1 Arbeiten mit selbst erstellten Fachkonzeptklassen

Ziel: eigene Fachkonzeptklassen erstellen und diese anwenden

Szenario „Auftragsverwaltung“:

Herr Sommer ist Inhaber der Firma HardSoft GmbH und handelt mit Computern Zubehör und Software. Er möchte in Zukunft seine komplette Auftragsverwaltung softwaregestützt abwickeln und benötigte dazu ein geeignetes Programmsystem.

Er wendet sich dazu an die Firma SoftGut GmbH, die ihm bei der Erstellung behilflich sein soll.

Herr Sommer schildert zunächst seine Anforderungen. Kurz zusammengefasst möchte Herr Sommer folgende Informationen verwalten:

Kunden: Name, Vorname, Anschrift, E-mail, Telefon, Rabattsatz, Zahlungsziel in Tagen, Gesamtbestellsumme

Aufträge: AuftragsNr, Datum, Auftragswert

Lieferanten: Name, Anschrift, Gesamtumsatz, Lieferbedingungen

Artikel: ArtikelNr, Bezeichnung, Verkaufspreis, Lagerbestand, Mindestbestand, Artikelgruppe, mittlerer Einstandspreis

1.1. Erstellen Sie zunächst auf Papier ein UML-Klassendiagramm für die Klasse `Kunde`.

Überlegen Sie sich geeignetet Datentypen für die Attribute.

Die Klasse enthält neben den genannten Attributen einen expliziten Standardkonstruktor, der den Namen des Kunden mit "-leer-" initialisiert und alle numerischen Attribute mit 0.

Für alle Attribute soll es entsprechende Accessor- und Mutator-Methoden geben.

1.2. Erstellen Sie in Eclipse ein neues Package `auftragsVerwV1` und implementieren Sie darin die Klasse `Kunde` in Java.

1.3. Erstellen Sie eine UI-Klasse `AuftragsVerwUI`, die gleichzeitig Startklasse ist und erzeugen Sie ein Kunden-Objekt.

1.4. In der UI-Klasse werden dann in einem Benutzerdialog die Attributwerte des Kunden eingelesen und in dem Kunden-Objekt gespeichert.

1.5. Erweitern Sie die Klasse `Kunde` um eine Methode `public String toString()`, die alle Attribute eines Kunden in einem String zurückliefert. Innerhalb dieses Strings sollen die einzelnen Werte durch Semikolon getrennt werden:
"Attribut1; Attribut2; Attribut3; usw.".

1.6. Verwenden Sie diese Methode in der main-Methode der UI-Klasse, um die Daten des Kunden vor den Benutzereingaben und nochmals danach am Bildschirm anzuzeigen.

1.7. Ergänzen Sie das Package `auftragsVerwV1` um die Klasse `Auftrag`.
Benutzen Sie für das Datum ein `LocalDate` Objekt.
Der Standardkonstruktor initialisiert die numerischen Attribute mit 0 und das Datum mit dem aktuellen Datum.
Ein zusätzlicher überladener Konstruktor initialisiert alle Attribute der Klasse mit Parameterwerten. Für das Datum erhält der Konstruktor Jahr, Monat und Tag und erzeugt daraus ein entsprechendes `LocalDate` Objekt.
Hinweis: Die Accessor- und Mutator-Methoden können von Eclipse automatisch erzeugt werden: *→Menü:Source→Generate Getters und Setters…*

1.8. Erzeugen Sie in der UI-Klasse `AuftragsVerwUI` zwei Auftrags-Objekte. Einmal mit Hilfe des Standardkonstruktors und dann mit dem überladenen Konstruktor. Lassen Sie jeweils die Auftragsdaten am Bildschirm anzeigen (Accessor-Methoden verwenden).

1.9. Ergänzen Sie das Package `auftragsVerwV1` um die Klasse `Lieferant`.
Zusätzlich zu den genannten Attributen und den entsprechenden Accessor- und Mutator-Methoden und einer `toString`-Methode, soll die Klasse noch um folgende Methoden ergänzt werden:

Einen expliziten Standardkonstruktor, der den Namen des Lieferanten mit "-leer-" initialisiert und alle numerischen Attribute mit 0.

