Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Technik & Architektur

Programmieren 2

Test 1, Frühlingssemester 2009

R. Diehl, R. Gisler, M. Klaper; Version 1.0

Name: Galliker Kurs: 3

Vorname: Thomas

Rahmenbedingungen:

1. **Prüfungszeit: 90 Minuten**
2. In diesem Test können 68 Punkte erreicht werden. Jeder Aufgabe ist eine maximal erreichbare Punktezahl zugeordnet.
3. Schreiben Sie Ihren Namen und Ihren Kurs auf dieses Blatt. Blätter ohne Namensangabe werden nicht bewertet.
4. Es handelt sich um einen schriftliche Test ohne Einsatz des Computers oder elektronischer Hilfsmittel. Sie dürfen keine Unterlagen verwenden.
5. Sollte die Problemstellung Unklarheiten aufweisen, dürfen Sie eigene Annahmen treffen. Führen Sie diese in der Lösung auf.
6. Schreiben Sie möglichst verständlich und gut leserlich. Missverständliche Lösungen werden nicht berücksichtigt.
7. Benutzen Sie den Freiraum unter den Aufgaben für Ihre Lösung.

Für die Korrektur (nicht ausfüllen!)

1	2	3	4	5	6	7	Punkte	Visum
3	5	5	5	6	1	0.5	57	
8	**9**	**10**	**11**	**12**	**13**	**14**		
5	8	3.5	7	2	2	4		

Inhaltsverzeichnis:

Aufgabe 1: Vererbung in Java (6 Punkte) 3
Aufgabe 2: Interfaces (5 Punkte) 4
Aufgabe 3: Polymorphie (5 Punkte) 5
Aufgabe 4: Vererbung in Java II (5 Punkte) 6
Aufgabe 5: Typhierarchie (6 Punkte) 7
Aufgabe 6: equals() und hashCode() (3 Punkte) 8
Aufgabe 7: Hashtabellen (2 Punkte) 8
Aufgabe 8: equals() (5 Punkte) 9
Aufgabe 9: Implemention und Verwendung einer Queue (8 Punkte) 10
Aufgabe 10: Bäume: Definitionen (4 Punkte) 12
Aufgabe 11: Binäre Suchbäume (8 Punkte) 13
Aufgabe 12: AVL Bäume (5 Punkte) 15
Aufgabe 13: Ereignissteuerung (2 Punkte) 16
Aufgabe 14: GUI Komponenten (4 Punkte) 16

Aufgabe 1: Vererbung in Java (6 Punkte)

3

Beantworten Sie die folgenden Multiple Choice Fragen zur Vererbung in Java.

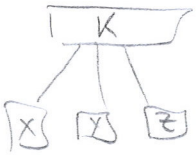
Hinweis: Kreuzen Sie alle Aussagen an, die richtig sind.
Pro richtiges Kreuz erhalten Sie 0.5 Punkte. Für ein falsches Kreuz erhalten Sie -1 Punkt.
Bei dieser Aufgabe können Sie nicht weniger als 0 Punkte erhalten.

Eine abstrakte Klasse A

- ☒ definiert einen Typ. Bsp: `A myVar;`
- ☐ kann nur ein Interface implementieren.
- ☒ kann von einer konkreten Klasse abgeleitet werden.
- -0,5 ! ☒ muss abstrakte Methoden beinhalten.
- ! ☒ darf keine Attribute haben.
- ☐ kann instantiiert werden. Bsp: `a = new A();`
- ✓ ☒ kann abstrakte Methoden beinhalten.

Eine konkrete Klasse K

- ☐ kann abstrakte Methoden beinhalten.
- ☒ kann von einer abstrakten Klasse abgeleitet werden.
- 2,5 ☒ kann mehrere Interfaces implementieren.
- ☒ kann mehrere Unterklassen haben. Bsp: `X extends K`
- ☒ kann instantiiert werden. Bsp: `k = new K();`
- ☒ definiert einen Typ. Bsp: `K myVar;`
- ☐ kann nur eine Unterklasse haben. Bsp: `X extends K`



