

1. Aufgabe: Versionierung / Versionsverwaltung

In der Veranstaltung wurden drei Versionen vorgestellt, die bei der (klassischen Form der) Weiterentwicklung eines Softwareprodukts, welches bereits produktiv ist, mindestens vorhanden sein müssen. Geben Sie die drei Versionen an und erläutern Sie diese bezüglich der Arbeitsschritte (1 – 2 Sätze je Version).

2. Aufgabe: Design Patterns

a) MVC Pattern

Gegeben seien die folgenden Klassen (ohne get- und set-Methoden), die eine Variante des MVC Pattern realisieren. Geben Sie das zugehörige UML Klassendiagramm an!

Klasse View

```
public class View{  
  
    private Control control;  
    private ModelX modelX;  
    private ModelY modelY;  
}
```

Klasse Control

```
public class Control{  
  
    private View view;  
    private ModelX modelX;  
    private ModelY modelY;  
}
```

Klasse ModelX

```
public class ModelX{  
}
```

Klasse ModelY

```
public class ModelY{  
}
```

b) Singleton Pattern

Ändern Sie die Klasse *ModelX* aus dem Aufgabenteil a) ab. Geben Sie denjenigen Quellcode an, mit welchem man erreicht, dass man maximal ein Objekt vom Typ *ModelX* erzeugen kann.

Geben Sie weiterhin die Anweisung zum Erhalt eines *ModelX*-Objekts an.

c) Fabrik-Methode Pattern

Gegeben seien die Klassen *Dateienverwaltung* und *DateienService*, erstere siehe unten.

Klasse Dateienverwaltung

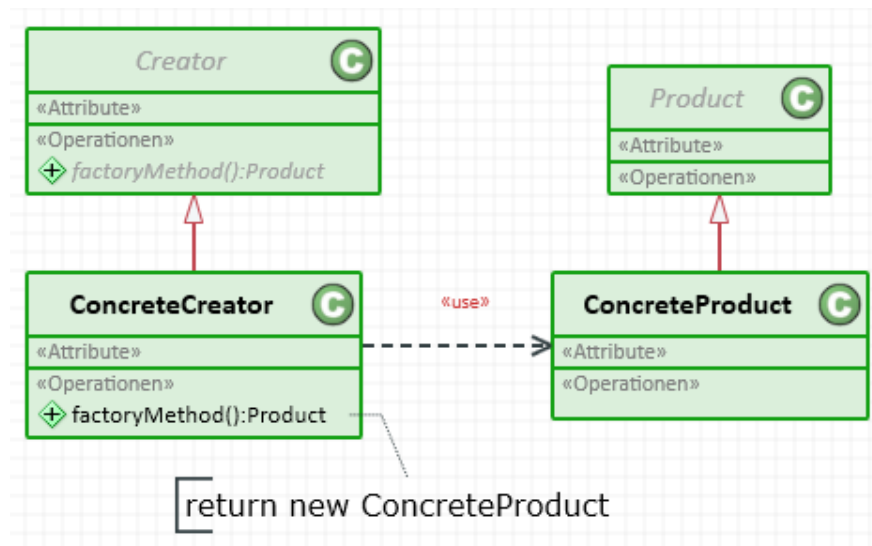
```
import java.io.IOException;

public class Dateienverwaltung {

    private DateienService dateienService = new DateienService();

    public static void main(String[] args) {
        Dateienverwaltung dv = new Dateienverwaltung();
        try {
            dv.dateienService.leseDatumAusCsvDatei();
            dv.dateienService.gibDatumInKonsoleAus();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Gegeben seien die Klassen *Creator*, *Product*, *ConcreteCsvReaderCreator* und *ConcreteCsvReaderProduct*, die das Fabrik-Methode Pattern implementieren.



Karl Eilebrecht, Gernot Starke, Patterns kompakt

Die Klasse *Product* enthält weiterhin eine abstrakte Methode *public abstract String leseDatumAusCsvDatei() throws IOException*. Deren Implementierung in der Klasse *ConcreteCsvReaderProduct* liest eine Zeile aus einer Datei *Datum.csv* und gibt diese als String zurück.

Die Datei *Datum.csv* enthält die Zeile *24;12;2000*. Das Programm liest diese Zeile und gibt sie in der Form *24.12.2000* in der Konsole aus, siehe main-Methode der Klasse *Dateienverwaltung*.

Geben Sie den Quellcode der Klasse *ConcreteCsvReaderCreator* an und ergänzen Sie den Quellcode der Klasse *DateienService*.

Klasse *ConcreteCsvReaderCreator*

Klasse *DateienService*

