Unterrichtskonzept

Didaktisch-methodische Vorüberlegungen

Der Lehrplan der Jahrgangsstufe 10 thematisiert drei große Modellierungskonzepte: Das Zustands- und das Algorithmenmodell zur Beschreibung von Abläufen und die objektorientierte Modellierung zur Beschreibung großer Systeme. Dabei werden die Ablaufmodelle in der objektorientierten Modellierung als Detailbeschreibungen benötigt (Lebenszyklus der Objekte / Beschreibung der Methoden). Aufgrund dieser Verzahnung bietet es sich an, die einzelnen Lehrplanthemen im Zusammenhang und nicht sequentiell zu betrachten; die Lehrplanforderungen können bei der Entwicklung eines oder weniger großer Beispiele abhängig von den Notwendigkeiten der Programmentwicklung an jeweils passender Stelle behandelt werden.

In diesem Kapitel wird ein Beispiel dargestellt, an dem sich der gesamte Lehrplaninhalt veranschaulichen und anwenden lässt. Um dafür geeignet zu sein, muss das Thema nicht nur allen Lehrplaninhalten gerecht werden, es muss auch gestatten, das Zielprodukt in kleinen Schritten aufzubauen, so dass die Schüler jede neue Erkenntnis sofort umsetzen, ausprobieren und anwenden können. Weiter muss sich aus dem Thema heraus ein abgeschlossenes Produkt entwickeln lassen, das keine offenen Bereiche lässt, die man erst mit späterem Wissen ausfüllen kann. Die Themenstellung muss auch berücksichtigen, dass Modellierung immer zielorientiert erfolgt. Abgrenzung und Reduktion sind nur bezüglich der Aufgabenstellung möglich. Technische Geräte (z. B. CD-Spieler) sind diesbezüglich schon vormodelliert und bringen daher kaum Modellierungserkenntnis beim „Nachbau“. Simulationen dagegen erfordern meistens deutlich Abgrenzung, Abstraktion, Idealisierung, Aggregation und Strukturanalyse. Da Simulationen in der Regel keine Datenpersistenz benötigen, können mit den Mitteln der Jahrgangsstufe 10 vollständige, anwendbare Produkte erstellt werden.

Überblick über die Sequenz

Folgende Aufgabenstellung wird in diesem Unterrichtskonzept als Thema gewählt: Simulation des Kassenbereichs eines Supermarktes mit dem Ziel, bei bekannter Kundenfrequenz die nötige Zahl offener Kassen, d. h. den Personalbedarf, zu ermitteln.

Für die Umsetzung des Beispiels wird von einem Unterrichtsaufbau in Doppelstunden (DS) ausgegangen. Der Überblick über die Unterrichtssequenz ist in kleine, logische Einheiten gegliedert, die zur Erarbeitung ein bis zwei Doppelstunden beanspruchen. Von den im Lehrplan vorgeschlagenen 46 Wochenstunden für die Kapitel Inf 10.1 und Inf 10.2 werden damit ca. 38 belegt, so dass noch viel Raum für zusätzliche Übungen und Vertiefungen bleibt. Stunden für organisatorische Arbeiten, Lernzielkontrollen und Ähnliches sind hier nicht zu berücksichtigen, da die im Lehrplan vorgeschlagenen Stunden nicht den vollen Jahresstundenumfang ausnützen. Zu jeder Einheit existiert bei den Materialien eine Beispiellösung.

Im Einzelnen wird folgender, in 13 Einheiten untergliederter Ablauf vorgeschlagen:

1. Erster Einblick (1 DS)
2. Bekanntschaft mit dem Werkzeug – Objekte (1 DS)
3. Eigene Klassen (1 DS)
4. Methoden (1 DS)
5. Zustände (2 DS)

6. Einfache Warteschlange (2 DS)

7. Felder (2 DS)

8. Der Supermarkt (2 DS)

9. Automatischer Ablauf (1 DS)

10. Zufälligkeiten (1 DS)

11. Vererbung und Polymorphismus (2 DS)

12. Besondere Klassen(1 DS)

13. Graphische Oberfläche (2 DS)

In der ersten Einheit werden anhand der Erstellung eines Basispflichtenhefts zunächst die gesamte bekannte OO-Begrifflichkeit und die Darstellung eines Modells als Klassendiagramm wiederholt. In der zweiten Einheit wird das neu benötigte Werkzeug vorgestellt, gleichzeitig wiederholen die Schüler den Umgang mit Objekten und die Verwendung von Methoden. In den Einheiten 3 und 4 werden von den Jugendlichen erste eigene Klassen für das Zielprodukt erstellt. Für die Beschreibung des Verhaltens der Kassen lernen die Schüler in der 5. Einheit das Zustandsmodell (Automat) und dessen graphische Darstellung als Zustandsübergangsdiagramm kennen. Sie setzen dieses Modell in Methoden um. Die Einheiten 1 mit 5 decken das Lehrplankapitel „Inf 10.1.1 Zusammenfassung und Festigung der bisher erlernten objektorientierten Konzepte" ab und den ersten Teil von Lehrplankapitel „Inf 10.1.2 Zustände von Objekten und algorithmische Beschreibung von Abläufen". Der Schwerpunkt liegt auf der Zustandsmodellierung einschließlich des Variablenmodells und der Zuweisung. Algorithmische Elemente wie bedingte Anweisung werden nur so weit behandelt, wie sie zur Umsetzung des Modells in die Programmiersprache notwendig sind.

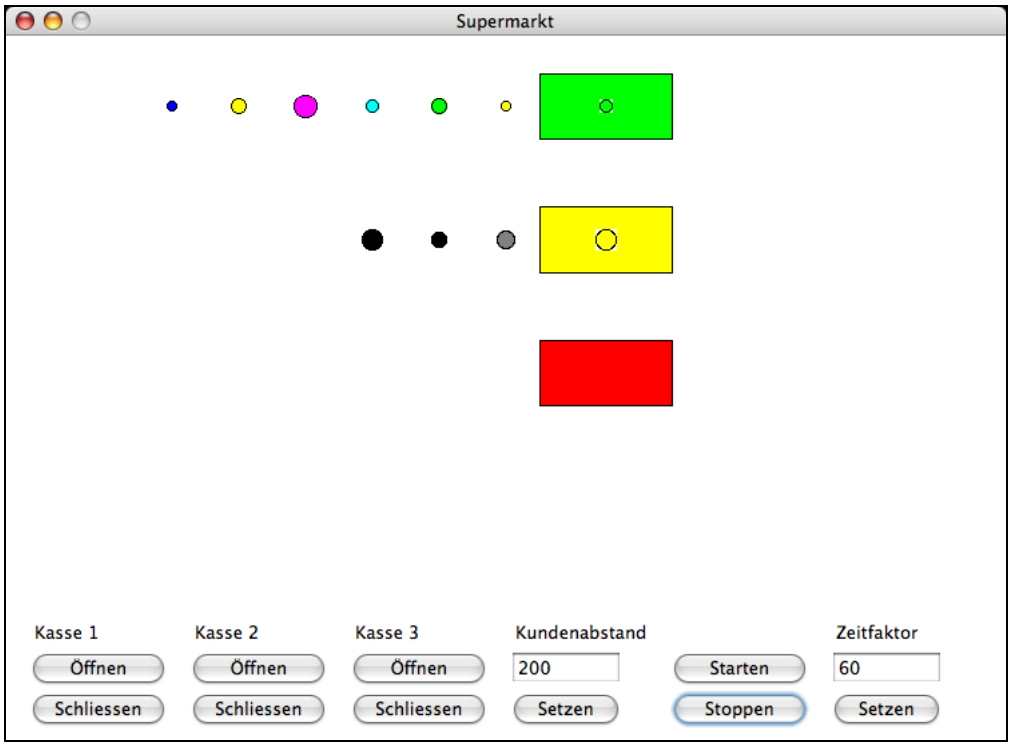
In den Einheiten 6 mit 9 steht die algorithmische Beschreibung von Abläufen im Vordergrund. Dabei werden anhand der Warteschlangenverwaltung (Einheiten 6 und 7) auch die Notwendigkeit strukturierter Datentypen, insbesondere von Feldern, und der Umgang damit thematisiert sowie die restlichen Inhalte des Lehrplankapitels Inf 10.1.2 behandelt. Dem Thema Beziehungen zwischen Objekten, d. h. dem Lehrplankapitel Inf 10.1.3, wird in dieser Einheitengruppe ebenfalls breiter Raum gewidmet.

In den Einheiten 10 mit 12 werden die Mechanismen Vererbung und Polymorphismus eingeführt und in einem breiten Anwendungsspektrum genutzt. Einheit 12 ist dabei eine Vertiefung der Anwendung von Vererbung, die zusätzliche Konzepte für den Umgang mit graphischen Benutzeroberflächen und für das Projekt liefert. Eine Thematisierung von Entwurfsmustern, die sich hier konsequent anschließen könnte, ist erst in der Jahrgangsstufe 11 vorgesehen.

