Objektorientierte Systementwicklung:

Grundlagen

Inhaltsverzeichnis:

1. Objektorientierte Systemanalyse- und entwicklung

1.1 Einführungsbeispiel

Sachverhalt:
Der Getränkemarkt GeLa GmbH entschließt sich, sein Sortiment mittels einer zu entwickelnden objektorientierten Software zu verwalten.
Für das Einstiegsbeispiel wird davon ausgegangen, dass die Kunden nur einen Kassenzettel erhalten. Die Kassenverwaltung (z. B.: Kassenbestand berechnen) ist aus Gründen der Vereinfachung ebenfalls ausgeklammert.
In einer ersten Projektphase soll folgendes Szenario analysiert werden:
Der Getränkemarkt bietet diverse Getränke an. Von diesen Getränken werden die Bezeichnung, der Verkaufspreis, ein Mindest- und ein Höchstbestand sowie der aktuelle Lagerbestand erfasst. Die Verkäufe erfolgen zur Zeit ausschließlich gegen Barzahlung. Jeder Kunde erhält nach getätigtem Umsatz einen Kassenzettel. Es ist zu beachten, dass eine Bestellung dann erfolgt, wenn der aktuelle Bestand den Mindestbestand unterschritten hat. Die zu bestellenden Einheiten ergeben sich dann aus der Differenz zwischen dem Höchstbestand und dem aktuellen Lagerbestand.

In der objektorientierten Softwareentwicklung wird die zu entwickelnde Anwendung in Diagrammen modelliert. Diese Diagramme werden durch die Modellierungssprache UML bereitgestellt. „Die Unified Modeling Language (UML, engl. Vereinheitlichte Modellierungs-Sprache) ist eine von der Object Management Group (OMG) entwickelte und standardisierte Sprache für die Modellierung von Software und anderen Systemen.“1
Welche Diagramme das Entwicklerteam auswählt, hängt von der Art der zu entwickelnden Programme ab. Im Folgenden werden

- Anwendungsfall-Diagramme (auch Use-Case-Diagramm)
- Aktivitätsdiagramme
- Klassendiagramme

erstellt, um zu einem Modell der Anwendung zu gelangen.

[1] Wikipedia zu UML

2 Anwendungsfall-Diagramm

2.1 Eingangsbeispiel

Zur Analyse der verschiedenen Abläufe, die bei der GeLa GmbH täglich anfallen, sollen deren Strukturen und Zusammenhänge untereinander beschrieben werden. Außerdem sollen die Zusammenhänge zwischen den Geschäftsfällen und den beteiligten Personen aufgezeigt werden. Die systematische Darstellung der Strukturen und Zusammenhänge soll mit Hilfe von Diagrammen der standardisierten Modellierungssprache UML erfolgen.
Mittels eines Anwendungsfall-Diagramms (USE-CASE-Diagramm) soll die reale Gegebenheit für das zu erstellende Programm näher analysiert werden.
Ein Anwendungsfall-Diagramm zeigt die Akteure (Menschen aber auch technische Systeme), Anwendungsfälle (Tätigkeiten, Geschäftsvorfälle, Geschäftsprozesse) und die Beziehungen zwischen Akteuren und Anwendungsfällen. Das Anwendungsfall-Diagramm zwingt den Entwickler, die Realität aus der Sicht des zukünftigen Systems zu betrachten. Akteur ist in unserem Beispiel der kaufmännische Angestellte und der Verkäufer. Die Tätigkeiten des Lagerarbeiters sind nicht durch das zu entwickelnde System zu unterstützen, weshalb sie im Anwendungsfall-Diagramm nicht aufzuführen sind.
Akteur ist auch der Kunde, der durch seine Einkäufe Prozesse in Gang setzt.
Anwendungsfälle im Einstiegsbeispiel sind: Getränke neu aufnehmen, Bestellmenge eines Getränks berechnen, Kassenzettel drucken und eine bestimmt Anzahl an Einheiten eines Getränks verkaufen. Anwendungsfälle beschreiben, was ein System leisten soll. Wie diese Leistung vom System zu erbringen ist, darüber treffen Anwendungsfälle keine Aussagen.
Das Anwendungsfall-Diagramm ist für den Unterricht so zu interpretieren, dass die zu entwickelnde Anwendung die abgebildeten Anwendungsfälle berücksichtigen sollte. Auf Softwareergonomie, Datenspeicherung und graphische Oberfläche wird hier noch nicht eingegangen.
Das Anwendungsfall-Diagramm trägt aus didaktischer Sicht zum besseren Verständnis der Aufgabenstellung bei. Die Problematik wird aus Sicht der Anwendung formal beschrieben, entsprechend sollte der Name des Anwendungsfalles aus der Perspektive des betrachteten Systems formuliert werden. Die im Anwendungsfall-Diagramm rigide Einschränkung der Ausdruckmöglichkeiten zwingt den Entwickler zu klaren Definitionen, die auch ein zukünftiger Nutzer verstehen kann. Ist das Anwendungsfall-Diagramm gezeichnet, so besteht eine grobe Vorstellung von der zukünftigen Software.

Bemerkung:

Für die **computergestützte Modellierung** der im im Folgenden dargestellten Anwendungsfall- und Aktivitätsdiagramme gibt es eine große Auswahl an Tools. Wir möchten an dieser Stelle exemplarisch auf das Tool Violet (http://violet.sourceforge.net) hinweisen, welches sowohl lokal installiert als auch direkt aus dem Internet eingesetzt werden kann.

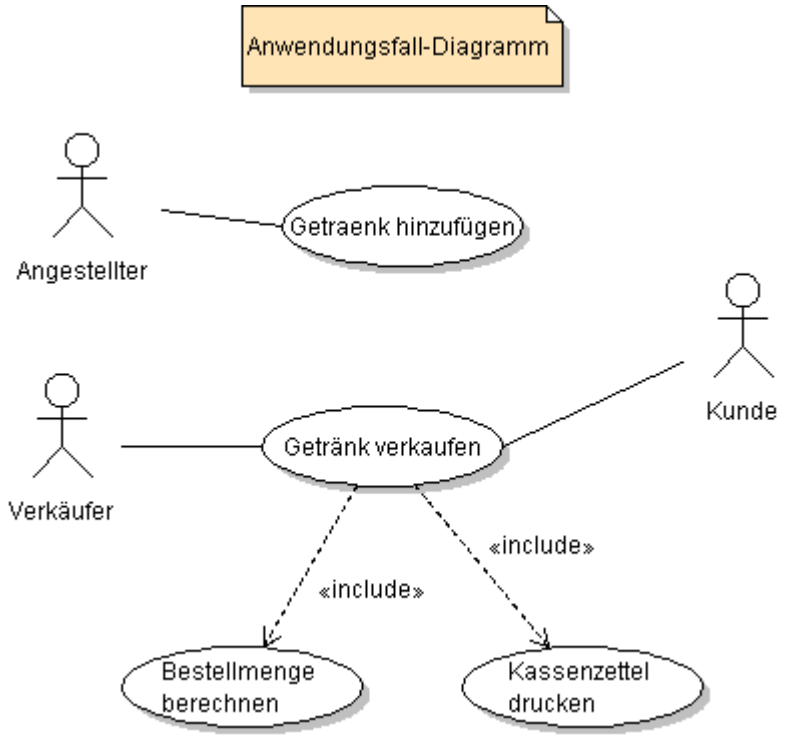


Abb. 1: Anwendungsfall-Diagramm zum Einstiegsbeispiel

Werdegang des Anwendungsfall-Diagrammes:

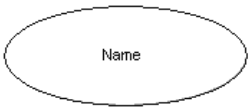
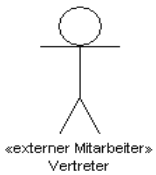
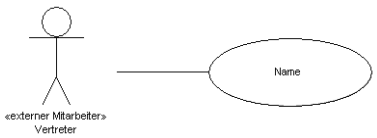
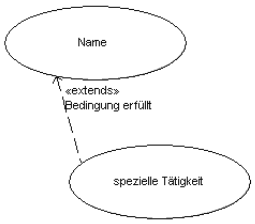
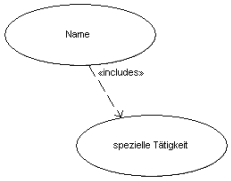
- Finden und Definieren der Akteure.
  Die Akteure stellen alle Beteiligten an einem bestimmten Vorgang dar. Im Informatikunterricht werden die Probleme oft didaktisch reduziert, sodass die Akteure in der Regel Menschen sind. Akteure können aber auch Fremdsysteme oder Zeitereignisse sein. Im gegebenen Fall wurden als Akteure Verkäufer, Angestellte und Kunden ausgemacht.

- Finden und Definieren von Anwendungsfällen.
  Die Kernidee besteht in der Beschreibung eines Ausschnitts der Software in einem sehr frühen Stadium des Projekts - möglichst noch vor den Prototypen, auf jeden Fall aber vor der Programmierung des Codes. Die Namen der Anwendungsfälle ergeben sich aus Tätigkeiten, die ein Anwendungsfall einfordert. Anwendungsfälle werden durch Ellipsen dargestellt.

- Aufzeigen der Beziehungen zwischen Akteuren und Anwendungsfällen.
  Sobald die Akteure und die Anwendungsfälle festgelegt sind, können auch die Beziehungen zwischen den Akteuren und Anwendungsfällen definiert werden. Dies geschieht durch Linien, welche die beteiligten Elemente verbinden.

Nach Durchführung der obigen Schritte liegt als Ergebnis das Anwendungsfall-Diagramm vor. Dieser Typ von Diagramm beschreibt keine Abläufe und auch kein Verhalten eines Akteurs. Es dient nur der Ermittlung der Anforderungen an das System. Die Granularität eines Anwendungsfall-Diagramms, damit ist die Feinheit der Beschreibung und der Zerstückelungsgrad des Gesamtprojekts gemeint, liegt im Ermessen des Entwicklers. Eine Faustregel jedoch besagt, dass ein Anwendungsfall sowohl hinsichtlich der zeitlichen Dauer als auch vom Umfang der zu verrichtenden Tätigkeiten überschaubar bleiben sollte.

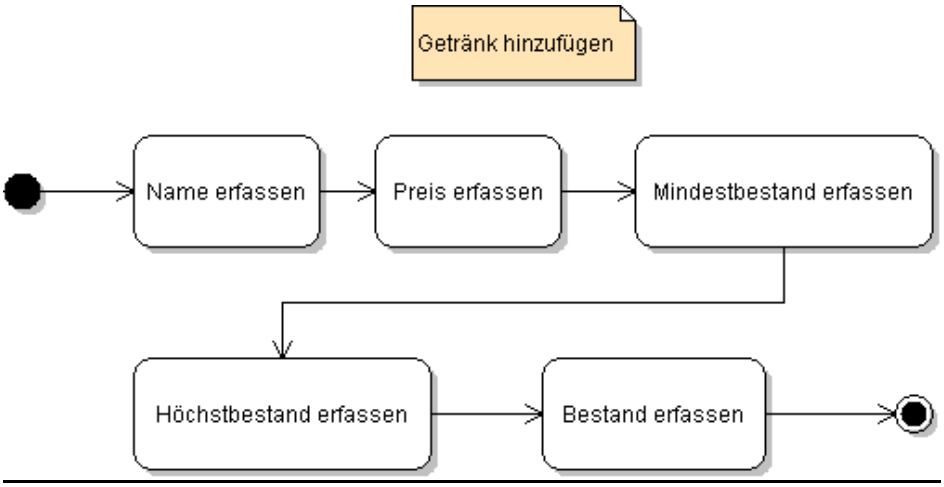
2.2 Gestaltungselemente des Anwendungsfall-Diagramms (unter Berücksichtigung der Möglichkeiten von eclipse):

Name/Symbol	Erklärung
Anwendungsfall 	Ein Anwendungsfall wird durch eine Ellipse dargestellt. Sie enthält den Namen des Anwendungsfalles. Dieser ergibt sich durch die Tätigkeit, die durchzuführen ist.
Akteur 	Ein Akteur löst einen Anwendungsfall aus. Ein Akteur wird durch ein Strichmännchen repräsentiert. Unterhalb der Abbildung kann ein <<Stereotyp>> angegeben sein, der den Akteur kategorisiert. Der Name des Akteurs kommt seiner Rolle im Anwendungsfall gleich.
Beziehung 	Ein Akteur steht in Beziehung zu einem Anwendungsfall. Diese Beziehung wird durch eine Linie zwischen den beiden Komponenten dargestellt.
Erweiterung 	Wird ein Anwendungsfall unter einer bestimmten Bedingung durch einen weiteren Anwendungsfall ergänzt, wird vom ergänzenden Anwendungsfall ein Pfeil nach dem ergänzten Anwendungsfall gezeichnet. Der Pfeil erhält die Beschriftung „extends“.
Einschluss 	Ist ein Anwendungsfall in einem anderen Anwendungsfall enthalten, werden beide Anwendungsfälle durch einen Pfeil mit der Beschriftung „include“ verbunden. Die Pfeilspitze zeigt auf den enthaltenen Anwendungsfall.

