Hochschule Harz	FB Automatisierung und Informatik
Programmierung2	Dipl.-Inf., Dipl.-Ing. (FH) M. Wilhelm
Hausaufgabe08:	„Programmierung 2“ für MI / WI Thema: **Singleton, Log-Datei und Queue**

Versuchsziele

Kenntnisse in der Anwendung von:

- Singleton
- Schreiben einer Log-Datei
- Verwendung einer Queue (Warteschlange)
- Der Student soll erkennen, dass die Verwendung eines Singleton gegenüber eines normalen Objektes erhebliche Vorteile hat (absolute Lokalität)

Hausaufgabe08a:

In dieser Aufgabe soll eine grafische Oberfläche mit Menüs und Schaltern für das Testen einer Log-Datei mit Singleton-Instanz implementiert werden. Jede Aktion (Menü oder Schalter) soll in einer Log-Datei mit Datum/Uhrzeit gespeichert werden.

Aufgaben

1. Teilaufaufgaben: Projekt erstellen und aufbauen:

- Projektname: Aufgabe08a

2. Teilaufaufgaben: vorgegebner Quellcode:

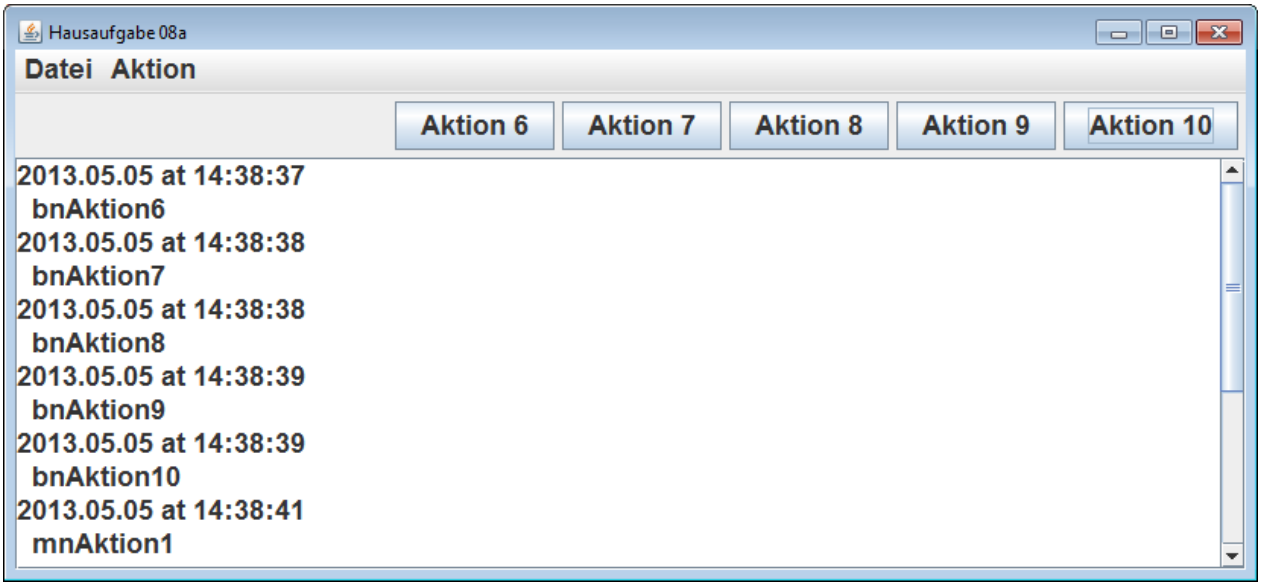


Abbildung 1 Überblick der GUI: Aufgabe08a

- Bauen Sie in Ihr Projekt den vorgegeben Quellcode ein (siehe Homepage).

3. Teilaufaufgabe: Klasse LogFile

Diese Aufgabe implementiert eine Klasse „LogFile“, die als Singleton implementiert werden soll.
Eigenschaften:

- **Name der Klasse:**
  - LogFile
- **Konstante:**
  - public static String fileName = "log.txt";
- **Attribute:**
  - privates Attribut:
    - instance à la Singleton
- **Methoden:**
  - clearFile
    - Keine Parameter
    - Löscht die vorhandene Datei (Öffnen und Schließen)
    - Verwendet FileOutputStream und DataOutputStream
  - write
    - Parameter:
      - String text
    - Schreibt das aktuelle Datum und die Uhrzeit in die Datei (anhängen).
    - Schreibt den Text in die Datei (anhängen).