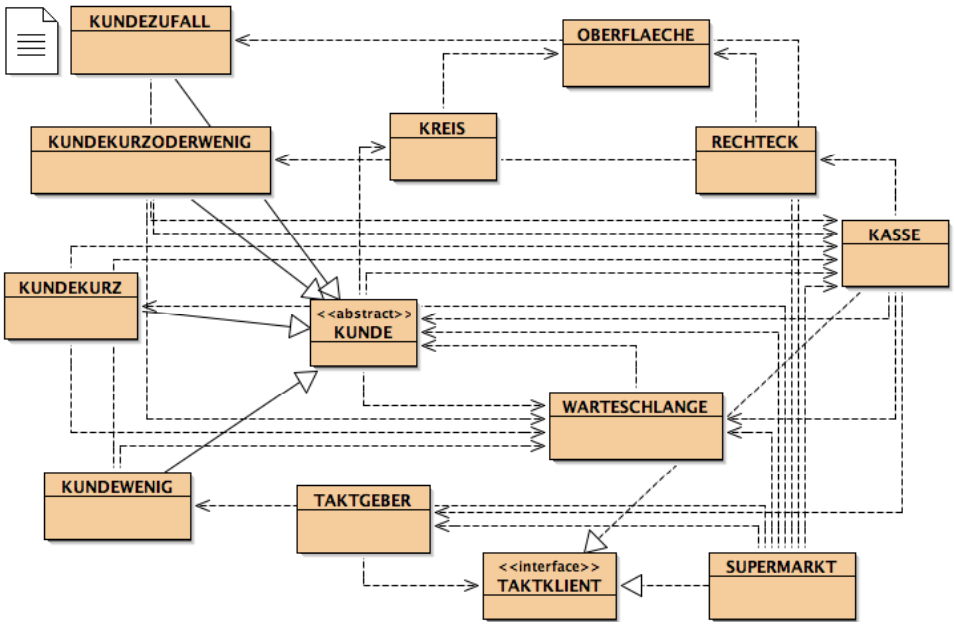
Mögliches Ergebnis der Unterrichtssequenz

Die Fragestellung in der Einstiegsstunde mündet in einen präzisen Auftrag, der aber über den logischen Ansatz und die Gestaltung des Programms nur wenige Aussagen macht. Deshalb soll hier kurz die Endfassung der Beispiellösung vorgestellt werden, um den Kontext der einzelnen Schritte besser herauszustellen.

Die fertige Simulation stellt die drei Kassen des Supermarkts mit ihren Warteschlangen in einer einfachen Graphik dar. Unter dem „Kassenbereich" sind die Bedienelemente des Programms angeordnet. Zu erkennen ist die unterschiedliche Größe der Kundensymbole als Darstellung für die Anzahl der Artikel im Warenkorb. Die obere Kasse (grün) ist offen, die mittlere (gelb) schließt gerade, die untere Kasse (rot) ist geschlossen.



Das Klassendiagramm ist in der Endfassung sehr umfangreich. Für die Schüler reicht zu Beginn ein Diagramm der zentralen Klassen (siehe Hefteintrag zur 1. Einheit) aus. Für die Lehrkraft ist hier ein Überblick über die Klassen des fertigen Produkts anhand des zugehörigen BlueJ-Projekts gegeben.



1. Erster Einblick (1. Doppelstunde)

In dieser Einheit lernen die Schüler den Inhalt des Schuljahres und seine Einbindung in den roten Faden der Mittelstufe, d. h. in den Themenbereich „Modellierung“, kennen. Anhand der Analyse des von der Lehrkraft als Auftrag vorgegebenen Beispiels wiederholen die Schüler die Vorgehensweise objektorientierter Analyse und die dazu gehörende Begrifflichkeit (Objekt, Attribut, Attributwert, Methode, Klasse, Beziehung).

Einstieg

Wiederholung des Begriffs Modellierung anhand einer Präsentation zum Thema „Modellierung“ (Präsentation „Modellierung“) und anhand von Beispielen aus der Datenmodellierung, die den Schülern bereits aus der vorangegangenen Jahrgangsstufe bekannt sind.

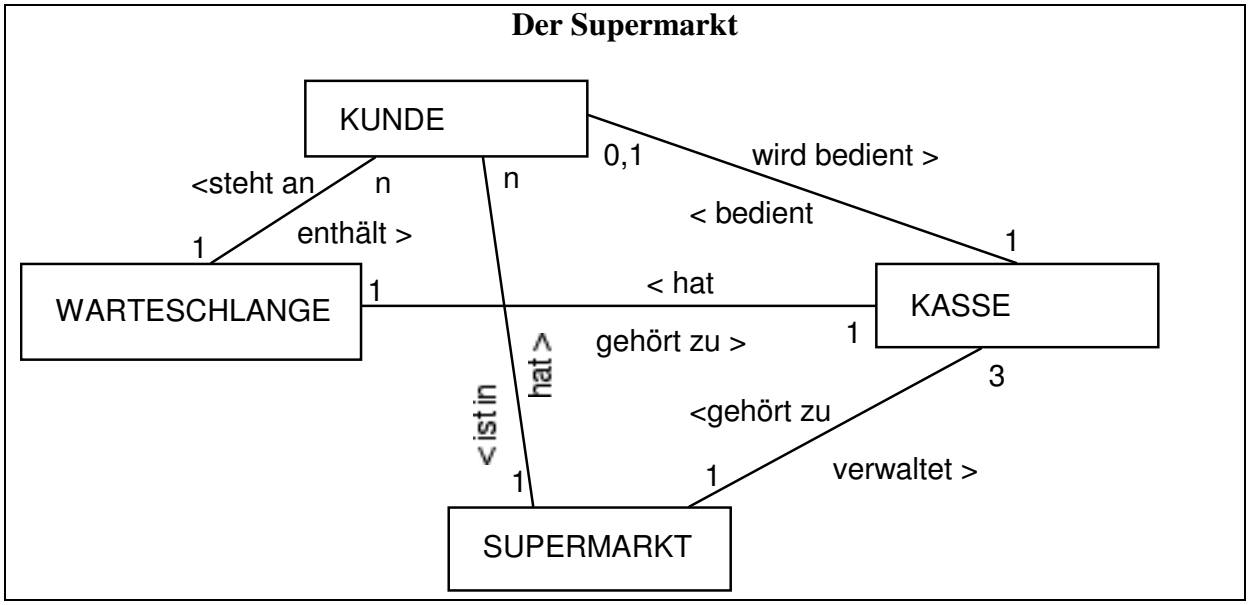
Impuls

Der Lehrer stellt als „Auftraggeber“ an die Firma „Klasse 10x“ das Thema vor: „Der Supermarkt Oldi möchte für eine optimale Personalplanung ein Programm, das die Länge der Warteschlangen an den Kassen in Abhängigkeit von Kundendichte und Anzahl offener Kassen simuliert. Der Supermarkt hat drei Kassen, das Programm soll aber so ausgelegt sein, dass es auch für Märkte mit mehr Kassen schnell erweitert werden kann. Die Kassen und die eventuell dort wartenden Kunden sollen in einer einfachen Graphik übersichtlich dargestellt werden. Die empirischen Daten über die Anzahl und das Einkaufsverhalten der Kunden liegen bereits aus einer zuvor durchgeführten Kundenerhebung vor.“

Analyse und Auswertung

In einem kurzen Brainstorming – evtl. unterstützt von einem „unscharfen“ Bild einer speziellen Kassensituation (Präsentation „Supermarkt“) – denken sich die Schüler in die Situation ein. Die Ergebnisse der Vorüberlegungen werden zu einem groben Pflichtenheft (siehe Hefteintrag) mit einem ersten Klassendiagramm und einer informellen Beschreibung der Aufgabe der einzelnen Objekte zusammengefasst. Gegebenenfalls muss die Lehrkraft durch Impulsfragen deutlich machen, dass Kassen und Warteschlangen verschiedene Aufgaben haben, also durch getrennte Objekte zu modellieren sind. Zur Veranschaulichung eignen sich auch Rollenspiele der Schüler bzw. das Nachstellen der Kassensituation mit Spielfiguren.

Hefteintrag



Kunden haben ein Darstellungssymbol, das die Artikelanzahl im Einkaufskorb wiedergibt.
Kassen haben ein Darstellungssymbol, das erkennen lässt, ob die Kasse offen oder geschlossen ist, und verwalten den Kunden, der gerade bedient wird.
Warteschlangen verwalten die Kunden vor einer Kasse.
Der Supermarkt „organisiert" das Öffnen und Schließen der Kassen und hat Kunden.

2. Bekanntschaft mit dem Werkzeug – Objekte (2. Doppelstunde)

In dieser Einheit verwenden die Schüler erstmals das Werkzeug BlueJ. Sie lernen, Objekte vorgegebener Klassen zu instanzieren und mit diesen Objekten zu arbeiten (Methodenaufruf). Dabei verbinden die Schüler relevante Begriffe objektorientierter Sichtweise – Objekt, Attribute, Methode, Botschaft – mit dem Softwarewerkzeug.

Impuls

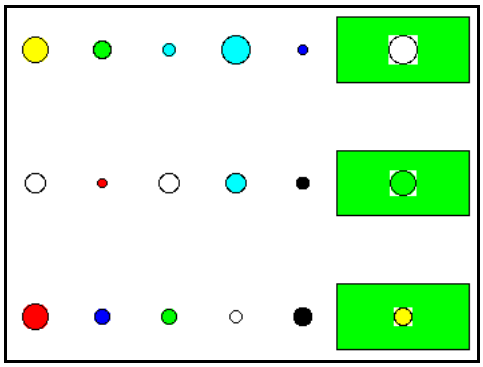
Der Lehrer verteilt ein BlueJ-Projekt „Supermarkt_1" mit den Klassen KREIS und RECHTECK.

