieser Programmzeile auch unser Programm ordnungsgemaess beendet.
72           */
73          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
74      }
75
76      @Override
77      public void actionPerformed(ActionEvent e) {
78          // je nach Button entsprechende Methode ausfuehren
79          Object source = e.getSource();
80          if (source == this.postkarteHinzufuegenButton) {
81              this.postkarteHinzufuegen();

```

```

82     } else if (source == this.siegerBerechnenButton) {
83         this.siegerBerechnen();
84     }
85 }
86
87 /** implementiert die Funktionalitaet des "Postkarte hinzufuegen"-Buttons */
88 private void postkarteHinzufuegen() {
89     String name = JOptionPane.showInputDialog("Name des Teilnehmers: ");
90
91     String einlesenEntfernung =
92         JOptionPane.showInputDialog("Zurueckgelegte Entfernung in km: ");
93     int entfernung = Integer.parseInt(einlesenEntfernung);
94
95     Postkarte neuePostkarte = new Postkarte(name, entfernung);
96     this.postkartenListe.addFirst(neuePostkarte);
97
98     this.ausgabeBereich.setText(
99         "Die Postkarte des Teilnehmers "
100         + name
101         + " mit einer Entfernung von "
102         + entfernung
103         + " km wurde hinzugefuegt.");
104 }
105
106 /** implementiert die Funktionalitaet des "Sieger berechnen"-Buttons */
107 private void siegerBerechnen() {
108     int groessteEntfernung = -1;
109     MeinePostkartenListe siegerListe = new MeinePostkartenListe();
110
111     for (int i = 0; i < this.postkartenListe.size(); i++) {
112         Postkarte postkarte = this.postkartenListe.get(i);
113         int entfernung = postkarte.getEntfernung();
114
115         if (entfernung > groessteEntfernung) {
116             groessteEntfernung = entfernung;
117             siegerListe = new MeinePostkartenListe();
118             siegerListe.addFirst(postkarte);
119         } else if (entfernung == groessteEntfernung) {
120             siegerListe.addFirst(postkarte);
121         }
122         // else: diese Postkarte ist nicht weiter gereist
123         // als die bisherigen Siegerpostkarte
124     }
125
126     this.ausgabeBereich.setText("Die Sieger sind: \n");
127     for (int i = 0; i < siegerListe.size(); i++) {
128         Postkarte postkarte = siegerListe.get(i);
129         this.ausgabeBereich.append(
130             postkarte.getTeilnehmer()
131             + " mit einer Entfernung von "
132             + postkarte.getEntfernung()
133             + " km");
134         this.ausgabeBereich.append("\n");
135     }
136 }
137 }

```

- d) Zur Lösung der Aufgabe sollen nun die generischen Klassen `LinkedList<E>` und `Iterator<E>` der Java-Bibliothek verwendet werden. Nehmen Sie die Benutzeroberfläche aus Aufgabe c) zur Grundlage und schreiben Sie eine neue Klasse `PostkartenFrame`. Das Aussehen der Benutzeroberfläche sowie die Funktionsweise der Buttons soll nicht verändert werden, allerdings soll anstelle eine Liste vom Typ `MeinePostkartenListe` nun eine Liste vom Typ `LinkedList<Postkarte>` verwendet werden, um alle Postkarten zu speichern. Eben-

so soll zum Durchlaufen einer Liste stets ein Objekt der Klasse `Iterator<Postkarte>` benutzt werden. Um Ihr Programm ausführen zu können, schreiben Sie eine weitere Klasse `PostkartenFrameMain`, die Sie wie gewohnt im gleichen Ordner wie Ihre Klasse `PostkartenFrame` abspeichern.

Lösung:

Eine Implementierung der Klassen `PostkartenFrameMain` und `PostkartenFrame` finden Sie auch im ZIP-Archiv.