Ein Interface I

- ☐ kann Instanzvariablen beinhalten.
- ☒ kann abstrakte Methoden beinhalten. => ??? sind ohnehin schon alle "public abstract"!
- ☐ kann Klassenvariablen beinhalten.
- 1 ☒ kann Klassenkonstanten beinhalten. Bsp: `static final...`
- ☐ kann Konstanten (`final`) beinhalten.
- ☒ dient als Spezifikation (WAS).
- ☐ kann private Methoden beinhalten.
- ☒ definiert einen Typ. Bsp: `I myVar;`
- ☐ kann Klassenmethoden beinhalten.
- ☐ kann instantiiert werden. Bsp: `i = new I();`

Aufgabe 2: Interfaces (5 Punkte)

5

Gegeben ist die folgende Klasse:

implements IMitarbeiter

```
public class Dozent {
    private long String id;
    private String name;

    public Dozent(long id, String name) {
        this.id = id;
        this.name = name;
    }

    public long getId() {
        return this.id;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void print() {
        System.out.println(this.name + " (" + this.id + ")");
    }
}
```

4 a) Schreiben Sie ein Interface `IMitarbeiter`, das *alle* Methoden ausser der Methode `print()` der Klasse `Dozent` definiert. (4 Punkte)

```
public interface IMitarbeiter
{
    long getId();
    String getName();
    void setName(String name);
}
```

1 b) Ändern Sie die Klasse `Dozent` direkt im Code so ab, dass diese das Interface `IMitarbeiter` implementiert. (1 Punkt)

Aufgabe 3: Polymorphie (5 Punkte)

5

Gegeben ist folgende Klasse:

```
public class Person
{
        private String name;
        private int age;

        public Person(String name, int age) {
                this.name = name;
                this.age = age;
        }

        public String toString() {
            return (name + " " + age);
        }
        // other methods omitted
}
```

3 a) Ergänzen Sie in der folgenden Klasse den Konstruktor und die Methode `toString()`. Der Konstruktor soll *alle* Instanzvariablen korrekt initialisieren und die Methode `toString()` soll *alle* Instanzvariablen ausgeben. (3 Punkte)

```
public class Student extends Person
{
    private String fach;

    public Student(String name, int age, String fach) {
      super(name, age);
      this.fach = fach;

    }

    public String toString() {
      return (super.toString() + " " + fach);
    }
}
```

1 b) Ergänzen Sie die folgenden Sätze:

Die Methode `toString()` in der Klasse `Student` überschreibt die Methode `toString()` der Klasse Person. (0.5 Punkte)

Es handelt sich dabei um Statische Polymorphie. (0.5 Punkte)

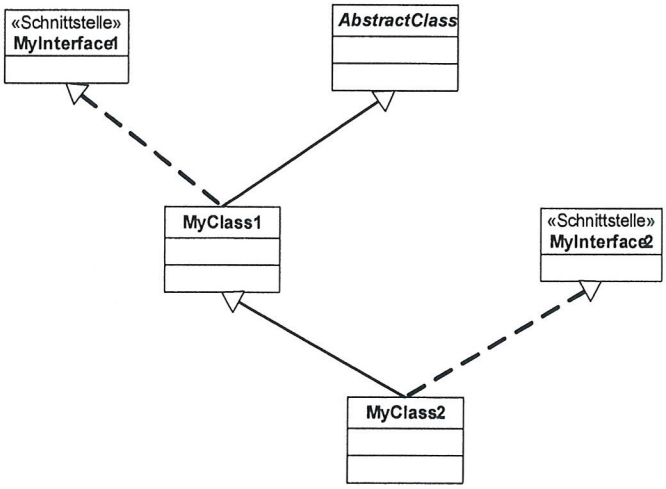
1 c) Für welche Typen ist die Methode `toString()` definiert? (1 Punkt)

- Object
  - Person
    - Student

Aufgabe 4: Vererbung in Java II (5 Punkte)

5

Gegeben ist folgendes Klassendiagramm:



Geben Sie für jede der oben stehenden Klassen und Schnittstellen (Interfaces) nur den Klassenrahmen[1] an, d.h. nur die Definition einer Klasse *ohne* Felder, Konstruktoren und Methoden. (5 Punkte)