Vorstellung des Werkzeugs

Unter Anleitung des Lehrers starten die Schüler das Werkzeug, öffnen das Projekt und erhalten Einblick in die Elemente der Oberfläche und der Menüs. Insbesondere werden das Kontextmenü der Klassensymbole, das Erzeugen neuer Objekte, die Objektleiste und das Kontextmenü der Objekte zum Methodenaufruf näher betrachtet.

Arbeit am Computer

Die Schüler erzeugen nun selbständig Objekte der vorgegebenen Klassen KREIS und RECHTECK und organisieren diese zu einem ungefähren Layout des Supermarkts. Dabei notieren sie ihre Schritte (Punktschreibweise; evtl. auch im Codepad-Fenster) als Vorbereitung für die nächste Stunde. Hier können durchaus unterschiedliche Lösungsideen entstehen, die anschließend vorgestellt und diskutiert werden. Eine mögliche Darstellung ist:



1. Figuren testen
   a) Erzeuge je ein Objekt der Klassen KREIS und RECHTECK (z. B. k und r).
   b) Positioniere das Rechteck an den rechten Rand des Fensters.
   c) Verändere Farbe und Größe des Rechtecks so, dass es eine offene Kasse repräsentiert.
   d) Notiere die nötigen Methodenaufrufe in der Punktschreibweise (z. B. `r. FarbeSetzen( __ ),r.PositionSetzen( __ , __ )`).
   e) Positioniere den Kreis etwa in der Mitte des Rechtecks; er soll den Kunden darstellen, der gerade bedient wird. Notiere die nötigen Methodenaufrufe.

2. Schlange bauen

   Eine Warteschlange besteht aus Kunden.

   a) Erzeuge mehrere Objekte der Klasse KREIS mit den Namen k1, k2, … Gib den Objekten verschiedene Größen und Farben.

   b) Positioniere die Objekte so, dass sie mittig zur Höhe des Kassensymbols hintereinander stehen. Schreibe auch hier für alle Objekte die notwendigen Methodenaufrufe auf. Verwende dabei folgendes Schema:

   ```
   k1.PositionSetzen( _____, _____ ),
   k2.PositionSetzen( _____, _____ ),
   ```

   usw.

3. Eigene Klassen (3. Doppelstunde)

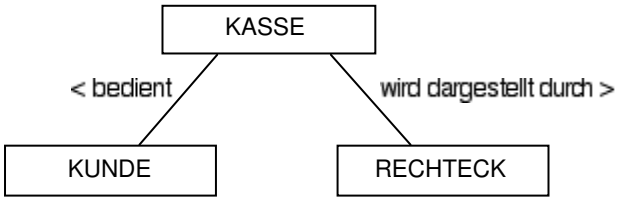
In dieser Stunde erstellen die Schüler erste eigene Klassen. Sie lernen dabei auch die Vereinbarung von Attributen, den Konstruktor als besondere Methode, das Erzeugen neuer Objekte und die Zuweisung von Objektreferenzen an Attribute kennen.

Impuls

Kunden und Kassen sind keine Objekte der Klassen KREIS bzw. RECHTECK, da sie eigene Eigenschaften, wie z. B. die Artikelzahl des Kunden und (später) eigene Methoden besitzen. Allerdings werden sie in der graphischen Ansicht durch Objekte der Klassen KREIS bzw. RECHTECK dargestellt.

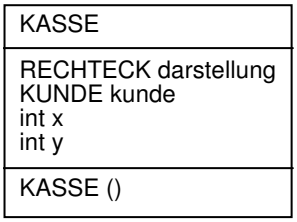
Die Attribute der Klasse KASSE / Anlegen einer Klassenbeschreibung

In einer Vorüberlegung werden ausgehend vom folgenden Klassendiagramm mögliche Attribute der Klasse KASSE angedacht.



Ein Objekt der Klasse KASSE benötigt Attribute x und y zur Speicherung seiner Position, um daraus später z. B. die Position für die Anzeige des gerade bedienten Kunden ermitteln zu können.

Ähnlich wie in der Datenmodellierung werden die Beziehungen zwischen den Klassen auch hier über zusätzliche Attribute in der Implementierung umgesetzt. In der Datenmodellierung werden die Fremdschlüssel ins Tabellenschema aufgenommen. In der objektorientierten Modellierung ist ein vergleichbarer Zwischenschritt nicht vorhanden. Diese Rolle übernimmt deshalb ein erweitertes Klassendiagramm, bei dem neben den Referenzattributen für die Beziehungen auch die Datentypen der übrigen Attribute angegeben sind. Der Vorteil der zusätzlichen Information wird allerdings durch den Nachteil einer geringeren Allgemeingültigkeit erkauft. Es ist darauf zu achten, dass die Schüler Attribute vordefinierter Datentypen (z. B. Datentyp `int`) und Attribute, deren Datentyp eine selbst angelegte Klasse



erweitertes Klassendiagramm

ist, nicht als zwei unterschiedliche Arten von Attributen auffassen. Die Schüler müssen nicht zwischen primitiven Datentypen und „echten“ Klassen unterscheiden.

Die Trennung zwischen der KASSE als Klasse des eigentlichen Modells und ihrem Darstellungssymbol (Rechteck) ergibt sich aus den Übungen der Vorstunden. Damit wird implizit, ohne diesen Ansatz ausdrücklich zu thematisieren, auch die Trennung zwischen Logikklassen (model) und Darstellungsklassen (view) vorbereitet (vgl. Einheit 12 und13).

Nach Erarbeitung der benötigten Attribute und Festhalten der Ergebnisse als Klassendiagramm teilt die Lehrkraft mit, wie dieser Bauplan in der Sprache Java beschrieben wird. Gemeinsam wird die Java-Darstellung der Klasse KASSE erarbeitet. Dabei muss auf die Typisierung der Attribute (im Gegensatz z. B. zu SmallTalk oder Python) und die vorgegebene Klasse `int` eingegangen werden. Zudem muss für die Umsetzung der Beziehungen der Begriff der Objektreferenz mit der Vorstellung eingeführt werden, dass ein Attribut nicht das ganze Objekt der angegebenen Klasse beinhaltet, sondern – wie die Fremdschlüssel bei den Datenbanken – eine Referenz auf das jeweilige Objekt speichert.

Die Schüler übertragen nun die erarbeitete Klassenbeschreibung (Quelltext) in ihr Projekt. Die Lehrkraft teilt mit, dass aus der für den Menschen lesbaren Notation eine für die Maschine ausführbare Darstellung erzeugt werden muss (Compilierung, Übersetzung). Die Schüler übersetzen ihre Klassenbeschreibung und legen Objekte der Klasse KASSE an. Dazu muss – da das Attribut `kunde` vereinbart ist – auch eine leere Klasse KUNDE existieren; es genügt, hierfür den von BlueJ erzeugten Rahmen zu verwenden.

Anfangswerte der Attribute / Konstruktor

Die Tatsache, dass von den erzeugten Objekten der Klasse KASSE nichts zu sehen ist, wirft die Frage auf, welche Werte die Attribute der Objekte haben. Mithilfe des Objektinspektors zeigt sich sofort, dass das Attribut `darstellung` den Wert `null` besitzt. Dies bedeutet, dass die Kasse noch kein Darstellungssymbol hat. Der Ausdruck „null“ ist nicht mit der Ziffer 0 zu verwechseln.

Damit ein Objekt der Klasse KASSE einen „vernünftigen Zustand“ aufweist, ist es nötig, dass alle Attribute geeignete Anfangswerte besitzen. Aus diesem Grund hat jedes Objekt eine spezielle Methode, den so genannten Konstruktor, der automatisch bei der Erzeugung des Objekts aufgerufen wird. Die Schüler erfahren nun die Syntax des Konstruktors, die Notation für die Erzeugung neuer Objekte, z. B. des zugehörigen Darstellungsobjekts der neuen Kasse, und die Zuweisung von Startwerten an Attribute. Der Aufruf von Methoden des Darstellungsobjekts wird mit der bekannten Punktschreibweise formuliert; mit den Ergebnissen der letzen Stunde wissen die Schüler, wie die Objekte der neuen Klasse günstig dargestellt werden. Die neuen Kenntnisse werden jetzt erprobt.

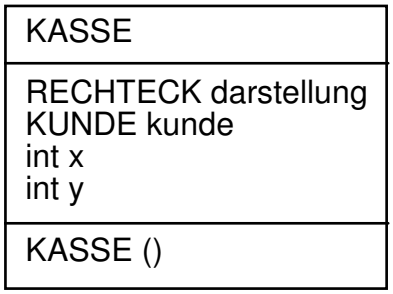
Bei der Vorstellung des Operators `new` zur Instanzierung neuer Objekte sollte darauf eingegangen werden, dass es in Java und verwandten Sprachen kein Gegenstück gibt, mit dem der Lebenszyklus („erzeugen – verwenden – löschen“) eines Objekts vom Programmierer gesteuert beendet wird. In Java wird der Lebenszyklus von Objekten durch den so genannten „garbage collector“ beendet, ein Programmteil, das den Speicherplatz all der Objekte wieder freigibt, zu denen keine Referenz mehr besteht. Das Objekt kann aber auch so lange „leben“, bis das Programm beendet wird.

Hefteintrag Teil 1

Eigene Klassen

Informationen über Beziehungen werden durch eigene Attribute (Referenzattribute) gespeichert. Im erweiterten Klassendiagramm werden diese mit ihren Datentypen dargestellt.

