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.

```java
/* Kunde.java
package oopAuftragsVerwV1;

/**
 * @author stk
 *         Kurzbeschreibung: Die Klasse verwaltet
 *         die Daten eines Kunden
 */
public class Kunde
{
    // Anfang Attribute
    private String sName;
    private String sVorname;
    private String sAnschrift;
    private String sEmail;
    private int iZahlungsZiel;
    private String sTelefon;
    private double dRabattsatz;
    private double dGesamtBestellsumme;
    private int iAnzahlBestellungen = 0;
    // Ende Attribute

    public Kunde()
    {
        this.sName = "-leer-";
        this.iZahlungsZiel = 0;
        this.dRabattsatz = 0;
    }

    // Anfang Methoden
    public String getSName()
    {
        return this.sName;
    }

    public int setSName(String sNeuerName)
    {
        if (sNeuerName.length() > 20)
        {
            this.sName = sNeuerName.substring(0, 20);
        }
        else
        {
            this.sName = sNeuerName;
        }
        return (this.sName.length());
    }
```

```
52      public String getSVorname()
53      {
54          return this.sVorname;
55      }
56
57      public void setSVorname(String sVorname)
58      {
59          this.sVorname = sVorname;
60      }
61
62      // usw.
63      public int erhöheGesamtbestellsumme(double dBetrag)
64      {
65          int status = 1; // 1 = OK; -1 = dBetrag zu klein; -2 = dBetrag zu groß
66          if (dBetrag < 200)
67          {
68              status = -1;
69          }
70          else
71              if (dBetrag > 2000)
72              {
73                  status = -2;
74              }
75              else
76              {
77                  dGesamtBestellsumme += dBetrag;
78                  iAnzahlBestellungen++;
79              }
80          return status;
81      }
82
83      public int erniedrigeGesamtbestellsumme(double dBetrag)
102
103     public int getIAnzahlBestellungen()
107
108     public void setIAnzahlBestellungen(int iAnzahlBestellungen)
112
113     public String toString()
114     {
115         return (sName + "; " + sVorname + "; " + sAnschrift + "; "
116                 + sEmail + "; " + sTelefon + "; " + iZahlungsZiel
117                 + "; " + dRabattsatz + "; " + dGesamtBestellsumme);
118     }
119     // Ende Methoden
120
121 }
```

```
11  * @author stk
16  public class Auftrag
17  {
18      // Anfang Attribute
19      private int iAuftragsNr;
20      private LocalDate datum;
21      private double dAuftragswert;
22      // Ende Attribute
23
24      // Anfang Methoden
25      public Auftrag()
26      {
27          this.iAuftragsNr = 0;
28          this.datum = LocalDate.now();
29          this.dAuftragswert = 0;
30      }
31
32      public Auftrag(int iAuftragsNr, int iJahr, int iMonat, int iTag,
33                     double dAuftragswert)
34      {
35          this.iAuftragsNr = iAuftragsNr;
36          this.datum = LocalDate.of(iJahr, iMonat, iTag);
37          this.dAuftragswert = dAuftragswert;
38      }
39
40      public int getIAuftragsNr()
44
45      public void setIAuftragsNr(int iAuftragsNr)
49
50      public LocalDate getDatum()
54
55      public void setDatum(LocalDate datum)
59
60
61      public double getDAuftragswert()
65
66      public void setDAuftragswert(double dAuftragswert)
