**Projekttag**

Eine vollständige Aufgabe umsetzen



|  |  |
| --- | --- |
| **Name:** | *Manuel Doebel* |
| **Kurs :** | *Winter FIAE B* |
| **Datum:** | *Donnerstag, 13. Juli 2023* |

Inhaltsverzeichnis

Vorgaben 3

Aufgabenstellung 3

UML-Diagramme nach vorgegeben Angaben erstellen 3

UML-Klassendiagramm 3

Zusammenfassung 3

Klassenbeschreibung 3

Enthaltene Assoziationen 4

Weiteres 5

UML-Sequenzdiagramm 5

Weiteres 6

Quellcode schreiben 6

Weiteres 6

# Vorgaben

1. Erklärung der Aufgabe etwa 15 Minuten
2. UML-Diagramme nach vorgegeben Angaben erstellen etwa 70 Minuten
3. UML-Diagramme codieren etwa 105 Minuten
4. **Insgesamt etwa 190 Minuten**

# Aufgabenstellung

## UML-Diagramme nach vorgegeben Angaben erstellen

Es sind zwei Arten von UML-Diagrammen zu erstellen: ein UML-Klassendiagramm und ein zugehöriges UML-Sequenzdiagramm. Die Anforderungen für die Erstellung der beiden Diagramme sind nachstehend aufgeführt.

Anmerkung. Bei der folgenden Aufgabe handelt es sich um eine vereinfachte Aufgabenstellung. Eine reale Umsetzung dieses Falles würde etwas anders aussehen.

### UML-Klassendiagramm

**Erstellen Sie ein UML-Klassendiagramm nach den folgenden Vorgaben.**

#### Zusammenfassung

Es sind **Vererbungen**, **Assoziationen** (gerichtete Assoziation und Aggregation), **Instanzvariablen** und **Methoden** (abstrakte und gewöhnliche) enthalten. Es sind abstrakte und konkrete Klassen zu erstellen.

#### Klassenbeschreibung

* Konkrete Klasse ***Kunde***

Die Klasse ***Kunde*** soll ein nur klassenintern zugreifbares Objekt ***eineWare*** (Datentyp Ware) sowie zwei private Attribute ***Gesamtkosten*** (Datentyp Int) und ***Kundenwunschdauer*** (Datentyp Float/Double) enthalten. (Anmerkung, bei der Codierung beachten: Ist der Datentyp Int für die Gesamtkosten geeignet?) Die Klasse ***Kunde*** enthält den öffentlichen **Konstruktor** und die nur klassenintern sichtbare Methode ***ausgabe***. Dem Konstruktor muss ein Objekt der Klasse *Ware* mit dem das private Objekt ***eineWare*** gesetzt wird und ein Float/Double-Wert zum initialisieren des *Kundenwunschdauer-*Attributs übergeben werden. Die Methode ***ausgabe*** enthält weder Eingabe- (bzw. Übergabe-) noch Rückgabeparameter.

* Abstrakte Klasse ***Ware***

Die abstrakte Klasse ***Ware*** soll zwei nur innerhalb der Klassenhierarchie sichtbare Instanzvariablen ***ArtikelNr*** und ***Kosten*** enthalten (ihre Datentypen sind String bzw. Float/Double). Für das Attribut ***ArtikelNr*** soll je eine abstrakte paketübergreifende sichtbare Set- und Get-Methode bereitgestellt werden. Die öffentliche abstrakte Methode ***berechneKosten*** soll einen Float/Double Parameter übergeben bekommen und zurückgeben (Anmerkung: Die Implementierung dieser Methoden erfolgt in den Subklassen).

* Konkrete Klasse ***Dienstleistung***

Die von der abstrakten Klasse *Ware* erbende Klasse ***Dienstleistung*** soll zusätzlich zu den geerbten, nur klassenintern sichtbare Instanzvariablen ***Verantwortlicher*** und ***Dauer*** beinhalten (die Instanzvariablen sind jeweils vom Datentyp Integer bzw. Float/Double). Dem öffentlichen **Konstruktor** der Klasse *Dienstleistung* sollen zwei Material-Objekte übergeben werden, die in einem nur klassenintern sichtbaren Container ***Materialverbrauch*** gespeichert werden. Die nur klassenintern sichtbare Methode ***istGewinn*** soll einen booleschen Wert (*true* oder *false*) zurückgeben, ist aber Eingabeparameterlos.

* Konkrete Klasse ***Material***

Die von der abstrakten Klasse ***Ware*** erbende Klasse ***Material*** soll zusätzlich zu den geerbten, nur klassenintern sichtbare Instanzvariablen ***Name***, ***Art***, ***Qualitaet*** und ***Verfallsdatum*** enthalten (alle diese Instanzvariablen sind *Strings*). Der paketübergreifend erreichbare **Konstruktor** der Klasse ***Material*** soll Übergabeparameter (=Eingabeparameter) enthalten, mit denen alle diese vier Instanzvariablen (d. h. *Name*, *Art*, *Qualitaet* und *Verfallsdatum*) der Klasse initialisiert werden.