Klassendiagramm der Klasse KASSE und dessen Umsetzung in Java:




```
class KASSE
{
    RECHTECK darstellung;
    KUNDE kunde;
    int x;
    int y;
    Kasse ()
    {
        darstellung = new RECHTECK();
        kunde = null;
        x = 400;
        y = 150;
        darstellung.GroesseSetzen (100, 50);
        darstellung.PositionSetzen (x, y);
        darstellung.FarbeSetzen ("gruen");
    }
}
```

Mit dem Operator **new** werden neue Objekte erzeugt.

Hinweis: Da die Schüler das Struktogramm aus Jahrgangsstufe 7 als Darstellungsform bereits kennengelernt haben, ist es durchaus möglich, dieses bereits zur Formulierung des Konstruktors heranzuziehen; damit lässt sich das Tafelbild (bzw. der Hefteintrag) besser strukturieren. Spätestens beim Auftauchen der ersten Kontrollstrukturen sollte das Struktogramm bei der Herleitung des jeweiligen Algorithmus konsequent benutzt werden, da diese Darstellung flexibler, klarer und universeller ist als jede Programmiersprache.

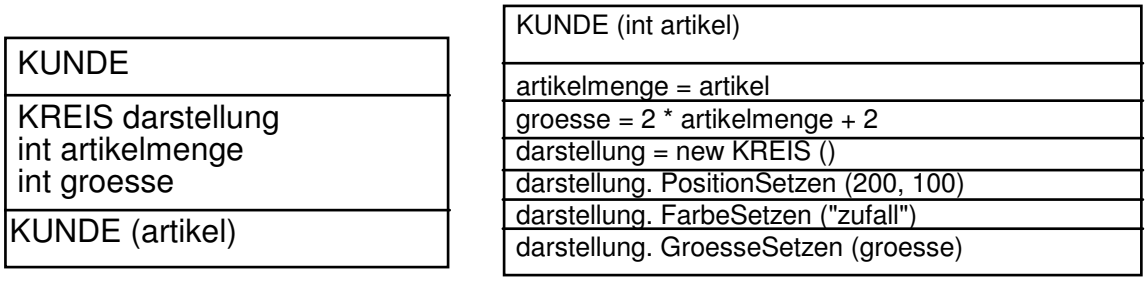
Die Klasse KUNDE / Parameter

Bei der Modellierung der Klasse KUNDE muss festgelegt werden, welches Darstellungssymbol für deren Objekte gewählt wird und wie die Anzahl der Artikel im Warenkorb (Attribut `artikelmenge`) in der Darstellung repräsentiert wird, z. B. über die Größe des Darstellungssymbols „Kreis“. Die Frage „Wie viele Artikel sind beim Instanzieren eines neuen Kunden im Warenkorb?“ führt auf die Zweckmäßigkeit, Methodenparameter auch beim Konstruktor zur Verfügung zu haben.

Das Klassendiagramm für KUNDE wird erarbeitet; in einem einfachen Struktogramm werden die (Arbeits-)Abläufe im Konstruktor festgelegt. Für Rechenterme genügt der Hinweis, dass die Operatoren für die Grundrechenarten in der gewohnten Weise geschrieben werden.

Hefteintrag Teil 2

Klassendiagramm der Klasse KUNDE und der Konstruktor als Struktogramm:



Hinweis: Die Farbe der Kundendarstellung hat für das Modell keine Bedeutung. Deshalb kann sie zur besseren Unterscheidbarkeit der Kunden zufällig gewählt werden. Der Parameterwert "zufall" beim Aufruf der Methode `FarbeSetzen()` bewirkt eine zufällige Farbgebung.

Weitere Arbeit am Computer

Die Schüler erstellen nun selbständig die Klasse KUNDE gemäß dem vorhandenen Diagramm und testen sie.

Hausaufgabe

Die Schüler überlegen, welche Methoden die Klassen KUNDE und KASSE benötigen und welche Aufgabe diese Methoden zu erfüllen haben. Sie notieren Methodennamen und eine kurze, informelle Beschreibung der Semantik (z. B. in Spiegelstrichen).

4. Methoden (4. Doppelstunde)

In dieser Stunde erstellen die Schüler Methoden in den Klassen KUNDE und KASSE. Neben einer Vertiefung der Parameterangabe lernen sie auch, die Rückgabeinformation einer Methode zu beschreiben.

Impuls

Die in der Hausaufgabe für die Klassen KASSE und KUNDE gefundenen Methoden werden besprochen. Im Unterrichtsgespräch einigt man sich auf einheitliche Namen, Parameter und deren Bedeutung. Möglich sind:

Klasse	Methode	Beschreibung
KUNDE	PositionSetzen (x, y)	setzt die Position des Kunden an die Stelle (x | y).
KASSE	PositionSetzen (x, y)	setzt die Position der Kasse an die Stelle (x | y).
KASSE	KundeSetzen (kunde)	ordnet den zu bedienenden Kunden der Kasse zu und positioniert ihn dabei in das Kassensymbol.

Ergänzen der Klassendiagramme und Methodensyntax

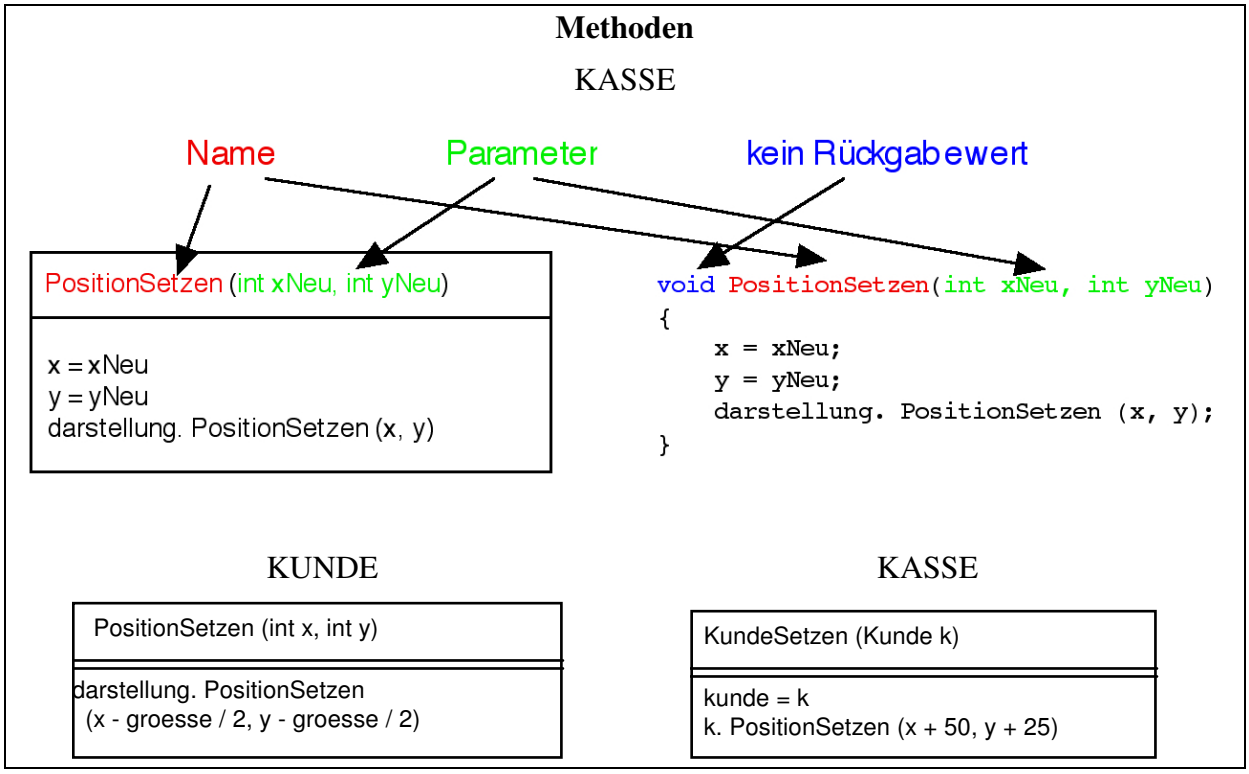
Die gefundenen Methoden werden im Klassendiagramm der Klassen KUNDE und KASSE eingetragen.

Für die Methoden wird jeweils ein Struktogramm erstellt. Dabei müssen auch die Datentypen der verwendeten Parameter (und eventuell des Rückgabewerts) angegeben werden. Es empfiehlt sich, bereits hier die Zielsprachensyntax zu verwenden. Allerdings benötigt eine Methode mit dem Rückgabewert `void` keine besondere Angabe im Struktogramm. Die Behand-

lung des Rückgabetyps kann daher bis zur Einführung der Methodensyntax verschoben werden. Für die Schüler ist die Unterscheidung zwischen Methoden mit und ohne Rückgabewert nicht prinzipiell neu, da schon bei Karol Methoden verwendet wurden, die keinen Wert zurückmelden (z. B. Schritt, LinksDrehen) und solche, die ein Ergebnis melden (IstWand, IstZiegel).

Der Hefteintrag wird ab hier schrittweise entwickelt.

Hefteintrag



Hinweis: Die DIN-Norm für Struktogramme beschreibt nur den eigentlichen Algorithmus, nicht aber die Vereinbarungen. Der Doppelstrich trennt deutlich zwischen diesen beiden Teilen und wird in diesem Beispiel durchgehend verwendet.