Eine Methode `erhöheUmsatz` und eine Methode `verringereUmsatz`. Diese Methoden erhalten einen Auftragswert als Übergabeparameter. Beide Methoden haben folgende Funktionalität:

- Aktualisieren eines Auftragszählers (zusätzliches Attribut).
- Prüfung des übergebenen Auftragswerts: muss positiv sein, Gesamtumsatz darf nicht negativ werden; falls diese Bedingungen nicht erfüllt sind, wird `false` zurückgegeben, sonst `true`.

Desweiteren ist eine Methode `getMittlerenAuftragswert` zu erstellen. Diese liefert den aktuellen durchschnittlichen Auftragswert aller Lieferaufträge.

1.10. Testen Sie die Lieranten-Klasse, indem Sie in der UI-Klasse `AuftragsVerwUI` ein Lieferanten-Objekt erzeugen. Setzen Sie in diesem Objekt die Attribute mit sinnvollen Werten und lassen sich die Daten am Bildschirm anzeigen. Ändern Sie den Lieferantenumsatz und lassen sich die Daten erneut am Bildschirm anzeigen.

1.11. Ergänzen Sie das Package `auftragsVerwV1` um die Klasse `Artikel`.
Zusätzlich zu den genannten Attributen und den entsprechenden Accessor- und Mutator-Methoden und einer `toString`-Methode, soll die Klasse noch um folgende Methoden ergänzt werden:

Einen expliziten Standardkonstruktor, der die Bezeichnung des Artikels mit "-leer-" initialisiert und alle numerischen Attribute mit 0.

Einen überladenen Konstruktor, mit dessen Hilfe man die Artikelnummer, die Bezeichnung und den Verkaufspreis beim Erzeugen eines Objektes festlegen kann.

Außerdem die Methoden:

- `public void erhoehePreis (double dBetrag)`

- `public double ermittleGesamtLagerwert()`
- `public boolean bestellen ()`
  Die Methode vergleicht den Lagerbestand mit dem Mindestbestand. Wenn eine Bestellung notwendig ist, wird `true` zurückgegeben, sonst `false`.

1.12. Testen Sie die Artikel-Klasse, indem Sie in der UI-Klasse `AuftragsVerwUI` ein Artikel-Objekt mit dem überladenen Konstruktor erzeugen. Setzen Sie restlichen Attribute in diesem Objekt mit sinnvollen Werten und lassen sich die Daten am Bildschirm anzeigen. Lassen Sie sich den Gesamtlagerwert am Bildschirm anzeigen. Erhöhen Sie danach den Preis und lassen Sie sich den Gesamtlagerwert erneut am Bildschirm anzeigen

1.13. Ergänzen Sie die Klasse `Kunde` um eine Methode `erhöheGesamtbestellsumme` und eine Methode `erniedrigeGesamtbestellsumme`. Diese Methoden erhalten einen Bestellwert als Übergabeparameter. Beide Methoden haben folgende Funktionalität:

- Aktualisieren eines Bestellungszählers (zusätzliches Attribut).
- Es wird nur ein Bestellwert akzeptiert, der zwischen 200 und 2000 Euro liegt. Liegt er darunter, wird -1 geliefert, darüber -2, sonst 1

1.14. Erweitern Sie die Methode `setName` der Klasse `Kunde` so, dass sie nur Namen mit einer maximalen Länge von 20 Zeichen akzeptiert. Ist der Name länger, so wird der Name auf 20 Zeichen gekürzt gespeichert. Testen Sie die geänderte Methode.