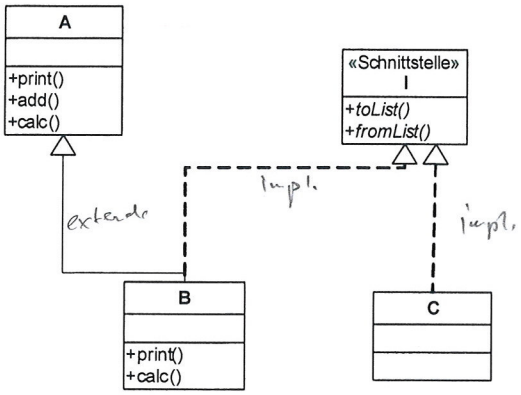
```
public interface MyInterface1
{...}
public abstract class AbstractClass
{...}
public class MyClass1 extends AbstractClass implements MyInterface1
{...}
public interface MyInterface2
{...}
public class MyClass2 extends MyClass1 implements MyInterface2
{...}
```

[1] Beispiel: Der Klassenrahmen der Klasse `Object` lautet: `public class Object {...}`

Aufgabe 5: Typhierarchie (6 Punkte)

6

Gegeben ist folgendes Klassendiagramm:



Gegeben sind ausserdem die folgenden Deklarationen:

```
A  a;
B  b;
I  i;
```

Entscheiden Sie für die folgenden Anweisungen, ob sie *compilieren* (je 0.5 Punkte):

2

`b = new A();`	☐ compiliert	☒ compiliert nicht
`a = new B();`	☒ compiliert	☐ compiliert nicht
`i = new C();`	☒ compiliert	☐ compiliert nicht
`i = new I();`	☐ compiliert	☒ compiliert nicht

Gegeben sind folgende Anweisungen:

```
A a1 = new A();
A a2 = new B();
I i1 = new C();
```

Ergänzen Sie die folgende Tabelle: Bestimmen Sie für die folgenden Methodenaufrufe jeweils den statischen und den dynamischen Typ der Variable und geben Sie an, welche Implementation (d.h. von welcher Klasse) ausgeführt wird (pro vollständig korrekter Zeile je 1 Punkt)

4

	statischer Typ	dynamischer Typ	Methodenimplementation der Klasse …
`a1.print()`	A	A	A
`a2.print()`	A	B	B
`a2.add()`	A	B	A
`i1.toList()`	I	C	C

✓

Vor Ausführung definiert: A a2;

Zur Laufzeit initialisiert: a2 = new B();

Aufgabe 6: equals() und hashCode() (3 Punkte)

1

Sind untenstehende Aussagen richtig? Kreuzen Sie entsprechend an.

Hinweis: Jedes richtige Kreuz ergibt 0.5 Punkte; jedes falsche -0.5 Punkte. Insgesamt weniger als Null Punkte gibt es aber nicht!

- ☒ ja ☐ nein `equals()` ist eine Methode der Klasse `Object`.
- ☒ ja ☒ nein Eine Klasse muss `equals()` überschreiben, wenn sie einen "Entity Type" repräsentiert. Identitätsvergleich wird in Object/equals() durchgeführt ?
- ☐ ja ☒ nein `equals()` liefert bei Gleichheit den int-Wert `0` zurück. boolean!
- ☒ ja ☐ nein Die Klasse `HashSet` baut darauf auf, dass `equals()` adäquat implementiert ist.
- ☒ ja ☐ nein `hashCode()` muss für gleiche Objekte zwingend denselben Rückgabewert liefern. Nicht zwingend => muss aber dokumentiert werden
- ☒ ja ☒ nein `hashCode()` muss für ungleiche Objekte zwingend verschiedene Rückgabewerte liefern.

Aufgabe 7: Hashtabellen (2 Punkte)

0.5

Inwiefern sind Hashtabellen vorteilhafte Datenstrukturen?
Erklären Sie kurz:

- HashTables enthalten eineindeutige Keys zur Identifizierung eines Datensatzes. Es wird gewährleistet, dass keine Doppeleinträge vorkommen.
- HashTabellen besitzen effiziente Zugriffsoperationen mit praktisch O(1) für Einfügen, Entfernen und Löschen
- Unabhängig von der Anzahl Elemente (also O(1))

Aufgabe 8: equals() (5 Punkte)

Ballone werden als "gleich" betrachtet, wenn sie in Farbe und Grösse übereinstimmen. Korrigieren Sie unmissverständlich die 3 Fehler in untenstehender Methode.

```
public class Balloon
{
    private Color color;
    private int size;

    ...

    public boolean equals(Object other)
    {
        if (this == other)
            return true;
        if (other == null)
            return ~~true~~; false
        if (other.getClass() != this.getClass())
            return false;
        if (!color.equals(other.color) && !size.equals(other.size))
            return false;
        if (size != other.size)
            return false;
        return true;
    }

}
```