```

import java.io.*;

public class DateienService {

    // enthaelt ein Datum in der Form TT.MM.JJJJ
    private String datum;
    
```

```

public void leseDatumAusCsvDatei()
    throws IOException{
    // Hier ergaenzen! Mit Hilfe des Pattern Fabrik-Methode
    // soll das Datum aus Datum.csv gelesen werden (erste
    // Zeile) und dann formatiert und im Attribut datum
    // abgespeichert werden.

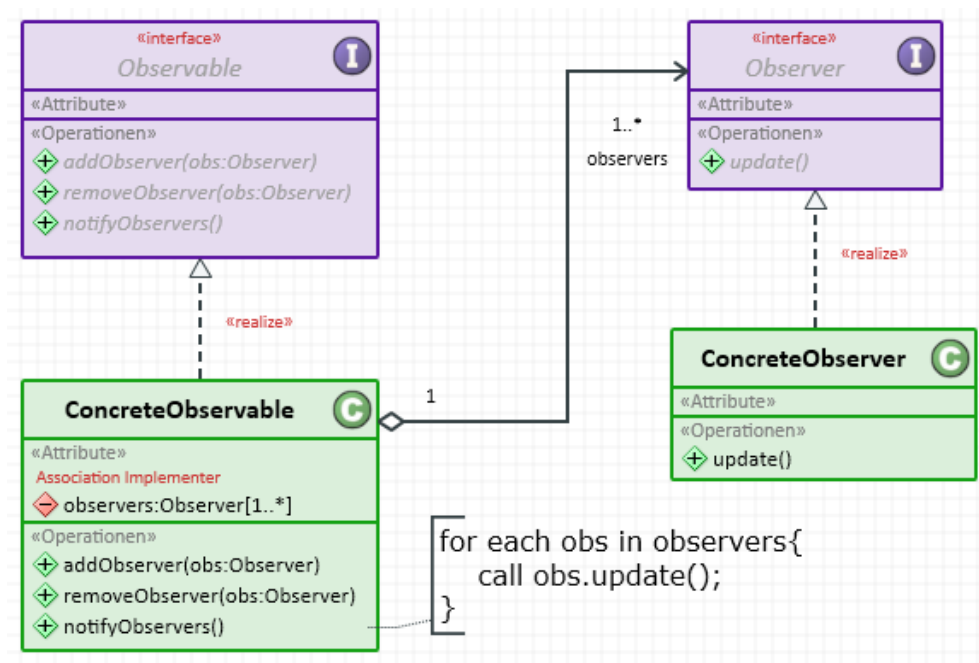
}

public void gibDatumInKonsoleAus() {
    System.out.println(this.datum);
}
}

```

d) Observer Pattern

Gegeben seien die Klassen *DateienService* und *Dateienverwaltung* aus dem Aufgabenteil c). Erweitern Sie diese. *DateienService* soll *Observable* werden entsprechend der folgenden Variante des Observer Patterns. *Dateienverwaltung* soll ein Observer von *DateienService* sein und immer, wenn das Attribut *datum* aus *DateienService* aktualisiert wird, dieses in die Konsole schreiben. Insbesondere wird der Aufruf *dv.dateienService.gibDatumInKonsoleAus();* in der main-Methode nicht mehr benötigt. Sie können davon ausgehen, dass die Interfaces *Observable* und *Observer* vorhanden sind.



Karl Eilebrecht, Gernot Starke, Patterns kompakt

Klasse DateienService

```

import java.io.*;

public class DateienService {

    // enthaelt ein Datum in der Form TT.MM.JJJJ
    private String datum;

    public void leseDatumAusCsvDatei()
        throws IOException{
        // siehe Aufgabenteil c)
        ...
    }

    public void gibDatumInKonsoleAus() {
        System.out.println(this.datum);
    }
}
  
```

