| Hochschule Harz | FB Automatisierung und Informatik |
|---|---|
| Programmierung2 | Dipl.-Inf., Dipl.-Ing. (FH) M. Wilhelm |
| Tutorial / Hausaufgabe 07: | „Programmierung 2“ für MI / WI<br>Thema:<br>**ArrayList, Performancetest und Generische Klassen** |

# Versuchsziele

Kenntnisse in der Anwendung von:

- Implementierung einer eigenen ArrayList
- Performancetest der eigenen ArrayList mit der Java Klasse ArrayList
- Benutzung der generischen Klassen
- Performancetest mit Klassen des Collection-Frameworks
- Der Student soll erkennen, dass für große Datenmengen die internen Kenntnisse extrem wichtig sind.
- Beispiele:
    - addLast vs. add,
    - LinkedList und get

# Tutorial- Hausaufgabe07:

In dieser Aufgabe soll eine grafische Oberfläche für das Testen einer zu implementierenden ArrayList entwickelt werden. Benötigt werden drei Aktionen für das Tutorial und drei für die Hausaufgabe. Neben dem Schließen sind es also sieben Aktionen die „verdrahtet“ werden müssen. Die Technik dafür ist komplett Ihnen überlassen.

## Aufgaben

1. Teilaufaufgaben: Projekt erstellen und aufbauen:

- Projektname: Aufgabe07

2. Teilaufaufgaben: Event-Methoden:

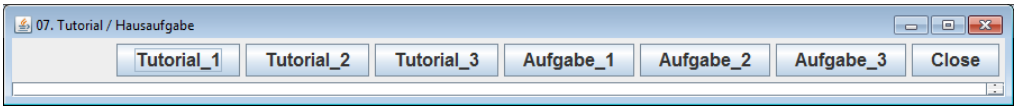


**Abbildung 1 Überblick der GUI**

- Bauen Sie in Ihr Projekt sieben Auswahl-Elemente ein. Damit sollen sieben Event-Methoden angesteuert werden. Die Technik (Menüs, Schalter oder andere Technik) ist beliebig.
- Namen der Events:
    - Tutorial_test1
    - Tutorial_test2
    - Tutorial_test3
    - Aufgabe_test1
    - Aufgabe_test2
    - Aufgabe_test3
    - Close
- Erstellen und verknüpfen Sie die sieben Event-Methoden
- **Sie können die auf meiner Homepage zur Verfügung gestellten Java-Beispiele benutzen.**
- **ODER Sie benutzen eigene Quellen.**

3. Teilaufaufgabe: Tutorial_test1

Diese Aufgabe implementiert eine Klasse „SimpleArrayList“, in der die Methoden „add“, „size“, „clear“, „get“, „set“, „insert“ und „remove“ implementiert werden sollen. Nach dem Aufbau der Klasse sollen die Methode getestet werden.

- Erstellen Sie die Klasse „SimpleArrayList“
- **SimpleArrayList**
  - Verwaltet ein dynamisches Array
  - Attribute:
    - step // Größe für das Wachsen des Feldes, kann intern vergrößert werden
    - feld // internes Array
    - n // aktuelle Größe des Feldes für den Anwender
  - Methoden:
    - add (Object obj)
      - Prüft, ob die Größe des Feldes ausreicht. Im „Fehlerfall“ muss ein neues Feld angelegt werden. Die Daten werden kopiert und das globale Feld bekommt eine neue Referenz.
    - size()
    - remove(index)
      - Erzeugt ein neues lokales Feld. Die Daten werden kopiert und das globale Feld bekommt eine neue Referenz.
    - clear
    - get(index)
    - set(index, obj)

**Testszenario für Tutorial_test1:**

- Erstellen Sie eine Instanz der Klasse „SimpleArrayList“.
- Fügen Sie **10 Strings** in die Liste (à la „Item 123“).
- Geben Sie die Liste aus.
- Löschen Sie den 7. Eintrag.
- Geben Sie die Liste aus.
- Setzen Sie den 5. Eintrag mit „Neuer Eintrag“.
- Geben Sie die Liste aus.