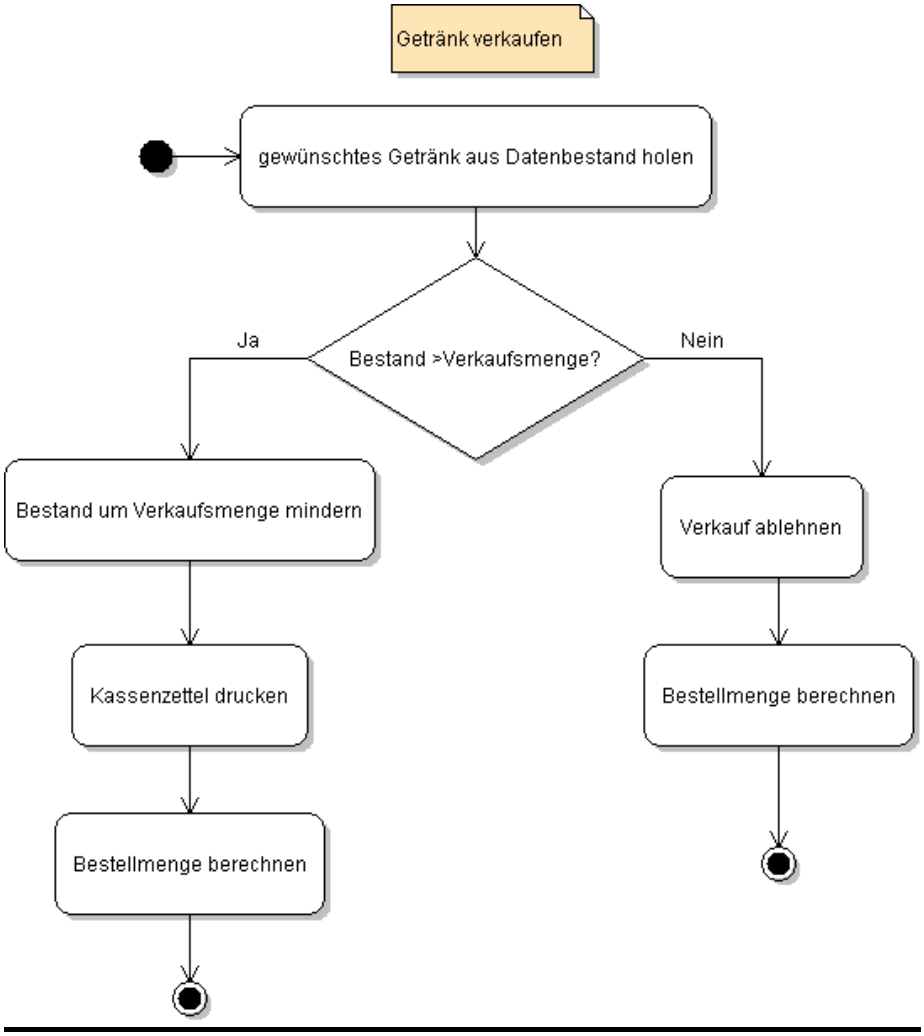
3. Aktivitätsdiagramm

3.1 Fortführung des Beispiels

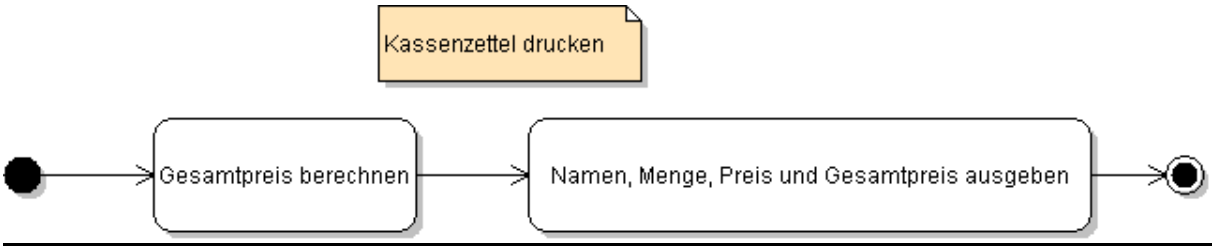
Nachdem bei der GeLa GmbH mit Hilfe eines Use-Case-Diagramms die Strukturen und Zusammenhänge der anfallenden Geschäftsfälle modelliert wurden, sollen nun die Arbeitsabläufe dargestellt werden. Dabei sollen die Aktivitäten, die den Ablauf eines Geschäftsprozesses ausmachen, in ihren Abhängigkeiten zueinander analysiert werden.
Im Anwendungsfall-Diagramm werden die Anwendungsfälle nicht beschrieben, lediglich ihre Existenz wird aufgeführt. Ein Aktivitätsdiagramm ist ein Ablaufdiagramm und dient der Modellierung der Abläufe in einem Anwendungsfall. Jeder Anwendungsfall wird durch ein Aktivitätsdiagramm erläutert.
Aus dem obigen Anwendungsfall-Diagramm lassen sich Aktivitätsdiagramme ableiten. Dabei erhält jeder Anwendungsfall ein Aktivitätsdiagramm. Hier soll mit dem Aktivitätsdiagramm für den Anwendungsfall „Getränk hinzufügen“ begonnen werden.

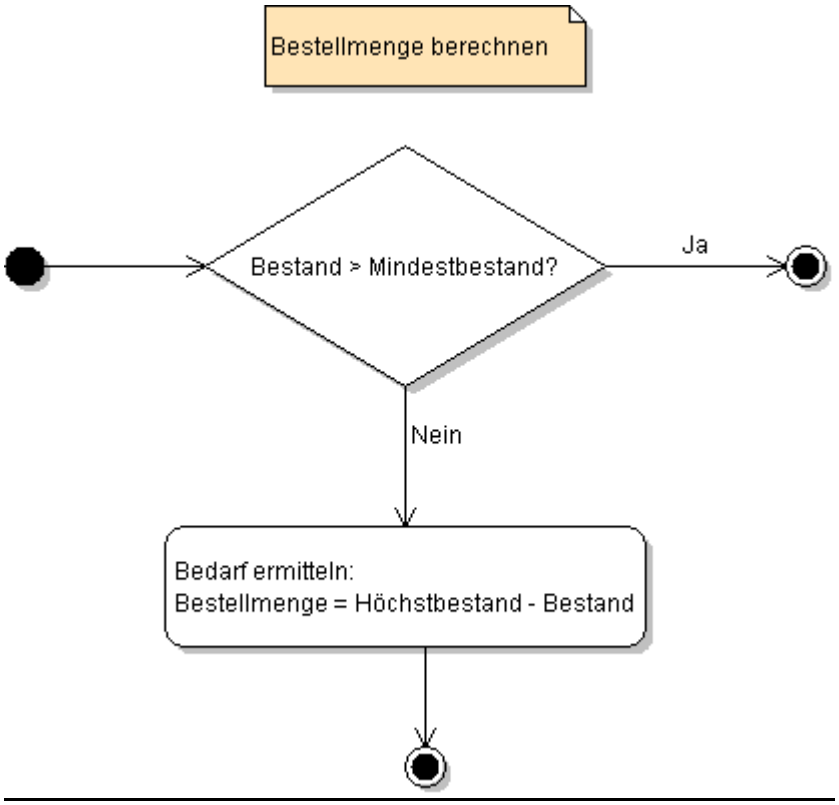


Das Aktivitätsdiagramm führt die Tätigkeiten auf (hier Erfassen der Werte, die ein Getränk beschreiben), die im entsprechenden Anwendungsfall durchgeführt werden. Gewissermaßen als Ergebnis der oben dargestellten Aktivität entsteht ein Objekt, welches ein Getränk repräsentiert.
Der dargestellte Ablauf ist später programmtechnisch zu realisieren. Zunächst aber sollen die verbleibenden Anwendungsfälle in Aktivitätsdiagramme überführt werden.

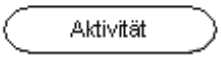
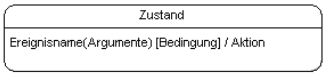
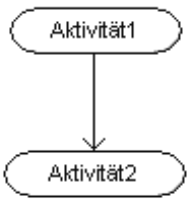
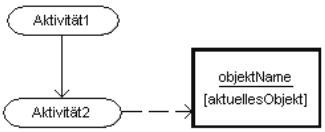
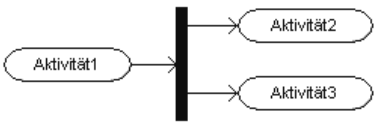
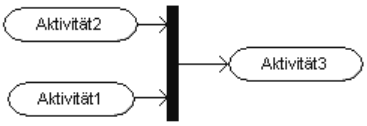


In Abhängigkeit davon, ob der im Lager vorrätige Bestand die Verkaufsmenge überschreitet, verzweigt das Aktivitätsdiagramm, was durch die Raute zum Ausdruck kommt.
Zur vollständigen Umsetzung des Anwendungsfall-Diagramms müssen jetzt noch die Anwendungsfälle „Kassenzettel drucken“ und „Bestellmenge berechnen“ jeweils in ein Aktivitätsdiagramm überführt werden.




Bestellmenge berechnen
Bestand > Mindestbestand?
Ja
Nein
Bedarf ermitteln:
Bestellmenge = Höchstbestand - Bestand

3.2 Gestaltungselemente des Aktivitätsdiagramms (toolabhängig):

Name/Symbol	Erklärung
Aktivität 	Symbol für eine Aktivität. Eine Aktivität beschreibt einen Ablauf. Der Name der Aktivität beschreibt die Tätigkeit des Systems.
Status 	Ein Status beschreibt das Eintreten eines Ereignisses. Der Status ist statisch und definiert einen Zustand. In der Regel werden hier Zustände beschrieben, die einer Entscheidung bedürfen. Die Kriterien, die der Entscheidung zu Grunde liegen, sind angegeben.
Programmfluss 	Aktivitäten, die nacheinander durchgeführt werden, sind durch einen Pfeil verbunden. Auf Aktivität 1 erfolgt hier Aktivität 2.
Raute 	Die Raute symbolisiert eine Entscheidung. Hier trennt sich der Programmfluss oder er wird zusammengeführt.
Objekt 	Hier werden ein bearbeitetes Objekt und dessen Zustand beschrieben.
Objektfluss 	Hier wird gezeigt, welche Aktivitäten ein Objekt verändern. Der Pfeil verbindet die das Objekt verändernde Aktivität und das Objekt.
Startpunkt 	Der Eintritt in die Aktivitätskette.
Endpunkt 	Der Endpunkt der Aktivitätskette.
Splitter 	Hier wird ein Programmfluss in parallele Abläufe aufgeteilt.
Synchronisierer 	Hier können parallele Abläufe, die ein Splitter auftrennte, wieder zusammengeführt werden.

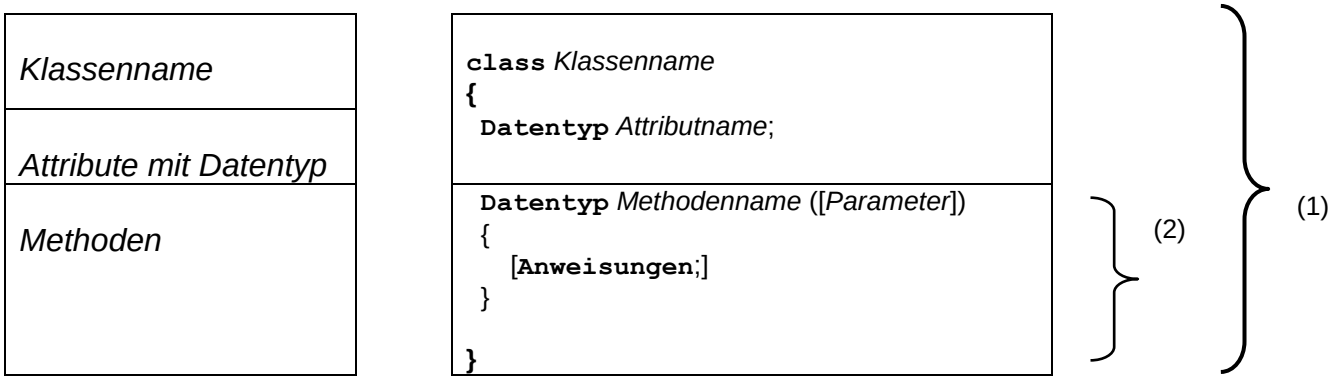
4. Klassendiagramm

4.1 Fortführung des Beispiels

Das Software-Entwicklerteam der GeLa GmbH möchte in einem weiteren Schritt der Problemanalyse die Objekte, die an den realen Geschäftsprozessen beteiligt sind, identifizieren. Objekte, die gleiche oder ähnliche Eigenschaften aufweisen, sollen zusammengefasst werden.
Die Aktivitätsdiagramme zeigen, dass sich die vorliegende Softwareproblematik in erster Linie auf Getränke bezieht. Ein Getränk ist hierbei ein Objekt unserer Welt, etwas das es in der Wirklichkeit gibt. In einem Computer kann natürlich nicht das reale Objekt erfasst werden, sondern nur die Beschreibung des Objekts. Durch sie wird das Objekt repräsentiert. Diese Repräsentationen im Rechner werden ebenfalls Objekte genannt. Sie abstrahieren die konkreten Objekte.
Objektorientierte Softwareentwicklung bedeutet nun, dass der Softwareaufbau an solchen Objekten ausgerichtet ist und Objekte im Zentrum der Betrachtung stehen.
Ein Objekt ist für uns damit eine Abstraktion eines konkreten Gegenstandes (Mitarbeiter, Artikel, Ort, Gebäude etc.) unserer Welt. Ein Objekt kann aber auch einen Vorgang unserer Anschauung (Zahlung, Mahnung, Produktion, Unterricht etc.) repräsentieren.
Objekte, die gleich beschrieben werden können, werden in Klassen zusammengefasst. Werden z.B. alle Mitglieder eines Vereins mit Namen, Vornamen, Geschlecht, Geburtstag usw. erfasst, dann formuliert man das in der objektorientierten Analyse wie folgt: Der Verein hat Objekte, die der Klasse Mitglied angehören (vom Typ Mitglied sind). Die Klasse Mitglied hat die Attribute (Eigenschaften der Objekte) Namen, Vornamen, Geschlecht, Geburtstag usw.. Alle Objekte einer Klasse werden nach der gleichen Regel erstellt. Das bedeutet hier, dass alle Mitglieder mit den gleichen Attributen beschrieben werden, jeder hat einen Namen, Vornamen, Geschlecht, Geburtstag usw..

Die Darstellung von Klassen erfolgt mit Hilfe der UML (Unified Modeling Language) im so genannten Klassendiagramm.

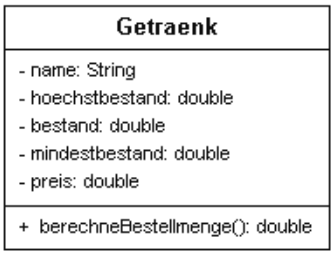
In der folgenden Darstellung ist links der Aufbau eines UML-Klassendiagramms, rechts daneben der Aufbau einer Klasse in Java abgebildet.



Festlegungen:
- Klassennamen beginnen mit Großbuchstaben.
- Attributs- und Methodennamen beginnen mit einem Kleinbuchstaben
- Namen dürfen keine Leerzeichen enthalten.

- Die Klassendefinition erfolgt mit dem Schlüsselwort *class,* der Klassenrumpf wirddurch {….} begrenzt **(1)**.
- An Methoden können Parameter zur Verarbeitung übergeben werden. Andererseits können Methoden auch am Ende der Verarbeitung Parameter zurückgeben (Rückgabewerte). Der Methodenrumpf steht zwischen {…..} **(2)**

Für unser konkretes Beispiel bedeutet dies, dass wir eine Klasse Getränk konstruieren müssen. Jedes Getränk wird als Objekt dieser Klasse abgebildet und wird mit Name, Höchstbestand, Mindestbestand, Bestand und Preis beschrieben.
Das folgende Schaubild steht für die Klasse Getränk:

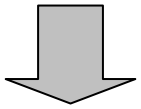


Im ersten Teil des Rechtecks steht der Name der Klasse, im zweiten Teil sind die Attribute aufgeführt, die ein Objekt der Klasse beschreiben. Von jedem Attribut wird auch der Datentyp genannt: Das Attribut Name ist vom Typ String (Buchstaben und Zahlen sind erlaubt). Die Mengen und der Preis sind vom Typ Double (Dezimalzahlen).
Die Minuszeichen vor den Attributnamen weisen darauf hin, dass die Attribute nur lokal (private) definiert sind. Das bedeutet, die Attribute können nur innerhalb der Klasse gesehen werden.
Im unteren Teil des Klassendiagramms sind die Methoden aufgeführt. Man kann sie als kleine Programme verstehen, die in der Verantwortung der Klasse liegen. Für die Klasse Getraenk ist bisher nur eine Methode vorgesehen, dies ist die Methode „berechneBestellmenge“. Das Pluszeichen vor dem Methodennamen (public) bedeutet, dass die Methode auch außerhalb der Klasse aufgerufen werden kann, der Eintrag „double“ bedeutet, dass die Methode einen Wert vom Typ Double (Kommazahl) zurückgibt.
Der wesentliche Unterschied zur klassischen Datenhaltung und dem Entity-Relationship-Modell besteht hier darin, dass die Klassen auch für die Operationen, mit welchen die Attribute verändert werden, verantwortlich sind. Das wird später beim Programmieren deutlich.

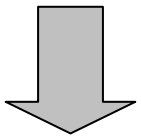
4.2 Ergänzungen zur objektorientierten Analyse

Ausschnitte der realen Welt			Erklärung:
			Ein Objekt ist eine Abstraktion von Dingen der realen Welt mit einer – eindeutigen Bezeichnung (Namen) – Eigenschaften und einem – Verhalten
Irish Ale	**Mineralwasser**	**Portwein**	

Abstrakte Beschreibung: Klasse



Getraenk	Name der Klasse
Name Bestand Mindestbestand Höchstbestand Preis	Eigenschaften (Attribute)
berechneBestellmenge()	Verhalten (Methoden)



- Eine Klasse ist der Plan zum Erzeugen von Objekten.
- Alle Objekte der Klasse haben die gleichen Attribute sowie das gleiche Verhalten.
- Elemente von Klassen werden als Objekte oder als Exemplare der Klasse bezeichnet.