Für eine der Methoden der Klasse KASSE, z. B. `PositionSetzen()` wird beispielhaft die Java-Notation entwickelt. In dieser Methode wird mithilfe der Punktschreibweise auch eine Methode des Darstellungsobjekts aufgerufen: `darstellung.PositionSetzen(x,y)`. Dabei kann nochmals thematisiert werden, dass sich die Koordinaten auf die linke, obere Ecke beziehen.

In der Methode `PositionSetzen()` der Klasse KUNDE ist es sinnvoll, dass die Position des Mittelpunkts übergeben wird; beim Aufruf von `darstellung.PositionSetzen()` muss dann innerhalb der Methode umgerechnet werden, da die übergebenen Parameter von `PositionSetzen()` der Klasse KREIS sich auf die linke obere Ecke des umgebenden Quadrats beziehen.

Die Schüler müssen wissen, wo innerhalb der Klassenbeschreibung die Methodenbeschreibungen zu stehen haben. Java erlaubt hier einige Freiheiten, es empfiehlt sich aber, die Reihenfolge „Attribute – Konstruktor(en) – weitere Methoden“ als verbindlich festzulegen.

Arbeit am Computer

Die Schüler fügen die gemeinsam erstellte Methode in den Quelltext ein und testen sie, wobei sie auch Attributwerte mit dem Objektinspektor einsehen. Anschließend werden die anderen Methoden der Klassen von den Schülern selbständig in Java übertragen, eingegeben und getestet.

Zusatz: Was passiert beim Aufruf von `PositionSetzen()` einer Kasse, an der ein Kunde bedient wird? (Hinweis: Es wird nur das Darstellungssymbol der Kasse (Rechteck) verschoben, das Darstellungssymbol des Kunden (Kreis) behält seine Position.)

Anmerkung: Dieser Methodenaufruf kann z. B. in der Designphase nötig sein, ist aber auch denkbar, wenn das Projekt in einem Browserfenster veränderlicher Größe dargestellt werden soll.

Hausaufgabe

Wie könnte das oben im Zusatz angesprochene Problem der Methode `PositionSetzen()` in Klasse KASSE beseitigt werden? Worauf muss dabei geachtet werden?

5. Zustände (5. und 6. Doppelstunde)

In der ersten Doppelstunde dieser Unterrichtseinheit stellen die Schüler fest, dass es verschiedene Werte eines Attributs notwendig machen, auf Ereignisse (Methodenaufrufe) verschieden zu reagieren: Das Objekt ist in unterschiedlichen Zuständen. Sie lernen, das Zustandsübergangsdiagramm und die Zustandsübergangstabelle als geeignetes Mittel zur graphischen Darstellung von Zuständen und Übergängen anzuwenden, und erkennen die Wertzuweisung als die konkrete Stelle des Zustandsübergangs. Sie setzen das Zustandsübergangsdiagramm in eine einfache algorithmische Beschreibung um.

In der zweiten Doppelstunde vertiefen die Schüler ihre Kenntnisse durch die Untersuchung komplexerer Situationen und lernen Attribute von Aufzählungstypen als einfache Möglichkeit zur Darstellung von Zuständen kennen.

Impuls

In der Hausaufgabe haben die Schüler Lösungsvorschläge für das Problem im Zusammenhang mit der Methode `PositionSetzen()` der Klasse KASSE entworfen. Eine vorgeschlagene Lösung ist sicher das Einfügen von `kunde.PositionSetzen(x + 50, y + 25);` als letzte Zeile. Ein Test dieser Lösung führt zu einer Ausnahmesituation und einem Abbruch des Methodenaufrufs, falls das Attribut `kunde` den Wert `null` hat.

Zustände des Objekts kasse1

Die Schüler erkennen als Absturzursache, dass das Attribut `kunde` den Wert `null` hat, d. h., es existiert kein Adressat für die Botschaft „PositionSetzen“ an den Kunden. Beim Empfang der Botschaft „PositionSetzen“ an die Kasse ist es notwendig, zwischen den Zuständen „mitKunde“ und „ohneKunde“ zu unterscheiden. Die Lehrkraft stellt das Zustandsübergangsdiagramm als Möglichkeit zur Visualisierung der Zustände und ihrer Übergänge (einschließlich aller notwendigen Bedingungen) vor. Dabei wird mit der Symbolisierung der beiden Zustände durch Kreise begonnen. Schrittweise wird nun an der Tafel erarbeitet, was die Methode `PositionSetzen()` der Klasse KASSE in jedem der beiden Zustände zu leisten hat und wie diese Aufgaben im Zustandsübergangsdiagramm notiert werden – in untenstehender Graphik durch die beiden Übergangspfeile oberhalb der Zustände. Dabei wird auch angesprochen, dass bei der Verwendung von Zustandsübergangsdiagrammen statt von „Botschaften“ auch von „Ereignissen“ gesprochen wird.

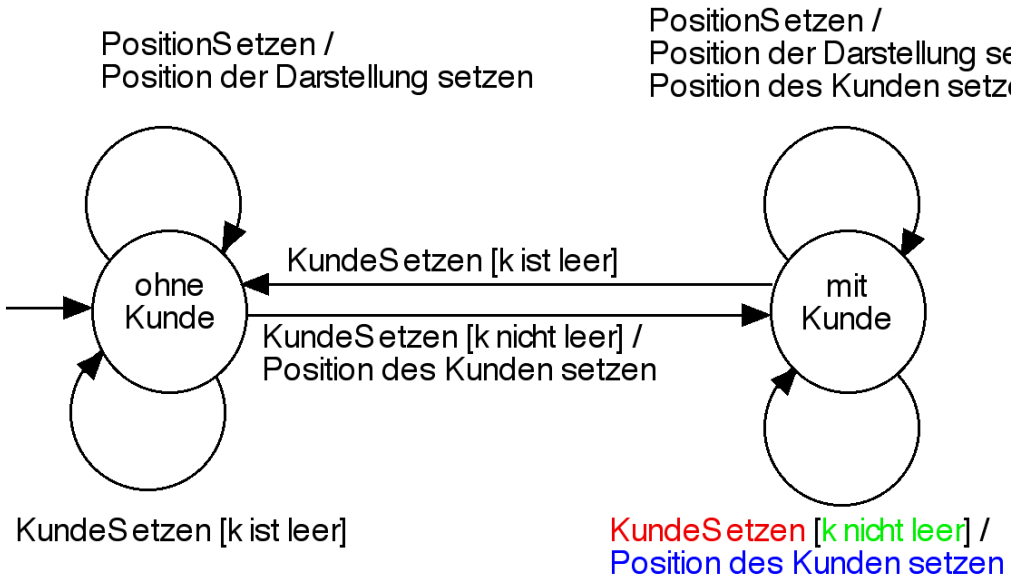
Da der Übergang zwischen den beiden Zuständen durch die Botschaft „KundeSetzen“ ausgelöst wird, bietet es sich an, den zugehörigen Methodennamen in das Diagramm mit aufzunehmen. Hier muss noch die Notation von Bedingungen angegeben werden, da der Zielzustand nach Empfang der Botschaft „KundeSetzen“ vom Wert des übergebenen Parameters „k“ abhängt: hat der Parameter den Wert „null“ (leere Referenz), so ist anschließend kein Kunde mehr an der Kasse; jeder andere Wert bedeutet, dass ein neuer Kunde übergeben wurde und jetzt auf Bedienung wartet. Dabei erkennen die Schüler, dass Zustandsübergänge auch den Wert des Attributs `kunde` verändern. Eine genauere Untersuchung folgt nach der Umsetzung des Zustandsmodells in algorithmische Beschreibung.

Die Möglichkeit, Zustandsübergänge auch in Form von Tabellen (Zustandsübergangstabellen; fakultativer Lehrplaninhalt) anzugeben, kann hier bereits kurz angesprochen werden. Diese Notationsform ist aber nicht so gut geeignet (und eigentlich auch nicht dafür gedacht), Bedingungen abzubilden. Zustandsübergangstabellen sind eher für die Darstellung endlicher Automaten geeignet, wie sie in der Jahrgangsstufe 12 thematisiert werden. Beim nächsten Beispiel ist jedoch exemplarisch auch die Tabelle angegeben.

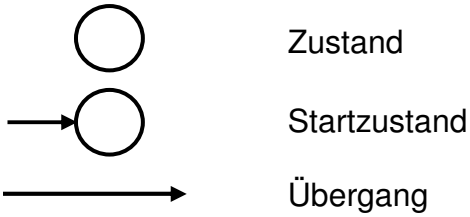
Hefteintrag 1

Zustände

Zustandsmodell für die Behandlung des Kunden an der Kasse:



Symbole:



KundeSetzen: auslösende Aktion
[k nicht leer]: Bedingung
Position des Kunden setzen: ausgelöste Aktion

Abbildung des Zustandsmodells in algorithmische Beschreibung

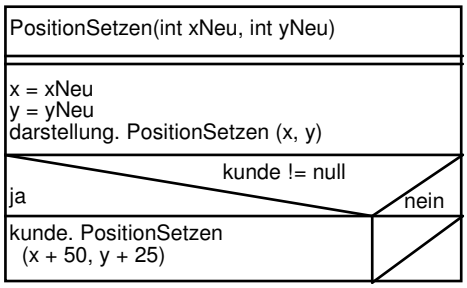
Die Schüler beschreiben die verfeinerte Arbeitsweise der Methode `PositionSetzen()` der Klasse KASSE in Umgangssprache und kommen dabei zu Formulierungen wie „wenn ein Kunde vorhanden ist, dann muss auch dessen Position gesetzt werden“. Ähnlich lässt sich `KundeSetzen()` näher spezifizieren: „Wenn der neue Kunde nicht den Wert null hat, dann muss seine Position gesetzt werden.“ Diese „Wenn-Dann“-Formulierungen sind den Schülern aus der Jahrgangsstufe 7 bekannt. Sie wiederholen die graphische Darstellung der bedingten Anweisung im Struktogramm; neu sind die zugehörige Java-Syntax sowie die Schreibweise der Vergleichsoperatoren.