```
}
```

Klasse Dateienverwaltung

```
import java.io.IOException;
```

```
public class Dateienverwaltung {
```

```
    DateienService dateienService = new DateienService();
```

```
    public static void main(String[] args) {  
        Dateienverwaltung dv = new Dateienverwaltung();  
        try {  
            dv.dateienService.leseDatumAusCsvDatei();  
            // dv.dateienService.gibDatumInKonsoleAus();  
        }  
        catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

3. Aufgabe: Vertiefung Java

a) Enumeration

Erstellen Sie eine Enumeration *GeometrischeFigur* mit einem Kreis, Dreieck, Quadrat, einer Kugel und einem Würfel. Diese haben die Eigenschaften *dimension* und *anzahlEcken*, welche im Konstruktor belegt werden. Die Werte der Eigenschaften werden mittels get-Methoden herausgegeben.

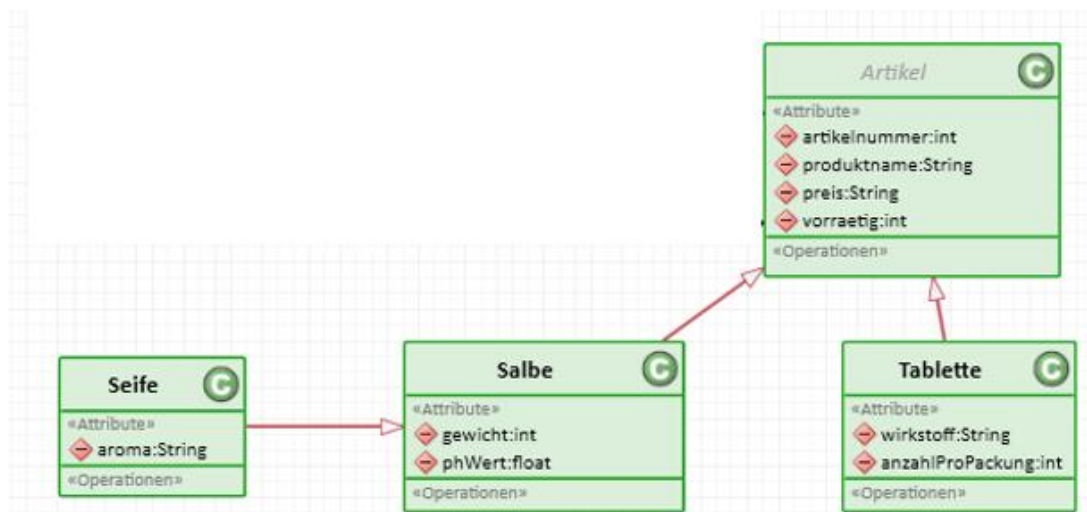
Erstellen Sie eine Klasse *Anwendersystem* mit einer main-Methode, welche die folgende Konsolenausgabe erzeugt. Benutzen Sie eine for each – Schleife für die Ausgabe.

```
Problems @ Javadoc Declaration Console x
<terminated> Anwendersystem [Java Application] C:\Program Files\Java\jdk-11.0.2\
Die geometrische Figur KUGEL

Alle vorhandenen geometrischen Figuren mit Eigenschaften:
KREIS: Dimension: 2, Anzahl Ecken: 0
DREIECK: Dimension: 2, Anzahl Ecken: 3
QUADRAT: Dimension: 2, Anzahl Ecken: 4
KUGEL: Dimension: 3, Anzahl Ecken: 0
WUERFEL: Dimension: 3, Anzahl Ecken: 8
```

b) Generics

- i) Erstellen Sie die Klassen zu dem folgenden UML Klassendiagramm inklusive Konstruktoren, in welchen die Attribute mit Werten belegt werden, und get-und set-Methoden.



Weiterhin sollen die Klassen jeweils eine Methode *public String gibAttributeZurueck()* enthalten, die die Werte der Attribute, mit Komma getrennt, zurückgibt. Verwenden Sie Überschreibung.