```
1  /**
2   * Diese Klasse startet den PostkartenFrame
3   *
4   * @author Annabelle Klarl
5   */
6  public class PostkartenFrameMain {
7
8      /**
9       * Dieses Programmstueck startet das Programm.
10      *
11      * @param args
12      */
13     public static void main(String[] args) {
14         PostkartenFrame frame = new PostkartenFrame();
15         frame.setVisible(true);
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  import java.util.Iterator;
6  import java.util.LinkedList;
7  import javax.swing.JButton;
8  import javax.swing.JFrame;
9  import javax.swing.JOptionPane;
10 import javax.swing.JScrollPane;
11 import javax.swing.JTextArea;
12 import javax.swing.ScrollPaneConstants;
13
14 /**
15  * Diese Klasse repraesentiert das Hauptfenster der Postkarten-Praesenzaufgabe,
16  * wobei {@link LinkedList} verwendet wird.
17  *
18  * @author Annabelle Klarl
19  */
20 public class PostkartenFrame extends JFrame implements ActionListener {
21     private JButton postkarteHinzufuegenButton;
22     private JButton siegerBerechnenButton;
23
24     private JTextArea ausgabeBereich;
25
26     private LinkedList<Postkarte> postkartenListe;
27
28     /** In diesem Programmstueck wird das Fenster erzeugt */
29     public PostkartenFrame() {
30         /* Verbindung zum Modell der GUI */
31         this.postkartenListe = new LinkedList<Postkarte>();
32
33         /* In der Kopfleiste des Fenster steht "PostkartenFrame" */
34         this.setTitle("PostkartenFrame");
35
36         /* Das Fenster ist 500 Pixel breit und 250 Pixel hoch. */
37         this.setSize(500, 250);
```



```

38
39  /* Hier werden alle Buttons erzeugt. */
40  this.postkarteHinzufuegenButton =
41      new JButton("Neue Postkarte hinzufuegen");
42  this.siegerBerechnenButton =
43      new JButton("Sieger aus allen bisherigen Postkarten berechnen");
44
45  /* Hier wird der Ausgabe-Bereich erzeugt. */
46  this.ausgabeBereich = new JTextArea(10, 100);
47  JScrollPane scrollPane = new JScrollPane(this.ausgabeBereich);
48  scrollPane.setHorizontalScrollBarPolicy(
49      ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
50
51  /*
52      * Der ContentPane ist der Ausschnitt des Fensters, auf dem Widgets
53      * d.h. Interaktionselemente (wie eine TextArea oder ein Button)
54      * platziert werden koennen.
55      */
56  Container contentPane = this.getContentPane();
57  contentPane.setLayout(new GridLayout(3, 1));
58  /* Hier wird der Button platziert. */
59  contentPane.add(this.postkarteHinzufuegenButton);
60  contentPane.add(this.siegerBerechnenButton);
61  /* Hier wird der Ausgabebereich platziert. */
62  contentPane.add(scrollPane);
63
64  /*
65      * Hier wird der Frame als Listener fuer Knopfdruck-Ereignisse bei
66      * jedem der Buttons registriert.
67      */
68  this.postkarteHinzufuegenButton.addActionListener(this);
69  this.siegerBerechnenButton.addActionListener(this);
70
71  /*
72      * Wird das Fenster geschlossen (d.h. auf X gedrueckt), wird mit Hilfe
73      * dieser Programmzeile auch unser Programm ordnungsgemaess beendet.
74      */
75  this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
76  }
77
78  @Override
79  public void actionPerformed(ActionEvent e) {
80      // je nach Button entsprechende Methode ausfuehren
81      Object source = e.getSource();
82      if (source == this.postkarteHinzufuegenButton) {
83          this.postkarteHinzufuegen();
84      } else if (source == this.siegerBerechnenButton) {
85          this.siegerBerechnen();
86      }
87  }
88
89  /** implementiert die Funktionalitaet des "Postkarte hinzufuegen"-Buttons */
90  private void postkarteHinzufuegen() {
91      String name = JOptionPane.showInputDialog("Name des Teilnehmers: ");
92
93      String einlesenEntfernung =
94          JOptionPane.showInputDialog("Zurueckgelegte Entfernung in km: ");
95      int entfernung = Integer.parseInt(einlesenEntfernung);
96
97      Postkarte neuePostkarte = new Postkarte(name, entfernung);
98      this.postkartenListe.addFirst(neuePostkarte);
99
100     this.ausgabeBereich.setText(
101         "Die Postkarte des Teilnehmers "
102         + name

```