70
71      public String toString()
72      {
73          String sDatum = datum.format(DateTimeFormatter.ofPattern("dd.MM.yy"));
74          return iAuftragsNr + "; " + sDatum + "; " + dAuftragswert;
75      }
76
77      // Ende Methoden
78  }
```

```java
/*
 * Lieferant.java
 */
package oopAuftragsVerwV1;

/**
 * @author stk
 *         Kurzbeschreibung: Verwalten der Daten eines Lieferanten
 */
public class Lieferant
{
    // Anfang Attribute
    private String sName;
    private String sAnschrift;
    private double dGesamtumsatz;
    private String sLieferbedingungen;
    private int iAnzahlAuftraege;
    // Ende Attribute

    // Anfang Methoden

    // Konstruktoren
    public Lieferant()
    {
        this.sName = "-leer-";
        this.iAnzahlAuftraege = 0;
        this.dGesamtumsatz = 0.0;
    }

    // Get- und Set- und toString Methoden ...

    public boolean erhoeheUmsatz(double dBetrag)
    {
        boolean bStatus = true;

        if (dBetrag >= 0)
        {
            this.dGesamtumsatz = this.dGesamtumsatz + dBetrag;
            this.iAnzahlAuftraege++;
        }
        else
        {
            bStatus = false;
        }

        return bStatus;
    }

    public boolean erniedrigeUmsatz(double dBetrag)
    {
        boolean bStatus = true;

        if (dBetrag >= 0 && (this.dGesamtumsatz - dBetrag >= 0)
            && this.iAnzahlAuftraege > 0)
        {
            this.dGesamtumsatz = this.dGesamtumsatz - dBetrag;
            this.iAnzahlAuftraege--;
        }
        else
        {
            bStatus = false;
        }
        return bStatus;
    }

    public double getMittlerenAuftragswert()
    {
        return ((int) ((this.dGesamtumsatz
                        / this.iAnzahlAuftraege) * 100)) / 100.0;
    }
    // Ende Methoden
}
```

```java
/* Artikel.java...
package oopAuftragsVerwV1;

/**
 * @author stk Kurzbeschreibung: Verwaltet die Daten eines Artikels
 */
public class Artikel
{
    // Anfang Attribute
    private int iArtikelNr;
    private String sBezeichnung;
    private String sArtikelgruppe;
    private double dVerkaufsPreis;
    private double dLagerbestand;
    private double dMindestbestand;
    private double dMittlEinstPreis;
    // Ende Attribute

    // Konstruktoren
    public Artikel()
    {
        this.sBezeichnung = "-leer-";
        this.iArtikelNr = 0;
        this.dLagerbestand = 0;
        this.dMindestbestand = 0;
        this.dMittlEinstPreis = 0.0;
        this.dVerkaufsPreis = 0.0;
    }

    public Artikel(int iArtikelNr, String sBez, double dVerkaufsPreis)
    {
        this.iArtikelNr = iArtikelNr;
        this.sBezeichnung = sBez;
        this.dVerkaufsPreis = dVerkaufsPreis;
    }

    // Get- Set- und toString Methoden

    public void erhöhePreis(double dBetrag)
    {
        this.dVerkaufsPreis = this.dVerkaufsPreis + dBetrag;
    }

    public double ermittleGesamtLagerwert()
    {
        return ((int) ((this.dLagerbestand
                        * this.dMittlEinstPreis) * 100)) / 100.0;
    }

    public boolean bestellen()
    {
        boolean bErgebnis;
        if (dLagerbestand < dMindestbestand)
            bErgebnis = true;
        else
            bErgebnis = false;
        return bErgebnis;
    }

    // Ende Methoden
}
```