Ergänzen Sie die vorliegende Klasse *Anwendersystem*, um die folgende Ausgabe in der Konsole zu erhalten. Benutzen Sie ein Attribut *artikelliste* vom Typ *ArrayList* und für die Ausgabe eine for each - Schleife.

```
1, Tablette gegen Entzündungen, 12,90 Euro, 50, Eisenkraut, 50
2, Zinksalbe, 8,90 Euro, 30, 75, 5.5
3, Kernseife, 3,90 Euro, 20, 100, 6.5, neutral
```

Klasse Anwendersystem

```
import java.util.ArrayList;

public class Anwendersystem {

    // Hier ergaenzen

    public void fuelleArtikelliste() {
        this.artikelliste.add(new Tablette(
            1, "Tablette gegen Entzündungen",
            "12,90 Euro", 50, "Eisenkraut", 50));
        this.artikelliste.add(new Salbe(
            2, "Zinksalbe", "8,90 Euro", 30, 75, 5.5f));
        this.artikelliste.add(new Seife(
            3, "Kernseife", "3,90 Euro", 20, 100, 6.5f,
            "neutral"));
    }

    public void gibArtikellisteAus() {
        // Hier ergaenzen
    }

    public static void main(String[] args) {
        Anwendersystem anw = new Anwendersystem();
        anw.fuelleArtikelliste();
        anw.gibArtikellisteAus();
    }
}
```

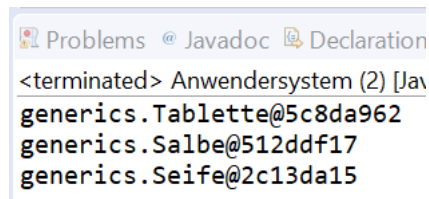
- ii) Erstellen Sie eine Klasse *ArrayListMitAusgabe*, welche von *ArrayList* ableitet und die folgende Methode *gibListeAlsZeilenZurueck()* enthält. Sie gibt ein Array von Strings zurück, welches die einzelnen Listenelemente als Strings enthält.


```

public String[] gibListeAlsZeilenZurueck() {
    String[] ergebnis = new String[this.size()];
    for(int i = 0; i < this.size(); i++) {
        ergebnis[i] = this.get(i).toString();
    }
    return ergebnis;
}

```

Ändern Sie das Attribut *artikelliste* und die Methode *gibArtikellisteAus* der Klasse *Anwendersystem* ab. Sie soll *ArrayListMitAusgabe* benutzen. Allerdings erhalten Sie noch nicht die gewünschte Konsolenausgabe sondern die folgende (, *generics* ist hier der Name des packages, in welchem die Klassen liegen).



```

<terminated> Anwendersystem (2) [Jav
generics.Tablette@5c8da962
generics.Salbe@512ddf17
generics.Seife@2c13da15

```

Nehmen Sie Änderungen in der Klasse *ArrayListMitAusgabe* vor, so dass Sie die gewünschte Ausgabe erhalten. Benutzen Sie eine Typeinschränkung mittels *extends*.

- iii) Kommentieren Sie die Änderungen für den Aufgabenteil ii) in der Klasse *Anwendersystem* aus. Basis ist das Ergebnis des Aufgabenteils i).

Überladen Sie die Methode *gibArtikellisteAus* mit einer Methode, welche die Artikel einer als Parameter vorgegebenen Artikelliste in der Konsole ausgibt. Die Elemente der vorgegebenen Artikelliste sollen vom Typ *Artikel* oder einer Superklasse von *Artikel* sein.

Erstellen Sie eine Methode *selektiereSeifenSalben*, welche aus einer vorgegebenen Liste *quelle* von Artikeln die Seifen und Salben in eine vorgegebene Liste *ziel* hineinkopiert. Verwenden Sie Upper- und Lower bounded wildcards.