```

103         + " mit einer Entfernung von "
104         + entfernung
105         + " km wurde hinzugefuegt.");
106     }
107
108     /** implementiert die Funktionalitaet des "Sieger berechnen"-Buttons */
109     private void siegerBerechnen() {
110         int groessteEntfernung = -1;
111         LinkedList<Postkarte> siegerListe = new LinkedList<Postkarte>();
112
113         Iterator<Postkarte> iteratorPostkarten = this.postkartenListe.iterator();
114         while (iteratorPostkarten.hasNext()) {
115             Postkarte postkarte = iteratorPostkarten.next();
116             int entfernung = postkarte.getEntfernung();
117
118             if (entfernung > groessteEntfernung) {
119                 groessteEntfernung = entfernung;
120                 siegerListe = new LinkedList<Postkarte>();
121                 siegerListe.addFirst(postkarte);
122             } else if (entfernung == groessteEntfernung) {
123                 siegerListe.addFirst(postkarte);
124             }
125             // else: diese Postkarte ist nicht weiter gereist
126             // als die bisherigen Siegerpostkarten
127         }
128
129         this.ausgabeBereich.setText("Die Sieger sind: \n");
130         Iterator<Postkarte> iteratorGewinner = siegerListe.iterator();
131         while (iteratorGewinner.hasNext()) {
132             Postkarte postkarte = iteratorGewinner.next();
133             this.ausgabeBereich.append(
134                 postkarte.getTeilnehmer()
135                 + " mit einer Entfernung von "
136                 + postkarte.getEntfernung()
137                 + " km");
138             this.ausgabeBereich.append("\n");
139         }
140     }
141 }

```

Aufgabe 13-2

Rechnungsverwaltung

Hausaufgabe

Eine Firma möchte beliebig viele Rechnungen erfassen, die sie an Kunden ausgestellt hat. Für jede einzelne Rechnung sollen die Rechnungsnummer und der Rechnungsbetrag angegeben werden. Die Firma möchte außerdem ihre besten Kunden herausfinden, indem sie den/die Kunden mit dem höchsten Rechnungsbetrag ermittelt.

- a) Schreiben Sie eine Klasse **Rechnung**, die eine Rechnung wie oben beschrieben repräsentiert. Die Klasse soll Attribute für die Rechnungsnummer und den Rechnungsbetrag (als Ganzzahl vom Typ `int`) haben. Fügen Sie einen Konstruktor zur Initialisierung der Attribute sowie "Getter" für beide Attribute hinzu.

Lösung:

Eine Implementierung der Klasse **Rechnung** finden Sie auch im ZIP-Archiv.

```

1  /**
2   * Diese Klasse repraesentiert eine Rechnung bestehend aus einer
3   * Rechnungsnummer und dem zu zahlenden Betrag in Euro,
4   * wobei hier nur ganzzahlige Betraege erlaubt sind.
5   *
6   * @author Annabelle Klarl

```

```

7  */
8  public class Rechnung {
9
10     private int rechnungsnr;
11     private int betrag;
12
13     /**
14      * Konstruktor
15      *
16      * @param rechnungsnr
17      * @param betrag Rechnungsbetrag in Euro als ganzzahliger Betrag
18      */
19     public Rechnung(int rechnungsnr, int betrag) {
20         this.rechnungsnr = rechnungsnr;
21         this.betrag = betrag;
22     }
23
24     /**
25      * Diese Methode gibt die Rechnungsnummer zurueck.
26      *
27      * @return die Rechnungsnummer
28      */
29     public int getRechnungsnr() {
30         return this.rechnungsnr;
31     }
32
33     /**
34      * Diese Methode gibt den Rechnungsbetrag zurueck.
35      *
36      * @return der Rechnungsbetrag
37      */
38     public int getBetrag() {
39         return this.betrag;
40     }
41 }

```

- b) Implementieren Sie nun eine verkettete Liste zur Speicherung von Rechnungen nach dem Beispiel in der Vorlesung. Nennen Sie die Klasse, die Ihre verkettete Liste darstellt, `MeineRechnungsListe` und die Klasse für Elemente in dieser Liste `MeinRechnungsListenelement`.

Lösung:

Eine Implementierung der Klassen `MeineRechnungsListe` und `MeinRechnungsListenelement` finden Sie auch im ZIP-Archiv.

```

1  /**
2   * Diese Klasse implementiert eine verkettete Liste, die Rechnungen speichert.
3   * Dazu werden als Elemente Objekte vom Typ {@link MeinRechnungsListenelement}
4   * benutzt.
5   *
6   * @author Annabelle Klarl
7   */
8  public class MeineRechnungsListe {
9
10     private MeinRechnungsListenelement first;
11
12     /** Konstruktor: erstellt eine leere Liste */
13     public MeineRechnungsListe() {
14         this.first = null;
15     }
16
17     /**
18      * Fuegt die gegebene Rechnung an erster Stelle in die Liste ein.

```

