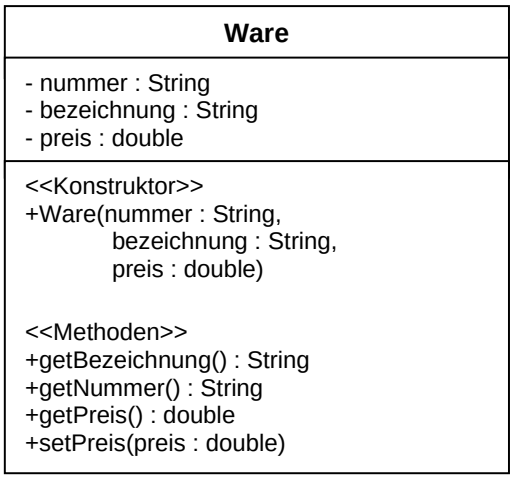
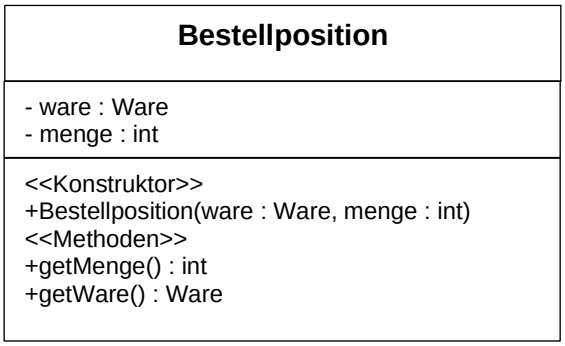
Übungen – Klassen und Objekte

Aufgabe 1 – Warenbestellungen

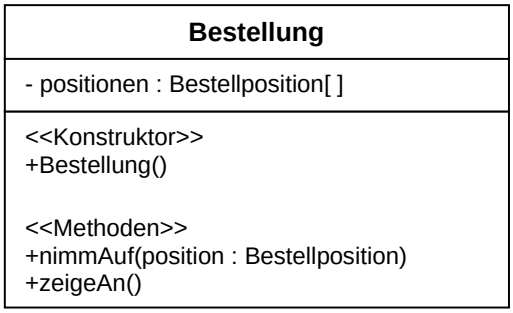
In einem Programm zur Ausfertigung von Warenbestellungen benötigt man die Klassen `Ware`, `Bestellposition` und `Bestellung`.
In einer `Bestellung` werden sog. `Bestellposition`*en* aufgeführt, wobei eine `Bestellposition` eine bestimmte `Ware` angibt und die Menge, die bestellt wird.
Die Klasse `Ware`:



Als Währungseinheit wird EUR angenommen; sie wird aber nicht gespeichert.
Die Klasse `Bestellposition`:



In einem Objekt der Klasse `Bestellposition` soll lediglich die bestellte Menge einer Ware gespeichert werden. Für den Bezug auf die bestellte `Ware` muss es eine Referenz auf die bestellte `Ware` aufnehmen.
Die Klasse `Bestellung`:



In einem Objekt der Klasse `Bestellung` soll eine Reihe von `Bestellpositionen` gespeichert werden, z. B. in einem Array (**`positionen`**). Dass ein Array nur eine beschränkte Anzahl von

Bestellpositionen aufnehmen kann, nehmen wir in Kauf. Für den Bezug auf die Bestellpositionen muss das Array Referenzen auf die Bestellpositionen aufnehmen.

Die Methode nimmAuf() fügt eine Bestellposition in die Bestellung ein.

Die Methode zeigeAn() gibt die Bestellung in Form einer Auflistung auf der Konsole aus, so wie unten im Beispiel gezeigt. Geben Sie sich bei der Formatierung keine allzu große Mühe. Wir werden in der nächsten Vorlesung einige Möglichkeiten kennenlernen.

Testen Sie die Klassen in einer kleinen Anwendung. Die Anwendung könnte so aussehen:

```
Bestellung best = new Bestellung();
Bestellposition pos;
Ware w;

w = new Ware("01019010", "Hammer", 19.00);
pos = new Bestellposition(w, 30);
best.nimmAuf(pos);

w = new Ware("01019020", "Zange", 17.00);
pos = new Bestellposition(w, 20);
best.nimmAuf(pos);

... // weitere Bestellpositionen...

best.zeigeAn();
```

Die Ausgabe könnte so aussehen:

```
01019010 Hammer, Preis: 19.0 EUR, Menge: 30
01019020 Zange, Preis: 17.0 EUR, Menge: 20
01019030 Schraubendreher, Preis: 12.0 EUR, Menge: 25
03073073 Reifen, Preis: 33.0 EUR, Menge: 15
03073074 Schlauch, Preis: 8.0 EUR, Menge: 16
03073103 Luftpumpe, Preis: 13.0 EUR, Menge: 5
```

Aufgabe 2 – Eine Klasse `Bruchzahl`

In Smalltalk gibt es als Bestandteil des Systems eine Klasse für ***rationale Zahlen***. Sie heißt `Fraction`. Zu den rationalen Zahlen kann man auch ***Bruchzahlen*** sagen.
Bekanntlich besteht eine Bruchzahl aus einem Zähler z und einem Nenner n, beides ganze Zahlen. Statt mit einem Bruchstrich, kann man eine Bruchzahl auch einfach als ein in Klammer geschriebenes Zahlenpaar angeben:

(z, n), z. B.: (3, 4), (5, 7)

In Java wird man Bruchzahlen als Objekte einer Klasse einführen, wobei dann Zähler und Nenner als Attribute gehalten werden.
Entwickeln Sie eine Klasse `Bruchzahl`, die mit Bruchzahlen umgeht. Die Methoden der Klasse stehen für die uns bekannten Rechenoperationen mit Bruchzahlen.