Auf eine Systematik der Kontrollstrukturen kann hier verzichtet werden, da sich diese Zusammenschau in späteren Einheiten (vgl. Einheit 8) organischer anbietet. Die Schreibweise der Vergleichsoperatoren, insbesondere Gleichheit „==“ und Ungleichheit „!=“ muss mitgeteilt werden.

Hefteintrag 2

Die Methode PositionSetzen() der Klasse KASSE

Algorithmisches Modell:



Java-Notation:

```
void PositionSetzen (int xNeu, int yNeu)
{
    x = xNeu;
    y = yNeu;
    darstellung. PositionSetzen (x, y);
    if (kunde != null)
    {
        kunde. PositionSetzen (x + 50, y + 25);
    }
}
```

Hinweis: Da die Methode `darstellung.PositionSetzen()` für beide Zustände aufgerufen werden muss, steht sie hier bereits vor der bedingten Anweisung.

Arbeit am Computer

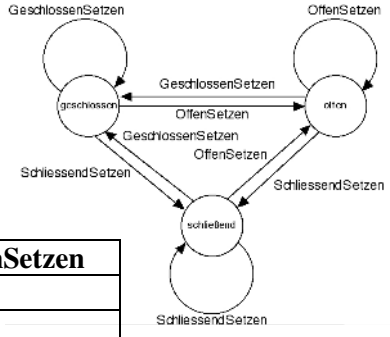
Die Schüler setzen die gefundene Lösung am Computer um und testen sie. Anschließend entwerfen sie ein Struktogramm der erweiterten Methode `KundeSetzen()`, übertragen dieses in Java-Notation, geben es in ihr Projekt ein und testen ebenfalls.

Impuls

Eine kurze Diskussion darüber, ob man sich in einem Supermarkt immer an allen Kassen anstellen kann/darf zeigt, dass eine Kasse drei Zustände hat. Sie ist offen, geschlossen oder wird gerade geschlossen (schließend, d. h. die anstehenden Kunden werden noch bedient, aber neue Kunden dürfen sich nicht mehr anstellen).

Zustandsmodell einer Kasse

Ein erstes, einfaches Modell erlaubt beliebige Übergänge durch die drei Ereignisse OffenSetzen, SchliessendSetzen und GeschlossenSetzen. Dieses Modell könnte auch mithilfe einer Zustandsübergangstabelle angegeben werden.



	OffenSetzen	**SchliessendSetzen**	**GeschlossenSetzen**
offen	offen	schliessend	geschlossen
schliessend	offen	schliessend	geschlossen
geschlossen	offen	schliessend	geschlossen

Eine genauere Diskussion des Ablaufs an der Kasse zeigt, dass dieses einfache Modell, in dem nur die Ereignisse, aber keine Bedingungen angegeben sind, der Situation nicht gerecht wird. Zwar kann eine Kasse jederzeit geöffnet werden – bei schon offenen Kassen ändert sich dann deren Zustand nicht –, eine offene Kasse wird jedoch nicht einfach geschlossen, sie wird nur in den Zustand schließend gesetzt. Erst wenn der letzte Kunde bedient ist, wird sie geschlossen. Das Ereignis „BedienvorgangAbschliessen“ setzt dann die Kasse in den Zustand „geschlossen“. Um die Situation beschreiben zu können, muss das neue Ereignis „BedienvorgangAbschliessen“ in die Betrachtung aufgenommen werden. Die Zustandsübergangstabelle ist für diese Situation eigentlich nicht mehr vorgesehen; möglich wäre, für das Ereignis „BedienvorgangAbschliessen“ zwei Spalten vorzusehen, je eine, wenn die Bedingung erfüllt bzw. nicht erfüllt ist.

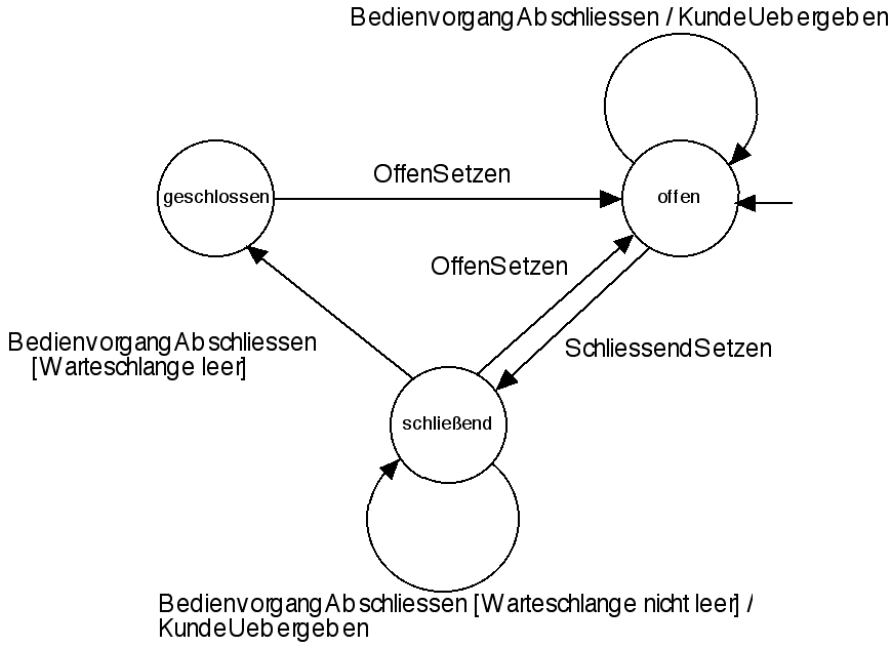
	OffenSetzen	**SchliessendSetzen**	**BedienvorgangAbschliessen [Kunde vorhanden]**	**BedienvorgangAbschliessen [kein Kunde]**
offen	offen	schließend	offen	offen
schließend	offen	schliessend	schliessend	geschlossen
geschlossen	offen			

Ein weiterer zu berücksichtigender Aspekt für die Umsetzung am Computer entsteht durch die notwendige Interaktion mit der zur Kasse gehörenden Warteschlange. Diese wird zwar bei der Modellierung berücksichtigt, momentan aber bei der Implementierung noch ausgespart; entsprechende Methodenaufrufe werden nur als Kommentare geschrieben. Diese offenen Posten dienen als Einstieg in die folgende Einheit „6. Einfache Warteschlange“.

Hefteintrag 3

Ablauf an der Kasse

Zustandsmodell:



Umsetzungsstrategie / Aufzählungstypen

Da z. B. das Ereignis „BedienvorgangAbschliessen“ in verschiedenen Zuständen verschiedene Aktionen auslöst, muss man bei der Umsetzung die zugehörige Methode den aktuellen Zustand bestimmen können. Das bedeutet, dass das KASSEN-Objekt um ein Attribut erweitert wird, das den Zustand repräsentiert. Eine kurze Diskussion zeigt, dass Repräsentationen

der Zustände durch Texte oder Zahlen möglich sind, dies das Programm aber oft unübersichtlich macht und vor allem durch evtl. undefinierte Werte fehleranfällig ist. Benötigt wird ein Attribut mit wenigen, festgelegten Werten; diese Situation kann mithilfe von Aufzählungstypen gelöst werden. Viele Programmiersprachen, Java ab Version 5, kennen diese Funktionalität.

Hier bietet es sich an, zurückgreifend auf die Überlegung der letzen Stunde, eine allgemeine Strategie zur Umsetzung des Zustandsmodells in ein algorithmisches Modell zu entwickeln.

Hefteintrag 2

Aufzählungstypen in Java:

Aufzählungstypen bieten sich zur Beschreibung von Zuständen an.

In Java können sie innerhalb einer Klasse oder als eigene Klasse vereinbart werden.

In die Beschreibung der Klasse KASSE wird vor den Attributvereinbarungen eingefügt:

```
enum Zustaende {offen, geschlossen, schliessend};
```

Bei den Attributen wird ergänzt

```
Zustaende zustand;
```

Jeder Wert wird mittels Punktnotation angesprochen, z. B. `Zustaende.offen`.

Eine Wertzuweisung lautet z. B.: `zustand = Zustaende.offen;`

Umsetzungsstrategie für Zustandsmodelle:

- Zustände werden bei Bedarf durch Aufzählungswerte dargestellt.
- Auslösende und ausgelöste Ereignisse werden durch Methoden realisiert.
- In den Methoden wird zuerst durch bedingte Anweisungen nach den möglichen Zuständen unterschieden; Bedingungen werden ebenfalls durch bedingte Anweisungen umgesetzt.