**Ausgabe:**

```
Tutorial_test1:

Testausgabe
i: 0 liste.get: Item 0
i: 1 liste.get: Item 1
i: 2 liste.get: Item 2
i: 3 liste.get: Item 3
i: 4 liste.get: Item 4
i: 5 liste.get: Item 5
i: 6 liste.get: Item 6
i: 7 liste.get: Item 7
i: 8 liste.get: Item 8
i: 9 liste.get: Item 9

remove-Test
i: 0 liste.get: Item 0
i: 1 liste.get: Item 1
i: 2 liste.get: Item 2
i: 3 liste.get: Item 3
i: 4 liste.get: Item 4
i: 5 liste.get: Item 5
i: 6 liste.get: Item 6
i: 7 liste.get: Item 8
i: 8 liste.get: Item 9

get-Test
i: 0 liste.get: Item 0
i: 1 liste.get: Item 1
i: 2 liste.get: Item 2
i: 3 liste.get: Item 3
i: 4 liste.get: Item 4
i: 5 liste.get: Neuer Eintrag
i: 6 liste.get: Item 6
i: 7 liste.get: Item 8
i: 8 liste.get: Item 9
```

4. Teilaufaufgabe: Tutorial_test2

Diese Aufgabe testet Ihre eigene Klasse „SimpleArrayList“ mit der Java-Klasse „ArrayList und gibt die benötigte Zeit der einzelnen Tests aus (System.nanoTime).

**Testszenario für Tutorial_test2:**

- Erstellen Sie eine Instanz der Klasse „SimpleArrayList“.
- Verwenden Sie eine Konstante MAX mit einem Wert von 50000.
- Fügen Sie **50000 Integer** in die Liste (new Integer).
- Zeit ausgeben
- Erstellen Sie eine Instanz der Klasse „ArrayList“.
- Fügen Sie **50000 Integer** in die Liste (new Integer).
- Zeit ausgeben
- Die Zeiten können durch die „Vergrößerungstechnik“ stark variieren. Die untere MAX-Methode arbeitet durch einen zweiten Konstruktor sehr effizient.
- „addMAX“ und „addFast“ muss nicht implementiert werden.
- **Die Zeitmessung ist hier nicht exakt.**

**Ergebnisse:**

```
Test ArrayList add Step=100:
Zeit SimpleArrayList add:          27773197 ns
Zeit SimpleArrayList MAX add:      1003943 ns
Zeit SimpleArrayList addFast add:  1636218 ns
Zeit ArrayList add:                4507298 ns

Test ArrayList add Step=1000:

Zeit SimpleArrayList add:          27741307 ns
Zeit SimpleArrayList MAX add:       1250255 ns
Zeit SimpleArrayList addFast add:   1152757 ns
Zeit ArrayList add:                 1371946 ns

Test ArrayList add Step=5000:
Zeit SimpleArrayList add:           2558058 ns
Zeit SimpleArrayList MAX add:       14276959 ns
Zeit SimpleArrayList addFast:       2501244 ns
Zeit ArrayList add:                 2220477 ns müsste schneller sein
```

Resumé:

- Die Anzahl der Kopiervorgänge sind entscheidend, siehe MAX.
- Die Technik des Kopierens ist entscheidend, siehe addFast.

5. Teilaufaufgabe: Tutorial_test3

Diese Aufgabe testet Ihre eigene Klasse „SimpleArrayList“ mit der Java-Klasse „ArrayList und gibt die benötigte Zeit der einzelnen Tests aus (System.nanoTime).

**Testszenario für Tutorial_test3:**

- Erstellen Sie eine Instanz der Klasse „SimpleArrayList“.
- Verwenden Sie eine Konstante MAX mit einem Wert von 50000.
- Fügen Sie **2*50000 Integer** in die Liste (new Integer) (Initialisierung).
- Löschen Sie **50000 Integer** aus der Liste (liste.remove(i)).
- Geben Sie Zeit im Editor aus.
-
- Erstellen Sie eine Instanz der Klasse „ArrayList“.
- Fügen Sie **2*50000 Integer** in die Liste (new Integer) (Initialisierung)
- Löschen Sie **50000 Integer** aus der Liste (liste.remove(i)).
- Geben Sie Zeit im Editor aus.
- Die Zeiten sind nur durch die Kopiertechnik bestimmt.

**Ergebnisse:**

```
Zeit SimpleArrayList remove:      35705013338 ns
Zeit SimpleArrayList removeFast:   2027569402 ns
Zeit ArrayList remove:              1511985007 ns
```

Resumé:

- Die Technik des Kopierens ist entscheidend, siehe removeFast.