Hinweise:

String speichern mittels DataOutputStreams:

```
boolean anhaengen = false;
FileOutputStream fout = new FileOutputStream(fileName, anhaengen);
DataOutputStream dout   = new DataOutputStream(fout);
dout.writeBytes(" hier ist ein Text "+"\n");
dout.flush();
dout.close();
```

Datum / Uhrzeit holen:

```
SimpleDateFormat form = new SimpleDateFormat ("yyyy.MM.dd 'at' HH:mm:ss ");
Date currentTime = new Date();
String s = form.format(currentTime);
```

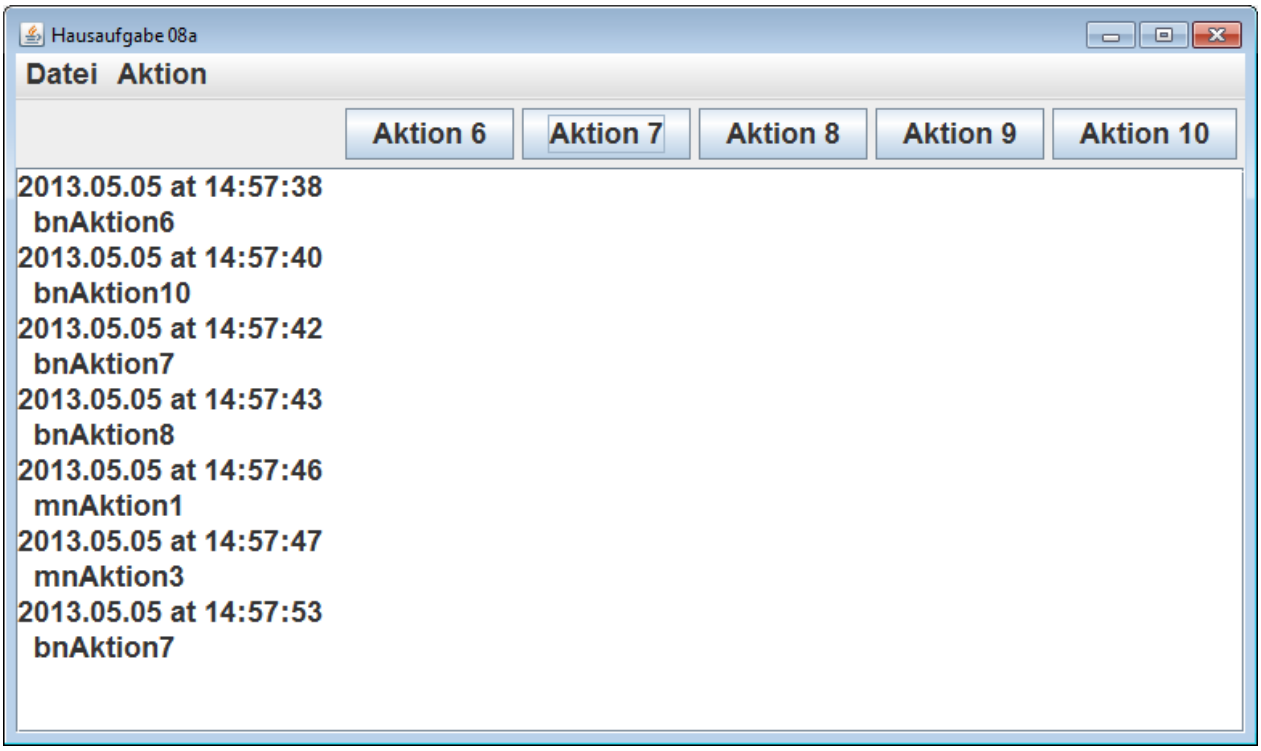
4. Teilaufaufgabe: Hauptframe, Methode implementieren