Erweitern Sie die main-Methode folgendermaßen, um die folgende Ausgabe in der Konsole zu erhalten.

```

// Aufgabenteil iii)
System.out.println("");
ArrayList<Salbe> listeSeifenSalben
    = new ArrayList<Salbe>();
anw.selektiereSeifenSalben(
    listeSeifenSalben, anw.artikelliste);
anw.gibArtikellisteAus(listeSeifenSalben);

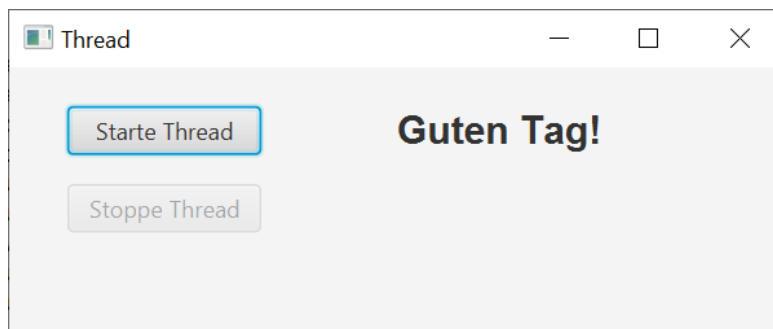
```

```
Problems @ Javadoc Declaration Console
<terminated> Anwendersystem (2) [Java Application] C:\Program Files\Java\jdk-11.0.2\bin
1, Tablette gegen Entzündungen, 12,90 Euro, 50, Eisenkraut, 50
2, Zinksalbe, 8,90 Euro, 30, 75, 5.5
3, Kernseife, 3,90 Euro, 20, 100, 6.5, neutral

2, Zinksalbe, 8,90 Euro, 30, 75, 5.5
3, Kernseife, 3,90 Euro, 20, 100, 6.5, neutral
```

c) Innere anonyme Klasse und Lambda-Ausdruck

Gegeben ist die Vorgabe zu einer Anwendung mit der folgenden Oberfläche.



Beim Klick auf *Starte Thread* wandert das Label *Guten Tag!* nach unten. Mit dem Klick auf *Stoppe Thread* hört diese Aktion wieder auf. Es werden für die Aktionen beider Buttons innere anonyme Klassen verwendet.

Erstellen Sie eine innere, aber nicht anonyme Klasse für die Aktion des Buttons *Starte Thread*. Erstellen Sie einen Lambda-Ausdruck für die Aktion des Buttons *Stoppe Thread*.

4. Aufgabe: Softwaretests

Gegeben sei die Methode

```
public double berechne(int a, int b)
    throws Exception {...}
```

Sie berechnet: $1 / (\sqrt{a} * (b - 15)^2)$

Falls a kleiner 0 ist, oder der Nenner gleich 0 ist, wird eine *IllegalArgumentException* geworfen.

- a) Ergänzen Sie den vorliegenden Quellcode eines JUnit Tests. Die Methode *berechne* liegt in der Klasse *Berechnungen* und soll durch diese JUnit Testklasse getestet werden. Implementieren Sie drei Testfälle. Die ersten beiden Testfälle sollen eine gewünschte Berechnung testen, der dritte Testfall soll eine gewünschte Exception testen. Sie können davon ausgehen, dass die Klasse *Berechnungen* einen Default-Konstruktor besitzt. Benutzen Sie aus der Klasse *Assertions* *assertTrue(boolean condition, String message)*, *assertEquals(double expected, double actual, double delta, String message)* und *fail(String message)*.

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.*;

public class BerechnungenTest{

    // Ergaenzen!!!

    @BeforeEach
    public void setUp() throws Exception{
        // Ergaenzen!!!
    }

    @AfterEach
    public void tearDown() throws Exception{
        // Ergaenzen!!!
    }
}
```

```

@Test

public void testBerechne() throws Exception{

    // Ergaenzen!!!