# Hausaufgabe 07

6. Teilaufaufgabe: Aufgabe_test1 und Aufgabe_test2

In dieser Aufgabe wird die Klasse „SimpleArrayList“ in eine generische Klasse umgebaut und in test1 und test2 benutzt.

**a) Neue Klasse „SimpleArrayList_T:**

- Erstellen Sie eine neue Klasse „SimpleArrayList_T“.
- Kopieren Sie den Inhalt aus der Klasse „SimpleArrayList“ (inkl. Änderungen).
- Das interne Feld kann weiterhin vom Typ Objekt sein.
- Wandeln Sie die Klasse in eine generische Klasse um.
    - Klassendefinition
    - Methode get
    - Methode set
    - Methode add

**b) Methode Aufgabe_test1:**

- Erstellen Sie eine Instanz der Klasse „SimpleList_T“ mit dem Datentyp **Integer**.
- Fügen Sie fünfzehn Elemente in die Liste.
- Geben Sie die komplette Liste aus.

**c) Methode Aufgabe_test2:**

- Erstellen Sie eine Instanz der Klasse „SimpleList_T“ mit dem Datentyp **String**.
- Fügen Sie fünfzehn Elemente in die Liste.
- Fügen Sie einen String in die Liste (probehalber).
- Geben Sie die komplette Liste aus.

**Ergebnis:**

```
1. Test Generic SimpleArrayList_T
mit Integer:

i: 0    get(): 0
i: 1    get(): 1
i: 2    get(): 2
i: 3    get(): 3
i: 4    get(): 4
i: 5    get(): 5
i: 6    get(): 6
i: 7    get(): 7
i: 8    get(): 8
i: 9    get(): 9
i: 10    get(): 10
i: 11    get(): 11
i: 12    get(): 12
i: 13    get(): 13
i: 14    get(): 14
```

```
2. Test Generic SimpleArrayList_T
mit String:

i: 0    get(): "Node 0"
i: 1    get(): "Node 1"
i: 2    get(): "Node 2"
i: 3    get(): "Node 3"
i: 4    get(): "Node 4"
i: 5    get(): "Node 5"
i: 6    get(): "Node 6"
i: 7    get(): "Node 7"
i: 8    get(): "Node 8"
i: 9    get(): "Node 9"
i: 10    get(): "Node 10"
i: 11    get(): "Node 11"
i: 12    get(): "Node 12"
i: 13    get(): "Node 13"
i: 14    get(): "Node 14"
```

7. Teilaufaufgabe: Aufgabe_test3

In dieser Aufgabe sollen fast alle Klassen des Collection-Frameworks getestet werden. Da die Listen die Schnittstelle „List“ und die verschiedenen Set die Schnittstelle „Set“ implementieren, benötigt man nur zwei Testmethoden (testList und testSet).

```
private void Aufgabe_test3() {
  MAX =100000
}  // Aufgabe_test3

private void testList(java.util.List liste) {
  // append
  // get
  // remove
}
private void testSet(java.util.Set set) {
  // append
  // iterator
  // remove
}
```

**a) Methode Aufgabe_test3:**

- Erstellen Sie jeweils eine Instanz der folgenden Klassen und rufen die Testmethode auf.
- ArrayList
- ArrayList mit MAX
- Vector
- Vector mit MAX
- LinkedList
- HashSet
- TreeSet
- LinkedHashSet

**b) Methode testList:**

- Programmieren Sie folgende Aufgaben und berechnen Sie den Zeitaufwand.
- Eintragen von MAX-Elementen
- Ausgabe von MAX-Elementen (get)
- Löschen der **Hälfte** der Elemente mittels Index (einzeln in einer For-Schleife).
    - Hier kurz nachdenken

**c) Methode testSet:**

- Programmieren Sie folgende Aufgaben und berechnen Sie den Zeitaufwand.
- Eintragen von MAX-Elementen
- Ausgabe mittels eines Iterators
    - siehe Anhang
- Löschen der **Hälfte** der Elemente mit jeweils einem Objekt (For-Schleife).