```

19  * Alle bisherigen Listen-Elemente werden um eins nach hinten gerueckt.
20  *
21  * @param rechnung
22  */
23  public void addFirst(Rechnung rechnung) {
24      this.first = new MeinRechnungsListenelement(rechnung, this.first);
25  }
26
27  /**
28   * Bestimmt die Anzahl der Elemente in der Liste
29   *
30   * @return die Anzahl der Elemente in der Liste
31   */
32  public int size() {
33      int size = 0;
34      MeinRechnungsListenelement element = this.first;
35      while (element != null) {
36          element = element.getNext();
37          size++;
38      }
39      return size;
40  }
41
42  /**
43   * Gibt das Element an Position i in der Liste zurueck
44   * (ohne es aus der Liste zu loeschen!).
45   *
46   * @param i
47   * @return das Listen-Element an Position i
48   */
49  public Rechnung get(int i) {
50      MeinRechnungsListenelement element = this.first;
51      while (element != null && i > 0) {
52          element = element.getNext();
53          i--;
54      }
55      if (element == null) {
56          throw new IndexOutOfBoundsException();
57      }
58      return element.getRechnung();
59  }
60 }

```

```

1  /**
2   * Diese Klasse repraesentiert ein Element in einer
3   * {@link MeineRechnungsListe}.
4   * Sie verweist auf die gespeicherte Rechnung fuer dieses Element
5   * und verweist ebenso auf das naechste Element vom Typ
6   * {@link MeinRechnungsListenelement} in der Liste.
7   *
8   * @author Annabelle Klarl
9   */
10 public class MeinRechnungsListenelement {
11
12     private Rechnung rechnung;
13     private MeinRechnungsListenelement next;
14
15     /**
16      * Konstruktor: erstellt ein neues Element mit einem Verweis
17      * auf die entsprechende Rechnung
18      *
19      * @param rechnung
20      */
21     public MeinRechnungsListenelement(Rechnung rechnung) {
22         this.rechnung = rechnung;