Erste Ausbaustufe

Konstruktor:
`public Bruchzahl(long z, long n)`
z ist der Zähler und n der Nenner der neuen Bruchzahl.

Methoden:
`public Bruchzahl addiere(Bruchzahl q)`
Addiert q zur *aktuellen Bruchzahl* und liefert ein neues Objekt `Bruchzahl`. Die *aktuelle Bruchzahl* und q bleiben unverändert.

`public Bruchzahl multipliziere(Bruchzahl q)`
Multipliziert die *aktuelle Bruchzahl* mit q und liefert ein neues Objekt `Bruchzahl`. Die *aktuelle Bruchzahl* und q bleiben unverändert.

Für die Ausgabe ist es nützlich, eine Methode zur Verfügung zu stellen, welche die Bruchzahl in netter Form anzeigt:

```
public void zeigeAn
{
      // Serie von System.out.print(...);
}
```

Schließlich wollen wir diese Klassen noch ausprobieren. Implementieren Sie die Klasse `BruchzahlTest` mit einer `main()`-Methode, in der alle Operationen durchgeführt werden. Zur Kontrolle kann man sich die Brüche und die Ergebnisse anzeigen lassen, z. B. in der Form:

```
Erste Bruchzahl:  (5, 7)
Zweite Bruchzahl: (3, 5)
(5, 7) + (3, 5) = (46, 35)
(5, 7) * (3, 5) = (15, 35)
```

Zweite Ausbaustufe

Fügen Sie weitere Methoden hinzu:

`public Bruchzahl subtrahiere(Bruchzahl q)`
Subtrahiert q von der *aktuellen Bruchzahl* und liefert ein neues Objekt `Bruchzahl`.
Die *aktuelle Bruchzahl* und q bleiben unverändert.
Die Subtraktion wird auf die Addition zurückgeführt, indem man das Negative von q zur *aktuellen Bruchzahl* addiert.

Wir führen für „das Negative von“ den Begriff „Gegenwert von“ ein und legen die Bildung des Gegenwertes in eine eigene Methode. Sie ist hier beschrieben:

`private Bruchzahl bildeGegenwert()`
Bildet das Negative der aktuellen Bruchzahl und liefert ein neues Objekt `Bruchzahl`.
Die *aktuelle Bruchzahl* bleibt unverändert.

`public Bruchzahl dividiere(Bruchzahl q)`
Dividiert die *aktuelle Bruchzahl* durch q und liefert ein neues Objekt `Bruchzahl`.
Die *aktuelle Bruchzahl* und q bleiben unverändert.
Die Division wird auf die Multiplikation zurückgeführt, indem man die *aktuelle Bruchzahl* mit dem Kehrwert von q multipliziert.
Für die Bildung des Kehrwerts gibt es wieder eine eigene Methode.

`private Bruchzahl bildeKehrwert()`
Bildet den Kehrwert von der *aktuellen Bruchzahl* und liefert ein neues Objekt `Bruchzahl`.
Die *aktuelle Bruchzahl* bleibt unverändert.
Den Kehrwert bekommt man, indem man Zähler und Nenner vertauscht.

Ergänzen Sie die Testroutine entsprechend.

```
Erste Bruchzahl:  (5, 7)
Zweite Bruchzahl: (3, 5)
(5, 7) + (3, 5) = (46, 35)
(5, 7) - (3, 5) = (4, 35)
(5, 7) * (3, 5) = (15, 35)
(5, 7) / (3, 5) = (25, 21)
```

Dritte Ausbaustufe

Vom Konstruktor und von den Methoden `addiere(), subtrahiere(), multipliziere()` und `dividiere()` soll die zurückzugebende Bruchzahl „normiert“ werden. Dazu wird für eine Bruchzahl die folgende Methode aufgerufen:

`private Bruchzahl normiere()`
Alle Veränderungen werden an der *aktuellen Bruchzahl* durchgeführt, und abschließend wird die *aktuelle Bruchzahl* zurückgegeben.
Falls der Nenner negativ ist, werden Zähler und Nenner mit `−1` multipliziert.
Falls Zähler und Nenner nicht beide `0` sind:
Falls der Zähler `0` ist, wird der Nenner `1`.
Falls der Nenner `0` ist wird der Zähler `1`.
Kürzen. Dazu wird die Methode `kuerze()` aufgerufen.

`private void kuerze()`
Kürzt die *aktuelle Bruchzahl*.
Ein Bruchzahl wird gekürzt, indem man Zähler und Nenner durch den größten gemeinsamen Teiler (ggT) dividiert. Eine Methode, die das ausführt, ist hier beschrieben:

`private long ggT(long a, long b)`
Ermittelt den ggT von a und b und liefert ihn zurück.
Wenn a und b zwei ganze positive Zahlen sind, dann berechnet sich der ggT der beiden Zahlen nach Euklid so:

Man bildet den Rest der Division von a und b. Ist der Rest 0, dann ist b der ggT. Andernfalls vertauscht man a durch b und b durch den Rest und führt Schritt 1 erneut durch. Dies wird so lange wiederholt, bis im Schritt 1 der Rest 0 ist.

Jetzt sieht das Resultat so aus:

```
Erste Bruchzahl:  (5, 7)
Zweite Bruchzahl: (3, 5)
(5, 7) + (3, 5) = (46, 35)
(5, 7) - (3, 5) = (4, 35)
(5, 7) * (3, 5) = (3, 7)
(5, 7) / (3, 5) = (25, 21)
```