Universität Augsburg
Institut für Informatik

Betreutes Programmieren zur Vorlesung Informatik 2 SS 2018
Fakultät für Angewandte Informatik
Lehrprofessur für Informatik
Prof. Dr. Lorenz, Marius Brendle, Johannes Metzger 02.05.2018

Betreutes Programmieren 3

Aufgabe 3 * *(Die Simulation des Biertrinkens)*

In der heutigen Aufgabe sollen Sie ein Klassendiagramm implementieren. Dieses modelliert die Daten und Operationen einer Anwendung zur Simulation des Biertrinkens.

Setzen Sie dazu das unten abgebildete UML-Klassendiagramm in Java-Code um, berücksichtigen Sie dabei die nachfolgende Systembeschreibung und die weiteren Beschreibungen unter dem Diagramm, und halten Sie sich an die Vorgaben. Im Gegensatz zur Vorlesung wird hier u.a. keine Ausnahmebehandlung gefordert und die Operationsnamen folgen nicht den in der Vorlesung vorgestellten Konventionen.

Systembeschreibung: Um das Trinken eines Bieres durchzuführen, benötigt man Bier – im vorliegenden Fall eine Flasche Bier, die man einem Kasten entnimmt, der entweder 6, 10 oder 20 solcher Flaschen beherbergt. Eine Flasche Bier ist in der Regel von einem bestimmten Hersteller und hat einen bestimmten Namen, bspw. **Corona**, **Guinness**, **Budweiser**, ..., der im Folgenden einfach verallgemeinert als Marke bezeichnet wird. So eine Bierflasche kann man in zwei verschiedenen Zuständen erhalten: nämlich offen und geschlossen. Ist Letzteres der Fall, muss man die Flasche zunächst öffnen, um daraus trinken zu können. Das Trinken selbst hat Auswirkungen auf die in der Flasche verbleibende Füllmenge, wodurch die Flasche im Laufe des Trinkvorgangs Schritt für Schritt leer wird. Sollte der Trinkvorgang nicht mehr durchführbar sein aufgrund der Leere der Bierflasche, so sollte man sich ggf. eine neue Flasche aus dem Kasten besorgen.

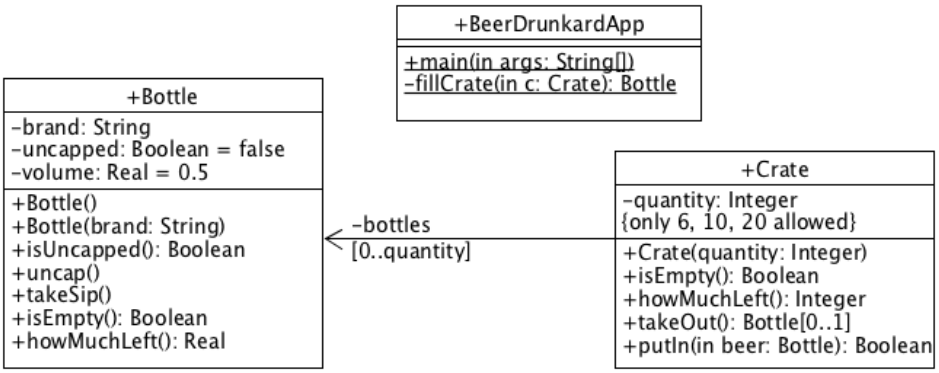

Abbildung 1: UML-Klassendiagramm zum Biertrinker

Der Pfeil zwischen von der Klasse `Crate` zu der Klasse `Bottle` symbolisiert eine sog. Assoziation zwischen den beiden Klassen - in diesem Fall, dass ein Kasten eben Flaschen enthält (Assoziationen werden in Kapitel 6 der Vorlesung besprochen). Die Beschriftung der Assoziation mit `-bottles` wird in Java als privates Attribut der Klasse `Crate` umgesetzt (welches man als Referenzattribut bezeichnet). Die zusätzliche Angabe `[0..quantity]` ist eine Multiplizität und bedeutet, dass ein Kasten mit `0` bis *quantity* vielen Flaschen gefüllt sein kann (`quantity` kann gemäß der Einschränkung entweder der Wert 6, 10 oder 20 sein). Über das Attribut `bottles` können damit die Flaschen in einem Kasten in der Klasse `Crate` verwaltet werden.