}

}

```

- b) Ergänzen Sie vier weitere Testfälle innerhalb einer neuen Testmethode. Die ersten drei Testfälle sollen eine gewünschte Berechnung testen, der vierte Testfall soll eine gewünschte Exception testen. Sie können davon ausgehen, dass die Klasse Berechnungen einen Default-Konstruktor besitzt. Benutzen Sie aus der Klasse Assertions
- assertTrue(boolean condition, Supplier<String> messageSupplier),*
assertTrue(BooleanSupplier conditionSupplier, String message),
assertTrue(BooleanSupplier conditionSupplier,
Supplier<String> messageSupplier) und
T assertTrueThrows(Class<T> clazz, Executable executable). Testen Sie bei der Exception sowohl den Typ als auch die Nachricht.

Weitere Aufgaben zur Softwaretests aus SE / SEoderOP /

- a) [SoftwareEngineering/Vorlesung/SEoderOP/Musterklausur.docx](#)
(Use Case Test)
- b) [SoftwareEngineering/Uebung/SEoderOP/AufgabenZurKlausur.docx](#)
(Zustandsbasierter Test und Entscheidungstabellen)

5. Aufgabe: SOLID

a) Single Responsibility Principle

Gegeben sei die folgende Klasse *Mitarbeiter*. Gegeben sei eine Klasse *Anwendersystem*, welche die folgende Ausgabe erzeugt.

- i) Inwiefern wird das Single Responsibility Principle nicht erfüllt?
- ii) Geben Sie eine mögliche Änderung an, so dass das Single Responsibility Principle erfüllt wird.

Klasse Mitarbeiter

```
public class Mitarbeiter {

    private int personalnummer;
    private String vorname;
    private String nachname;
    private String abteilung;
    private String gebaeude;
    private int bueronummer;
    private String strasseHnr;
    private String plz;
    private String ort;

    // Konstruktor, der Attribute mit Werten belegt
    ...
    // get- und set-Methoden
    ...

    public String gibAttributeZurueck() {
        return this.getPersonalnummer() + "\n"
            + this.getVorname() + " " + this.getNachname() + "\n"
            + this.getAbteilung() + "\n"
            + this.getGebaeude() + ": Raum "
            + this.getBueronummer() + "\n"
            + this.getStrasseHnr() + "\n"
            + this.getPlz() + " " + this.getOrt();
    }
}
```

```
Problems @ Javadoc Declaration Console
<terminated> Anwendersystem (4) [Java Application] C:\
Mitarbeiter:

1
Elke Musterfrau
Entwicklung
Turm-Gebäude: Raum 32
Kurze Straße 1
44795 Bochum
2
Max Mustermann
Vertrieb
L-Gebäude: Raum 13
Lange Straße 103
44795 Bochum
```

b) Open Closed Principle

Gegeben sei die folgende Klasse *Holzbrett*. Gegeben sei eine Klasse *Anwendersystem*, welche die folgende Ausgabe erzeugt.

- i) Nennen Sie einen Grund dafür, dass das Open Closed Principle eingehalten werden sollte (1-2 Sätze).
- ii) Inwiefern wird das Open Closed Principle nicht erfüllt?
- iii) Geben Sie eine mögliche Änderung an, so dass das Open Closed Principle erfüllt wird.

Klasse Holzbrett

```
public class Holzbrett {

    private int nrHolzbrett;
    private String holzart;

    // Konstruktor, der Attribute mit Werten belegt
    ...
    // get- und set-Methoden
    ...
    public double berechnePreis(int laenge, int breite) {
        double ergebnis = laenge * breite * 0.002;
        if(getHolzart() != null) {
            if(getHolzart().equals("Douglasie")
                || getHolzart().equals("Buche")) {
                ergebnis = ergebnis * 1.2;
            }
            if(getHolzart().equals("Eiche")
                || getHolzart().equals("Lärche")) {
                ergebnis = ergebnis * 2.3;
            }
        }
    }
}
```

```

        return ergebnis;
    }
}

```

```

<terminated> Anwendersystem (8) [Java Application] C:\Program Files\Ja
Preise von Holzbrettern, 2500 mm lang, 60 mm breit:
null: 300.0
Fichte: 300.0
Buche: 360.0
Eiche: 690.0

```

c) Liskov Substitution Principle

Gegeben sei die folgende Klasse *Artikel*, *Salbe* und *Anwendersystem*.

- i) Inwiefern wird das Liskov Substitution Principle nicht erfüllt?
- ii) Welche negative Auswirkung hat es, dass das Liskov Substitution Principle nicht erfüllt wird?
- i) Geben Sie eine mögliche Änderung in den Klassen an, so dass das Liskov Substitution Principle erfüllt wird. Eine fachliche Änderung ist erlaubt. (Weitere Änderungen / Optimierungen brauchen Sie nicht vornehmen.)

Klasse Artikel