2

3 ⇒ if(!color.equals(((Balloon)other).color) && !size.equals(((Balloon)other).size))
return false

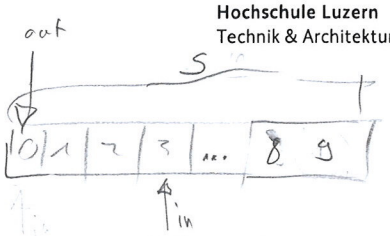
⇒ Zugriff auf color und size verletzen das Konzept von Information Hiding.

⇒ Gleiche Klassen können auf private-Attribute anderer Objekte zugreifen.

⇒ Das ist schlecht, aber es hat ja niemand behauptet, dass Java gut ist! :)

8

Hochschule Luzern
Technik & Architektur



FiFo

Aufgabe 9: Implemention und Verwendung einer Queue (8 Punkte)

a) Implementieren Sie in der gegebenen Klasse `MyQueue` die Methode `enqueue()`. `enqueue()` und `dequeue()` sollen nicht prüfen, ob die Queue voll bzw. leer ist. (4 Punkte)

```
public class MyQueue
{
    private int size;           // Anzahl Plaetze der Queue
    private int n = 0;          // Anzahl Elemente in der Queue (≙ belegte El.)
    private int in = 0;         // Zeiger auf leeren Platz
    private int out = 0;        // Zeiger auf belegten Platz
    private Object[] queue;

    public MyQueue(int s)
    {
        size = s;
        queue = new Object[size];
    }

    public void enqueue(Object obj)
    {
1       n++;
1       if(in == size) {
            in = 0;
        }
1       queue[in] = obj;
1       in++;



    }

    public Object dequeue()
    {
        n--;
        if(out == size) {
            out = 0;
        }
        Object o = queue[out];
        out++;
        return o;
    }

    public boolean isEmpty() { return (n == 0); }

    public boolean isFull()  { return (n == size); }
}
```

b) In einem Programm gibt es folgende Definition:

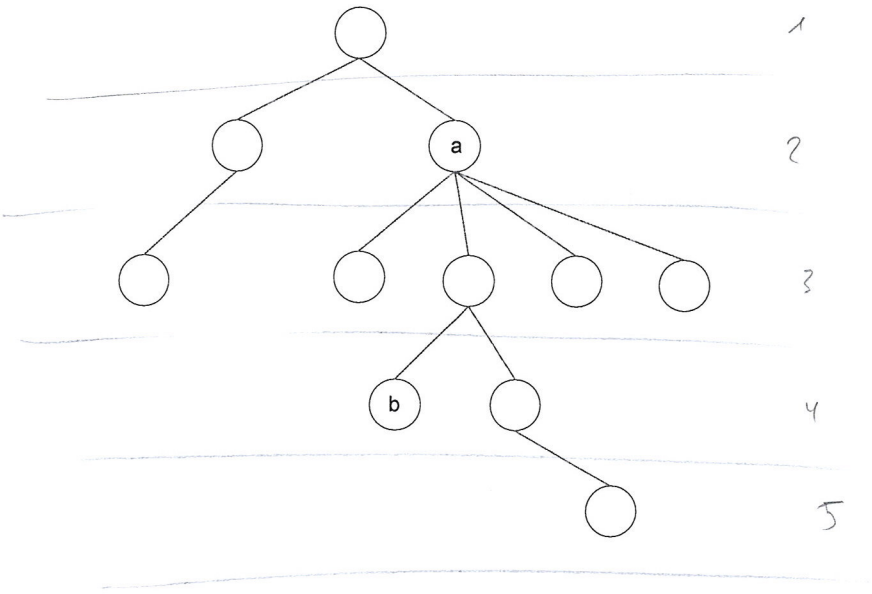
```
MyQueue myQueue = new MyQueue(12);
```

Zu einem späteren Zeitpunkt möchten Sie im Programm das Objekt `myObject` in die Queue `myQueue` einfügen. Schreiben Sie das entsprechende Code Fragment. Achten Sie darauf, dass zur Laufzeit kein Fehler entstehen kann. (4 Punkte)

```
try {
2       if(!myQueue.isFull())
        {
2         myQueue.enqueue(myObject);
        }
    }
    catch(Exception ex)
    {
      //...
    }
```