Main Methode in Start-/UI-Klasse `AuftragsVerwUI`

```
public class AuftragsVerwUI
{
    /**
     * @param args
     * Kurzbeschreibung:
     */
    public static void main(String[] args)
    {
        // Aufgabe 1.5 und 1.6 ----------------------------------------
        Kunde kundeSchmid = new Kunde();
        String sName;
        String sVorname;
        String sAnschrift;
        String sEmail;
        int iZahlungsZiel;
        String sTelefon;
        double dRabattsatz;
        double dGesamtBestellsumme;
        int iAnzahlBestellungen = 0;

        System.out.println("Kunde 1 (ohne konkrete Daten):");
        System.out.println(kundeSchmid.toString() + "\n");

        System.out.println("Daten für Kunde 1:");
        sName = Eingabe.getString("Name = ");
        kundeSchmid.setSName(sName);
        sVorname = Eingabe.getString("Vorname = ");
        kundeSchmid.setSVorname(sVorname);
        sAnschrift = Eingabe.getString("Anschrift = ");
        kundeSchmid.setSAnschrift(sAnschrift);
        sEmail = Eingabe.getString("E-Mail = ");
        kundeSchmid.setSEmail(sEmail);
        sTelefon = Eingabe.getString("Telefon = ");
        kundeSchmid.setSTelefon(sTelefon);
        iZahlungsZiel = Eingabe.getInt("Zahlungsziel = ");
        kundeSchmid.setIZahlungsZiel(iZahlungsZiel);
        iAnzahlBestellungen = Eingabe.getInt("Anzahl Bestellungen = ");
        kundeSchmid.setIAnzahlBestellungen(iAnzahlBestellungen);
        dGesamtBestellsumme = Eingabe.getDouble("Bestellsumme = ");
        kundeSchmid.setDGesamtBestellsumme(dGesamtBestellsumme);
        dRabattsatz = Eingabe.getDouble("Rabattsatz = ");
        kundeSchmid.setDRabattsatz(dRabattsatz);

        System.out.println("\nKunde 1:");
        System.out.println(kundeSchmid.toString() + "\n");
```

```
// Aufgabe 1.8 ------------------------------------------------------------
Auftrag auftrag1 = new Auftrag();
Auftrag auftrag2 = new Auftrag(1, 2019, 1, 8, 1225);
LocalDate datum;
String sDatum;

datum = auftrag1.getDatum();
// Formatierung des Datums mit selbt geschriebener Methode
sDatum = A0301KalenderKlasse.getDatum(datum.getYear(), datum.getMonthValue(),
                                      datum.getDayOfMonth());

System.out.println("AuftragsNr = " + auftrag1.getIAuftragsNr() + "\n"
                 + "Datum = " + sDatum + "\n"
                 + "Auftragswert = " + auftrag1.getDAuftragswert());
// Formatierung des Datums mit DateTimeFormatter
datum = auftrag2.getDatum();
sDatum = datum.format(DateTimeFormatter.ofPattern("EE, dd. MMM. yy"));
System.out.println("AuftragsNr = " + auftrag2.getIAuftragsNr() + "\n"
                 + "Datum = " + sDatum + "\n"
                 + "Auftragswert = " + auftrag2.getDAuftragswert());

System.out.println("\nAuftrag 1:");
System.out.println(auftrag1.toString()+"\n");

System.out.println("\nAuftrag 2:");
System.out.println(auftrag2.toString()+"\n");

// Aufgabe 1.10 -----------------------------------------------------------------
Lieferant lieferer1 = new Lieferant();

lieferer1.setSName("Fa. Lieferant");
lieferer1.setSAnschrift("Industriestr. 22, 70031 Stuttgart");
lieferer1.setSLieferbedingungen("frei Haus");
lieferer1.setDGesamtumsatz(45000);
lieferer1.setAnzahlAuftraege(4);

System.out.println("\nLieferant 1:");
System.out.println(lieferer1.toString());
System.out.printf("Umsatz/Auftrag = %6.2f\n", lieferer1.getMittlerenAuftragswert());

lieferer1.erhoeheUmsatz(3500.50);

System.out.println("\nLieferant 1 (nach zusätzlichem Auftrag):");
System.out.println(lieferer1.toString());
System.out.printf("Umsatz/Auftrag = %6.2f\n", lieferer1.getMittlerenAuftragswert());

// Aufgabe 1.12 -----------------------------------------------------------------
Artikel artikel1 = new Artikel(4711, "Ladenhüter", 25.60);
artikel1.setINr(4711);
artikel1.setDMittlEinstPreis(150);
artikel1.setDLagerbestand(40);
artikel1.setDMindestbestand(10);
artikel1.setDVerkaufsPreis(210);
artikel1.setSArtikelgruppe("Haushalt");

System.out.println("\nArtikel 1:");
System.out.println(artikel1.toString());
System.out.printf("Gesamtlagerwert = %6.2f\n", artikel1.ermittleGesamtLagerwert());
artikel1.erhöhePreis(25);
System.out.println("\nArtikel 1 mit erhöhtem Preis:");
System.out.println(artikel1.toString());
System.out.printf("Lagerbestand = %f; Mindestbestand = %f\n",
                  artikel1.getDLagerbestand(), artikel1.getDMindestbestand());
System.out.printf("Bestellen? %b\n", artikel1.bestellen());
```

```
    // Aufgabe 1.15 ------------------------------------------------------------
    double dBestellwert;
    System.out.println("\nKunde 1 tätigt neue Bestellung:");
    dBestellwert = Eingabe.getDouble("Bestellwert = ");
    if (kundeSchmid.erhöheGesamtbestellsumme(dBestellwert) != 1)
        System.out.println("Bestellwert muss zwischen 200 und 2000 liegen!");

    System.out.println("Neue Daten Kunde 1:");
    System.out.println(kundeSchmid.toString() + "\n");

    Kunde neuerKunde = new Kunde();
    neuerKunde.setSName(Eingabe.getString("Neuer Kunde: Kundenname = "));
    System.out.println(neuerKunde.toString());
}
```

