

### 3 Fragen aus dem Unterricht zu „Fass“

In diesem Zusatzkapitel werde ich Probleme ansprechen, die sich im Laufe des Unterrichtsgesprächs ergeben haben. Ich werde Lösungen anbieten, ohne dass diese im Einzelnen besprochen werden. Vielmehr ist jeder Schüler hiermit aufgefordert, die vorgeschlagene Implementierung in Ruhe zu studieren und die Vorschläge in seine Hausaufgaben zu übernehmen.

Ich möchte noch einmal betonen, dass es nicht unbedingt notwendig ist, alles auf Anhieb zu verstehen. Es werden in diesen Vorschlägen immer wieder Strukturen verwendet, die im Unterricht an dieser Stelle noch nicht besprochen worden sind. Aber es schadet auf keinen Fall, wenn du dir darüber Gedanken machst.

Hausaufgabe war die Programmierung einer Klasse *Fass*. Die Implementierung konntest du analog der Klasse *Konto* erstellen.

Ein Fass hat jedoch noch weitere Eigenschaften, die bei der Implementierung zu berücksichtigen sind.

1. Ein Fass hat nur ein begrenztes Volumen von z.B. 5 Litern. Somit ist es nicht möglich, in ein Fass mehr hineinzuschütten, denn es würde überlaufen.
2. Aus einem Fass lässt sich nicht mehr herausnehmen als den aktuellen Inhalt, denn dann ist das Fass leer.

In der Hausaufgabe sollten diese Fälle durch eine Fehlermeldung behandelt werden.

#### 3.1 Implementierung der Klasse *Fass*

Eine mögliche Lösung wäre:

```
/**
 * Eine Basisklasse Fass
 *
 * @author Ralph Henne
 * @version 19.10.2003
 */

public class Fass
{
    ..... geeignete Datenfelder .....

    /** Konstruktor dient zur Erzeugung des Fasses */
    Fass(int volumenH, int inhaltH)
    {
```

```
        ..... geeigneter Konstruktor .....
    }

    /**
     * Liefert den aktuellen Fassinhalt.
     */
    int gibAktuellenFassinhalt()
    {
        ..... geeignete Anweisungen .....
    }

    /**
     * Setzt den aktuellen Fassinhalt.
     * @param menge ist gesetzter Inhalt des Weinfasses
     */
    void setzeAktuellenFassinhalt(int menge)
    {
        ..... geeignete Anweisungen .....
    }

    /**
     * Vergroessert den aktuellen Fassinhalt.
     * @param menge ist hinzugefuellter Wein
     */
    void fuellen(int menge)
    {
        if (rechneFuellRestmenge() < menge) {
            System.out.println("Das Fass wird ueberlaufen.");
        }
        else
            inhalt += menge;
    }

    /**
     * Verkleinert den aktuellen Fassinhalt.
     * @param menge ist ausgeschuetteter Wein
     */
    void leeren(int menge)
    {
        if (rechneLeerRestmenge() < menge) {
            System.out.println("So viel ist nicht mehr im Fass.");
        }
        else
            inhalt -= menge;
    }

    /**
     * Fuellt aus einem Fass um in ein anderes Fass.
     * @param neuFass in dieses Fass wird umgefuehrt
     */
```

```
* @param menge ist umgefüllter Wein
*/
void umfuellen(Fass neuFass, int menge)
{
    ..... geeignete Anweisungen .....
}

/**
 * Berechnet, wie viele Liter noch in das Fass
 * gefüllt werden koennen.
 */
private int rechneFuellRestmenge()
{
    int menge = volumen - inhalt;
    return menge;
}

/**
 * Berechnet, wie viele Liter noch aus dem Fass
 * geleert werden koennen.
 */
private int rechneLeerRestmenge()
{
    int menge = inhalt;
    return menge;
}
}
```

### Übung 4z.1:

Übernimm nun die oben gezeigten Methoden und untersuche diese mit Hilfe von **BlueJ**.

Übertrage die hier gezeigten Ideen in dein Hausaufgaben-Projekt und suche nach Verbesserungen.

## 3.2 Zugriffsmodifikatoren

Die Abbildung 4.1 zeigt die öffentlich zugänglichen Methoden des Objekts *fass1*.

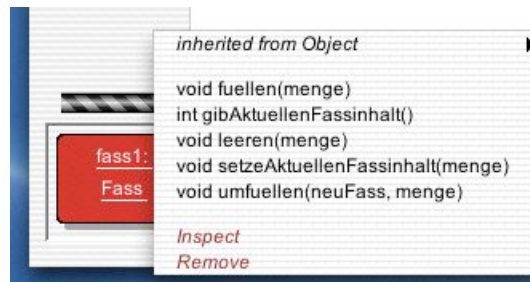


Abbildung 3.1: Öffentliche Methoden der Klasse *Fass*

Hier erscheinen nicht die Methoden *rechneFuellRestmenge()* und *rechneLeerRestmenge()*. Im Listing sind diese beiden Methoden mit dem Schlüsselwort *private* bezeichnet. Stillschweigend haben wir bisher mit dem Schlüsselwort *public* die Klasse *Fass* implementiert. Diese beiden Schlüsselwörter *public* und *private* heißen *Zugriffsmodifikatoren*.

Solche Zugriffsmodifikatoren definieren die Sichtbarkeit eines Datenfeldes, eines Konstruktors oder einer Methode. Wenn eine Methode als *public* vereinbart ist, kann sie innerhalb der Klasse und von jeder anderen Klasse aufgerufen werden. Methoden, die als *private* vereinbart sind, können nur innerhalb der Klasse aufgerufen werden, in der sie definiert wurden. Sie sind für andere Klassen nicht sichtbar und damit auch nicht zugreifbar.

In unserem Beispiel werden die Methoden *rechneFuellResmenge()* und *rechneLeerRestmenge()* benutzt, um eine größere Aufgabe in kleinere zu zerlegen. Diese Unteraufgaben sind also nicht dafür vorgesehen, von außerhalb der Klasse aufgerufen zu werden, sondern sie dienen nur einer übersichtlichen Strukturierung der Klasse. Somit werden also die Methoden als *private* vereinbart und erscheinen in Abbildung 4.1 auch nicht in der Liste aller von außerhalb der Klasse aufrufbaren Methoden.

In den folgenden Programmbeispielen werde ich Datenfelder meistens als *private* vereinbaren. Zugriffe auf solche privaten Datenfelder sollen also grundsätzlich nur noch erlaubt sein über sondierende und verändernde Methoden.

Methoden, die auch von anderen Klassen verwendet werden dürfen, werden als *public* vereinbart. Werden jedoch Methoden ausschließlich innerhalb einer Klasse verwendet, um die Strukturierung übersichtlicher zu gestalten, werden sie als *private* vereinbart.

#### Hinweis:

Dieses Prinzip wird auch als *Geheimnisprinzip* bezeichnet (engl.: *information hiding*)

Java definiert zwei weitere Zugriffsstufen. Die eine wird mit dem Schlüsselwort *protected* deklariert, die andere wird wirksam, wenn gar kein Modifikator angegeben ist. Weitere Einzelheiten dazu werde ich im Kapitel 5 „Bahn“ erwähnen.