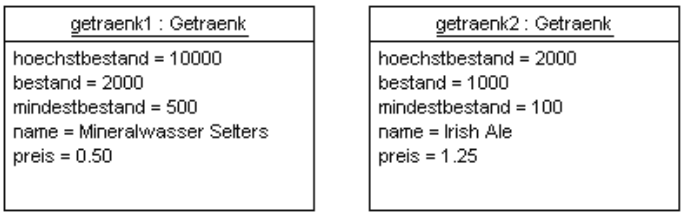
Der Zustand eines Objekts ist durch die Attributswerte gegeben.

- Mit Methoden können die Attributswerte verändert werden.

Aus der Klasse Getraenk erzeugte Objekte:

getraenk1	getraenk2	getraenk3
name=Irish Ale preis= 1,25 usw.	name= Mineralwasser preis = 0,35 usw.	name=Portwein preis=32,50 usw.

Objekte können in Objektdiagrammen dargestellt werden. Für die konkreten Objekte getraenk1 und getraenk2 sieht das Objektdiagramm wie folgt aus:

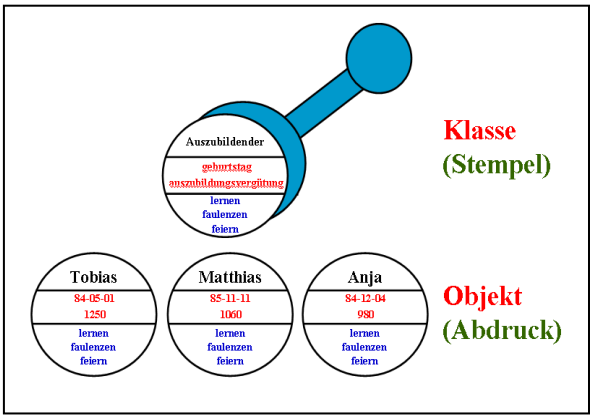


Das Aussehen eines Objektdiagramms erinnert an das Klassendiagramm. Im Objektdiagramm werden exemplarisch Objekte, die zu einem bestimmten Zeitraum existieren, angezeigt (Schnappschuss von den Objekten zu einem bestimmten Zeitpunkt). Die Attributswerte werden angeführt. Methoden werden im Objektdiagramm nicht angezeigt. Objektdiagramme werden seltener als Klassendiagramme verwendet.

Im Folgenden werden Klassen, Objekte und Attribute noch einmal zusammenfassend allgemein beschrieben:

Klassen

Wie beschreibt man Objekte? Der Schlüssel dazu kommt vom Ansatz der abstrakten Datentypen. Man beschreibt nicht individuelle Objekte, sondern beschränkt sich auf die einer ganzen Klasse von Objekten gemeinsamen Muster – die Klasse der Bücher, die Klasse der Lehrer, die Klasse der Kundendaten. Der Begriff Klasse ist der anerkannte Begriff zur Beschreibung solcher Muster, welche die Basis für objektorientierte Programme sind.
Die Klasse beschreibt die Struktur einer (eventuell unendlichen) Menge von Objekten. Jedes dieser Objekte wird Exemplar (instance) dieser Klasse genannt. Jedes Objekt ist ein Exemplar irgendeiner Klasse.
Es ist wichtig, die Unterscheidung zwischen Objekten und Klassen im Kopf zu behalten: Objekte sind Laufzeitelemente, die während der Ausführung eines Systems erzeugt werden; Klassen sind rein statische Beschreibungen einer Menge möglicher Objekte – der Exemplare dieser Klasse. Der Klassenbegriff der UML bezeichnet eine Menge von Objekten, die über gleiche Eigenschaften, dasselbe Verhalten und dieselbe Bedeutung verfügen. Im Rahmen der UML wird eine Klasse als Typ interpretiert, dessen Ausprägungen Objekte heißen. Mit Ausprägung ist die Zugehörigkeit eines Objekts zu einer Klasse gemeint. Das Objekt stellt also eine Ausprägung der durch die Klasse formulierte Schablone dar.
In dem untenstehenden Bild wird zur Verdeutlichung der Beziehung zwischen einer Klasse und den aus ihr erzeugten Objekten die Metapher des Stempels herangezogen. Die Besonderheit des Stempels ist, dass nicht nur eine vollständige Kopie der Attribute und Operationen erzeugt wird, sondern dass die Attribute mit Attributwerten versehen werden können.



Der Stempel entspricht einer Klasse, die unterschiedliche Exemplare von Objekten erzeugt. Die Stempelabdrücke symbolisieren die Objekte. Das Problem an der Metapher besteht darin, dass die Abdrücke mehr Informationen besitzen können als der Stempel. Ein Attributwert z.B. für das Geburtsdatum ist in der Klasse Auszubildender noch nicht enthalten. Die gezeichneten Objekte haben jeweils einen solchen Attributwert.
Eine Klasse dient zur Erzeugung von Objekten. In der Umgangssprache wird der Klassenbegriff eher zur Gruppierung von Objekten genützt. In der objektorientierten Welt dient eine Klasse primär zur Erzeugung von Objekten (Metapher der Klasse als Objektfabrik).

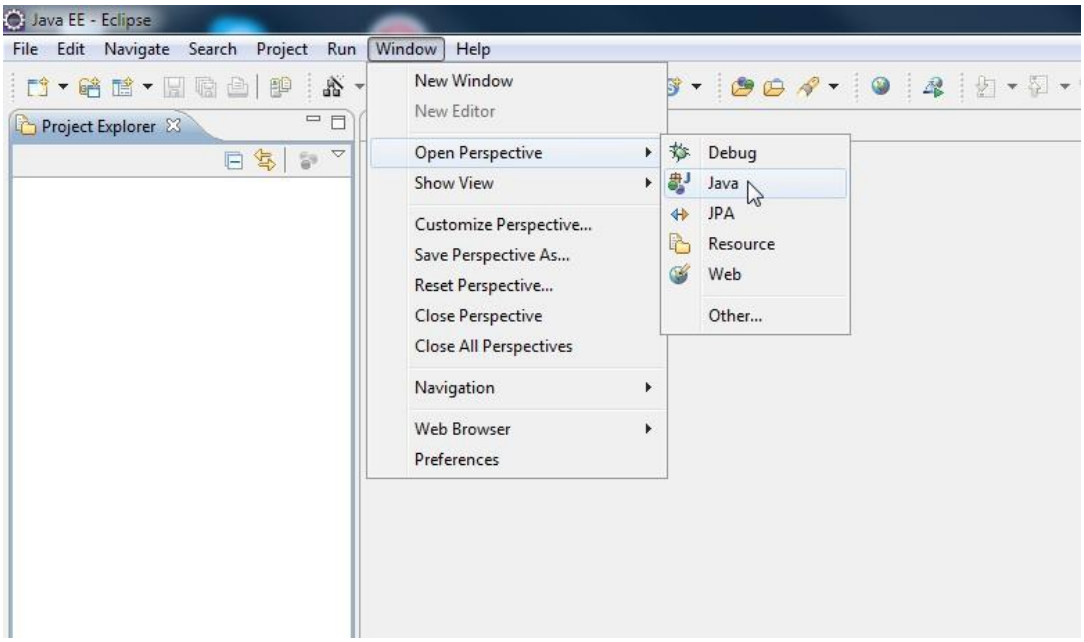
Attribute

Die Attribute beschreiben die Eigenschaften (Daten – Zustandsinformationen) eines Objekts. Alle Objekte einer Klasse besitzen dieselben Attribute, jedoch unterschiedliche Attributwerte. Das bedeutet, dass jedes Objekt Speicherplatz für alle seine Attribute erhalten muss.

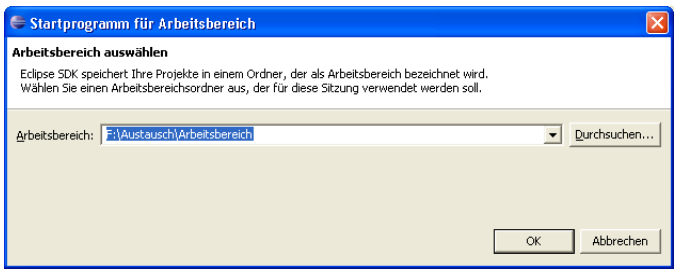
5 Realisierung des Einführungsbeispiels in der Arbeitsumgebung Eclipse (Eclipse 3.7 – Indigo)

5.1 Einstellung der Arbeitsumgebung

In Eclipse werden die verwalteten Dateien eines Projektes im *Workspace* zusammengefasst. Hiervon ist die Workbench zu unterscheiden, welche letztlich die Arbeitsoberfläche darstellt. Das konkrete Aussehen der *Workbench* wird über die jeweils gewählte *Perspektive* bestimmt. In einer *Perspektive* sind Ansichten mit speziellen Menüs und Werkzeugleisten sowie Editoren für bestimmte Anwendungsbereiche, wie zum Beispiel die Java – Programmierung, zusammengefasst.
Dem Einsteiger sei empfohlen, sich nicht der vielfältigen Möglichkeiten zur Gestaltung der *Workbench* zu widmen, sondern sich auf die Java – Perspektive zu konzentrieren;
Über das Menü Fenster Perpektive öffnen kann die Workbench vorbereitet werden.
Starten Sie Eclipse und richten Sie in der Workbench die Perspektive Java ein.



Der Workspace, in dem alle verwalteten Dateien eines Projektes zusammengefasst werden, wird wie folgt eingestellt:
Beispiel: Beim Starten von Eclipse soll z.B. als Standardverzeichnis für die Eclipse-Projekte das Verzeichnis F:\Austausch\Arbeitsbereich verwendet werden.
Dazu wird nach dem ersten Start von Eclipse das Menü *Datei Arbeitsbereich wechseln* aufgerufen:



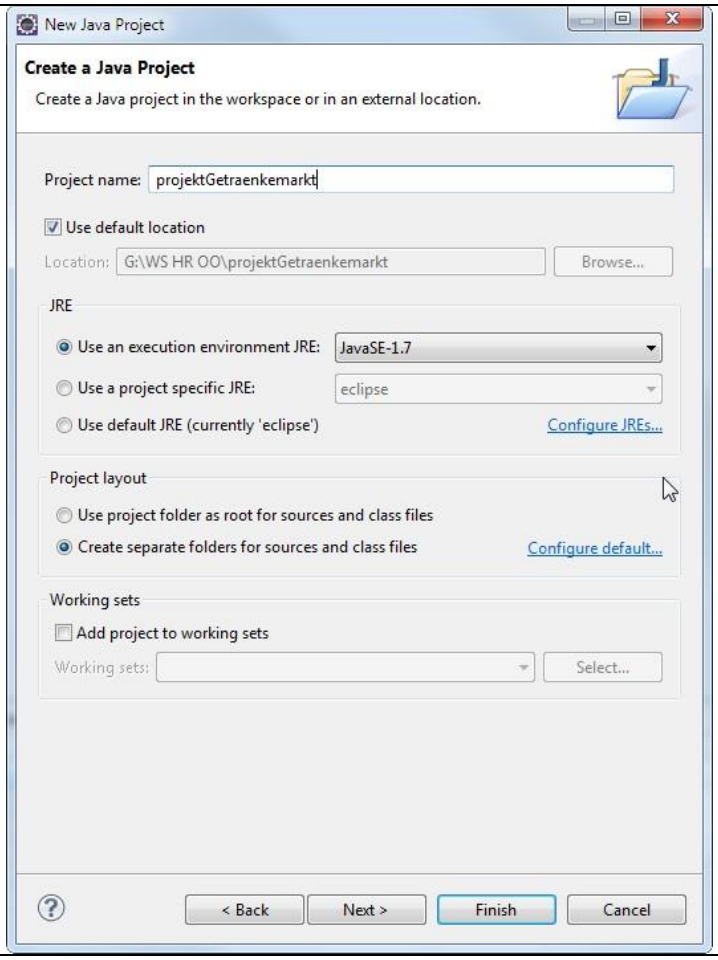
In dieses Verzeichnis wird dann von Eclipse automatisch der Ordner *.metadata* aus dem Verzeichnis *Programe\Eclipse\EclipseWorkbench\eclipse\workspace* kopiert. Im Ordner *.metadata* mit seinen Unterverzeichnissen werden alle Einstellungen zur Eclipse Arbeitsumgebung (Editoren, Perspektiven usw.) verwaltet.
Die vorgenommene Einstellung bleibt erhalten bis sie wieder gewechselt wird.

5.2 Fortführung des Beispiels

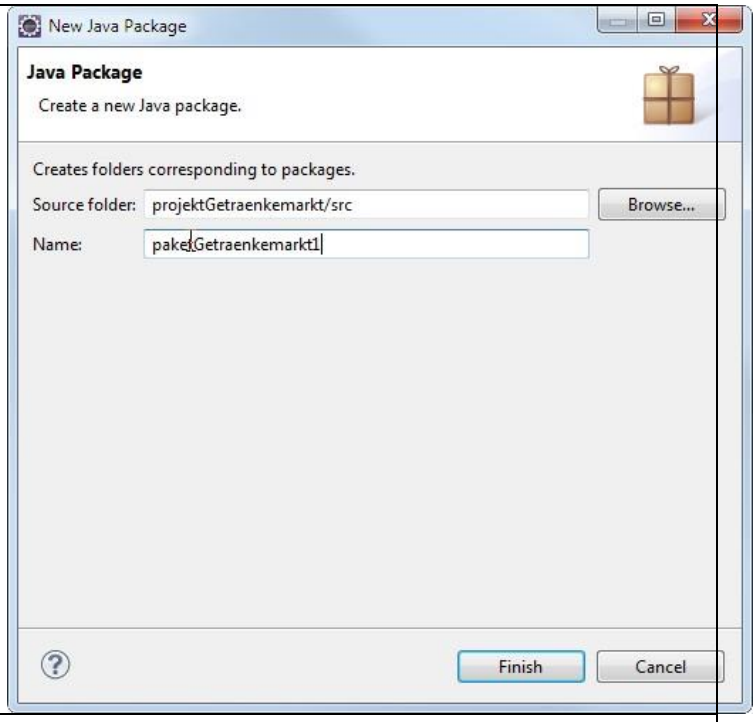
Das Software-Entwicklerteam der GeLa GmbH erhält den Auftrag, die Klasse *Getraenk* mit Hilfe des Software-Tools Eclipse zu modellieren.
Zunächst wird ein neues Projekt angelegt. Danach wird mit dem UML-Editor für die Klasse *Getraenk* das Klassendiagramm erstellt, dabei wird automatisch der Java-Quellcode für die Klasse generiert.
Anschließend wird die *Startklasse* mit der Methode *main()* eingerichtet. In der *Startklasse* wird festgelegt, wie die Klasse *Getraenk* genutzt wird. Wenn dies alles erledigt ist, wird die *Startklasse* ausgeführt und getestet.

5.2.1 Anlegen des Projekts

(1) Neues Projekt anlegen
Zuerst wird mit dem Menübefehl ***File → Neu → Javaproject*** ein neues Projekt als Java-Projekt angelegt. Wenn der Arbeitsbereich bereits eingestellt wurde, kann der Standardwert übernommen werden.
Hinweis:
Projekte legen wird mit dem Präfix ***projekt****Name* angelegt, z.B. *projektGetraenkemarkt*

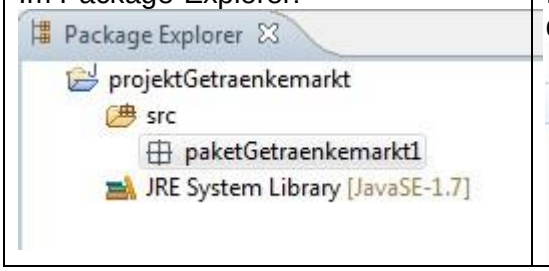


(2) **Paket anlegen**
Das Anlegen eines Pakets ist zwar optional, da aber ein Paketname im Verlauf des weiteren Projekts immer wieder gefordert wird, empfiehlt es sich, mit dem Menübefehl
Datei → New → Package
das Paket mit Namen *paketGetraenkemarkt1* anzulegen.