```

public class Artikel{

    private int artikelnummer;
    private String produktname;
    private String preis;
    private int vorraetig;

    public Artikel(int artikelnummer, String produktname,
        String preis, int vorraetig) {
        super();
        this.artikelnummer = artikelnummer;
        this.produktname = produktname;
        this.preis = preis;
        this.vorraetig = vorraetig;
    }

    // get- und set-Methoden
    ...
}

```

```

        public void gibAttributeAus() {
            System.out.print(getArtikelnummer() + ", "
                + getProduktname() + ", "
                + getPreis() + ", " + getVorraetig());
        }
    }
}

```

Klasse Salbe

```

public class Salbe extends Artikel{

    private int gewicht;
    private float phWert;

    public Salbe(int artikelnummer, String produktname,
        String preis, int vorraetig, int gewicht, float phWert) {
        super(artikelnummer, produktname, preis, vorraetig);
        this.gewicht = gewicht;
        this.phWert = phWert;
    }

    // get- und set-Methoden
    ...

    @Override
    public void gibAttributeAus() {
        if(getArtikelnummer() <= 0) {
            System.out.println(
                "Die Artikelnummer muss größer als 0 sein.");
        }
        else {
            super.gibAttributeAus();
            System.out.print(", "
                + getGewicht() + ", " + getPhWert());
        }
    }
}

```

Klasse Anwendersystem

```

public class Anwendersystem {

    public static void main(String[] args) {
        Artikel[] artikelArray = new Artikel[2];
        artikelArray[0] = new Artikel(1,
            "Tablette gegen Entzündungen",
            "12,90 Euro", 50);
        artikelArray[1] = new Salbe(0, "Zinksalbe",

```



```

        "8,90 Euro", 30, 75, 5.5f);
    for(int i = 0; i < artikelArray.length; i++) {
        artikelArray[i].gibAttributeAus();
        System.out.println("");
    }
}
}

```

d) Interface Segregation Principle

Gegeben seien das folgende Interface *Berechnungen* und die Klassen *Kreis* und *Kugel*. Gegeben sei die folgende Klasse *Anwendersystem*, welche die folgende Ausgabe erzeugt.

i) Inwiefern wird das Interface Segregation Principle nicht erfüllt?

ii) Geben Sie eine mögliche Änderung an, so dass das Interface Segregation Principle erfüllt wird.

Interface Berechnungen

```

public interface Berechnungen {

    public double berechneRadius();
    public double berechneUmfang();
    public double berechneFlaeche();
    public double berechneVolumen();
}

```

Klasse Kreis

```

public class Kreis implements Berechnungen{

    private double xWert;
    private double yWert;
    private double radius;

    // Konstruktor, der Attribute mit Werten belegt
    ...
    // get- und set-Methoden
    ...

    @Override
    public double berechneRadius() {
        return this.getRadius();
    }
}

```

```

@Override
public double berechneUmfang() {
    return 2 * Math.PI * this.getRadius();
}

@Override
public double berechneFlaeche() {
    return Math.PI * this.getRadius() * this.getRadius();
}

@Override
public double berechneVolumen() {
    return 0;
}
}

```

Klasse Kugel

```

public class Kugel extends Kreis implements Berechnungen{

    private double zWert;

    public Kugel(double xWert, double yWert, double zWert,
        double radius) {
        super(xWert, yWert, radius);
        this.zWert = zWert;
    }

    public double getZWert() {
        return zWert;
    }

    public void setZWert(double zWert) {
        this.zWert = zWert;
    }

    @Override
    // Berechnung der Oberflaeche
    public double berechneFlaeche() {
        return 4 * Math.PI * this.getRadius() * this.getRadius();
    }

    @Override
    public double berechneVolumen() {
        return (4 * Math.PI * this.getRadius() * this.getRadius()
            * this.getRadius()) / 3;
    }
}