In dieser Aufgabe sollen die Eventmenthoden (Menüs und Schalter) ihren Aufruf in der Log-Datei dokumentieren:

- Holen der Instanz der Klasse „LogFile“
- Schreiben der Aktion (zum Beispiel: mnAktion1)

5. Teilaufaufgabe: „Testablauf“

- Starten
- Schalter Aktion6 anklicken
- Schalter Aktion10 anklicken
- Schalter Aktion7 anklicken
- Schalter Aktion8 anklicken
- Menü Aktion1 anklicken
- Menü Aktion3 anklicken
- Schalter Aktion7 anklicken
- Laden des Datei



Aufgabe08b

In dieser Aufgabe muss eine Queue als Singleton implementiert werden. In der Warteschlange werden „Bestellungen“ eingefügt und mittels eines Timers entfernt.

1. Teilaufaufgaben: Neues Projekt erstellen und aufbauen:

- Projektname: Aufgabe08b

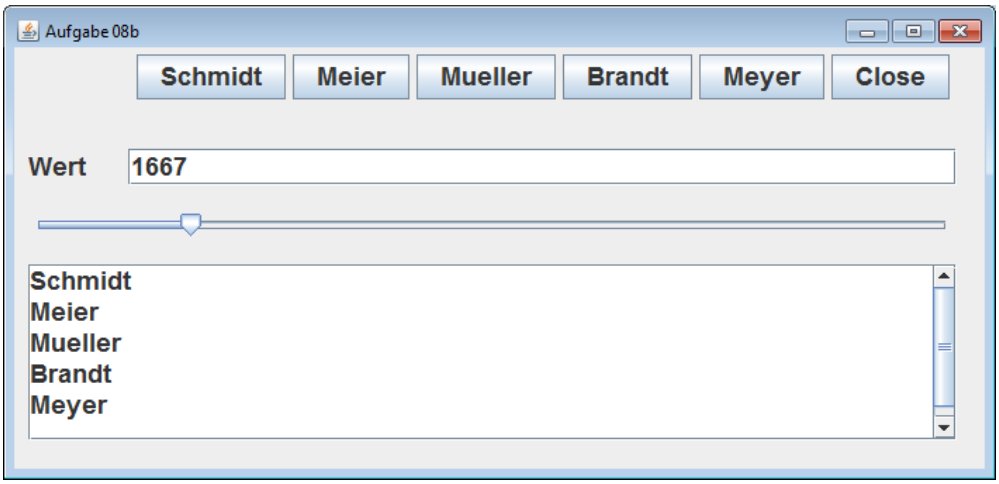


Abbildung 2 Überblick der GUI: Aufgabe08b

Erläuterung:

- Die Schalter „erzeugen“ jeweils eine Bestellung
- Der Slider bestimmt die Rechenkapazität. Der jeweilige Wert wird im JTextField angezeigt.
- Intern wird ein Timer verwendet, dessen Delay-Zeit über den Slider gesetzt wird.
- Nach jedem Aufruf des „Timerevents“ wird genau eine Bestellung aus der Queue entfernt (falls vorhanden).

2. Teilaufaufgaben: Vorhandener Quellcode:

- Bauen Sie in Ihr Projekt den vorgegeben Quellcode ein (Homepage Aufgabe08b).

3. Teilaufaufgabe: Klasse Bestellung

Diese Aufgabe implementiert eine Klasse „Bestellung“.
Eigenschaften:

- Attribut:
  - String name, kann öffentlich sein
- Methoden:
  - Keine
- Konstruktor:
  - String name

4. Teilaufaufgabe: Klasse MyQueue

In dieser Aufgabe Klasse soll die Klasse MyQueue als Singleton implementiert werden. Sie hat keine Oberklasse.

Eigenschaften:

- **Klassenname:**
  - MyQueue
- **Attribut:**
  - Privates statisches Attribut mit einer Instanz der Klasse MyQueue
  - queue (Warteschlange)
    - Zugriff: öffentlich
    - **Typ: Queue**
    - **Spezifischer Datentyp: Bestellung**
- **Methode**:
  - getInstance
- **Konstruktor:**
  - Privat, default-Konstruktor (muss sein, da er sonst öffentlich ist)

Hinweis:

- Eine Queue erstellt man, indem man zum Beispiel eine LinkedList als Basis nimmt.
- Beispiel:
  - public Queue<String> queue=new LinkedList<String>();
- Durch das Interface Queue verhält sich die LinkedList wie eine Queue.