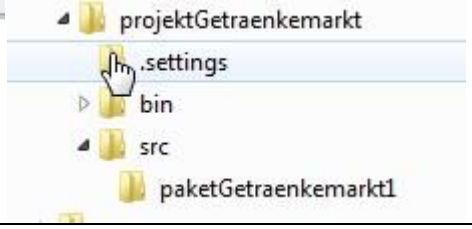


(3) Ergebnis des angelegten Projekts
Die Struktur des angelegten Projekts wird in Eclipse im *Package-Explorer* dargestellt und ist auf dem Datenträger über jeden beliebigen Datei-Explorer einsehbar

Im Package-Explorer:



In der Verzeichnisstruktur auf dem Datenträger (für das Projekt entsteht ein neuer Ordner):



5.2.2 Modellieren der Klasse Getraenk

Bei der automatischen Generierung von Quelltext entstehen Kommentare. Es gibt in Java drei Arten von Kommentare:

- `//…Der Rest der Zeile ist Kommentar`
- ```
  /*…alles zwischen diesen
  Zeichen ist Kommentar*/
  ```
- ```
  /**…hier handelt es sich um einen Kommentar, der durch spezielle
  Programme
  ausgewertet werden kann (Javadoc-Kommentare)
  Mit solchen Kommentaren werden z.B. die Klassendiagramme erzeugt, weshalb
  man sie nicht einfach löschen kann!*/
  ```

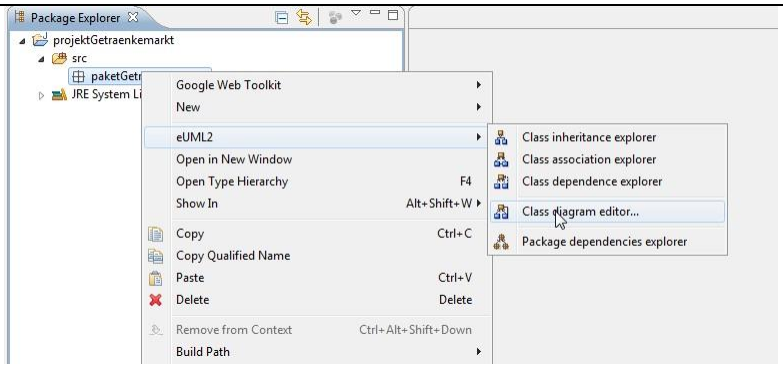
In der weiteren Darstellung von Quellcode werden die Kommentare in dieser Handreichung aus Gründen der Übersicht ausgeblendet.

In Eclipse kann für jede Problemstellung innerhalb eines Projekts ein der jeweiligen

Anforderung angepasster Editor aktiviert werden. Um eine neue Klasse mit UML modellieren zu können, wird daher zunächst der UML- Editor aktiviert (geöffnet):

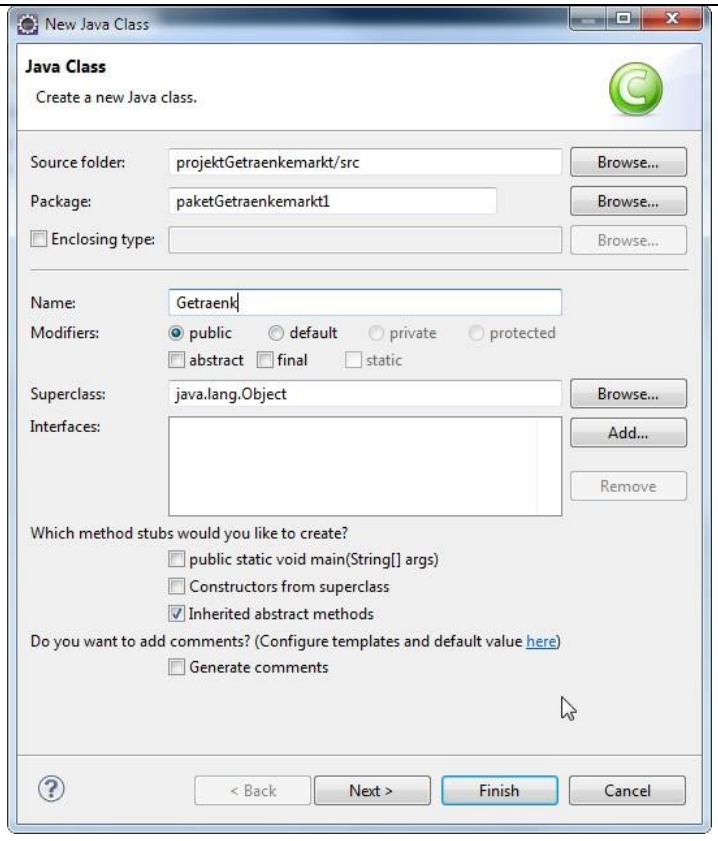
(1) UML-Editor aufrufen

Im Package-Explorer das Paket „paketGetraenkemarkt1“ markieren, dann das Kontextmenü (mit rechter Maustaste auf „paketGetraenkmarkt1“ klicken). Es öffnen sich nebenstehende Fenster. Bitte wie dargestellt auswählen.

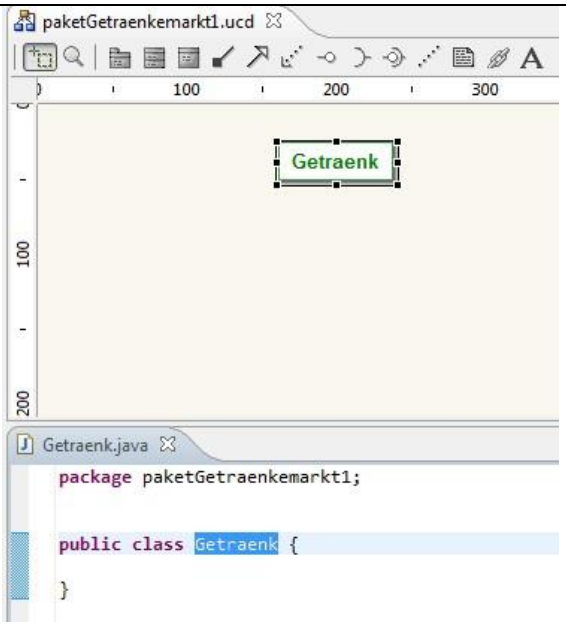
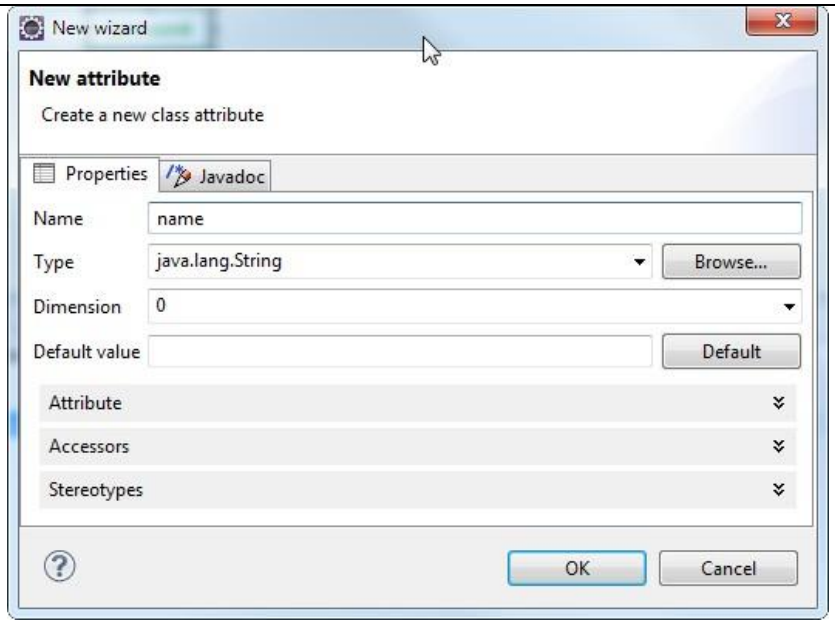


(2) Klasse anlegen

Durch Ziehen des Klassensymbols aus der Symbolleiste in den Arbeitsbereich des Editors wird das Erfassungsfenster für eine neue Klasse geöffnet. Man kann nun den Namen der Klasse angeben. Die übrigen Einstellungen sind zu übernehmen, sie werden später erklärt.



Die Klasse Getraenk erscheint nun auch im Ordner des Pakets

Im Klassendiagramm-Editor findet sich nun ein Klassendiagramm der Klasse Getraenk, das nun weiter bearbeitet wird. Sollten die Darstellung nicht wie rechts abgebildet dem Standard-UML-Design entsprechen , so stellten Sie unter der Menüfolge *Window → Preferences → Soyatec* als Diagram Presentation Style bitte um auf Basic (UML Standard)	
Im Quellcode-Editor kann die Entstehung des Java-Codes der erstellten Klasse verfolgt werden. Falls dieser Editor noch nicht angezeigt wird, kann er mit einem Doppelklick auf das Klassendiagramm erzeugt werden.	
Die Attribute kann man entweder im Quellcode oder über das Klassendiagramm einfügen. Hier wird gezeigt, wie man mit der Entwicklung des Klassendiagramms auch den Quellcode entstehen lassen kann. Mit Klick der rechten Maustaste in das Klassendiagramm kann man über das Kontextmenü *new Property* auswählen. Danach erscheint das nebenstehende Dialogfenster: Dabei sind die folunten stehendne Bemerkungen zu beachten:	

Attribute sollen nur innerhalb der Klasse gesehen werden können. Sie können von außen nur über die vom Programmierer vorgesehenen Methoden (Schnittstellen) bearbeitet werden. Deshalb werden sie mit „private“ deklariert.
Die Methoden, mit denen man die Variablen bearbeitet, müssen dagegen von außen erreichbar sein, weshalb sie mit „public“ deklariert werden.
Das Attribut „name“ ist ein Objekt der Klasse „String“. Diese Klasse befindet sich in der Bibliothek „java.lang.*“. Diese enthält den „Basisbefehlssatz“ von Java. Dieser Befehlssatz wird beim Start der Entwicklungsumgebung geladen und steht immer zur Verfügung.

Exkurs: Kapselung
Das Prinzip, wonach man die Attribute eines Objektes niemals von außen bearbeiten können darf, nennt man Kapselung. Nur die Klasse ist für ihre Objekte und deren Attributswerte zuständig und verantwortlich. Manipulationen dieser Werte sind daher nur mit Methoden möglich, die der Programmierer der Klasse zur Verfügung stellt.

Besondere Methoden sind hierbei die set- und get-Methoden.

Sie werden bei der eben dargestellten Art, Attribute zu benennen, automatisch erzeugt. Diese Methoden übergeben den Objektattributen konkrete Werte beziehungsweise lesen diese Werte aus.

Nach der Bestätigung des Fensters zur Definition eines Attributes ändert sich das Klassendiagramm wie rechts abgebildet. Die „Setter und Getter“, wie man die set- und get-Methoden auch nennt, müssen nicht im Diagramm abgebildet werden, da diese beiden Methoden standardmäßig für jedes Attribut vorhanden sind und ansonsten die Klassendiagramme zu überfrachtet dargestellt würden. Ein Blick in den Package-Explorer oder auch in den Quellcode belegt allerdings, dass die Settter und Getter ebenfalls erzeugt wurden.

Das Minuszeichen bedeutet, dass die Sichtbarkeit auf „private“, das Pluszeichen, dass die Sichtbarkeit auf „public“ gesetzt wurde. Hinter dem Doppelpunkt des Attributs steht der Datentyp, hinter dem Doppelpunkt des Namens der get-Methode steht der Typ des Rückgabewertes. Die Botschaft, die ein Objekt der Klasse Getränk mit dem Namen object1 erhalten kann, lautet z.B.: `objekt1.getName();`. Das bedeutet, dass das Objekt mit dem Namen „objekt1“ den Namen übergibt, dieser ist vom Typ String.
Die set-Methode hat bei der Deklaration in der Klammer stets den Wert, den sie dem Objekt übergibt, zusammen mit der Beschreibung des Typs für diesen Wert. Das „in“ in der Klammer steht für Input und meint, dass dem Objekt etwas übergeben wird.
Die Botschaft `objekt1.setName("Cola light");` besagt, dass das Objekt der Klasse Getraenk mit dem Namen „object1“ den Namen „Cola light“ erhält.
Die Klasse erhält im Quellcode die Zuordnung zu dem Paket (=package) welchem sie zugeordnet ist. Pakete dienen der Strukturierung von Programmen. Logisch zusammenhängende Klassen werden in Paketen zusammen gefasst.

set- und get-Methoden
Beim Anlegen des Attributs werden sofort zwei Methoden generiert:

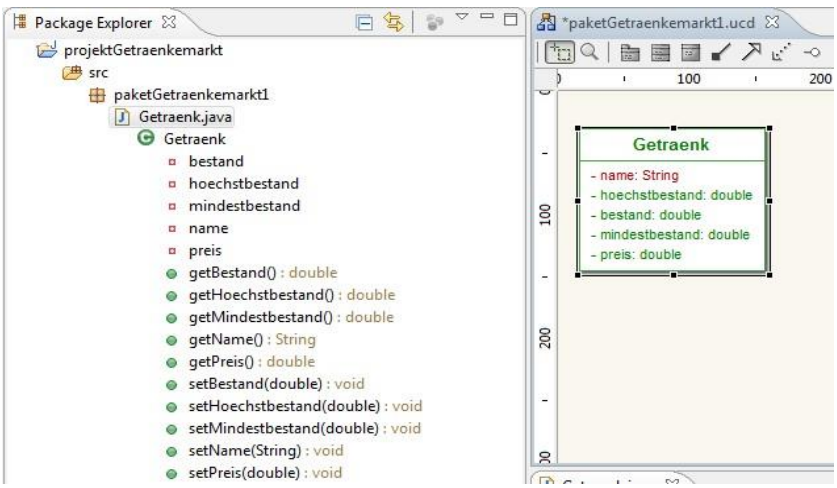
- **die get-Methode** mit ihr kann man den Attributswert des aktuellen Objekts auslesen.
- **die set-Methode** mit ihr kann man den Attributswert des aktuellen Objekts setzen. Die set-Methode hat in der Klammer den Wert, der geschrieben wird (Übergabewert, der von außen kommt). „this“ steht für das aktuelle Objekt. (Das aktuelle Objekt erhält für das Attribut name den Inhalt des Übergabewertes name vom Typ String) .
-