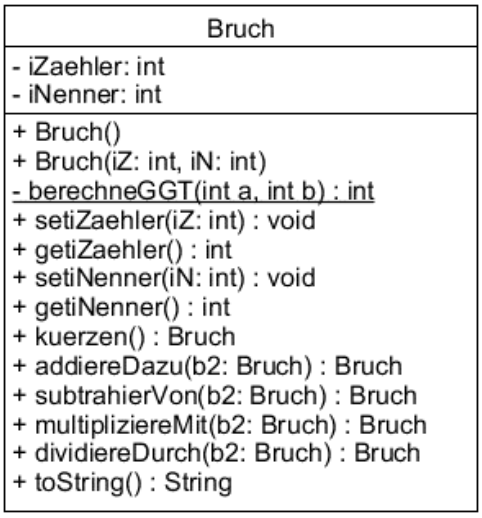
1.15. Ändern Sie die Methode `setName` so, dass sie alle Namen mit einer maximalen Länge über 20 Zeichen automatisch auf 20 Zeichen verkürzt. Die Länge des tatsächlich eingetragenen Namens soll als Rückgabewert geliefert werden. Testen Sie die geänderte Methode.

Klassenvariablen

1.16. Ergänzen Sie die Fachkonzeptklasse `Artikel` um die Klassenvariable `AnzahlErzeugterObjekte` sowie die dazugehörigen Accessor- und Mutatormethoden. Sorgen Sie in geeigneter Weise dafür, dass dieser Wert immer korrekt ist (→Konstruktor). Zusätzlich wird die Klasse `Artikel` erweitert um die Klassenvariable `Mehrwertsteuersatz` plus Accessor- und Mutatormethoden.
Eine zusätzliche Objektmethode `getVerkaufspreisBrutto` ermittelt den aktuellen Verkaufspreis inclusive Mehrwertsteuer und gibt diesen zurück.

1.17. Erstellen Sie eine neue Startklasse, in der zunächst der Mehrwertsteuersatz festgelegt wird und dann von Tastatur beliebig viele Artikel erfasst werden (nur Bezeichnung und Verkaufspreis). Ende bei Eingabe von Artikelnummer -1. Zu jedem Artikel wird der Bruttoverkaufspreis ausgegeben. Anschließend soll die Anzahl der erzeugten Artikel ermittelt werden. Der Mehrwertsteuersatz wird nun erhöht und der Bruttoverkaufspreis des letzten Artikels erneut ausgegeben.

2.1. Es soll eine Klasse zum Rechnen mit Brüchen implementiert werden. Die Klasse hat folgendes UML Klassendiagramm:



- Der Standardkonstruktor initialisiert den Zaehler mit 0 und den Nenner mit 1.
- Der überladene Konstruktor initialisiert den Bruch mit den gegebenen Parametern. Im Fall, dass der Nenner auf 0 gesetzt werden soll, wird eine Fehlermeldung auf dem Bildschirm ausgegeben. (Besser wäre es eine sogenannte „Exception“ auszulösen; siehe später)
- Die setiNenner() Methode prüft, ob der Nenner auf 0 gesetzt werden soll. In diesem Fall wird eine Fehlermeldung auf dem Bildschirm ausgegeben.
- Die Rechen-Methoden ändern den im Objekt (this) gespeicherten Bruch nicht und liefern das Ergebnis jeweils in gekürzter Form zurück!
- Für die Klassenmethode `berechneGGT` siehe „Lsg 02 JavaKontrollstrukturen A5 1BisA5 5“
- Die Objektmethode `toString()` hat folgende Funktionalität:
  - Bsp.: Ist der zaehler = 1 und der nenner = 2 dann ist die Rückgabe "1/2"
  - Bsp.: Ist der zaehler = 3 und der nenner = 2 dann ist die Rückgabe "1 1/2"
  - Bsp.: Ist der zaehler = 6 und der nenner = 2 dann ist die Rückgabe "2"

2.2. Programmieren Sie zum Testen der Bruch Klasse eine UIKlasse, die einen ähnlichen Dialog wie folgenden ermöglicht:

```
Zaehler 1 = 1
Nenner 1 = 2
Zaehler 2 = 2
Nenner 2 = 3

Rechenoperator ? +
Ergebnisbruch: 1 1/6
```