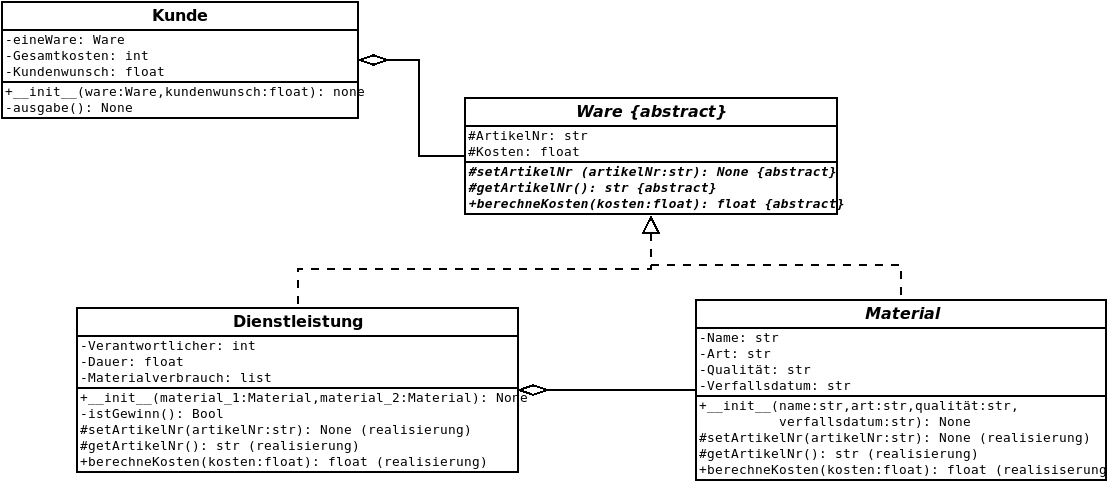
#### Enthaltene Assoziationen

* Zwischen der Klasse *Kunde* und *Ware* gibt es eine **gewöhnliche gerichtete Assoziation**.
* Zwischen der Klasse Material und der Klasse Dienstleistung gibt es **Aggregationsassoziation**. Dienstleistung ist die Aggregationsklasse.
* Die konkreten erbenden Klassen ***Material*** und ***Dienstleistung***sollen beide die *abstrakte* Klasse ***Ware*** *implementieren*.

#### Weiteres

Das UML-Klassendiagramm muss gemäß der von Ihnen gewählten Programmiersprache erstellt werden. Verwenden Sie dazu die Konventionen, die Sie in den Unterrichtsstunden des Moduls zur Programmiersprache vermittelt bekommen haben (z.B. Groß-/Kleinschreibung von Instanzvariablen). Zur Erstellung des UML-Klassendiagramms verwenden Sie wie im Unterricht die Softwareanwendung **Dia**. Wenn Sie fertig sind, exportieren Sie Ihre Lösung in eine Bilddatei und laden Sie sie hoch.

Geben Sie die von Ihnen gewählte Zielprogrammiersprache an: Python



### UML-Sequenzdiagramm

Erstellen Sie zu dem obigen UML-Klassendiagramm das dazugehörige UML-Sequenzdiagramm, das folgende Prozesse verdeutlicht:

* Der ***Kunde* ruft** im Konstruktor die Methode ***berechneKosten*** des ***Dienstleistung-Objekt*** auf und **übergibt** die Kundenwunschdauer, die die vom Kunden gewünschte Dienstleistung benötigt (ein Float/Double-Wert). (Anmerkung: Das Attribut ***Dauer*** kann durch Kundenwunschdauer in dieser Methode gesetzt werden.)
* Die Methode ***berechneKosten*** überprüft **mittels** der Methode ***istGewinn,*** ob die gewünschte Dienstleistung einen Gewinn hat.
* Im **Ja-Fall** gibt die Methode ***berechneKosten*** den Wert **-1.0** an den Kunden zurück.
* Im **Nein-Fall** wird die Methode ***berechneKosten*** des ***Material-Objekts*** aufgerufen, welche eine Berechnung der Gesamtkosten der günstigsten Materialien realisiert. (Anmerkung: Dabei wird ***Dauer*** (bzw. Kundenwunschdauer) an die ***berechneKosten*** Methode übergeben.)
* Die Gesamtkosten der günstigsten Materialien werden an den Kunden **weitergereicht**.
* Der ***Kunde* ruft** im **Konstruktor** die klasseneigene Methode ***ausgabe*** auf, die den *Ausgabetext1* ausgibt: "*Die Dienstleistung kann ohne zusätzliche Kosten in Anspruch genommen werden*", wenn die **Gesamtkosten gleich -1.0** sind, oder die den *Ausgabetext2* ausgibt: "*Die Dienstleistung wird mindestens [Gesamtkosten] kosten. Sie müssen mehr bezahlen als erwartet.*", wenn die **Gesamtkosten nicht gleich -1.0** sind.