*Getraenk.java

```
package paketGetraenkemarkt;

public class Getraenk {

    /**

    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

Aufgabe: Fügen Sie die anderen Attribute inklusive der zugehörigen Set- und Get-Methoden in das Klassendiagramm ein.

Die Definition der anderen Attribute führt zum nebenstehenden Klassendiagamm dargestellt mit Package-Explorer (für die Setter und Getter).

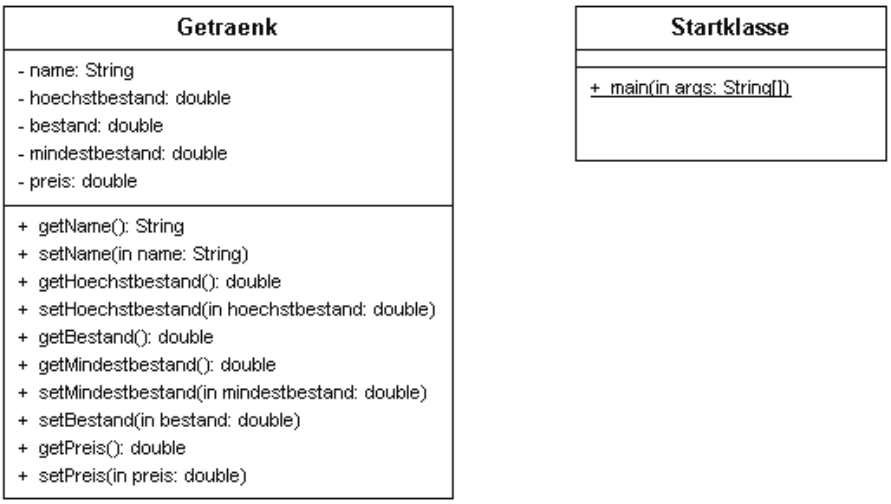


5.2.3 Die Startklasse

Die Klasse „Getraenk“ wird als „Fachklasse“ bezeichnet, weil in ihr das eigentliche Fachproblem gelöst wird. Die Fachklassen sollten immer im Zusammenwirken mit den zukünftigen Nutzern entwickelt werden, um deren Fachkompetenz mit einzubeziehen. Für diese Nutzer sind die Diagramme, die im Zuge der Modellierung entstehen, Veranschaulichungen des Problems. Die Kommunikation zwischen Entwicklern mit IT-Kompetenz und Fachmann mit Kompetenz für das eigentliche Problem wird erleichtert.
Wenn die Fachklassen (in unserer Einführungsphase liegt nur eine vor) entwickelt wurden, müssen diese auch benutzt werden. Um dies zu ermöglichen benötigt man eine Steuerung, die den Fachklassen Botschaften übersendet und diese selbst wieder zu Aktionen veranlasst. Solche Klassen nennt man dynamische Klassen. Sie veranlassen die Erzeugung von Objekten. In einer Startklasse stehen i.d.R. mehrere Anweisungen. Die Abarbeitung dieser Anweisungen nennt man Programmablauf.
Es macht keinen Sinn, der Startklasse Attribute zuzuweisen oder gar Objekte von ihr erzeugen.

Die Fachklasse „Getraenk“ legt fest, welche Attribute und Methoden die aus ihr erzeugten Objekte haben sollen.

Das erweiterte Klassenmodell besteht somit aus den Klassen:



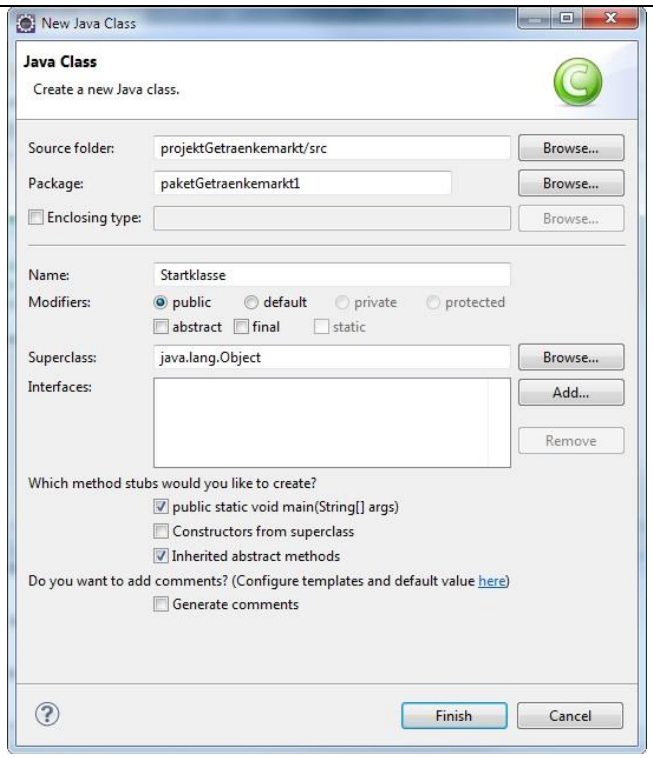
Die Klasse „Startklasse“ wird nun im Folgenden entwickelt. Dabei sind folgende Aufgaben zu lösen:

- Erweitern des Pakets „paketGetraenkemarkt“ um die Klasse Startklasse und erzeugen des Objekts „getraenk1“.
- Setzen der Attributswerte
- Ausgabe der Daten des Getränkeobjekts über die Konsole nach folgendem Muster:

```
Getränkenamen: Selters Preis: 0.25 Bestand: 2000.0 Höchstbestand: 4000.0
Mindestbestand: 1000.0
```

Entwicklung der Startklasse

(1) Die **Startklasse** ist inklusive der Methode **main()** zu erstellen. Zunächst ist das Paket zu markieren, dem die Klasse angehören soll. Danach wird im Menü File → New → Class das nebenstehende Fenster aufgerufen. Im Dialogfenster ist der Klassenname „Startklasse“ einzutragen. Ebenfalls ist ein Häkchen bei Methoden-Stubs zu setzen: Die main-Methode ist zu generieren.



Exkurs zur Methode main():
Die Startklasse hat eine Methode mit dem Namen „main“. Diese Methode ist dafür verantwortlich, dass das Programm abläuft. Jedes selbstständig laufende Java-Programm benötigt eine main-Methode.
Der Kopf der Methode „main“ wird an dieser Stelle noch nicht vollständig erklärt, da hierzu noch einige Grundlagen gelegt werden müssen. Vorerst genügt es zu wissen, dass mit dieser Zeile die Java-Maschine im Rechner geladen wird. Diese Java-Maschine ist eine gedachte (virtuelle), also nicht wirklich existierende Maschine. Dieser virtuelle Rechner baut auf dem Betriebssystem des eigentlichen Rechners auf und ergänzt bzw. beschneidet dieses so, dass ein genau definierter Rechner (Java-Virtual-Maschine oder kurz JVM) zur Verfügung steht. Da es für alle gängigen Betriebssysteme und Hardware solche virtuelle Maschine gibt, die den Rechner zu einer Java-Maschine werden läßt, verstehen nahezu alle Rechner die Programmiersprache Java. Man sagt deshalb auch Java ist plattformunabhängig. Die daraus resultierende Unabhängigkeit von Hard- und Software ist mit ein Grund für die enorme Verbreitung dieser Sprache. Mit der Zeile `public static void main(String[] args)` startet die virtuelle Maschine und führt die Anweisungen, die zwischen den geschweiften Klammern stehen, aus.

Es entsteht der nebenstehende Quellcode. Auch die Startklasse steht im Paket „paketGetraenkemarkt1“.

```
package paketGetraenkemarkt;

public class Startklasse {

    public static void main(String[] args) {

    }

}
```

(2) In der Methode main() werden wir nun die Anweisungen zum Erzeugen eines Objektes getraenk1 der Klasse Getraenk unterbringen:

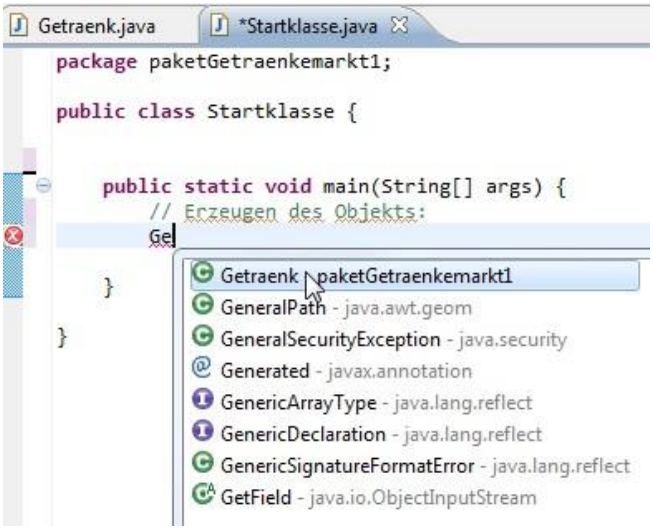
In der nun zu schreibenden Zeile

