

**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

- ☐ **private** ArrayList<int> numbers = **new** ArrayList<>();
  - ☐ **private** ArrayList<String> buchstaben = **new** HashMap<>();
  - ☐ **private** List<String> buchstaben = **new** ArrayList<String>();
  - ☐ **private** ArrayList<String> buchstaben = **new** ArrayList<>();
  - ☐ **private** ArrayList<String> buchstaben = **new** HashMap<String>();
  - ☐ **private** ArrayList<Integer> numbers = **new** HashMap<Integer>();
2. (4 Punkte) Welche der folgenden Aussagen über Interfaces, Klassen und abstrakten Klassen sind richtig?
- ☐ Interfaces und abstrakte Klassen haben gemeinsam, dass keine Instanzen von ihnen erstellt werden können.
  - ☐ Interfaces können Exemplarvariablen definieren, abstrakte Klassen nicht.
  - ☐ Interfaces definieren keine Typen, dass gilt nur für abstrakte Klassen.
  - ☐ Nur Klassen können von Interfaces erben, abstrakte Klassen nicht.
  - ☐ Interfaces definieren Typen.
  - ☐ Das Exemplar einer Klasse, die von einer abstrakten Klasse erbt, die wiederum ein bestimmtes Interface X implementiert, kann einer Variablen zugewiesen werden, die mit dem Typ des Interfaces X deklariert ist.
  - ☐ Abstrakte Klassen müssen nicht alle Methoden der von ihnen implementierten Interfaces implementieren.
  - ☐ Interfaces erlauben keine Mehrfachvererbung.
  - ☐ Klassen erweitern (extends) und implementieren nicht (implements) Interfaces. Sie implementieren 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 Gegensatz zu Klassen, \_\_\_\_\_. Schnittstellen können bei der Deklaration 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 werden, muss es aber nicht. Die \_\_\_\_\_ erlaubt es jeden Subtyp einer Schnittstelle dort zu verwenden wo die Schnittstelle als \_\_\_\_\_ angegeben ist. Schnittstellen helfen das Grundprinzip der \_\_\_\_\_ zwischen Bestandteilen eines Programms zu befolgen. Schnittstellendefinitionen werden nicht mit dem Schlüsselwort `class`, sondern mit dem Schlüsselwörtern \_\_\_\_\_, 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;	<input type="checkbox"/> richtig <input type="checkbox"/> falsch
IDoSomething doit = <b>new</b> DoSomething(); DoSomething doit2 = doit;	<input type="checkbox"/> richtig <input type="checkbox"/> falsch
IDoSomething doit = <b>new</b> IDoSomething();	<input type="checkbox"/> richtig <input type="checkbox"/> falsch
IDoSomething doit = <b>new</b> DoSomethingElse();	<input type="checkbox"/> richtig <input type="checkbox"/> falsch
IDoSomething doit = <b>new</b> DoSomething();	<input type="checkbox"/> richtig <input type="checkbox"/> falsch
DoSomething doit = <b>new</b> DoSomethingElse();	<input type="checkbox"/> richtig <input type="checkbox"/> falsch

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();  
    }  
}
```

- (5.1) (7 Punkte) Erläutern Sie die Begriffe *Überschreiben*, *Überladen* und *späte Bindung* an folgendem Code-Beispiel.

- (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 Entwurfsmuster soll folgendermaßen Anwendung finden: Exemplare 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.



(6.2) (2 Punkte) Begründen Sie Ihren Entwurf unter den in der Vorlesung genannten Entwurfsprinzipien.



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.

```
public interface Century {  
    public Collection<FemaleStudent> getFemaleMembers();  
    public Collection<MaleStudent> getMaleMembers();  
    public Student getSpeaker();  
    public Professor getCenturio();  
}  
  
public interface Student { String getDisplayName(); }  
  
public interface FemaleStudent extends Student {}  
  
public interface MaleStudent extends Student {}  
  
public interface Professor { String getDisplayName(); }
```

(7.1) (3 Punkte) Bewerten Sie die Kopplung der Klassen.

---

---

---

---

---

---

---

---

---

---

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

A large grid of graph paper, consisting of 20 columns and 30 rows of small squares, intended for drawing an alternative design.

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

---

---

---

---

---

---

---

---

---

---

---

---

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?

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