#### 

#### Weiteres

Zur Erstellung des UML-Sequenzdiagramm verwenden Sie wie im Unterricht die Softwareanwendung **Dia**. Wenn Sie fertig sind, exportieren Sie Ihre Lösung in eine Bilddatei und laden Sie sie hoch.

### Quellcode schreiben

Das gesamte UML-Klassendiagramm und das zugehörige UML-Sequenzdiagramm in Code umsetzen. Das Programm muss ausführbar sein. Nur wesentliches codieren.

Zu dem Inhalt der Methoden:

* Algorithmus mit Ausgabefunktionen ersetzten.
* If-else-Abfragen sollten vorhanden sein. Hierbei muss an einigen Stellen eine kleine Abfrage selbst erdacht werden.
* An einigen Stellen ist auch ein selbst erdachter Rückgabewert notwendig.

**Fortgeschritten:** In der Aufgabe gibt es zwei fortgeschrittene Punkte.

* Fehler. Wenn möglich, finden Sie den Fehler und vermerken Sie ihn in einem Kommentar. Wenn möglich, korrigieren Sie den Fehler (vergessen Sie auch hier nicht, einen Kommentar hinzuzufügen, der auf die Fehlerkorrektur aufmerksam macht).
* Polymorphismus. Dies ist besonders schwer. In Python aber kaum erkennbar.

#### Weiteres

Verwenden Sie die im Programmiermodul verwendete Umgebung, um den Quellcode zu erstellen. Exportieren Sie den erstellten Quellcode in eine Textdatei und laden Sie diese hoch. Beim hochladen wird die Dateiendung vom System geändert. Das ist kein Problem.

import random

from abc import ABC, abstractmethod

class Ware(ABC):

def \_\_init\_\_(self) -> None:

super().\_\_init\_\_()

self.\_artikelNr : str

self.\_kosten: float

@abstractmethod

def setArtikelNr(self, artikelNr: str) -> None:

pass

@abstractmethod

def getArtikelNr(self) -> str:

pass

@abstractmethod

def berechneKosten(self, basiswert: float) -> float:

pass

class Material(Ware):

def \_\_init\_\_(self, name: str, qualität: str, verfallsdatum: str) -> None:

super().\_\_init\_\_()

self.\_\_name = name

self.\_\_qualität = qualität

self.\_\_verfallsdatum = verfallsdatum

def setArtikelNr(self, artikelNr: str) -> None:

self.\_artikelNr = artikelNr

def getArtikelNr(self) -> str:

return self.\_artikelNr

def berechneKosten(self, basiswert: float) -> float:

if self.\_\_qualität != "mies":

return basiswert\*2

return basiswert

class Dienstleistung(Ware):

def \_\_init\_\_(self, material\_1: Material, material\_2: Material, dauer: float) -> None:

super().\_\_init\_\_()

self.\_\_materialverbrauch = []

self.\_\_verantwortlicher : int

self.\_\_dauer = dauer

def istGewinn(self) -> bool:

return random.randint(0,2) % 2 == 0

def setArtikelNr(self, artikelNr: str) -> None:

self.\_artikelNr = artikelNr

def getArtikelNr(self) -> str:

return self.\_artikelNr

def berechneKosten(self, basiswert: float) -> float:

self.\_\_dauer = basiswert

if self.istGewinn():

return -1.0

# berechnung der GESAMTkosten hier, nicht in einem

# einzelnen material, weil GESAMT

gesamtkosten = 0

for material in self.\_\_materialverbrauch:

gesamtkosten += material.berechneKosten(self.\_\_dauer)

return gesamtkosten

# nicht Ware sondern Dienstleistung als Datentyp für self.\_\_eineWare

class Kunde():

def \_\_init\_\_(self, ware : Dienstleistung, kundenwunschdauer: float):

self.\_\_eineWare = ware

self.\_\_gesamtkosten : int

self.\_\_kundenwunschdauer = kundenwunschdauer

self.\_\_gesamtkosten

=self.\_\_eineWare.berechneKosten(self.\_\_kundenwunschdauer)

self.ausgabe(self.\_\_gesamtkosten)

def ausgabe(self,kostenwert):

if kostenwert == -1.0:

return print("Die Dienstleistung kann ohne zusätsliche Kosten in Anpsruch genommen werden")

return print(f"Die Dienstleistung wird Mindestens {kostenwert} kosten. Sie müssen mehr bezahlen als erwartet" )

class Main():

def \_\_init\_\_(self) -> None:

kühlflüssigkeit = Material("Kühlflüssigkeit", "gut", "Ende 2026")

schmierstoff = Material("schmiere", "mies", "2024")

zahnräder = Material("Zahnräder", "gut", "2248")

zahnräder\_pflegen = Dienstleistung(schmierstoff, zahnräder, 2.2)

kühlung\_erneuern = Dienstleistung(kühlflüssigkeit, kühlflüssigkeit, 2.2)

kunde1 = Kunde(kühlung\_erneuern, 2)

kunde2 = Kunde(zahnräder\_pflegen, 5)

if \_\_name\_\_ == "\_\_main\_\_":

start = Main()