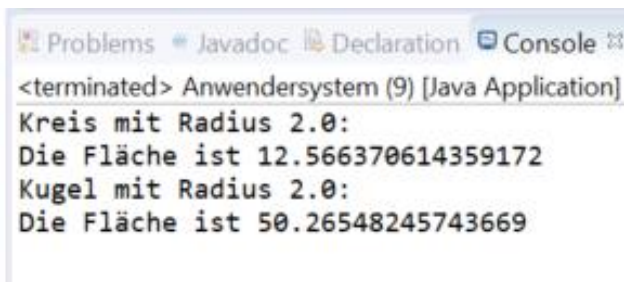
```

Klasse Anwendersystem

```
public class Anwendersystem {

    public static void gibFlaecheAus(
        Berechnungen[] geometrischeFiguren) {
        for(int i = 0; i < geometrischeFiguren.length; i++) {
            System.out.println(geometrischeFiguren[i]
                .getClass().getSimpleName()
                + " mit Radius "
                + geometrischeFiguren[i].berechneRadius()
                + ": ");
            System.out.println("Die Fläche ist "
                + geometrischeFiguren[i].berechneFlaeche());
        }
    }

    public static void main(String[] args) {
        Berechnungen[] geometrischeFiguren = new Berechnungen[2];
        geometrischeFiguren[0] = new Kreis(0, 0, 2);
        geometrischeFiguren[1] = new Kugel(0, 0, 0, 2);
        gibFlaecheAus(geometrischeFiguren);
    }
}
```



The screenshot shows a Java IDE console window with the following output:

```
<terminated> Anwendersystem (9) [Java Application]
Kreis mit Radius 2.0:
Die Fläche ist 12.566370614359172
Kugel mit Radius 2.0:
Die Fläche ist 50.26548245743669
```

e) Dependency Inversion Principle

Gegeben seien das Interface *Berechnungen* und die Klassen *Kreis*, *Kugel* und *Anwendersystem* zur Aufgabe zum Interface Segregation Principle.

Welche Klassen sind high level, welche low level? Was sind in dem Beispiel Abstraktionen? In wie weit wird das Dependency Inversion Principle erfüllt? Geben Sie für Erfüllung und auch Nichterfüllung Begründungen an.

6. Aufgabe: Clean Code

Gegeben ist eine Klasse *Produkt*.

```
public class Produkt {

    private int identnummer;
    private String produktname;
    private String preis;

    public Produkt(int identnummer, String produktname) {
        super();
        this.identnummer = identnummer;
        this.produktname = produktname;
    }

    public Produkt(int identnummer, String produktname,
        String preis) {
        super();
        this.identnummer = identnummer;
        this.produktname = produktname;
        this.preis = preis;
    }

    // get- und set-Methoden
    ...
    public float gibAttributeUndPreisBeiMengenrabattAus(
        int anzahl) {
        float ergebnis = this.preis;
        if(anzahl >= 10) {
            ergebnis = ergebnis * 0.9f;
        }
        gibAttributeAus();
        System.out.println(
            "Mengenrabatt bei " + anzahl + " Stück: "
            + ergebnis + " Euro");
        return ergebnis;
    }

    public void gibAttributeAus() {
        System.out.println(
            "Werte der Eigenschaften des Produkts");
        System.out.println("Identnummer: " + this.identnummer);
        System.out.println("Produktname: " + this.produktname);
        System.out.println("Preis:          " + this.preis
            + " Euro");
    }
}
```

- i) Geben Sie zu den Konstruktoren der Klasse *Produkt* den Quellcode zur Anwendung des Refaktorisierungsmusters *Chain Constructor* an.
- ii) Das *Integration Operation Segregation Principle (IOSP)*, welches zum Erreichen des roten Grades des Clean Code angewendet werden muss, wird in der Klasse *Produkt* nicht eingehalten. Geben Sie den Quellcode einer Möglichkeit zur Einhaltung des IOSP an.
- iii) Welche Tugend und welches Prinzip des roten Grades des Clean Code würden Sie verletzen, wenn Sie in dem vorherigen Aufgabenteil ii) dieser Aufgabe mehr als eine mögliche Änderung angeben? Geben Sie jeweils Begründungen an.
- iv) Wofür steht DRY und wieso sollte man DRY einhalten?