Aufgabe 10: Bäume: Definitionen (4 Punkte)

Gegeben ist folgender Baum:



Bestimmen Sie die folgenden Werte (je 1 Punkt):

- den Grad des Knoten a:

1 4

- die Ordnung des Baumes:

0.5 4 ⟹ >=4 (mind. 4)

- die Tiefe von Knoten a:

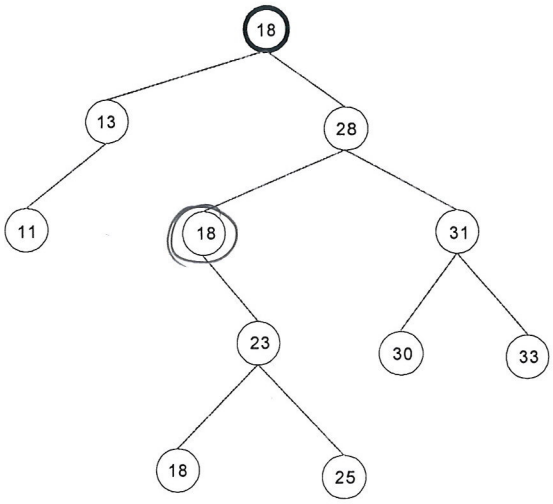
1 2

- die Höhe des Baumes:

1 5

Aufgabe 11: Binäre Suchbäume (8 Punkte)

Gegeben ist folgender, binärer Suchbaum:



1. Geben Sie die Ausgabe des Baumes in der Inorder Durchlaufordnung an. (1 Punkt)

   11, 13, ~~18, 25, 23, 18, 30, 33, 31, 28, 18~~

   18, 18, 18, 23, 25, 28, 30, 31, 33

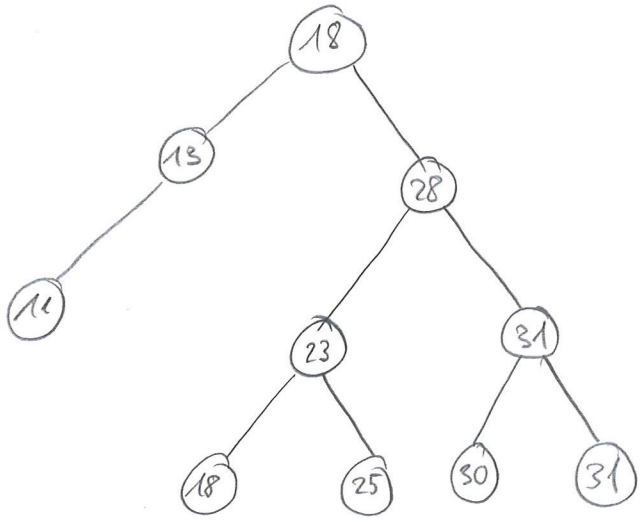
2. Geben Sie die Ausgabe des Baumes in der Preoder Durchlaufordnung an. (2 Punkte)

   2 18, 13, 11, 28, 18, 23, 18, 25, 31, 30, 33

3. In obigem Baum wird die Wurzel gelöscht (der Knoten ist fett umrandet).

   1
   - Markieren Sie in obigem Baum *deutlich* den Knoten, der die neue Wurzel wird. (1 Punkt)
   - Zeichnen Sie den Baum, der nach Löschen der Wurzel entsteht. (2 Punkte)