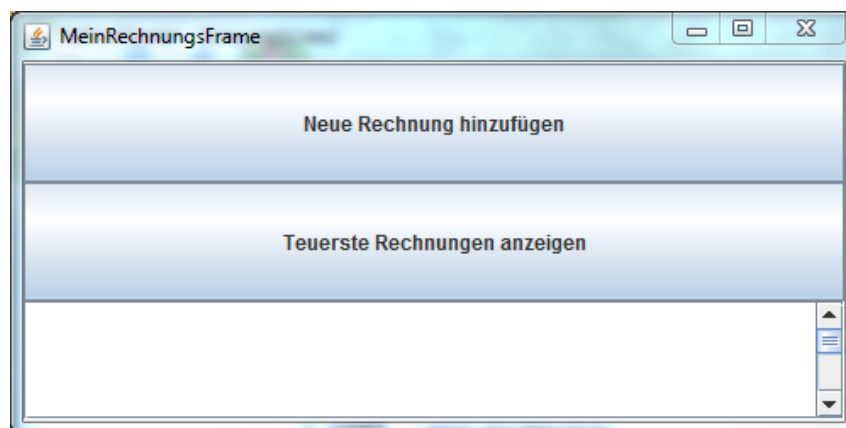
```

```

23     this.next = null;
24 }
25
26 /**
27  * Konstruktor: erstellt ein neues Element mit einem Verweis
28  * auf die entsprechende Rechnung und
29  * erstellt einen Verweis auf das naechste Listen-Element.
30  *
31  * @param rechnung
32  * @param next
33  */
34 public MeinRechnungsListenelement(
35     Rechnung rechnung, MeinRechnungsListenelement next) {
36     this.rechnung = rechnung;
37     this.next = next;
38 }
39
40 /**
41  * Diese Methode gibt die zu diesem Listen-Element gehoerende Rechnung
42  * zurueck.
43  *
44  * @return
45  */
46 public Rechnung getRechnung() {
47     return this.rechnung;
48 }
49
50 /**
51  * Diese Methode gibt das naechste Listen-Element zurueck.
52  *
53  * @return
54  */
55 public MeinRechnungsListenelement getNext() {
56     return this.next;
57 }
58 }

```

- c) Die grafische Benutzeroberfläche für die Rechnungsverwaltung soll wie folgt aussehen:



Es soll zwei Buttons mit der oben genannten Aufschrift geben. Darunter soll ein Ausgabebereich platziert werden.

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

Erweitern Sie Ihre Klasse `MeinRechnungsFrame` um eine Ereignisbehandlung für die beiden Buttons. Wird der Button “Neue Rechnung hinzufügen” gedrückt, soll der Benutzer zunächst mit Hilfe der Klasse `JOptionPane` nach der Rechnungsnummer und dem Rechnungsbetrag gefragt werden. Anschließend soll aus diesen Angaben ein Objekt vom Typ `Rechnung` erstellt werden und in einer Liste vom Typ `MeineRechnungsListe` gespeichert werden. Darüber soll der Benutzer im Ausgabebereich informiert werden.

Wird der Button “Teuerste Rechnungen anzeigen” gedrückt, sollen in der Liste aller bisher eingegebenen Rechnungen alle Rechnungen gesucht werden, die den höchsten Rechnungsbetrag aufweisen. Der Benutzer soll anschließend darüber informiert werden, welche Rechnungen den größten Betrag aufweisen. Beachten Sie, dass dies auch mehrere Rechnungen sein können.

Hinweis: In Ihrer Klasse `MeinRechnungsFrame` werden Sie ein Attribut brauchen, um die Liste von Rechnungen speichern zu können (Modell der GUI).

Lösung:

Bei der Implementierung der GUI geht man wie gewohnt vor. Da die Ausgabe der teuersten Rechnungen allerdings relativ lang 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(this.ausgabeBereich);
2 scrollPane.setHorizontalScrollBarPolicy(
3     ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
```

Vergessen Sie außerdem nicht Ihrer Klasse `MeinRechnungsFrame` ein Attribut vom Typ `MeineRechnungsListe` hinzuzufügen, das alle eingetragenen Rechnungen speichern soll und bei der Benutzung des Buttons “Neue Rechnung hinzufügen” um ein Element erweitert wird.

Um alle Rechnungen mit dem höchsten Rechnungsbetrag zu finden, geht man wie folgt vor:

1. Lege eine neue Liste zur Speicherung aller teuerster Rechnungen an.
2. Dekлариere eine lokale Variable `hoechsterBetrag`, die den Rechnungsbetrag der teuersten Rechnung speichern soll, und initialisiere sie mit -1.
3. Durchlaufe die Liste von gespeicherten Rechnungen und wiederhole für jedes Element:
 - Falls die Rechnung teurer ist als die teuerste Rechnung, lösche die bisherige Liste teuerster Rechnungen. Füge die aktuelle Rechnung in die nun leere Liste teuerster Rechnungen ein und speichere den Rechnungsbetrag dieser Rechnung als nun aktuell teuerste Rechnung in der lokalen Variablen `hoechsterBetrag`.
 - Falls die Rechnung genauso teuer ist wie die aktuell teuerste Rechnung, füge die aktuelle Rechnung in die Liste teuerster Rechnungen ein.
4. Gebe die Liste teuerster Rechnungen aus.

Eine Implementierung der Klassen `MeinRechnungsFrameMain` und `MeinRechnungsFrame` finden Sie auch im ZIP-Archiv.

```
1 /**
2  * Diese Klasse startet den MeinRechnungsFrame
3  *
4  * @author Annabelle Klarl
5  */
6 public class MeinRechnungsFrameMain {
7
8     /**
9      * Dieses Programmstueck startet das Programm.
10     *
11     * @param args
```

```

12  */
13  public static void main(String[] args) {
14      MeinRechnungsFrame frame = new MeinRechnungsFrame();
15      frame.setVisible(true);
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  import javax.swing.JButton;
6  import javax.swing.JFrame;
7  import javax.swing.JOptionPane;
8  import javax.swing.JScrollPane;
9  import javax.swing.JTextArea;
10 import javax.swing.ScrollPaneConstants;
11
12 /**
13  * Diese Klasse repraesentiert das Hauptfenster der Rechnungen - Hausaufgabe,
14  * wobei eine eigene verkettete Liste verwendet wird.
15  *
16  * @author Annabelle Klarl
17  */
18 public class MeinRechnungsFrame extends JFrame implements ActionListener {
19     private JButton rechnungHinzufuegenButton;
20     private JButton teuersteRechnungBerechnenButton;
21
22     private JTextArea ausgabeBereich;
23
24     private MeineRechnungsListe rechnungsListe;
25
26     /** In diesem Programmstueck wird das Fenster erzeugt */
27     public MeinRechnungsFrame() {
28         /* Verbindung zum Modell der GUI */
29         this.rechnungsListe = new MeineRechnungsListe();
30
31         /* In der Kopfleiste des Fenster steht "MeinRechnungsFrame" */
32         this.setTitle("MeinRechnungsFrame");
33
34         /* Das Fenster ist 500 Pixel breit und 250 Pixel hoch. */
35         this.setSize(500, 250);
36
37         /* Hier werden alle Buttons erzeugt. */
38         this.rechnungHinzufuegenButton =
39             new JButton("Neue Rechnung hinzufuegen");
40         this.teuersteRechnungBerechnenButton =
41             new JButton("Teuerste Rechnungen anzeigen");
42
43         /* Hier wird der Ausgabe-Bereich erzeugt. */
44         this.ausgabeBereich = new JTextArea(10, 100);
45         JScrollPane scrollPane = new JScrollPane(this.ausgabeBereich);
46         scrollPane.setHorizontalScrollBarPolicy(
47             ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
48
49         /*
50          * Der ContentPane ist der Ausschnitt des Fensters, auf dem Widgets d.h.
51          * Interaktionselemente (wie eine TextArea oder ein Button) platziert
52          * werden koennen.
53          */
54         Container contentPane = this.getContentPane();
55         contentPane.setLayout(new GridLayout(3, 1));
56         /* Hier wird der Button platziert. */
57         contentPane.add(this.rechnungHinzufuegenButton);
58         contentPane.add(this.teuersteRechnungBerechnenButton);