**Ergebnis:**

**Performance-Test**

**Test mit Klasse: class java.util.ArrayList**
Zeit add: 38193826 ns
Zeit get: **221755** ns
Zeit remove: 391171678 ns

**Test mit Klasse: MAX class java.util.ArrayList**
Zeit add: 45505861 ns
Zeit get: 209292 ns
Zeit remove: 371362218 ns

**Test mit Klasse: class java.util.Vector**
Zeit add: 19923452 ns
Zeit get: 1492902 ns
Zeit remove: 399747163 ns

**Test mit Klasse: MAX class java.util.Vector**
Zeit add: 40534527 ns
Zeit get: 295061 ns
Zeit remove: 369722334 ns

**Test mit Klasse: class java.util.LinkedList**
Zeit add: 25591938 ns
Zeit get: **14505668852** ns
Zeit remove: 14272106360 ns

**Test mit Klasse: class java.util.HashSet**
Zeit add: 69696163 ns
Zeit get: 3061312 ns
Zeit remove: 25021241 ns

**Test mit Klasse: class java.util.TreeSet**
Zeit add: 83543542 ns
Zeit get: 3906178 ns
Zeit remove: 24519453 ns

**Test mit Klasse: class java.util.LinkedHashSet**
Zeit add: 32761024 ns
Zeit get: 2694042 ns
Zeit remove: 105590340 ns

# Listen, Set und generische Klassen in Java

## Interface List extends Collection

- Geordnete Menge, Sequence
- Kann Elemente doppelt speichern
- Variable Länge
- Schnelles Einfügen und Löschen

**Wichtige Methoden:**

| **Name** | **Beschreibung** |
|---|---|
| boolean add(E e) | Fügt ein neues Element ans Ende |
| void add(int index, E element) | Fügt ein neues Element an einer bestimmten Position |
| void clear() | Löscht die Collection |
| boolean contains(Object o) | Prüft, ob ein Element in der Liste ist |
| boolean equals(Object o) | Parameter ist eine Liste. Prüft, ob die Listen identisch sind |
| E get(int index) | Gibt das it-te Element zurück |
| int indexOf(Object o) | Gibt den Index des Objektes in der Liste aus |
| boolean isEmpty() | Ist die Liste leer? |
| Iterator<E> iterator() | Gibt einen Iterator zurück |
| int lastIndexOf(Object o) | Gibt den letzten Index des Objektes in der Liste aus |
| ListIterator<E> listIterator() | Gibt einen Iterator zurück |
| E remove(int index) | Löscht das i-te Element |
| boolean remove(Object o) | Löscht das Objekt |
| int size() | Gibt die Anzahl der Elemente zurück |
| List<E> subList(int fromIndex, int toIndex) | Gibt eine Teilliste zurück |
| Object[] toArray() | Rückgabe der Liste als Array (for-each) |

**Klassen die diese Schnittstellen implementieren:**

- ArrayList
- Vector
- LinkedList
- RoleList
- RoleUnresolvedList
- Stack

**Algorithmen:**

- sort(List)
- binarySearch(List, Object)
- reverse(List)
- shuffle(List)
- fill(List, Object)
- copy(List dest, List src)
- min(Collection)
- max(Collection)
- rotate(List list, int distance)
- replaceAll(List list, Object oldVal, Object newVal)
- indexOfSubList(List source, List target)
- lastIndexOfSubList(List source, List target)
- swap(List, int, int)
- frequency(Collection, Object) – Bestimmt die Anzahl, wie oft ein Objekt vorhanden ist.

- disjoint(Collection, Collection) - Unterschiede
- addAll(Collection<? super T>, T...) - Adds all of the elements in the specified array to the specified collection.
- newSetFromMap(Map) - Creates a general purpose Set implementation from a general purpose Map
- asLifoQueue(Deque) - Returns a view of a Deque as a Last-in-first-out (Lifo) Queue

## Interface Set extends Collection

- Ist eine Menge
- Kann Elemente **nicht** doppelt speichern
- Schnelle Suche

**Wichtige Methoden:**

| **Name** | **Beschreibung** |
|---|---|
| boolean add(E e) | Fügt ein neues Element ans Ende |
| void add(int index, E element) | Fügt ein neues Element an einer bestimmten Position |
| void clear() | Löscht die Collection |
| boolean contains(Object o) | Prüft, ob ein Element in der Liste ist |
| boolean isEmpty() | Ist die Liste leer? |
| Iterator<E> iterator() | Gibt einen Iterator zurück |
| boolean remove(Object o) | Löscht das Objekt |
| int size() | Gibt die Anzahl der Elemente zurück |

