

## Musterartefakt: Einführung in die OO-Programmierung A23

**QUARTAL: II/2024** 

Dauer: 60 Minuten Anzahl Seiten ohne Deckblatt: 12 Datum: 01.01.1750

Hilfsmittel: keine.



1.	(3 Punkte) Was sind korrekte Initialisierungen (vom Compiler akzeptierte Initialisierungen von Instanzen von Collection Klassen?  Bewertung: Richtig = +,5; Falsch = -0,5; Min = 0				
	$\bigcirc$	private	ArrayList< <b>int</b> > numbers = <b>new</b> ArrayList<>();		
	$\bigcirc$	private	ArrayList <string> buchstaben = <b>new</b> HashMap&lt;&gt;();</string>		
	0	<pre>private &gt;();</pre>	List <string> buchstaben = <b>new</b> ArrayList<string< td=""></string<></string>		
	0	private <>()	ArrayList <string> buchstaben = <b>new</b> ArrayList;</string>		
	0	_	<pre>a ArrayList<string> buchstaben = new HashMap</string></pre>		
	$\circ$	•	<pre>e ArrayList<integer> numbers = new HashMap&lt; ger&gt;();</integer></pre>		
2.		e) Welche sind richti	e der folgenden Aussagen über Interfaces, Klassen und abstrakten g?		
	$\circ$		s und abstrakte Klassen haben gemeinsam, dass keine Instanzen n erstellt werden können.		
<ul> <li>Interfaces können Exemplarvariablen definieren, abstrakte Klassen nicht</li> </ul>					
	<ul> <li>Interfaces definieren keine Typen, dass gilt nur für abstrakte Klassen.</li> </ul>				
<ul> <li>Nur Klassen können von Interfaces erben, abstrakte Klassen nicht.</li> </ul>					
	$\bigcirc$	s definieren Typen.			
	0	ein bestir	nplar einer Klasse, die von einer abstrakten Klasse erbt, die wiederum nmtes Interface X implementiert, kann einer Variablen zugewiesen die mit dem Typ des Interfaces X deklariert ist.		
	$\bigcirc$		e Klassen müssen nicht alle Methoden der von ihnen implementierten simplementieren.		
	$\bigcirc$	Interfaces	s erlauben keine Mehrfachvererbung.		
	$\circ$		erweitern (extends) und implementieren nicht (implements) Interfaces. ementieren lediglich abstrakte Methoden abstrakter Vorfahren.		



## 3. (5 Punkte) Vervollständigen Sie den Text

Eine Schnittstelle beinhaltet die der Methoden, die eine die Schnittstelle Klasse zur Verfügung stellen muss. Schnittstellen unterstützen, im . Schnittstellen können bei der Deklaration Gegensatz zu Klassen, von Variablen als verwendet werden. Es gilt als guter Stil gegen Schnittstellen zu programmieren – der Quellcode wird dadurch unabhängig von der jeder konkreten der Schnittstelle. Eine Schnittstelle kann von jeder implementiert erlaubt es jeden Subtyp einer Schnittstelle werden, muss es aber nicht. Die angegeben ist. Schnittstellen helfen dort zu verwenden wo die Schnittstelle als zwischen Bestandteilen eines Programms zu das Grundprinzip der befolgen. Schnittstellendefinitionen werden nicht mit dem Schlüsselwort class, sondern mit dem Schlüssen , eingeleitet.



4. (6 Punkte) Gegeben seien folgendes Interface und folgende Klassen:

```
public interface IDoSomething{
    void doSomething();
}

public class DoSomething implements IDoSomething{
    public void doSomething() {
        System.out.println("Working hard");
    }
}

public class DoSomethingElse implements IDoSomething {
    public void doSomething() {
        System.out.println("Working even harder");
    }
}
```

Welche der Anweisungsfolgen sind richtig?

Kreuzen Sie für jede Anweisungsfolge entweder richtig oder falsch an.

- Für jede richtig gesetzte Markierung erhalten Sie einen Punkt (+1).
- Falsch oder nicht markierte Aussagen geben 0 Punkte.