```
Getraenk getraenk1;
```

wird zunächst ein Objekt der Klasse Getraenk angekündigt (deklariert).

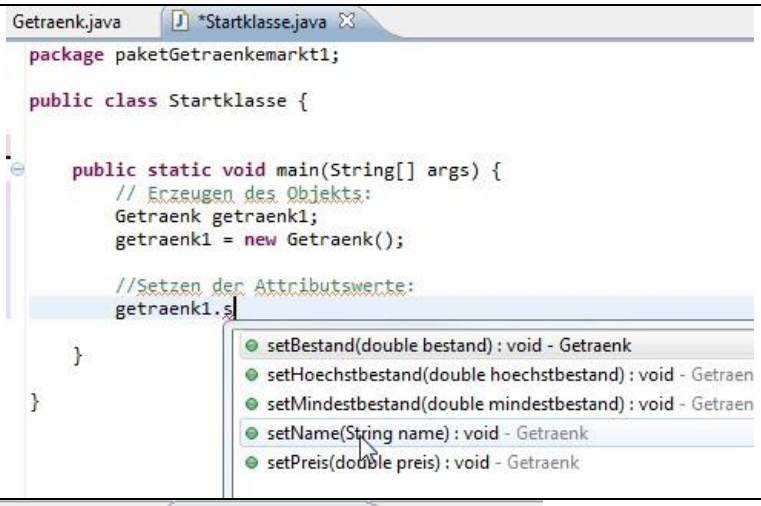
Bemerkung:

Um Quellcodezeilen zu schreiben, sollte (ab jetzt) die **Code-Ergänzungsfunktion** konsequent benutzt werden. Die Tastenkombination Strg+Leertaste bewirkt, dann an der aufrufenden Stelle kontextbezogen die Anweisungen angeboten werden, welche an der jeweiligen Stelle Sinn machen. Dadurch werden viele Fehler vermieden und der Unterrichtsfluss erheblich verbessert. Funktioniert die Code-Ergänzungsfunktion nicht, so liegt in aller Regel ein Syntaxfehler im bisherigen Quellcode vor.

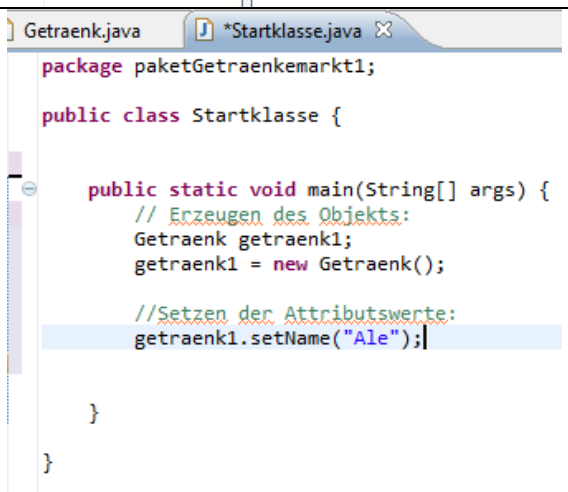


In der nächsten Zeile wird das Objekt erzeugt. Wenn man in dieser Zeile „ge“ gefolgt von Strg+Leer tippt, so erscheint bereits „getraenk1 in der Auswahl. Nun ist das Objekt „getraenk1“ bis auf die Festlegung der Attributswerte vorhanden. [Im Quellcode erscheint eine Methode Getraenk() – ein sogenannter „Kontstruktor“ – auf diesen Sachverhalt werden wir später eingehen.]	``` package paketGetraenkemarkt1; public class Startklasse { public static void main(String[] args) { // Erzeugen des Objekts: Getraenk getraenk1; getraenk1 = new Getraenk(); } } ```

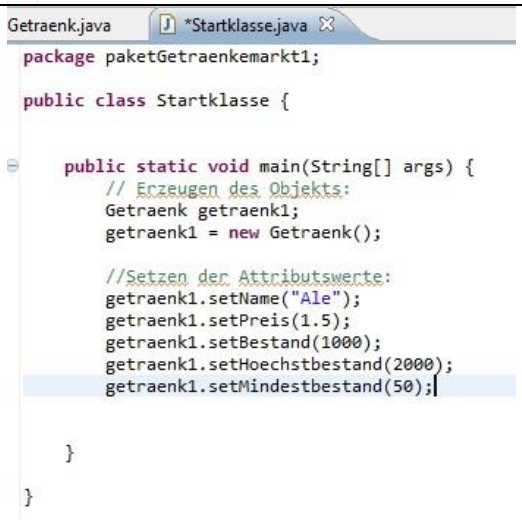
Nun werden die in der Klasse definierten Attribute für das aus dieser Klasse erzeugte Objekt mit Werten versehen.
Zum Setzen dieser Attributswerte benützen wir auch hier wiederum die Code-Ergänzung und setzen deshalb nach „getraenk1“ einen Punkt. Wir wählen die gewünschte Methode „set-Name()“ aus den angebotenen Möglichkeiten aus.



Wir vervollständigen unseren Quellcode wie nebenstehend abgebildet.
Der Übergabewert „Ale“ steht in Anführungszeichen, weil dies bei Strings so vorgesehen ist.



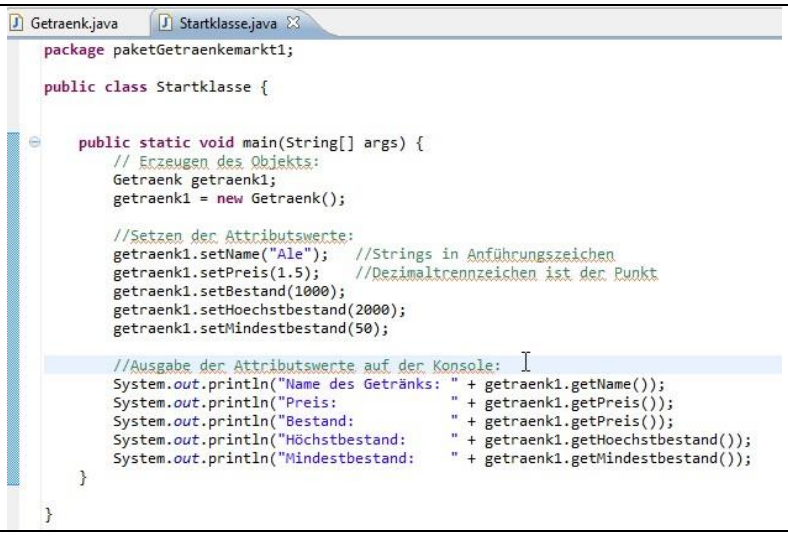
Anschließend werden auch die restlichen Attributswerte gesetzt.



Bildschirmausgabe:
Im Konsolefenster wird mit `System.out.println(String)` der in der Klammer stehende String ausgegeben.
Dieser Befehl ist wie folgt zu verstehen: Die Klasse System hat die Methode out, der ein PrintStream (`println()`) zugewiesen wird. Die Ausgabe erfolgt im Konsolefenster. Wird statt „println“ nur „print“ ausgegeben, so erfolgt nach der Stringausgabe keine Zeilenumschaltung.

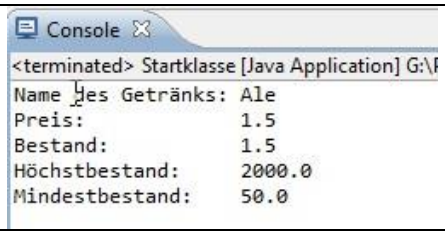
Die Zeile »`System.out.println("Name:" + p1.getName());`« wird dabei wie folgt erzeugt:
Man tippt „syso“ ein, danach STRG + Leertaste, danach erhält man: »`System.out.println();`«
Danach tippt man »`"Name:"+ p1.g`«.
Man erhält die get-Methoden zur Auswahl und sucht sich *getName()* aus.
Bemerkung: Das „+“ Zeichen ist in diesem Kontext eine Stringoperation, die bedeutet, dass Strings aneinander gefügt werden. Wird in dieser Syntax ein anderer Datentyp benutzt, wird er automatisch in einen String umgewandelt (=casting).

Quellcode der Startklasse mit Bildschirmausgabe:



Nun kann das Programm gestartet werden. Dazu wählen Sie Menü Run→ Run As → Java Applikation. Nun läuft die main-Methode ab.

Die Bildschirmausgabe der Konsole ist nebenstehend abgebildet.



Falls Sie die Konsole nicht sehen können, müssen Sie diese erst aktivieren. Der Befehl lautet: Window→Show View→Console

5.2.4 Konstruktoren

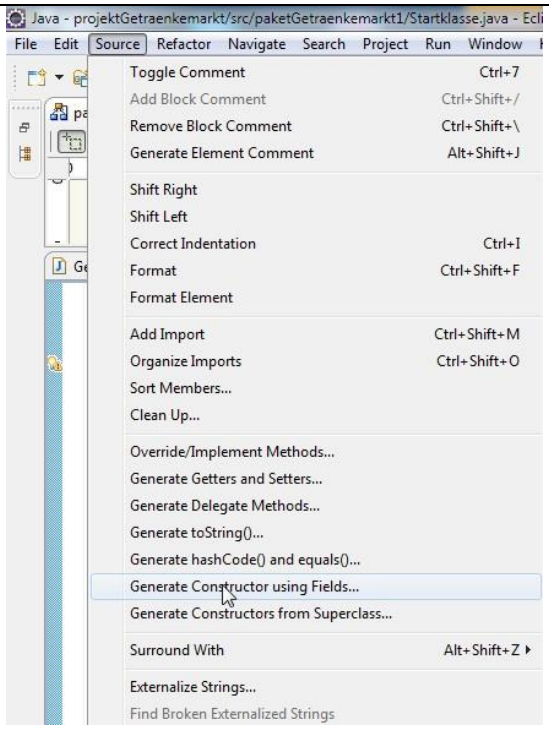
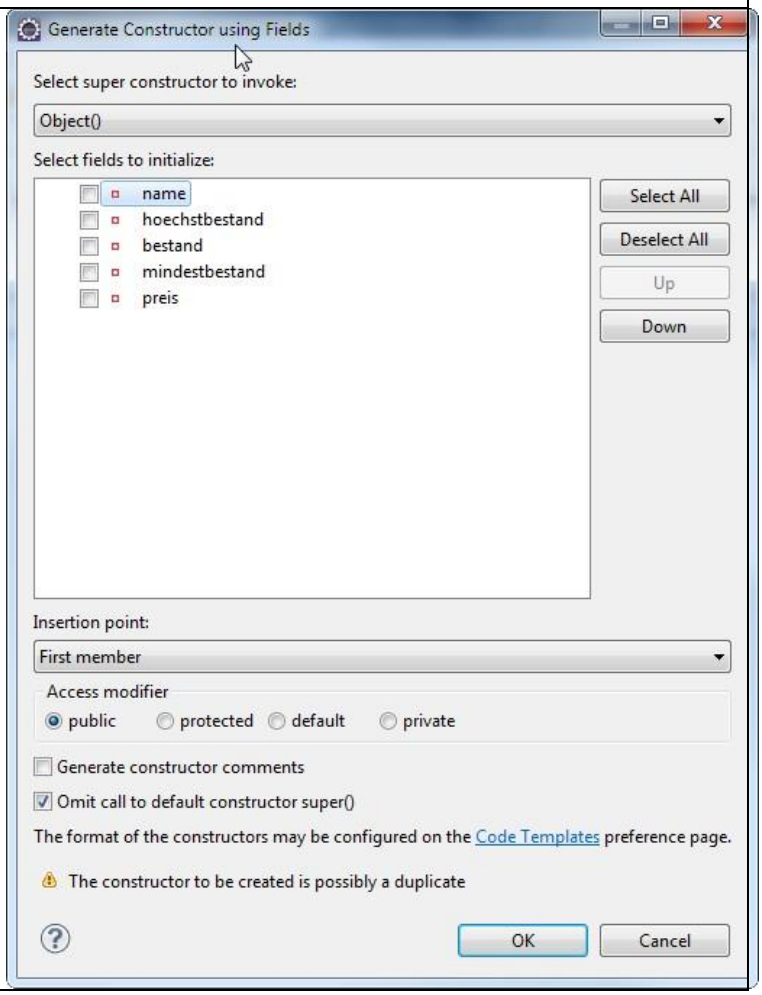
In der Startklasse wurde ein neues Objekt der Klasse Getraenk mit folgender Anweisung erzeugt: `getraenk1 = new Getraenk();`
Der Ausdruck `Getraenk()` ist eine besondere Methode, die den Namen der korrespondierenden Klasse trägt und im Unterschied zu sonstigen Methoden groß geschrieben ist. Diese Methode erzeugt ein Objekt der Klasse und wird Konstruktor genannt. Ist der Konstruktor nicht definiert, so hat die Klasse dennoch einen Konstruktor, den man Standardkonstruktor nennt. In der folgenden Tabelle sind mögliche Konstruktoren der Klasse Getraenk aufgeführt. Sie unterscheiden sich durch den Aufruf. Die Reihenfolge der Übergabeparameter in der Klammer ist für den Aufruf entscheidend.

Erklärung	Quellcode	Erzeugung
Ohne Übergabeparameter (=Standardkonstruktor)	`public Getraenk() { }`	`getraenk1=new Getraenk();`
Parameter mit Wert für Namen	`public Getraenk(String name) { this.name = name; }`	`getraenk1=new Getraenk(„Selters“)`
Parameter mit Werten für Name und Preis	`public Getraenk(String name, double preis) { this.name = name; this.preis = preis; }`	`getraenk1= new Getraenk(„Selters“, 0.25)`
usw.		

Das Wort „this“ stellt in diesem Zusammenhang einen Verweis auf das aktuelle (aufrufende) Objekt dar. Dies ist das gerade aktive Objekt. In der Klammer stehen die Parameter, sie sind Variable, die nach dem Aufruf des Konstruktors aus dem Hauptspeicher gelöscht werden. Sie werden in den Objekten zu Attributswerten.

Es gilt zu beachten: Existiert in einer Klasse ein Konstruktor mit Parameterübergabe, dann wird der Standardkonstruktor überschrieben. Der Aufruf `getraenk1=new Getraenk();` ist dann nicht mehr möglich. Will man nach wie vor diesen Konstuktor haben, muss er im Quellcode definiert sein.

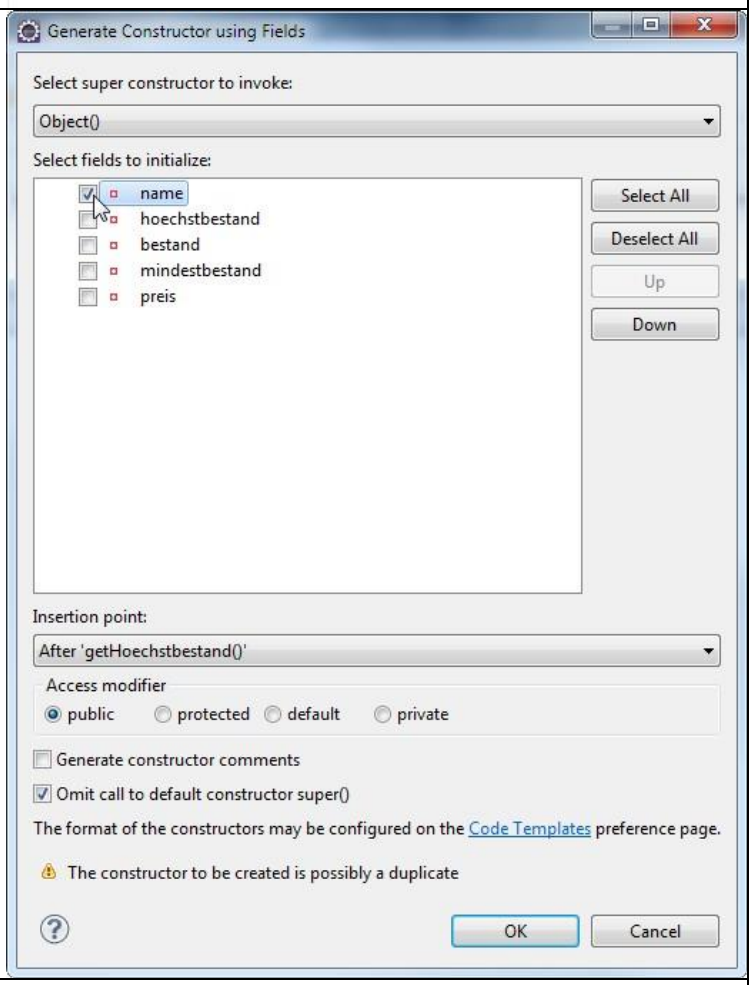
Für die oben beschriebene Startklasse kann man die Konstruktoren wie folgt generieren:

Im Menü Source nebenstehende Auswahl treffen	
Einfügen des Standardkonstruktors (=Konstruktor ohne Parameter): kein Kontrollkästchen aktivieren. [Um den Konstruktor einer möglichen Oberklasse nicht aufzurufen, muss unten das Kontrollkästchen „Omit call to default constructor super()“, aktiviert sein.]	

Diese Generierung des Standardkonstruktors führt zu nebenstehendem Quellcode.

```
public class Getraenk {


    public Getraenk() {
    }
```

Um einen Konstruktor mit einem Parameter für das Attribut „name“ zu erhalten, muss wie rechts dargestellt gehandelt werden.



Ferner fügen wir auf die gleiche Weise noch einen weiteren Konstruktor für alle Attribute ein.

Hier der um Konstruktoren ergänzte Quellcode der Klasse „Getraenk“:

```
package paketGetraenkemarkt1;

public class Getraenk {

	public Getraenk() {
	}

	private String name;

	public String getName() {
		return name;
	}


	public Getraenk(String name, double hoechstbestand, double bestand,
			double mindestbestand, double preis) {
		this.name = name;
		this.hoechstbestand = hoechstbestand;
		this.bestand = bestand;
		this.mindestbestand = mindestbestand;
		this.preis = preis;
	}

	public void setName(String name) {
		this.name = name;
	}

	/**
	 * @uml.property  name="hoechstbestand"
	 */
	private double hoechstbestand;

	/**
	 * Getter of the property <tt>hoechstbestand</tt>
	 * @return  Returns the hoechstbestand.
	 * @uml.property  name="hoechstbestand"
	 */
	public double getHoechstbestand() {
		return hoechstbestand;
	}

	public Getraenk(String name) {
		this.name = name;
	}

	/**
	 * Setter of the property <tt>hoechstbestand</tt>
	 * @param hoechstbestand  The hoechstbestand to set.
	 * @uml.property  name="hoechstbestand"
	 */
	public void setHoechstbestand(double hoechstbestand) {
		this.hoechstbestand = hoechstbestand;
	}

	/**
	 * @uml.property  name="bestand"
	 */
	private double bestand;

	/**
	 * Getter of the property <tt>bestand</tt>
	 * @return  Returns the bestand.
	 * @uml.property  name="bestand"
	 */
	public double getBestand() {
		return bestand;
	}

	/**
	 * Setter of the property <tt>bestand</tt>
	 * @param bestand  The bestand to set.
	 * @uml.property  name="bestand"
	 */
	public void setBestand(double bestand) {
		this.bestand = bestand;
	}

	/**
	 * @uml.property  name="mindestbestand"
```

```
	 */
	private double mindestbestand;

	/**
	 * Getter of the property <tt>mindestbestand</tt>
	 * @return  Returns the mindestbestand.
	 * @uml.property  name="mindestbestand"
	 */
	public double getMindestbestand() {
		return mindestbestand;
	}

	/**
	 * Setter of the property <tt>mindestbestand</tt>
	 * @param mindestbestand  The mindestbestand to set.
	 * @uml.property  name="mindestbestand"
	 */
	public void setMindestbestand(double mindestbestand) {
		this.mindestbestand = mindestbestand;
	}

	/**
	 * @uml.property  name="preis"
	 */
	private double preis;

	/**
	 * Getter of the property <tt>preis</tt>
	 * @return  Returns the preis.
	 * @uml.property  name="preis"
	 */
	public double getPreis() {
		return preis;
	}

	/**
	 * Setter of the property <tt>preis</tt>
	 * @param preis  The preis to set.
	 * @uml.property  name="preis"
	 */
	public void setPreis(double preis) {
		this.preis = preis;
	}
}
```

Im nächsten Schritt erzeugen wir nun in der Startklasse mittels unserer beiden neu erzeugten Konstruktoren je ein neues Getränke-Objekt und geben die gesetzten Attributswerte jeweils auf der Konsole aus:

```
package paketGetraenkemarkt1;
public class Startklasse {


        public static void main(String[] args) {


                // Erzeugen des ersten Objekts
                Getraenk getraenk1;
                getraenk1 = new Getraenk();

                //Setzen der Attributswerte via Set-Methoden::
                getraenk1.setName("Ale");   //Strings in Anführungszeichen
                getraenk1.setPreis(1.5);    //Dezimaltrennzeichen ist der Punkt
                getraenk1.setBestand(1000);
                getraenk1.setHoechstbestand(2000);
                getraenk1.setMindestbestand(50);

                //Ausgabe der Attributswerte des ersten Getränke-Objekts auf der Konsole:
                System.out.println("Name des Getränks: " + getraenk1.getName());
                System.out.println("Preis:             " + getraenk1.getPreis());
                System.out.println("Bestand:           " + getraenk1.getPreis());
                System.out.println("Höchstbestand:     " + getraenk1.getHoechstbestand());
                System.out.println("Mindestbestand:    " + getraenk1.getMindestbestand());

                //Erzeugen eines zweiten Objekts unter Benützung des ersten neuen Konstruktors:
                Getraenk getraenk2;
                getraenk2 = new Getraenk("Dorfbrunnen Mineralwasser");

                //Setzen der restlichen Attributswerte via Set-Methoden:
                getraenk2.setPreis(0.5);
                getraenk2.setBestand(3000);
                getraenk2.setHoechstbestand(5000);
                getraenk2.setMindestbestand(1000);

                //Ausgabe der Attributswerte des zweiten Getränke-Objekts auf der Konsole:
                System.out.println("Name des Getränks: " + getraenk2.getName());
                System.out.println("Preis:             " + getraenk2.getPreis());
                System.out.println("Bestand:           " + getraenk2.getPreis());
                System.out.println("Höchstbestand:     " + getraenk2.getHoechstbestand());
                System.out.println("Mindestbestand:    " + getraenk2.getMindestbestand());

                //Erzeugen eines dritten Objekts unter Benützung des zweiten
                //neuen Konstruktors:
                Getraenk getraenk3;
                getraenk3 = new Getraenk("Orangenlimonade", 2000, 1800, 700, 0.8);