```

```

59      /* Hier wird der Ausgabebereich platziert. */
60      contentPane.add(scrollPane);
61
62      /*
63       * Hier wird der Frame als Listener fuer Knopfdruck-Ereignisse bei jedem
64       * der Buttons registriert.
65       */
66      this.rechnungHinzufuegenButton.addActionListener(this);
67      this.teuersteRechnungBerechnenButton.addActionListener(this);
68
69      /*
70       * Wird das Fenster geschlossen (d.h. auf X gedrueckt), wird mit Hilfe
71       * dieser Programmzeile auch unser Programm ordnungsgemaess beendet.
72       */
73      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
74  }
75
76  @Override
77  public void actionPerformed(ActionEvent e) {
78      // je nach Button entsprechende Methode ausfuehren
79      Object source = e.getSource();
80      if (source == this.rechnungHinzufuegenButton) {
81          this.rechnungHinzufuegen();
82      } else if (source == this.teuersteRechnungBerechnenButton) {
83          this.teuersteRechnungBerechnen();
84      }
85  }
86
87  /** implementiert die Funktionalitaet des "Rechnung hinzufuegen"-Buttons */
88  private void rechnungHinzufuegen() {
89      String einlesenRechnungsnr =
90          JOptionPane.showInputDialog("Rechnungsnummer: ");
91      int rechnungsnr = Integer.parseInt(einlesenRechnungsnr);
92
93      String einlesenBetrag =
94          JOptionPane.showInputDialog("Rechnungsbetrag in Euro: ");
95      int betrag = Integer.parseInt(einlesenBetrag);
96
97      Rechnung neueRechnung = new Rechnung(rechnungsnr, betrag);
98      this.rechnungsListe.addFirst(neueRechnung);
99
100     this.ausgabeBereich.setText(
101         "Die Rechnung mit der Nummer "
102         + rechnungsnr
103         + " mit einem Betrag von "
104         + betrag
105         + " Euro wurde hinzugefuegt.");
106 }
107
108 /**
109  * implementiert die Funktionalitaet des
110  * "Teuerste Rechnung berechnen"-Buttons
111  */
112 private void teuersteRechnungBerechnen() {
113     int hoechsterBetrag = -1;
114     MeineRechnungsListe teuersteRechnungenListe = new MeineRechnungsListe();
115
116     for (int i = 0; i < this.rechnungsListe.size(); i++) {
117         Rechnung rechnung = this.rechnungsListe.get(i);
118         int betrag = rechnung.getBetrag();
119
120         if (betrag > hoechsterBetrag) {
121             hoechsterBetrag = betrag;
122             teuersteRechnungenListe = new MeineRechnungsListe();
123             teuersteRechnungenListe.addFirst(rechnung);

```



```

124     } else if (betrag == hoechsterBetrag) {
125         teuersteRechnungenListe.addFirst(rechnung);
126     }
127     // else: diese Rechnung ist billiger als die aktuell teuerste Rechnung
128 }
129
130 this.ausgabeBereich.setText("Die teuersten Rechnungen sind: \n");
131 for (int i = 0; i < teuersteRechnungenListe.size(); i++) {
132     Rechnung rechnung = teuersteRechnungenListe.get(i);
133     this.ausgabeBereich.append(
134         rechnung.getRechnungsnr()
135         + " mit einem Betrag von "
136         + rechnung.getBetrag()
137         + " Euro");
138     this.ausgabeBereich.append("\n");
139 }
140 }
141 }