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.

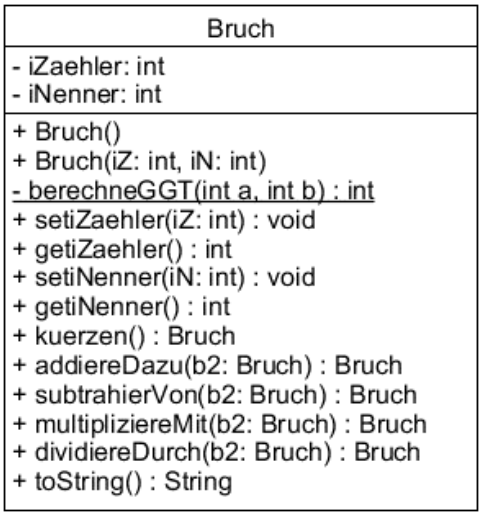
```java
/* Artikel.java
package oopAuftragsVerwV2;

/**
 * @author stk Kurzbeschreibung: Verwaltet die Daten eines Artikels
 *                               Erweiterte Version v2
 */
public class Artikel
{
    // Anfang Attribute
    private int iArtikelNr;
    private String sBezeichnung;
    private String sArtikelgruppe;
    private double dVerkaufsPreis;
    private double dLagerbestand;
    private double dMindestbestand;
    private double dMittlEinstPreis;
    // Klassenattribute
    private static double dMehrwertsteuersatz = 0.19;
    private static int iAnzahlErzeugterObjekte = 0;
    // Ende Attribute

    // Klassenmethoden
    public static double getdMehrwertsteuersatz()
    {
        return dMehrwertsteuersatz;
    }

    public static void setdMehrwertsteuersatz(double dMehrwertsteuersatz)
    {
        Artikel.dMehrwertsteuersatz = dMehrwertsteuersatz;
    }

    public static int getiAnzahlErzeugterObjekte()
    {
        return iAnzahlErzeugterObjekte;
    }

    public static void setiAnzahlErzeugterObjekte(int iAnzahlErzeugterObjekte)
    {
        Artikel.iAnzahlErzeugterObjekte = iAnzahlErzeugterObjekte;
    }

    // Konstruktoren
    public Artikel()
    {
        Artikel.iAnzahlErzeugterObjekte++;
    }

    public Artikel(String sBez)
    {
        this.sBezeichnung = sBez;
        Artikel.iAnzahlErzeugterObjekte++;

    }

    // Anfang Get- Set- und weitere Objektmethoden (s.o.) ...

    // Ende Methoden
}
```

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.

```
15  public class AuftragsVerwUI
16  {
17      /**
18       * @param args
19       * Kurzbeschreibung:
20       */
21      public static void main(String[] args)
22      {
23          Artikel einArtikel = null;
24          int iArtikelNr = 0;
25          double dVPreis;
26
27          do
28          {
29              iArtikelNr = Eingabe.getInt("ArtikelNr = ");
30              if (iArtikelNr != -1)
31              {
32                  dVPreis = Eingabe.getDouble("Verkaufspreis = ");
33                  einArtikel = new Artikel(iArtikelNr,
34                          "TollerArtikel" + (Artikel.getiAnzahlErzeugterObjekte() +1), dVPreis);
35                  System.out.printf("Artikel %s Bruttopreis: %f\n\n",
36                          einArtikel.getSBezeichnung(), einArtikel.getDVerkaufsPreisBrutto());
37              }
38          } while (iArtikelNr != -1);
39
40          System.out.printf("Anzahl erzeugter Artikel %d\n\n", Artikel.getiAnzahlErzeugterObjekte());
41          Artikel.setdMehrwertsteuersatz(0.20);
42          System.out.printf("Artikel %s Bruttopreis: %f\n\n",
43                  einArtikel.getSBezeichnung(), einArtikel.getDVerkaufsPreisBrutto());
44
45      }
46
47  }
```

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"

```java
 * Bruch.java
package oopBruchRechnen;

/**
 * @author stk
 * Kurzbeschreibung: Die Klasse Bruch dient zum Verarbeiten von Brüchen
 */
public class Bruch
{
    private int zaehler;
    private int nenner;

    // Standard Konstruktor
    public Bruch()
    {
        this.zaehler = 0;
        this.nenner = 1;
    }

    // überladener Konstruktor
    public Bruch(int einZaehler, int einNenner)
    {
        this.zaehler = einZaehler;  // Initialwert setzen
        if (einNenner != 0)
        {
            this.nenner = einNenner;
        }
        else
        {
            System.out.println("Fehler: Nenner darf nicht 0 sein");
        }
    }

    private static int berechneGGT(int a, int b)
    {
        int rest;

        if (a < 0) a = -a; // Algorithmus funktioniert nur für positive Zahlen
        if (b < 0) b = -b; // der ggT ist unabhängig vom Vorzeichen

        while (a % b != 0)
        {
            rest = b % a;
            b = a;
            a = rest;
        }
        return b; // b enthält den ggT
    }

    public void setiZaehler(int zaehler)
    {
        this.zaehler = zaehler; // Setze den Zähler des aktuellen
                                // Objektes auf den Wert des Parameters
    }

    public int getiZaehler()
    {
        return this.zaehler;
    }