                //Ausgabe der Attributswerte des dritten Getränke-Objekts auf der Konsole:
                System.out.println("Name des Getränks: " + getraenk3.getName());
                System.out.println("Preis:             " + getraenk3.getPreis());
                System.out.println("Bestand:           " + getraenk3.getPreis());
                System.out.println("Höchstbestand:     " + getraenk3.getHoechstbestand());
                System.out.println("Mindestbestand:    " + getraenk3.getMindestbestand());
    }
}
```

Das bisherige Klassenmodell in UML-Notation:

Die **Fachklasse** liefert lediglich die Vorlage (=Schablone) für die in der Startklasse initiierten Objekte. • Die Fachklasse ist eine **statische Klasse**	Die **Startklasse** ist eine **dynamische Klasse** Die Klasse *Startklasse* dient dazu • ein oder mehrere Objekte mit Hilfe der Fachklasse anzulegen • mit Hilfe der Methoden der Fachklasse - Attributswerte zu übernehmen - Attributswerte auszugeben

6. Methoden

Bisher wurden nur Methoden erwähnt, die man Getter und Setter nennt. Die set-Methoden haben lediglich die Aufgabe, einen Attributwert zu übergeben . Genauso soll get-Methoden nur die Aufgabe zukommen, einen Attributwert eines Objektes zu lesen und zur Verfügung zu stellen. Solche Methoden dürfen daher beispielsweise keine Rechenoperationen durchführen. Alle Methoden, die weder Setter noch Getter sind, dürfen konventionsgemäß nicht mit „set“ oder „get“ beginnen. Mit solchen Methoden können mannigfaltige Operationen durchgeführt werden.

6.1 Entwicklung eigener Methoden

Bevor die Methoden zur Realisierung des Einführungsbeispiels programmiert werden, ist es sinnvoll, sich den grundlegenden syntaktischen Aufbau von Methoden in Jave klarzumachen. Dieser soll am Beispiel einer einfachen Methode aufgezeigt werden.
Zu diesem Zweck soll unsere Klasse Getraenk um die Funktionalität der Bestandserhöhung erweitert werden. Diese Funktionalität soll mittels der Methode „bestanderhoehen(…)“ implementiert werden.
Methodennamen werden in Anlehnung an die mathematische Funktionenschreibweise gefolgt von zwei runden Klammen, innerhalb derer der Input (der sogenannte „Übergabewert“) der Methode steht.
Der Kopf der Methode beginnt mit „public“, weil die Methode von außerhalb der Klasse aufgerufen werden soll:

```
public void bestanderhoehen(double wert)
	{
		bestand = bestand + wert;
	}
```

Unsere Methode soll keinen Output liefern („keinen Wert zurückgeben“), daher benutzen wir für die Rückgabe den Datentyp „void“. Der Methodenrumpf steht in geschweiften Klammern.

Der grammatikalische Aufbau einer Methode in Java folgt somit immer genau dem gleichen syntaktischen Aufbau:

Die Syntax des Kopfes einer Methode in Java lautet:

```
[Sichtbarkeit] Rückgabetyp Methodenname(Parameterliste)
```

Der Rumpf ist in geschweiften Klammern eingeschlossen:

```
{
	Anweisungen…
	// gibt eine Methode einen Wert zurück, muss die return-Anweisung
	//benutzt werden
	return rückgabewert;
}
```

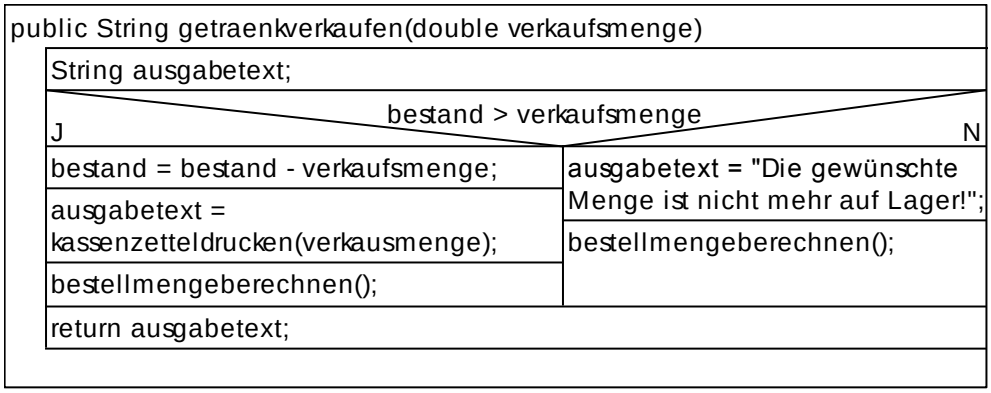
Möchten wir den Bestand unseres zweiten Getränkeobjekts beispielsweise um 200 Einheiten erhöhen, so könnten wir dies in der Startklasse mittels des folgenden Methodenaufrufes veranlassen:

```
getraenk2.bestanderhoehen(200);
```

6.2 Entwicklung der Methoden zur Realisierung des Einführungsbeispiels

Gemäß der Festlegungen im USE-CASE-Diagramm und in den Aktivitätsdiagrammen müssen nun die Aktivitäten „Getränk verkaufen“, „Kassenzettel drucken“ und „Bestellmenge berechnen“ realisiert werden. Es bietet sich an, diese Aktivitäten jeweils mittels einer eigenen Methode im System zu implementieren.

Zunächst soll die Methode „getraenkverkaufen(…)“ erarbeitet werden. Als erstes muss man sich darüber im Klaren sein, in welche Klasse die neue Methode zu integrieren ist. Da die Klasse Getraenk für ihre Objekte und damit auch für deren Attributswerte zuständig ist, gehört die Methode in die Klasse Getraenk.
Methoden werden oftmals in Struktogrammen veranschaulicht und für die Programmierung aufbereitet. Das Struktogramm unserer Methode sowie der zugehörige Quellcode könnten folgendermaßen aussehen:




```
public String getraenkverkaufen(double verkaufsmenge)
	{
		String ausgabetext;
		if (bestand > verkaufsmenge)
		{
			bestand = bestand - verkaufsmenge;
			ausgabetext = kassenzetteldrucken(verkaufsmenge);
			bestellmengeberechnen();
		}

		else

		{
			ausgabetext = "Die gewünschte Menge ist nicht mehr auf Lager!";
			bestellmengeberechnen();
		}

		return ausgabetext;

	}
```

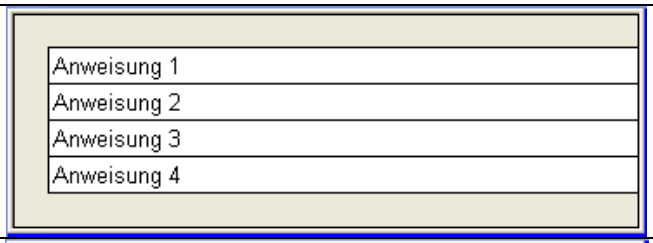
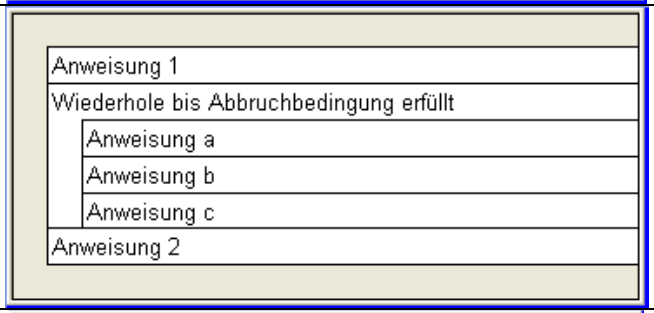
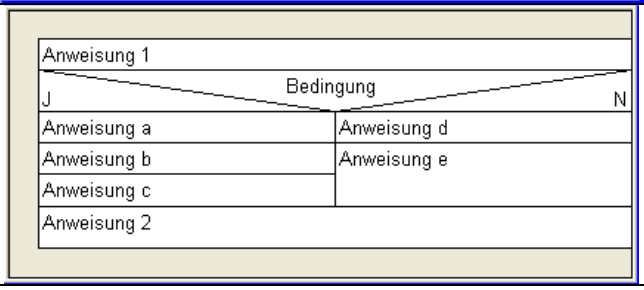
Der Kopf der Methode beginnt mit „public“, weil die Methode von außerhalb der Klasse aufgerufen werden soll. Die Methode gibt einen String-Wert (einen Text) zurück, deshalb muss als Rückgabetyp „String“ aufgeführt sein.
Im Anschluss folgt der Methodenname. Nach dem Methodennamen müssen die runden Klammern stehen, in ihnen steht der Übergabewert. In dieser Methode wird ein Double-Wert übergeben, dieser sogenannte Übergabeparameter wird in dieser Methode mit dem Variablennamen „verkaufsmenge“ bezeichnet.
Die im Methodenkopf angekündigte Rückgabe wird durch die Zeile `return ausgabetext;` realisiert, durch welche der Inhalt der String-Variablen **`ausgabetext`** zurückgegeben wird.

Um die in der Methode „getraenkverkaufen(…)“ beinhaltete Verzweigung verstehen zu können, werfen wir zunächst einen Blick auf Kontrollstrukturen in Java:

6.3 Kontrollstrukturen in Java

Um den in der Aktivität „Getränk verkaufen“ modellierten Verkaufsvorgang zu implementieren bedarf es einer Ablauflogik – in der Fachsprache oftmals „Kontrollstruktur“ genannt.

Leider wird das englische Verb „to control“ sehr oft falsch in die deutsche Sprache übersetzt. Es bedeutet zuerst „steuern“ und nicht „kontrollieren“. „Kontrollieren“ ist enger gefasst und kann als Untermenge von „steuern“ aufgefasst werden. So verhält es sich auch bei Kontrollstrukturen, die eigentlich Steuerungsstrukturen heißen müssten.
Wie der Name bereits sagt, soll der Programmablauf gesteuert werden. Die einfachste Kontrollstruktur ist die Serie. Darunter versteht man einen geradlinigen Verlauf des Programms. Kontrollstrukturen können mit Struktogrammen veranschaulicht werden.

Die Serie ist die einfachste Kontrollstruktur. Die Methode main() der Startklasse war nach diesem Muster aufgebaut. Das Programm arbeitete Anweisung für Anweisung linear ab.	
In der Schleife oder Wiederholung wird der Programmcode solange wiederholt, bis die Bedingung zum Abbruch erfüllt ist (z.B.: wiederhole solange Wert < 20). Ist das Abbruchkriterium erfüllt wird Anweisung 2 abgearbeitet.	
Die Auswahl prüft, ob eine Bedingung erfüllt ist. Im „ja“-Fall werden die Anweisungen a bis c, im „nein“-Fall die Bedingungen d und e ausgeführt. Im Anschluss wird in beiden Fällen Anweisung 2 ausgeführt.	

Von besonderer Bedeutung sind die folgenden beiden Ausdrücke im Quellcode unserer Methode:

```
kassenzetteldrucken(verkaufsmenge);
```

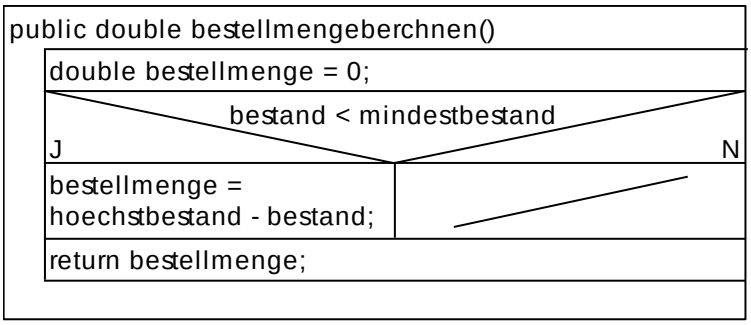
und

```
bestellmengeberechnen(verkaufsmenge);
```

Es handelt sich jeweils um Methodenaufrufe, in beiden Fällen wird der Inhalt der Übergabevariable „verkaufsmenge“ als Input in die aufgerufene Methode gegeben. Im ersten Fall wird die Methode „kassenzetteldrucken(…)“, im zweiten Fall die Methode „bestellmengeberechnen(…)“ aufgerufen.

Diese Vorgehensweise – Quellcode in Methoden zu strukturieren, an andere Stelle im Quellcode auszulagern und diesen durch einen Methodenaufruf zu aktivieren – nennt man **Modularisierung**.

Im jetzigen Projektzustand fehlt noch der Quellcode der beiden Methoden „bestellmengeberechnen(…)“ und „kassenzetteldrucken(…)“:



Quellcode der Methode „bestellmengeberchnen()“:

```
public double bestellmengeberechnen()
{
	double bestellmenge = 0;
	if (bestand < mindestbestand)
	{
		bestellmenge = hoechstbestand - bestand;
	}

	return bestellmenge;
}
```

public String kassenzetteldrucken(double verkaufsmenge)
String kassenzettel;
double gesamtpreis;
gesamtpreis = preis * verkaufsmenge;
kassenzettel = "Name: " + name + "\n" + "Verkaufsmenge: " + verkaufsmenge + "\n" + "Preis in Euro: preis + "\n" + "Gesamtpreis: " + gesamtpreis + "\n";
return kassenzettel;

Quellcode der Methode „kassenzetteldrucken(double verkaufsmenge)“:

```
public String kassenzetteldrucken(double verkaufsmenge)
	{
		String kassenzettel;
		Double gesamtpreis;
		gesamtpreis = preis * verkaufsmenge;

		kassenzettel = "Name: " + name + "\n" + "Verkaufsmenge: " +
		verkaufsmenge + "\n" + "Preis in Euro: " + preis + "\n" +
		"Gesamtpreis: " + gesamtpreis + "\n";

		return kassenzettel;
	}
```

Nachdem die Klasse Getraenk nun um die besprochenen drei Methoden ergänzt wurde, können diese drei Methoden auch benützt werden. Dazu ergänzen wir unsere bisherige Startklasse um den folgenden Quellcode:

```
		//Verkauf von 500 Einheiten des ersten Getränks:
		System.out.println(getraenk1.getraenkverkaufen(500));
		System.out.println("Von Getränk " + getraenk1.getName() + " sind " +
		getraenk1.bestellmengeberechnen() + " Einheiten zu bestellen.");
		System.out.println();

		//Verkauf von 4000 Einheiten des ersten Getränks:
		System.out.println(getraenk2.getraenkverkaufen(2800));
		System.out.println("Von Getränk " + getraenk2.getName() + " sind " +
		getraenk2.bestellmengeberechnen() + " Einheiten zu bestellen.");
		System.out.println();

		//Verkauf von 1500 Einheiten des ersten Getränks:
		System.out.println(getraenk3.getraenkverkaufen(1500));
		System.out.println("Von Getränk " + getraenk3.getName() + " sind " +
		getraenk3.bestellmengeberechnen() + " Einheiten zu bestellen.");
		System.out.println();
```

7. Einfache Dialogsteuerung über die Konsole

Der bislang erstellte Quellcode erlaubt es lediglich dem Programmierer, Daten dadurch zu erfassen, dass er diese direkt in den Quellcode eingibt.
Im eingangs definierten Anwendungsfalldiagramm hatten wir allerdings realitätsnäher verschiedene Akteure, die mit unserer Software interagieren sollen, ausfindig gemacht.
Vor diesem Hintergrund erweitern wir unsere Software nun um die Möglichkeit, dass nicht nur der Programmierer, sondern auch ein Akteur Eingaben in den Programmablauf machen kann.
Dies wollen wir hier am Beispiel der Eingabe der Verkaufsmenge vorführen.
Der am einfachsten zu realisierende Fall einer solchen Eingabe ist die Eingabe direkt in die Konsole, welche wir hier mit der Klasse „Scanner“ realisieren.

Um die Eingabe der Verkaufsmenge umsetzen zu können, benötigt das Programm eine Klasse, die im „Basisbefehlssatz“ nicht enthalten ist. Dies ist die Klasse „Scanner“, die in der Bibliothek „java.util.scanner“ steht. Die Klasse Scanner wird wie nebenstehend gezeigt importiert.