5. Teilaufaufgabe: Hauptframe, Methoden bnSchmidt_click() etc.

In diesen Methoden soll jeweils eine neue Bestellung erzeugt und in die Queue eingetragen werden. Sinnvollerweise geschieht das in der separaten Methode „insertBestellung“. In den Namens-Methoden soll nur die Methode „insertBestellung“ aufgerufen werden (Parameter String).

6. Teilaufaufgabe: Hauptframe, Methode insertBestellung.

In dieser Methode soll eine neue Bestellung in die Warteschlange eingetragen werden. **Die Instanz darf nur mittels Singleton geholt werden**.
Parameter: String

Interface Queque extends Collection

Wichtige Methoden:

Name	**Beschreibung**
boolean add(E e)	Fügt ein neues Element in die Menge ein
E element	Gibt das Element „head“ aus. Bleibt aber in der Menge
E remove	Gibt das Element „head“ aus, entfernt es aus der Menge
E peek	Gibt das Element „head“ aus oder null. Element bleibt in der Menge
E poll	**Gibt das Element „head“ aus oder null, entfernt es aus der Menge**
boolean offer(E e)	Fügt das Element in die Liste, Returnwert: T/F

Weitere wichtige Methoden aus Collection:

- **isEmpty**

	Throws Exception	Returns special value
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

Klassen die diese Schnittstellen implementieren:

- **LinkedList**
- Queue

7. Teilaufaufgabe: Hauptframe, Methode queue_aktion.

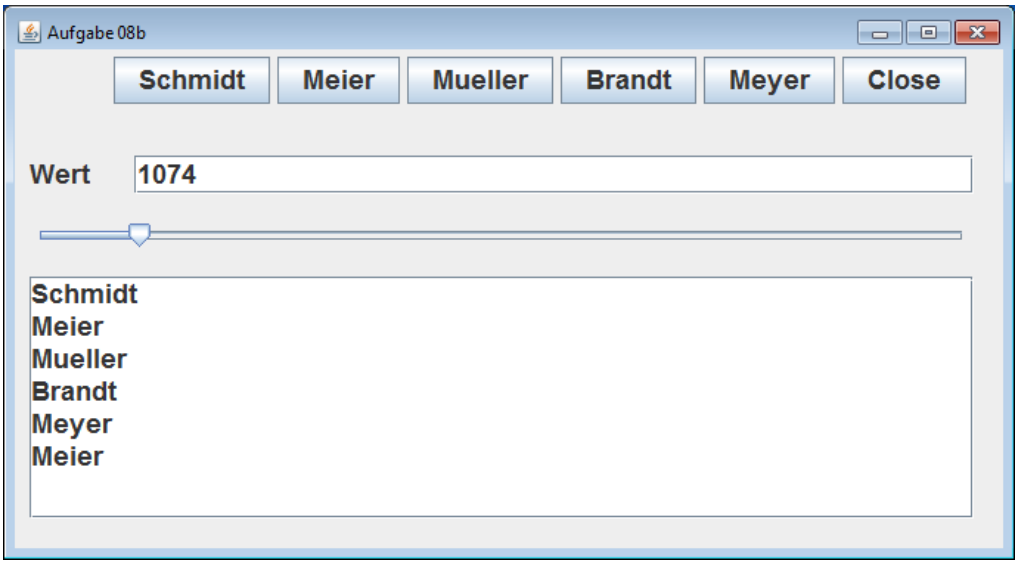
In dieser Methode soll aus der Queue, wenn möglich, eine Bestellung nach dem FIFO-Prinzip rausgeholt und in den Editor eingetragen werden. **Die Instanz darf nur mittels Singleton geholt werden**.

Ablauf:

- Holen der Instanz.
- Holen der internen Queue (mit Spezialisierung nach Bestellung).
- Abfrage ist die Queue voll oder so ähnlich.
- Holen des vordersten Elementes (siehe Tabelle oben).
- Eintragen in den Editor.