2



4. Welches der drei Codestück führt eine Preorder Traversierung aus? (2 Punkte)

2

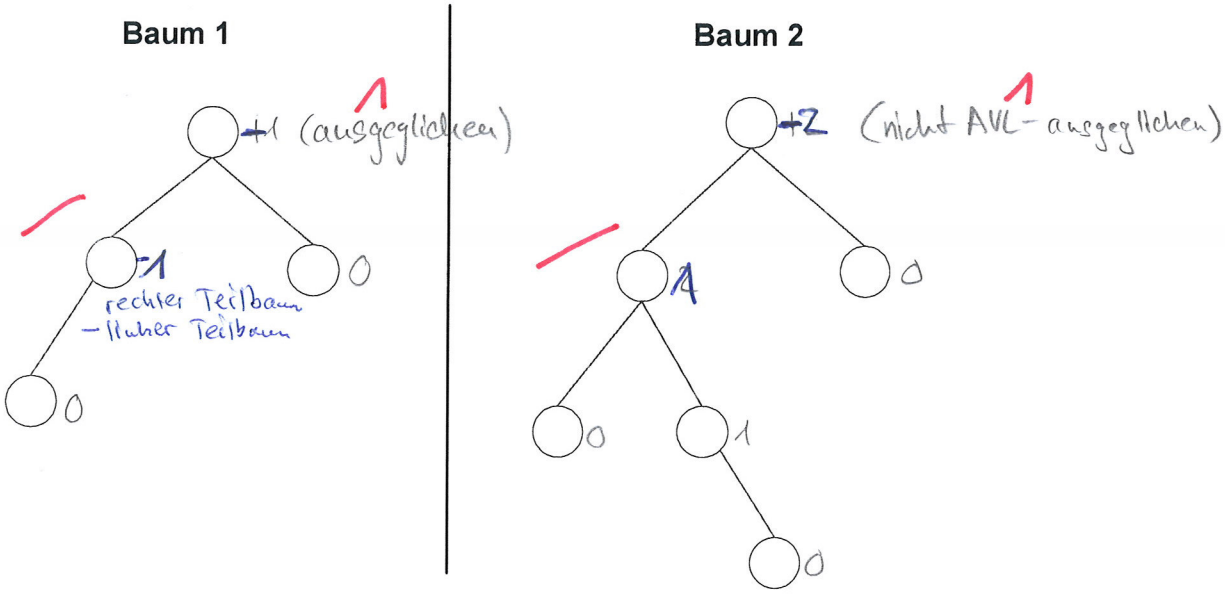
A	`private void aOrder(BinaryNode node)`
	`{`
	`  if (node!=null) {`
	`  System.out.print(node.key + " ");`
	`  aOrder (node.left);`
	`  aOrder (node.right);`
	`  }`
	`}`

B	`private void bOrder(BinaryNode node)`
	`{`
	`  if (node!=null) {`
	`  bOrder (node.left);`
	`  bOrder (node.right);`
	`  System.out.print(node.key + " ");`
	`  }`
	`}`

C	`private void cOrder(BinaryNode node)`
	`{`
	`  if (node!=null) {`
	`  cOrder (node.left);`
	`  System.out.print(node.key + " ");`
	`  cOrder (node.right);`
	`  }`
	`}`

Aufgabe 12: AVL Bäume (5 Punkte)

Gegeben sind die beiden folgenden Bäume:



1. Schreiben Sie rechts neben jeden Knoten den Balancefaktor des Knotens. (3 Punkte)

2. Geben Sie für *beide* Bäume an, ob sie AVL-ausgeglichen sind. (2 Punkte)

Aufgabe 13: Ereignissteuerung (2 Punkte)

Ordnen Sie die folgenden Merkmale [s]equentiellen bzw. [e]reignisgesteuerten Programmen zu:

2

[e] Nach der Ausführung fällt das Programm erneut in den "Schlafzustand".

[s] Benutzer/in kann nur bei Eingabeaufforderungen des Programms in den Ablauf eingreifen.

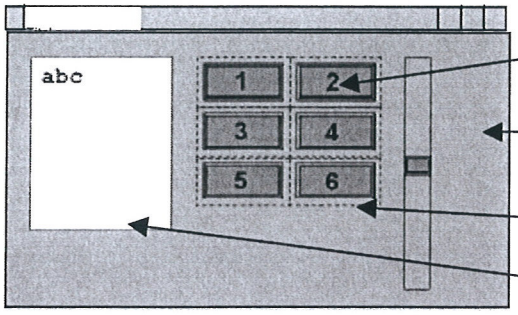
[s] fester Ablauf

[e] Was geschieht/ausgeführt wird, hängt vom Event ab.

Aufgabe 14: GUI Komponenten (4 Punkte)

Benennen Sie die im Bild mit Pfeilen markierten GUI Komponenten (AWT) und geben Sie zu jeder Komponente an, ob es sich dabei um eine elementare Komponente oder einen Container handelt:

4



Bezeichnung	elementar / Container
Button	elementar
Frame	Container
Panel mit GridLayout	Container
TextArea	elementar

---- Ende Test 1 ----