DoSomething doit = <b>new</b> DoSomething(); IDoSomething doit2 = doit;	<ul><li>☐ richtig</li><li>☐ falsch</li></ul>
<pre>IDoSomething doit = new DoSomething(); DoSomething doit2 = doit;</pre>	□ richtig □ falsch
IDoSomething doit = <b>new</b> IDoSomething();	<ul><li>☐ richtig</li><li>☐ falsch</li></ul>
IDoSomething doit = <b>new</b> DoSomethingElse();	□ richtig □ falsch
IDoSomething doit = <b>new</b> DoSomething();	<ul><li>☐ richtig</li><li>☐ falsch</li></ul>
DoSomething doit = <b>new</b> DoSomethingElse();	<ul><li>□ richtig</li><li>□ falsch</li></ul>

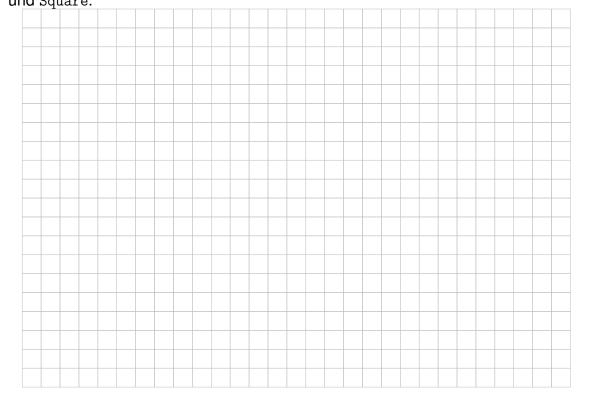


5. Betrachten Sie das folgende Code-Beispiel:

```
public abstract class Shape {
  private int posX, posY;
  protected Shape(int posX, int posY) {
    this.posX = posX;
    this.posY = posY;
  public abstract int getArea();
public class Rectangle extends Shape {
  private int len1, len2;
  public Rectangle(int posX, int posY, int len1, int len2) {
    super(posX, posY);
    this.len1 = len1;
    this.len2 = len2;
  public int getArea() {
    return len1 * len2;
}
public class Square extends Rectangle {
  public Square(int posX, int posY, int len) {
    super(posX, posY, len, len);
  public int getArea(int factor) {
    return factor * getArea();
}
```

(5.2) (4 Punkte) Überschreiben Sie die parameterlose Methode getArea() in der Subklasse Square.

Diskutieren Sie die Sinnfälligkeit des Überschreibens und der Status von Rectangle und Square.





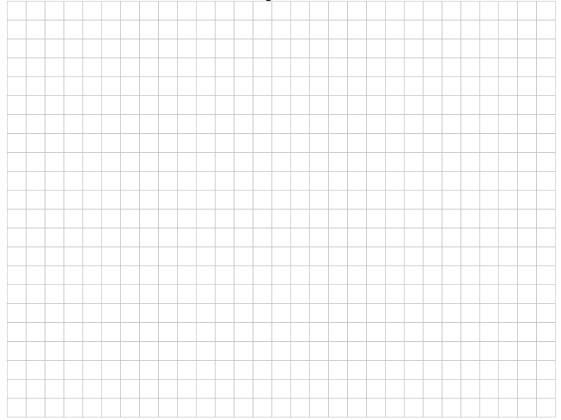
6. Das *Decorator* Entwurfsmuster ergänzt eine Methode um zusätzliches Verhalten. Dieses Entwufsmuster soll folgendermaßen Anwendung finden: Examplare einer Subklasse BachelorStudent der gegebenen Klasse Student sollen beim Erreichen des Abschlusses (Methode graduate()) zusätzlich ihren Hut in die Luft werfen.

```
public class Student {
   private ExaminationOfficeService examOffice;

public void graduate() {
   examOffice.graduate(this);
  }
}
```

public class BachelorStudent extends Student {}

(6.1) (3 Punkte) Setzen Sie das spezifizierte Verhalten entsprechend um. Sie brauchen nicht den gesamten vorgegebenen Code zu wiederholen. Machen Sie im Zweifel kenntlich, wo Sie Code ändern oder ergänzen.





 zipien.			