(a) (Klasse `Bottle`)
Implementieren Sie die Klasse `Bottle` nach den unten angegebenen Vorgaben. Wie in der Beschreibung erklärt und dem Diagramm entnehmbar, hat eine Flasche eine Marke (`brand`), den Zustand geöffnet oder geschlossen (`uncapped`) und darüber hinaus noch eine gewisse Füllmenge (`volume`).

- Der erste Konstruktor ohne Eingabeparameter soll eine ungeöffnete Halbliter-Flasche `Guinness` erzeugen und die Meldung **Freshly tapped beer! Cheers!** ausgeben.
- Der zweite Konstruktor mit einem Eingabeparameter soll die Bestellung einer Flasche Bier einer bestimmten Marke ermöglichen, sich sonst aber wie der erste Konstruktor verhalten.
- `isUncapped()`: Diese Methode gibt den Wert des Attributs `uncapped` zurück.
- `uncap()`: Mit dieser Methode wird die Flasche geöffnet, indem das Attribut `uncapped` auf `true` gesetzt wird sowie die Meldung **Uncapping good fresh <brand>** ausgegeben wird, wobei die passende Marke in die Meldung eingesetzt sein soll. Sollte die Flasche bereits geöffnet sein, soll die Methode einfach nichts tun.
- `takeSip()`: In dieser Methode wird ein Schluck aus der Flasche genommen, indem der Wert von `volume` um 0.1 reduziert wird, allerdings nur, wenn die Flasche bereits offen und noch nicht leer ist. Im Schluckfall wird der Wert `true` zurückgegeben und die Nachricht **Swig!** ausgegeben. Andernfalls erhält man `false` als Rückgabewert und die Meldung **Bottle is either capped or empty.** wird ausgegeben.
- `isEmpty()`: Der Rückgabewert dieser Methode ist `true`, wenn die Flasche leer ist (also das Attribut `volume` gegen Null geht), sonst `false`.
- `howMuchLeft()`: Diese Methode liefert den Wert des Attributs `volume`.

(b) (Klasse `Crate`)
Implementieren Sie die Klasse `Crate` nach den unten angegebenen Vorgaben. Laut Beschreibung und Diagramm hat eine Kiste ein maximales Fassungsvermögen (`capacity`) und Platz für maximal so viele Flaschen.

- Realisieren Sie die Rolle `bottles` der Assoziation zwischen den Klassen `Crate` und `Bottle` durch ein Attribut `private ArrayDeque<Bottle> bottles` der Klasse `Crate`. Die Klasse `java.util.ArrayDeque` ist eine der Collection-Klassen aus der Java-API und und verwaltet die Elemente ähnlich wie ein Kellerspeicher, der bereits in Informatik 1 eingeführt wurde.
- Der einzige Konstruktor soll alle Attribute passend initialisieren und dabei die Einschränkung bzgl. des Fassungsvermögen umsetzen. Abschließend soll die Meldung **A crate for <quantity> bottles of beer!** ausgegeben werden, wobei die passende Anzahl in die Meldung eingesetzt sein soll. Dabei soll je nach Wert von `quantity` auf das näheste erlaubte Fassungsvermögen einer Kiste auf- bzw. abgerundet werden.
- `isEmpty()`: Falls `bottles` keine Elemente mehr enthält, ist der Rückgabewert dieser Methode `true`, sonst `false`.
- `howMuchLeft()`: Der Rückgabewert dieser Methode ist die Anzahl der Elemente von `bottles`.

- `takeOut()`: Diese Methode gibt das nächste Element von `bottles` zurück. Sollte `bottles` leer sein, ist der Rückgabewert `null`.
- `putIn(..)`: Die Aufgabe dieser Methode ist es, die übergebene Flasche in `bottles` einzufügen und `true` zurückzugeben. Sollte `bottles` allerdings bereits voll sein, ist der Rückgabewert `false` und die Flasche bleibt draußen.

Die dritte Klasse `BeerDrunkardApp` enthält lediglich die `main(..)`-Methode und eine Methode, mit der eine Getränkekiste mit Flaschen befüllt wird. Diese Klasse wird Ihnen bereits fertig implementiert vorgegeben. Erklären Sie bei der Abnahme, was genau in diesen beiden Methoden passiert! Überprüfen Sie außerdem, wie viele Schlucke genau durchgeführt werden bis eine Flasche leer ist und versuchen Sie den Effekt zu erklären! Überlegen Sie sich eine Lösung, damit dieser Effekt nicht mehr auftritt und das eigentlich erwartete Verhalten vorhanden ist.