Die Methode BedienvorgangAbschliessen () der Klasse KASSE

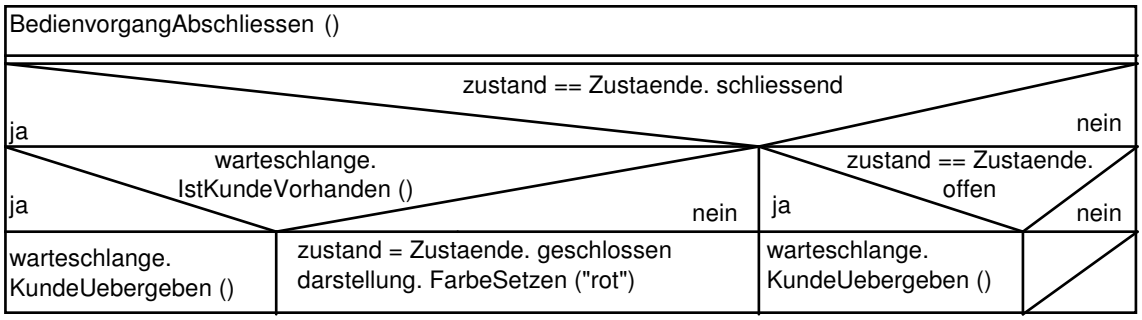
Wenn die Kasse geschlossen ist, löst das Ereignis „BedienvorgangAbschliessen“ keine Aktion aus. Daher wird es im Zustandsübergangsdiagramm nicht berücksichtigt. Bei der Umsetzung in eine Methode muss dieser Situation aber Rechnung getragen werden.

Der folgende Hefteintrag zeigt ein mögliches Ergebnis der Erarbeitung der Methode `BedienvorgangAbschliessen()`. Hier wird insbesondere deutlich, dass bei jedem Zustandsübergang der Wert des Attributs `zustand` entsprechend verändert werden muss. Dies führt zur Erkenntnis, dass für das Objekt der Zustandsübergang genau an der Stelle der Wertzuweisung erfolgt. Das kann auch bei den in der letzen Stunde behandelten Zustandsübergängen verifiziert werden.

Bei der Umsetzung in die Programmiersprache muss nochmals kurz wiederholt werden, dass in dem hier verwendeten Modell jeder Kasse eine eigene Warteschlange zugeordnet ist; diese wird später von der Kasse referenziert. Die Teile, die sich auf die Warteschlangeninteraktion beziehen, werden vorläufig informell durch Kommentare beschrieben. Die genaue Semantik für das Holen eines Kunden aus der Warteschlange wird erst in der nächsten Einheit festgelegt. Die Kasse holt einen Kunden, indem die Methode `KundeUebergeben()` der Warteschlange aufgerufen wird.

Hefteintrag 3

Die Methode BedienvorgangAbschliessen :



Arbeit am Computer

Die Schüler ergänzen den Aufzählungstyp und das Attribut `zustand`. Dabei ist insbesondere darauf zu achten, dass der Startwert (Startzustand „offen“) im Konstruktor entsprechend gesetzt wird.

Anschließend werden die Methoden `OffenSetzen()` (Darstellungssymbol wird grün), `SchliessendSetzen()` (Darstellungssymbol wird gelb) und `BedienvorgangAbschliessen()` (vgl. Hefteintrag) umgesetzt. Ein Test der neuen Methoden sollte – soweit schon möglich – folgen. Die Schüler beobachten dabei die Farbveränderungen der Darstellungsrechtecke als Folge der Zustandsübergänge; mit dem Objektinspektor kann der Wert des Attributs `zustand` auch direkt überprüft werden.

Hausaufgabe

Die Schüler analysieren den Ablauf an den Kassen. Kassen und Kunden werden dabei zweckmäßigerweise durch farbige Papiersymbole oder Spielfiguren repräsentiert, um Abläufe nachvollziehen zu können. Speziell sind folgende Fragen zu beantworten: „Was geschieht beim Anstellen? Was geschieht, wenn ein Kunde an die Kasse geht?“ Weiter entwerfen die Schüler ein Klassendiagramm für die neu zu erstellende Klasse WARTESCHLANGE (mit den notwendigen Attributen und Methoden), die eine Warteschlange vor einer Kasse verwaltet.

6. Einfache Warteschlange (7. und 8. Doppelstunde)

In dieser Unterrichtseinheit legen die Schüler die Grundfunktionalität einer Warteschlange fest und realisieren diese in einem ersten Ansatz mit maximal zwei Kunden. Dabei lernen sie lokale Attribute (Variable) kennen und erfahren, wie Methoden Werte zurückliefern können.

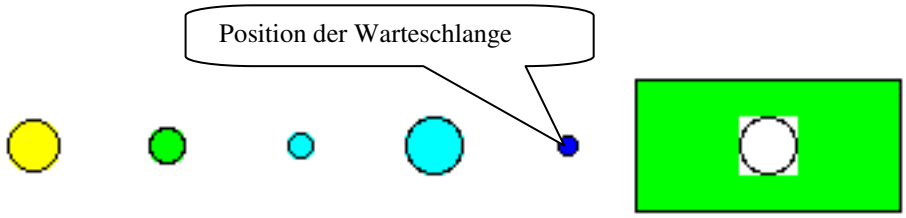
Impuls

Die Kasse benötigt für ihre Arbeit eine Warteschlange, aus der sie Kunden holen kann.

Eine einfache Warteschlange

Ausgehend von der Implementierung der Methode `BedienvorgangAbschliessen()` der letzten Stunden und den Überlegungen der Schüler in der Hausaufgabe wird zuerst die minimal benötigte Funktionalität eines Objekts der Klasse WARTESCHLANGE festgelegt. Vorüberlegungen der Schüler oder Beispiele mit konkreten Objekten (im Sinn der Hausaufgabe) zeigen, dass die Warteschlange nach jeder Veränderung und dem Setzen aller Kundenpositionen neu gezeichnet werden muss. Dazu benötigt die Warteschlange Information über die eigene Position und den Abstand der Kunden in der Warteschlange; sinnvoll ist hier, die x-Koordinate für das rechte Ende der Warteschlange und die y-Koordinate der Kunden-

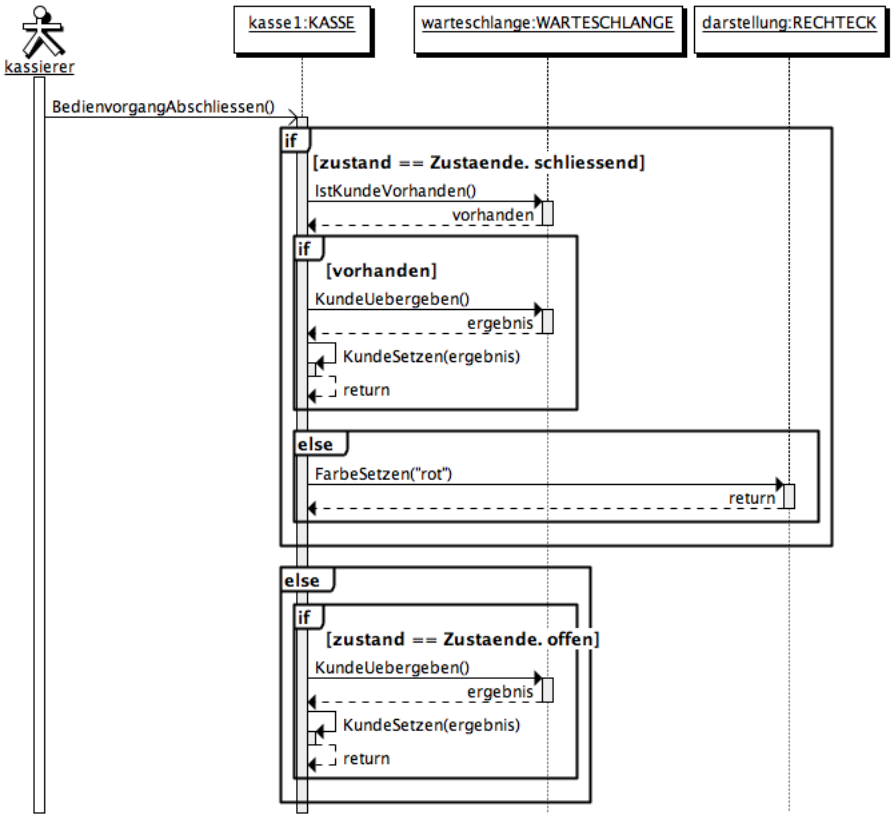
mittelpunkte als Attribute der Warteschlange einzuführen, da die Kunden „an der Kasse“ anstehen. Ein „Kundenabstand“ von 50 Punkten ergibt ein optisch ansprechendes Bild.



Aus den Vorarbeiten wird dann unter der vereinfachenden Vorgabe, dass nur zwei Kunden verwaltet werden, das Klassendiagramm erarbeitet. Die Ablaufdetails der Methoden werden durch Struktogramme näher beschrieben. Dabei muss auf die Rückgabe des Ergebniswertes bei den Methoden `IstKundeVorhanden()` und `KundeUebergeben()` eingegangen werden. Dafür benötigt `KundeUebergeben()` ein lokales Attribut (lokale Variable, typischer Name ist „resultat“ oder kurz „res“) zur Zwischenspeicherung. Hier muss auch die Angabe eines Ergebniswertes für eine Methode thematisiert werden (Ergebnistyp im Methodenkopf, return-Anweisung), da `KundeUebergeben()` die erste Methode mit Ergebniswert ist.