8. Teilaufaufgabe: „Testablauf“

1. Starten des Programms
2. Anklicken der Schalter
    a. Schmidt
    b. Meier
    c. Mueller
    d. Brandt
    e. Meyer
    f. Meier
3. Nach fünf Sekunden muss die erste Bestellung im Editor erscheinen.
4. Jetzt die Zeit auf ca. 1000 ms reduzieren.
5. Nun erscheinen die restlichen Bestellungen im Editor erscheinen.



Anhang

Interface Queque extends Collection

Wichtige Methoden:

Name	Beschreibung
boolean add(E e)	**Fügt ein neues Element in die Menge ein**
E element	Gibt das Element „head“ aus. Bleibt aber in der Menge
E remove	Gibt das Element „head“ aus, entfernt es aus der Menge
E peek	Gibt das Element „head“ aus oder null. Element bleibt in der Menge
E poll	Gibt das Element „head“ aus oder null, entfernt es aus der Menge
boolean offer(E e)	Fügt das Element in die Liste, Returnwert: T/F

Weitere Methoden aus Collection:

- addAll
- **clear**
- contains
- containsAll
- equals
- isEmpty
- iterator
- remove
- removeAll
- retainAll
- **size**
- toArray
- toArray

	Throws Exception	Returns special value
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

Klassen die diese Schnittstellen implementieren:

- ArrayBlockingQueue
- ArrayDeque
- ConcurrentLinkedQueue
- DelayQueue
- Deque
- LinkedBlockingDeque
- LinkedBlockingQueue
- **LinkedList**
- PriorityBlockingQueue
- PriorityQueue
- Queue
- SynchronousQueue

Erstellen und Benutzen einer Queue:

```
  private void test() {

    Queue<String>   queue  =  new   LinkedList<String>();

    queue.add("1 hallo");
    queue.offer("2 hallo");
    queue.add("3 hallo");
    queue.add("4 hallo");

    while(! queue.isEmpty() )
     {
       String value=(String)queue.poll();
       System.out.println(value);
     }

  }
```

Timer

Die Klasse Timer erlaubt das automatische Aufrufen einer Methode in einem JFrame.

Deklaration in der Klasse:

public class Timer1 extends JFrame **implements ActionListener** {

    javax.swing.Timer timer;

    timer = new Timer(500, this); // 500 ms

}

Timer muss gestartet werden:
    timer.start();

Abstellen des Timers:
    timer.stop();

Der Timer ruft die Methode actionPerformd auf:
    public void actionPerformed(ActionEvent e) {

    } // actionperformed

Intern wird ein Thread aufgerufen, der über ein Interface „ActionPerformed“ das JFrame aufruft.

Komplettes Beispiel Timer1.java

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

public class Timer1 extends JFrame implements ActionListener {

 Timer myTimer;  // Timer
 long summe = 0;

  // Konstruktor
 public Timer1 () {
  setSize(400, 300);  // Breite Höhe
  setLocation(300,300);
  setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  setGUI();
 }

  private void setGUI() {
  setTitle("Timertest");  // Titel
  myTimer = null;  // Null setzen
  startLoading();  // init und start des Timers
 }

    // init und Start des Timers animator
  public void startLoading() {
       // Sicherheitsbafrage
      if(myTimer == null) {
        myTimer = new Timer(1000, this); // 200 ms
        myTimer.start();  // starten
      }
  } // startLoading

    // Beendet den Timer und das gesamte Progamm !
  public void stopLoading() {
      // Sicherheitsabfrage
    if(myTimer != null) {
         myTimer.stop();  //
         myTimer = null;  // Anfangszustand
        System.exit(0);  // Beenden
   }
  }  // stopLoading

    // wird vom Timer aufgerufen
   public void actionPerformed(ActionEvent e) {
    if (e.getSource() == myTimer) {
      summe += 1;
      System.out.println(summe);
      if (summe>30) {
        stopLoading();
      }
    }
   }  // actionperformed
```

```
  public static void main(String[] args) {
    Timer1 timer;
    timer =  new Timer1();
    timer.setVisible(true);
  }  // main


}  // Timer1
```