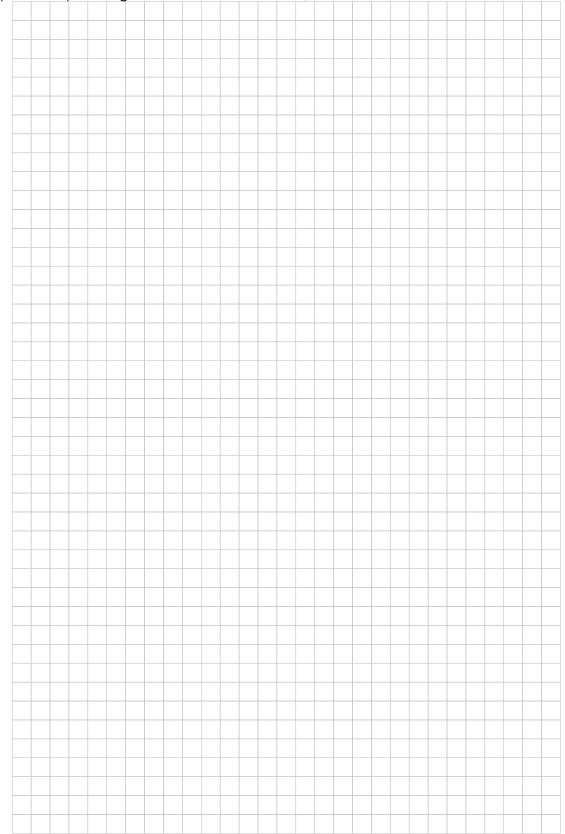


7. Gegeben sei folgende Schnittstellendefinition für eine Zenturie an der NORDAKADEMIE. Stellen Sie sich eine übliche Verwendung von Century-Objekten in Systemen wie dem CIS vor.

pub pub pub	lic Collecti lic Student	<pre>Century { on &lt; FemaleStudent &gt; getFemaleMembers(); on &lt; MaleStudent &gt; getMaleMembers(); getSpeaker(); or getCenturio();</pre>
public	interface	Student { String getDisplayName(); }
public	interface	FemaleStudent extends Student {}
public	interface	MaleStudent extends Student {}
public	interface	<pre>Professor { String getDisplayName(); }</pre>
(7.1) (3	Punkte) Bewei	ten Sie die Kopplung der Klassen.



(7.2) (5 Punkte) Schlagen Sie einen alternativen, besseren Entwurf vor.





(7.3)	(3 Punkte) ist.	Erläutern Sie, inwiefern Ihre Lösung aus der vorigen Teilaufgabe "besser"					



- 8. (8 Punkte) Kreuzen Sie für jede Aussage entweder richtig oder falsch an.
  - Für jede korrekt gesetzte Markierung erhalten Sie einen Punkt (+1).
  - Bei einer falschen oder fehlenden Markierung erhalten Sie keinen Punkt für die Teilaufgabe.

Welche der folgenden Aussagen über Konzepte der Programmiersprache Java sind richtig? □ richtig Subtyping erlaubt es, mehrere Klassen unter einer gemeinsamen Schnittstelle anzusprechen. Dabei können die Exemplarvariablen, □ falsch die im Interface deklariert werden, für alle das Interface implementierenden Klassen wiederverwendet werden. Eine Subklasse ist eine Klasse, die eine andere Klasse erweitert bzw. □ richtig von dieser Klasse erbt. Sie erbt alle Datenfelder und Methoden von □ falsch ihrer Superklasse. □ richtig Eine Subklasse ist eine Klasse, die in einer anderen Klasse als Typ einer Exemplarvariablen verwendet wird. □ falsch □ richtig Superklassen werden als "super" bezeichnet, da sie für viele unterschiedliche Facetten einer Software zuständig sind. Weitere □ falsch Spezialisierungen dieser Klasse werden in der Regel nicht mehr vorgenommen. Konstruktoren einer Spezialisierung können zu einem beliebigen □ richtig Zeitpunkt der Abarbeitung der Anweisungen ihres Konstruktors den □ falsch Konstruktor der Generalisierung aufrufen. Wenn im Quelltext kein solcher Aufruf angegeben ist, versucht Java automatisch einen parameterlosen Aufruf einzufügen. □ richtig Eine Variable kann ein Objekt halten, dessen Typ entweder gleich dem deklarierten Typ der Variablen oder ein beliebiger Subtyp des □ falsch deklarierten Typs ist. die Implementierung einer □ richtig Eine Subklasse kann Methode überschreiben. Dazu deklariert die Subklasse eine Methode □ falsch mit der gleichen Signatur wie in der Superklasse, implementiert diese jedoch mit einem anderen Rumpf. Die überschreibende Methode wird dann bei Aufrufen der Unterklasse vorgezogen. □ richtig Methodenaufrufe in Java sind niemals polymorph. Derselbe Methodenaufruf kann immer nur eine bestimmte Methode aufrufen. □ falsch Dies wird durch den statischen Typ der Variablen fest vorgegeben.



9. (2 Punkte) Erstellen Sie eine Methode, die einen Stream von Elements bekommt und diesen auf einen Stream von boolean-Objekten, entsprechend dem jeweiligen Attribut dead, abbildet.

```
public class Element {
   private boolean dead;
   private int iq;
   public boolean isDead() { return dead; };
   public int getIQ() { return iq; };
}
```