```

```
public void setiNenner(int nenner)
{
    if (nenner != 0)
    {
        this.nenner = nenner; // Setze den Nenner des aktuellen
                              // Objektes auf den Wert des Parameters
    }
    else
        System.out.println("Fehler: Nenner darf nicht 0 sein");
}

public int getiNenner()
{
    return this.nenner;
}

public Bruch kuerzen()
{
    Bruch erg = new Bruch();
    int ggt = berechneGGT(this.zaehler, this.nenner);

    erg.zaehler = this.zaehler / ggt;
    erg.nenner = this.nenner / ggt;

    return erg;
}

public Bruch addiereDazu(Bruch b2)
{
    Bruch erg = new Bruch();

    erg.zaehler = this.zaehler * b2.nenner + b2.zaehler * this.nenner;
    erg.nenner = this.nenner * b2.nenner;

    erg = erg.kuerzen();

    return erg;
}

public Bruch subtrahiereDavon(Bruch b2)

public Bruch multipliziereMit(Bruch b2)

public Bruch dividiereDurch(Bruch b2)

public String ToString()
{
    Bruch temp = this.kuerzen();
    String erg;
    int iZaehlerAbs = Math.abs(temp.zaehler);
    int iNennerAbs = Math.abs(temp.nenner);

    if (temp.zaehler % temp.nenner == 0) // ganze Zahl
        erg = String.format("%d", temp.zaehler / temp.nenner);
    else
        if (iZaehlerAbs > iNennerAbs) // unechter Bruch
            if (this.zaehler * this.nenner > 0) // Bruchwert ist positiv
                erg = String.format("%d %d/%d", iZaehlerAbs/ iNennerAbs,
                        iZaehlerAbs % iNennerAbs, iNennerAbs);
            else                                // Bruchwert ist negativ
                erg = String.format("%d %d/%d", temp.zaehler / temp.nenner,
                        iZaehlerAbs % iNennerAbs, iNennerAbs);
        else                            // echter Bruch
            if (this.zaehler * this.nenner > 0) // Bruchwert ist positiv
                erg = String.format("%d/%d", iZaehlerAbs, iNennerAbs);
            else                                // Bruchwert ist negativ
                erg = String.format("-%d/%d", iZaehlerAbs, iNennerAbs);


    return erg;
}
```

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
```

```java
import input.Eingabe;

/**
 * @author stk
 *
 *
 *         Kurzbeschreibung: UI- und Start-Klasse für Bruchrechnen
 */
public class BruchUI
{

    /**
     * @param args
     *            Kurzbeschreibung:
     */
    public static void main(String[] args)
    {
        int iZ1, iZ2, iN1, iN2;
        boolean bFehler = false;
        String sOp;

        Bruch erg = null;
        Bruch b1 = new Bruch();
        Bruch b2 = new Bruch();

        iZ1 = Eingabe.getInt("Zaehler 1 = ");
        iN1 = Eingabe.getInt("Nenner 1 = ");
        b1.setiZaehler(iZ1);
        b1.setiNenner(iN1);

        iZ2 = Eingabe.getInt("Zaehler 2 = ");
        iN2 = Eingabe.getInt("Nenner 2 = ");
        b2.setiZaehler(iZ2);
        b2.setiNenner(iN2);

        sOp = Eingabe.getString("\nRechenoperator ? ");

        switch (sOp)
        {
        case "+":
            erg = b1.addiereDazu(b2);// erg = b1+b2;
            break;
        case "-":
            erg = b1.subtrahiereDavon(b2);// erg = b1-b2;
            break;
        case "*":
            erg = b1.multipliziereMit(b2);// erg = b1*b2;
            break;
        case "/":
            erg = b1.dividiereDurch(b2);// erg = b1/b2;
            break;

        default:
            bFehler = true;
            System.out.println("Ungültiger Rechenoperator!\n");
            break;
        }
        if (!bFehler)
            System.out.println("Ergebnisbruch: " + erg.ToString());
    }

}
```