**Klassen die diese Schnittstellen implementieren:**

- HashSet Schnelle Mengenimplementierung durch eine interne HashMap.
- TreeSet Die Mengen werden durch **balancierte** Binärbäume realisiert (inkl. Sortierung).
- LinkedHashSet Schnelle Mengenimplementierung unter Beibehaltung der Einfügereihenfolge.
- EnumSet Eine spezielle Menge ausschließlich für Enum-Objekte.
- CopyOnWriteArraySet: Schnelle Datenstruktur für viele lesende Operationen durch **Threads**.
    - import java.util.concurrent.CopyOnWriteArraySet;
    - nur für kleine Mengen von Datensätzen
    - Datensätze können verändert werden (synchronized: Ampelsteuerung!!!)

## Vorteile der Architektur: Projekt „Prog2_TreeTest“:

```
 private void test6() {
  editor.setText("Test List Methoden\n\n");
  java.util.ArrayList<String> liste1 = new ArrayList<>();
  listMethoden(liste1);

  java.util.Vector <String> liste2 = new Vector<>();
  listMethoden(liste2);

  java.util.LinkedList<String> liste3 = new LinkedList<>();
  listMethoden(liste3);
 }


private void listMethoden(java.util.List liste) {
    editor.append("\nTest mit Klasse: "+  liste.getClass()+  "\n" );
    liste.add("Node 1");
    liste.add("Node 2");
```

```
    liste.add("Node 3");
    liste.add("Node 4");

    Object obj =  liste.get(0);
    obj =  liste.get(1);

    liste.remove(0);
    liste.remove(1);
  }


private void test7() {
    editor.setText("\n\nTest Set Methoden\n\n");
   java.util.HashSet<String> liste1 = new HashSet<>();
    setMethoden(liste1);

    java.util.TreeSet<String> liste2 = new TreeSet<>();
    setMethoden(liste2);

    // Menge und geordnete Liste
    java.util.LinkedHashSet<String> liste3 = new  LinkedHashSet<>();
    setMethoden(liste3);
  }


 private void setMethoden(java.util.Set set) {
    editor.append("\nTest mit Klasse: "+   set.getClass()+   "\n" );
     set.add( new Integer(1) );
     set.add( new Integer(2) );
     set.add( new Integer(3) );
     set.add( new Integer(4) );

    Iterator it = set.iterator();
    while (it.hasNext()) {
      it.next();
    }


    it = set.iterator();
    while (it.hasNext()) {
      Integer I =   (Integer)   it.next();
      editor.append("items: "+ I.toString()+  "\n" );
      //it.next();
    }
      set.remove( new Integer(1) );
      set.remove( new Integer(4) );

  }
```

## 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

## Generische Klassen, hier mit einer internen Klasse Item

```
public class TestGeneric<T> {

  private Item<T>[] feld;

  public TestGeneric(int size) {
    feld = new Item [size];
    for (int i = 0; i < feld.length; i++) {
      feld[i] = null;
    }
  }

  public void insert(int i, T data) {
    if (i >=0  &&  i<feld.length)
      feld[i]= new Item<>(data);
    else
      throw new IndexOutOfBoundsException("Fehlerhafter Index: " + i + "   Bereich: 0 bis " + (feld.length - 1));
  }

  public int length() {
    return feld.length;
  }

  public T get(int i) {
    if (i >= 0 && i < feld.length) {
      return feld[i].getData();
    } else {
      throw new IndexOutOfBoundsException("Fehlerhafter Index: " + i + "   Bereich: 0 bis " + (feld.length - 1));
    }
  }

  private class Item <T> {

    private T data;

    public Item (T data) {
      this.data = data;
    }

    public T getData() {
      return data;
    }

    public void setData(T data) {
      this.data = data;
    }
  }
}  // TestGeneric
```

```
private void test1() {
    final int MAX = 10;
    editor.setText("Test bezüglich Generic:\n");

    TestGeneric<Double> feld = new TestGeneric(MAX);

    for (int i = 0; i < feld.length(); i++) {
      feld.insert(i, 1200.4567 + i);
    }

    for (int i = 0; i < feld.length(); i++) {
      editor.append("i: " + i + "    get(): " + feld.get(i) + "\n");
    }

    editor.append("\nende");
  }  // Aufgabe_test1
```