```

- d) Zur Lösung der Aufgabe sollen nun die generischen Klassen `LinkedList<E>` und `Iterator<E>` der Java-Bibliothek verwendet werden. Nehmen Sie die Benutzeroberfläche aus Aufgabe c) zur Grundlage und schreiben Sie eine neue Klasse `RechnungsFrame`. Das Aussehen der Benutzeroberfläche sowie die Funktionsweise der Buttons soll nicht verändert werden, allerdings soll anstelle eine Liste vom Typ `MeineRechnungenListe` nun eine Liste vom Typ `LinkedList<Rechnung>` verwendet werden, um alle Rechnungen zu speichern. Ebenso soll zum Durchlaufen einer Liste stets ein Objekt vom Typ `Iterator<Rechnung>` benutzt werden. Um Ihr Programm ausführen zu können, schreiben Sie eine weitere Klasse `RechnungsFrameMain`, die Sie wie gewohnt im gleichen Ordner wie Ihre Klasse `RechnungsFrame` abspeichern.

Lösung:

Die Implementierung der Klassen `RechnungsFrame` und `RechnungsFrameMain` finden Sie auf der Webseite. **Eine Implementierung der Klassen `RechnungsFrameMain` und `RechnungsFrame` finden Sie auch im ZIP-Archiv.**

```

1  /**
2   * Diese Klasse startet den RechnungsFrame
3   *
4   * @author Annabelle Klarl
5   */
6  public class RechnungsFrameMain {
7
8      /**
9       * Dieses Programmstueck startet das Programm.
10      *
11      * @param args
12      */
13     public static void main(String[] args) {
14         RechnungsFrame frame = new RechnungsFrame();
15         frame.setVisible(true);
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  import java.util.Iterator;
6  import java.util.LinkedList;
7  import javax.swing.JButton;
8  import javax.swing.JFrame;

```

```

9 import javax.swing.JOptionPane;
10 import javax.swing.JScrollPane;
11 import javax.swing.JTextArea;
12 import javax.swing.ScrollPaneConstants;
13
14 /**
15  * Diese Klasse repraesentiert das Hauptfenster der Rechnungen-Hausaufgabe,
16  * wobei {@link LinkedList} verwendet wird.
17  *
18  * @author Annabelle Klarl
19  */
20 public class RechnungsFrame extends JFrame implements ActionListener {
21     private JButton rechnungHinzufuegenButton;
22     private JButton teuersteRechnungBerechnenButton;
23
24     private JTextArea ausgabeBereich;
25
26     private LinkedList<Rechnung> rechnungsListe;
27
28     /** In diesem Programmstueck wird das Fenster erzeugt */
29     public RechnungsFrame() {
30         /* Verbindung zum Modell der GUI */
31         this.rechnungsListe = new LinkedList<Rechnung>();
32
33         /* In der Kopfleiste des Fenster steht "RechnungsFrame" */
34         this.setTitle("RechnungsFrame");
35
36         /* Das Fenster ist 500 Pixel breit und 250 Pixel hoch. */
37         this.setSize(500, 250);
38
39         /* Hier werden alle Buttons erzeugt. */
40         this.rechnungHinzufuegenButton =
41             new JButton("Neue Rechnung hinzufuegen");
42         this.teuersteRechnungBerechnenButton =
43             new JButton("Teuerste Rechnungen anzeigen");
44
45         /* Hier wird der Ausgabe-Bereich erzeugt. */
46         this.ausgabeBereich = new JTextArea(10, 100);
47         JScrollPane scrollPane = new JScrollPane(this.ausgabeBereich);
48         scrollPane.setHorizontalScrollBarPolicy(
49             ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
50
51         /*
52          * Der ContentPane ist der Ausschnitt des Fensters, auf dem Widgets d.h.
53          * Interaktionselemente (wie eine TextArea oder ein Button) platziert
54          * werden koennen.
55          */
56         Container contentPane = this.getContentPane();
57         contentPane.setLayout(new GridLayout(3, 1));
58         /* Hier wird der Button platziert. */
59         contentPane.add(this.rechnungHinzufuegenButton);
60         contentPane.add(this.teuersteRechnungBerechnenButton);
61         /* Hier wird der Ausgabebereich platziert. */
62         contentPane.add(scrollPane);
63
64         /*
65          * Hier wird der Frame als Listener fuer Knopfdruck-Ereignisse bei jedem
66          * der Buttons registriert.
67          */
68         this.rechnungHinzufuegenButton.addActionListener(this);
69         this.teuersteRechnungBerechnenButton.addActionListener(this);
70
71         /*
72          * Wird das Fenster geschlossen (d.h. auf X gedrueckt), wird mit Hilfe
73          * dieser Programmzeile auch unser Programm ordnungsgemaess beendet.