```
package paketGetraenkemarkt1;

/*Der Standard-Umfang von Java wird um die Klasse Scanner,
die sich in der Bibliothek java.util befindet, erweitert:*/
import java.util.Scanner;


public class Startklasse {


    public static void main(String[] args) {


        // Erzeugen des ersten Objekts
        Getraenk getraenk1;
```

Unser neues Objekt, anhand dessen das Einlesen von Werten von der Tastatur gezeigt werden soll, wird nun in der Startklasse deklariert. Zusätzlich definieren wir noch eine Hilfsvariable „menge“ zur Zwischenspeicherung der gescannten Tastaturgeingabe:

```
//Eingabe der Verkaufsmenge von Getränk 2 über die Konsole:

//Erzeugen eines Objekts der Klasse Scanner für das Einlesen von der Tastatur:
Scanner tastatur;
tastatur = new Scanner(System.in);
//Deklarieren einer Hilfsvariable zur Abspeicherung der Tastatureingabe:
double menge;
```

Erläuterung zur Nutzung der Klasse Scanner:
Die Scannerklasse wird dazu veranlasst, ein neues Objekt zu erzeugen, das den Namen „tastatur“ erhält. Der Aufruf zur Erzeugung des Objektes übergibt den Parameter (System.in), was hier bedeutet, dass das Standardinputgerät (=Tastatur) des Computersystems auf Eingabe abgeprüft wird. Dieser „InputStream“ wird dem Objekt „tastatur“ übergeben.
Die Klasse System repräsentiert einen Teil der Hardware des zugrunde liegenden Rechners. Die Methoden der Klasse ermöglichen den Einsatz von Hardwarekomponenten. System.out steht für das Standardausgabegerät Bildschirm.

Nachfolgend ist der gesamte Javacode der abgeänderten Startklasse aufgelistet und kommentiert:

```
package paketGetraenkemarkt1;

/*Der Standard-Umfang von Java wird um die Klasse Scanner,
die sich in der Bibliothek java.util befindet, erweitert:*/
import java.util.Scanner;


public class Startklasse {

	public static void main(String[] args) {


		// Erzeugen des ersten Objekts
		Getraenk getraenk1;
		getraenk1 = new Getraenk();

		//Setzen der Attributswerte via Set-Methoden::
		getraenk1.setName("Ale");   //Strings in Anführungszeichen
		getraenk1.setPreis(1.5);    //Dezimaltrennzeichen ist der Punkt
		getraenk1.setBestand(1000);
		getraenk1.setHoechstbestand(2000);
		getraenk1.setMindestbestand(50);

		//Ausgabe der Attributswerte des ersten Getränke-Objekts auf der Konsole:
		System.out.println("Name des Getränks: " + getraenk1.getName());
		System.out.println("Preis:             " + getraenk1.getPreis());
		System.out.println("Bestand:           " + getraenk1.getPreis());
		System.out.println("Höchstbestand:     " + getraenk1.getHoechstbestand());
		System.out.println("Mindestbestand:    " + getraenk1.getMindestbestand());
		System.out.println();

		//Erzeugen eines zweiten Objekts unter Benützung des ersten neuen Konstruktors:
		Getraenk getraenk2;
		getraenk2 = new Getraenk("Dorfbrunnen Mineralwasser");

		//Setzen der restlichen Attributswerte via Set-Methoden:
		getraenk2.setPreis(0.5);
		getraenk2.setBestand(3000);
		getraenk2.setHoechstbestand(5000);
		getraenk2.setMindestbestand(1000);

		//Ausgabe der Attributswerte des zweiten Getränke-Objekts auf der Konsole:
		System.out.println("Name des Getränks: " + getraenk2.getName());
		System.out.println("Preis:             " + getraenk2.getPreis());
		System.out.println("Bestand:           " + getraenk2.getPreis());
		System.out.println("Höchstbestand:     " + getraenk2.getHoechstbestand());
		System.out.println("Mindestbestand:    " + getraenk2.getMindestbestand());
		System.out.println();

		//Erzeugen eines dritten Objekts unter Benützung des zweiten neuen
		//Konstruktors:
		Getraenk getraenk3;
		getraenk3 = new Getraenk("Orangenlimonade", 2000, 1800, 700, 0.8);


		//Ausgabe der Attributswerte des dritten Getränke-Objekts auf der Konsole:
		System.out.println("Name des Getränks: " + getraenk3.getName());
		System.out.println("Preis:             " + getraenk3.getPreis());
		System.out.println("Bestand:           " + getraenk3.getPreis());
		System.out.println("Höchstbestand:     " + getraenk3.getHoechstbestand());
		System.out.println("Mindestbestand:    " + getraenk3.getMindestbestand());
		System.out.println();

		//Eingabe der Verkaufsmenge von Getränk 2 über die Konsole:

		//Erzeugen eines Objekts der Klasse Scanner für das Einlesen von der Tastatur:
		Scanner tastatur;
```

```
                tastatur = new Scanner(System.in);
                //Deklarieren einer Hilfsvariable zur Abspeicherung der Tastatureingabe:
                double menge;

                //Beginn des Eingabe-Dialogs auf der Konsole:
                System.out.println("Geben Sie die Verkaufsmenge für das Getränk ---Dorfbrunnen
                Mineralwasser----- ein:");

        //Der Variablen "menge" wird die letzte Zeile aus dem Tastaturpuffer zugewiesen:
                menge = tastatur.nextDouble();
                System.out.println(getraenk2.getraenkverkaufen(menge));
                System.out.println("Von Getränk " + getraenk2.getName() + " sind " +
                getraenk2.bestellmengeberechnen() + " Einheiten zu bestellen.");
                System.out.println();

        }
}
```

8. Statische Attribute

Es kommt oft vor, dass Attribute für alle Objekte einer Klasse den gleichen Wert haben. Für das besprochene Beispiel kann man sich vorstellen, dass für alle Getränke der gleiche Mehrwertsteuersatz gilt. In diesem Fall existiert der Mehrwertsteuersatz unabhängig von einem konkreten Objekt. Man spricht daher von einem „statischen Attribut“ oder auch „Klassenattribut“. Die set- und get-Methoden, die es erlauben den Wert zu ändern, müssen auch statisch sein. Dieses Vorgehen hat folgenden Vorteil: Um den Mehrwertsteuersatz für alle Objekte der Klasse zu ändern, muss nur an einer Stelle eine Änderung vorgenommen werden.
In der Klasse Getraenk müssen folgende Zeilen ergänzt werden (man kann auch das Klassendiagramm bearbeiten, um die Änderungen zu bewirken):

```
        private static double mwsteuersatz;


        public static double getMwsteuersatz() {
                return mwsteuersatz;
        }


        public static void setMwsteuersatz(double mwsteuersatz) {
                Getraenk.mwsteuersatz = mwsteuersatz;
        }
```

Das Attribut und seine zugehörigen Methoden erhalten den Zusatz „static“

Im Klassendiagramm sind das statische Attribut und die statischen Methoden unterstrichen:

Die Zuweisung eines Wertes für eine statische Variable in der Startklasse benötigt kein bestimmtes Objekt. Die Anweisung für die Zuweisung funktioniert deshalb über die Klasse:
Allgemeine Zuweisung eines Objektattributs: `objektname.setMethode(Übergabewert);`
Allgemeine Zuweisung eines Klassenattributs: `Klassenname.setMethode(Übergabewert);`

Für das Beispiel gilt die Anweisung:

```
public class Startstatischundschleife {

    public static void main(String[] args) {

        //Festlegen der Umsatzsteuer für alle Objekte der Klasse Getraenk:
        Getraenk.setMwsteuersatz(0.19);

        // Erzeugen des ersten Objekts
        Getraenk getraenk1;
        getraenk1 = new Getraenk();
```

Diese Anweisung kann vor der Erzeugung oder Deklaration eines Objektes in der Startklasse stehen. Statische Attribute werden in Eclipse kursiv gedruckt.

Um die anteilige Mehrwertsteuer auf dem Kassenzettel auszuweisen könnte die Methode „kassenzetteldrucken(double verkaufsmenge)“ folgendermaßen ergänzt werden:

```
public String kassenzetteldrucken(double verkaufsmenge)
        {
                String kassenzettel;
                Double gesamtpreis;
                gesamtpreis = preis * verkaufsmenge;

                kassenzettel = "Name: " + name + "\n" + "Verkaufsmenge: " + verkaufsmenge + "\n"
                + "Preis in Euro: " + preis + "\n" + "Gesamtpreis: " + gesamtpreis + "\n";

                //Berechnung der anteiligen Umsatzsteuer:
                double mwsteueranteil;
```

```
            mwsteueranteil = (gesamtpreis * 19)/119;
            kassenzettel = kassenzettel + "Anteilige Mehrwertsteuer in EUR: " +
            mwsteueranteil;

            return kassenzettel;
    }
```

Die resultierende Konsolenausgabe samt Ausgabe des Mehrwertsteueranteils sieht dann folgendermaßen aus:

```
Name des Getränks: Dorfbrunnen Mineralwasser
Preis:             0.5
Bestand:           0.5
Höchstbestand:     5000.0
Mindestbestand:    1000.0

Name des Getränks: Orangenlimonade
Preis:             0.8
Bestand:           0.8
Höchstbestand:     2000.0
Mindestbestand:    700.0

Geben Sie die Verkaufsmenge für das Getränk ---Dorfbrunnen Mineralwasser----- ein:
150
Name: Dorfbrunnen Mineralwasser
Verkaufsmenge: 150.0
Preis in Euro: 0.5
Gesamtpreis: 75.0
Anteilige Mehrwertsteuer in EUR: 11.974789915966387
Von Getränk Dorfbrunnen Mineralwasser sind 0.0 Einheiten zu bestellen.
```

Die anteilige Mehrwertsteuer wird in der Konsole mit 15 Nachkommastellen ausgegeben. Hier wäre eine Ganzzahl sinnvoller. Aus diesem Grund soll die Variable *mwsteueranteil* vor der Ausgabe gerundet werden.
Um die entsprechende Funktionalität zur Verfügung zu haben, benutzen wir die Klasse „Math“ und ergänzen unsere Methode zum Drucken eines Kassenzettels um eine Zeile für das Runden der Variable *mwsteueranteil:*

```
//Berechnung der anteiligen Umsatzsteuer, gerundet:
            double mwsteueranteil;
            mwsteueranteil = (gesamtpreis * 19)/119;
            mwsteueranteil = Math.round(mwsteueranteil); //Runden!
            kassenzettel = kassenzettel + "Anteilige Mehrwertsteuer in EUR: " +
            mwsteueranteil;
```

Die Startklasse bleibt unverändert. Ein neuer Programmlauf zeigt folgende Konsole:

```
Name des Getränks: Dorfbrunnen Mineralwasser
Preis:             0.5
Bestand:           0.5
Höchstbestand:     5000.0
Mindestbestand:    1000.0

Name des Getränks: Orangenlimonade
Preis:             0.8
Bestand:           0.8
Höchstbestand:     2000.0
Mindestbestand:    700.0

Geben Sie die Verkaufsmenge für das Getränk ---Dorfbrunnen Mineralwasser----- ein:
150
Name: Dorfbrunnen Mineralwasser
Verkaufsmenge: 150.0
Preis in Euro: 0.5
Gesamtpreis: 75.0
Anteilige Mehrwertsteuer in EUR: 12.0
Von Getränk Dorfbrunnen Mineralwasser sind 0.0 Einheiten zu bestellen.
```

9. Wiederholstrukturen

Zur Steuerung der Ablauflogik innerhalb einer Methode sind neben Verzweigungen oft Wiederholstrukturen (Schleifen) nötig. Wir wollen hier die sogenannte „For-Schleife“ vorstellen.
Hierzu nehmen wir an, dass wir für das Objekt „getraenk2“ drei Verkaufsvorgänge hintereinander ablaufen lassen wollen. Um den dazu notwendigen Quellcode nicht dreimal hintereinander schreiben zu müssen, ist eine automatisierte Wiederholung des Quellcodes angebracht. Der relevante Quellcode in der Startklasse wird deshalb durch eine Schleifenstruktur umklammert:

```
//3-fache Wiederholung des folgenden Quellcodes:
        //Beginn der For-Schleife
        for (int i = 0; i < 3; i++)
        {

        //Eingabe der Verkaufsmenge von Getränk 2 über die Konsole:

        //Erzeugen eines Objekts der Klasse Scanner für das Einlesen von der Tastatur:
                        Scanner tastatur;
                        tastatur = new Scanner(System.in);
        //Deklarieren einer Hilfsvariable zur Abspeicherung der Tastatureingabe:
                        double menge;
                        //Beginn des Eingabe-Dialogs auf der Konsole:
                        System.out.println("Geben Sie die Verkaufsmenge für das Getränk -
                        --Dorfbrunnen Mineralwasser----- ein:");

        //Der Variablen "menge" wird die letzte Zeile aus dem Tastaturpuffer
        //zugewiesen:
                        menge = tastatur.nextDouble();
                        System.out.println(getraenk2.getraenkverkaufen(menge));
                        System.out.println("Von Getränk " + getraenk2.getName() + " sind
                        " + getraenk2.bestellmengeberechnen() + " Einheiten zu
                        bestellen.");
                        System.out.println();


        //Am Ende eines jeden Schleifendurchgangs wird der aktuelle Bestand von Getränk
        //2 ausgegeben:
                        System.out.println("Von Getränk 2 sind noch " +
                        getraenk2.getBestand() + " Einheiten im Lager.");
                        System.out.println();
                        System.out.println();
```

```
            }  //Ende der For-Schleife
```

Die Schleifensteuerung `for(int i = 0; i < 3; i++)` bedeutet, dass ein Zähler i (nur Ganzzahlen) von angegebenen Startwert Null zu laufen beginnt. Dieser Zähler wird am Ende eines jeden Schleifendurchgangs erhöht. „i++“ ist eine abkürzende Schreibweise für „i = i+1“. Dies bedeutet, dass i immer um 1 erhöht wird. Der durch die Schleife eingerahmte Quellcode (der Quellcode im sogenannten „Schleifenrumpf“) wird also genau dreimal durchlaufen, was zu folgender Konsolendialog führt:

```
Geben Sie die Verkaufsmenge für das Getränk ---Dorfbrunnen Mineralwasser----- ein:
50
Name: Dorfbrunnen Mineralwasser
Verkaufsmenge: 50.0
Preis in Euro: 0.5
Gesamtpreis: 25.0
Anteilige Mehrwertsteuer in EUR: 4.0
Von Getränk Dorfbrunnen Mineralwasser sind 0.0 Einheiten zu bestellen.

Von Getränk 2 sind noch 2950.0 Einheiten im Lager.


Geben Sie die Verkaufsmenge für das Getränk ---Dorfbrunnen Mineralwasser----- ein:
1000
Name: Dorfbrunnen Mineralwasser
Verkaufsmenge: 1000.0
Preis in Euro: 0.5
Gesamtpreis: 500.0
Anteilige Mehrwertsteuer in EUR: 80.0
Von Getränk Dorfbrunnen Mineralwasser sind 0.0 Einheiten zu bestellen.

Von Getränk 2 sind noch 1950.0 Einheiten im Lager.


Geben Sie die Verkaufsmenge für das Getränk ---Dorfbrunnen Mineralwasser----- ein:
500
Name: Dorfbrunnen Mineralwasser
Verkaufsmenge: 500.0
Preis in Euro: 0.5
Gesamtpreis: 250.0
Anteilige Mehrwertsteuer in EUR: 40.0
Von Getränk Dorfbrunnen Mineralwasser sind 0.0 Einheiten zu bestellen.

Von Getränk 2 sind noch 1450.0 Einheiten im Lager.
```