```

```

74     */
75     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
76 }
77
78 @Override
79 public void actionPerformed(ActionEvent e) {
80     // je nach Button entsprechende Methode ausfuehren
81     Object source = e.getSource();
82     if (source == this.rechnungHinzufuegenButton) {
83         this.rechnungHinzufuegen();
84     } else if (source == this.teuersteRechnungBerechnenButton) {
85         this.teuersteRechnungBerechnen();
86     }
87 }
88
89 /** implementiert die Funktionalitaet des "Rechnung hinzufuegen"-Buttons */
90 private void rechnungHinzufuegen() {
91     String einlesenRechnungsnr =
92         JOptionPane.showInputDialog("Rechnungsnummer: ");
93     int rechnungsnr = Integer.parseInt(einlesenRechnungsnr);
94
95     String einlesenBetrag =
96         JOptionPane.showInputDialog("Rechnungsbetrag in Euro: ");
97     int betrag = Integer.parseInt(einlesenBetrag);
98
99     Rechnung neueRechnung = new Rechnung(rechnungsnr, betrag);
100    this.rechnungsListe.addFirst(neueRechnung);
101
102    this.ausgabeBereich.setText(
103        "Die Rechnung mit der Nummer "
104        + rechnungsnr
105        + " mit einem Betrag von "
106        + betrag
107        + " Euro wurde hinzugefuegt.");
108 }
109
110 /**
111  * implementiert die Funktionalitaet des
112  * "Teuerste Rechnung berechnen"-Buttons
113  */
114 private void teuersteRechnungBerechnen() {
115     int hoechsterBetrag = -1;
116     LinkedList<Rechnung> teuersteRechnungenListe = new LinkedList<Rechnung>();
117
118     Iterator<Rechnung> iteratorRechnung = this.rechnungsListe.iterator();
119     while (iteratorRechnung.hasNext()) {
120         Rechnung rechnung = iteratorRechnung.next();
121         int betrag = rechnung.getBetrag();
122
123         if (betrag > hoechsterBetrag) {
124             hoechsterBetrag = betrag;
125             teuersteRechnungenListe = new LinkedList<Rechnung>();
126             teuersteRechnungenListe.addFirst(rechnung);
127         } else if (betrag == hoechsterBetrag) {
128             teuersteRechnungenListe.addFirst(rechnung);
129         }
130         // else: diese Rechnung ist billiger als die aktuell teuerste Rechnung
131     }
132
133     this.ausgabeBereich.setText("Die teuersten Rechnungen sind: \n");
134     Iterator<Rechnung> iteratorTeuersteRechnungen =
135         teuersteRechnungenListe.iterator();
136     while (iteratorTeuersteRechnungen.hasNext()) {
137         Rechnung rechnung = iteratorTeuersteRechnungen.next();
138         this.ausgabeBereich.append(

```

```

139         rechnung.getRechnungsnr()
140         + " mit einem Betrag von "
141         + rechnung.getBetrag()
142         + " Euro");
143     this.ausgabeBereich.append("\n");
144 }
145 }
146 }

```

Aufgabe 13-3

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). Die am meisten gestellten Fragen werden in der Zentralübung am 06.02.2019 behandelt.

*Besprechung der Präsenzaufgaben in den Übungen am 25.01.2019 und 28.01.2019. Abgabe der Hausaufgaben bis **Montag**, 04.02.2019, 14:00 Uhr über UniworX (siehe Folien der ersten Zentralübung).*

- Erstellen Sie zu jeder Aufgabe Klassen mit den entsprechenden Namen, die in der Aufgabe gefordert sind.
- Geben Sie nur die entsprechenden *.java*-Dateien ab. Wir benötigen **nicht** Ihre *.class*-Dateien.
- Geben Sie Java-Code nur in *.java*-Dateien ab. Java-Code in Bildern, PDF-Dokumenten und Text-Dateien wird nicht korrigiert.