Ablaufdarstellung durch Interaktionsdiagramme

Der Aufruf der Methode `BedienvorgangAbschliessen()` zieht mittlerweile eine ganze Reihe weiterer Methodenaufrufe nach sich. Um diese Methodenaufrufe und damit auch die beteiligten Objekte übersichtlich darzustellen, eignen sich Sequenzdiagramme (eine Form der Interaktionsdiagramme) sehr gut. Alle beteiligten Objekte werden mit Lebenslinien nach unten (Zeitachse nach unten) dargestellt. Die Methodenaufrufe und Punkte der Rückkehr werden durch waagrechte Linien dargestellt.



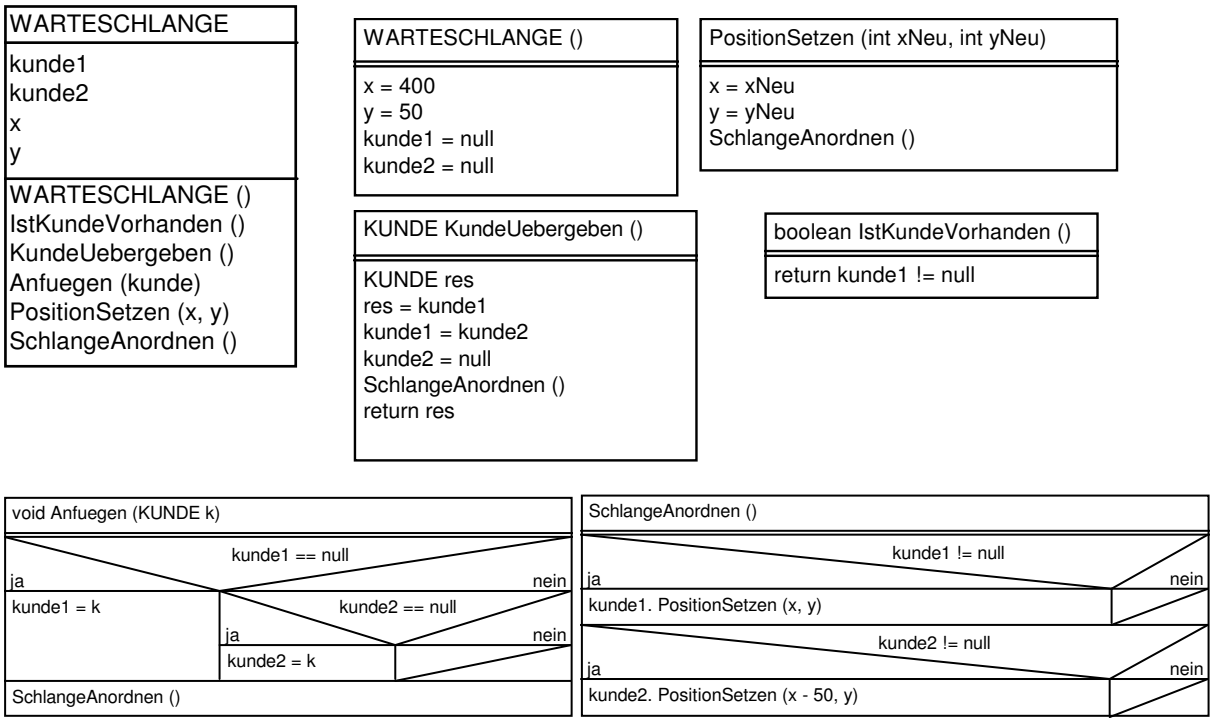
Hefteintrag

Die Klasse WARTESCHLANGE

Ein Objekt der Klasse WARTESCHLANGE muss folgende Leistungen erbringen:

IstKundeVorhanden ()	liefert WAHR, wenn in der Warteschlange noch Kunden anstehen.
KundeUebergeben ()	liefert den ersten Kunden in der Warteschlange, alle anderen rücken nach vorn.
Anfuegen (kunde)	stellt den angegebenen Kunden hinten an die Warteschlange an.
PositionSetzen (x, y)	setzt die Warteschlange an die übergebene Position und passt die Position der anstehenden Kunden an.
SchlangeAnordnen ()	zeichnet die Schlange durch korrektes Positionieren aller Kunden.

Details:



Methoden mit Rückgabewert:

Hat eine Methode einen Rückgabewert, so wird statt der Angabe `void` im Methodenkopf der Typ des Rückgabewerts angegeben. Mit der Anweisung `return` wird der Rückgabewert angegeben.

Arbeit am Computer

Die Schüler setzen die Überlegungen zur Klasse WARTESCHLANGE schrittweise am Computer um. Nach dem Konstruktor sollten zuerst die Methoden `SchlangeAnordnen()` und `Anfuegen()` implementiert werden, um neben dem Test mit dem Objektinspektor auch ein sichtbares Feedback zu erhalten. Es folgen mit jeweiligem Test die Methoden `KundeUebergeben()`, `PositionSetzen()` und `IstKundeVorhanden()`.

Anschließend wird der Kasse eine Warteschlange zugeordnet (Einführen eines Attributs für die Referenz, Besetzen des neuen Attributs im Konstruktor). In der Methode `BedienvorgangAbschliessen()` der Klasse KASSE werden die vorläufigen Kommentare durch Methodenaufrufe der Warteschlange ersetzt.

Beim Test der Methoden stellt sich heraus, dass der bediente Kunde anschließend aus der Darstellung entfernt werden muss. Dazu muss die Klasse KUNDE noch um eine Methode `Entfernen()` erweitert werden, die sich auf `Entfernen()` der Klasse KREIS abstützt.

Hausaufgabe

Was muss in der Klasse WARTESCHLANGE geändert werden, wenn auf diese Art vier Kunden pro Warteschlange verwaltet werden sollen? Lässt sich dieser Ansatz problemlos auf 10, 20 oder 30 Kunden erweitern?

Anmerkungen

1. Auf die Methode `PositionSetzen()` der Warteschlange kann verzichtet werden, wenn der Warteschlange die endgültige Position bereits beim Erzeugen (Parameter im Konstruktor) zugewiesen wird.
2. Auf die Methode `IstKundeVorhanden()` kann verzichtet werden, wenn der Rückgabewert `null` von `KundeUebergeben()` als „Warteschlange ist leer“ interpretiert wird.

7. Felder (9. und 10. Doppelstunde)

Die Schüler lernen Felder als eine Möglichkeit kennen, eine beliebig große, aber feste Zahl von Objektreferenzen in einem Attribut zu verwalten. Mit den Feldern steht ihnen ein erstes Mittel zur Verfügung, die 1:n-Beziehung umzusetzen.

Zur Bearbeitung der Felder werden verschiedene Formen der Wiederholung benötigt, so dass die Grundelemente von Algorithmen in dieser Einheit behandelt werden.

Impuls

In einer Warteschlange sind mehr als nur zwei Kunden.

Der Datentyp Feld

In der Hausaufgabe haben die Schüler erkannt, dass eine Vergrößerung der Warteschlange durch einfache Hinzunahme weiterer Attribute zwar prinzipiell möglich ist, aber sehr schnell an praktische Grenzen stößt. Benötigt wird ein Attribut, das viele Objektreferenzen verwalten kann. Diese Referenzen müssen aber geordnet (Reihenfolge der Warteschlange) und einzeln ansprechbar sein.

Mit der Klasse Feld steht in den meisten Programmiersprachen ein solcher Datentyp zur Verfügung.

Hefteintrag 1

Felder

Zur Speicherung vieler Objektreferenzen in einem Attribut gibt es spezielle Klassen, die Felder (engl. array). Ein Feld ist eine Zusammenfassung vieler Einzelwerte von gleicher Klasse zu einem Ganzen. Die einzelnen (Feld-)Elemente sind durchnummeriert. Diese Nummer nennt man Index.

Felder in Java

Die Ausprägung der Felder in Java muss die Lehrkraft mitteilen. Wichtige Aspekte sind:

– Die einzelnen Referenzen (Elemente) sind beginnend bei 0 durchnummeriert.
– Die Angabe des Index erfolgt in eckigen Klammern.
– Bei der Instanzierung eines Feldes muss die Anzahl der Elemente festgelegt werden.
– Attribute der Klasse Feld werden durch [] hinter der Klasse der Elemente vereinbart.
– Die Anzahl der Elemente eines Feldes kann aus dem Attribut `length` gelesen werden.

Die Syntax für die Verwendung von Feldern kann beispielhaft anhand der Klasse WARTESCHLANGE eingeführt werden. Dabei werden die bekannten Methoden systematisch durchleuchtet, notwendige Ergänzungen werden teils als Struktogramm, teils als Quelltext notiert.

Bei den Ergänzungen der Methoden der Klasse WARTESCHLANGE, in denen auf alle Feldelemente zugegriffen wird (Konstruktor, `SchlangeAnordnen()`, `KundeUebergeben()`, `Anfuegen()`), werden auch verschiedene Formen der Wiederholung thematisiert. Insbesondere wird die Zählwiederholung neu eingeführt, mit der in der Regel auf alle Feldelemente der Reihe nach zugegriffen wird. Die Methode `Anfuegen()` legt zusätzlich die Verwendung der Wiederholung mit Eingangsbedingung nahe.

Hefteintrag 2

Felder in Java:

Vereinbarung von Attributen der Klasse Feld:

```
KUNDE [] kunden;
```

Instanzieren eines Attributs der Klasse Feld mit 30 Elementen:

```
kunden = new KUNDE [30];
```

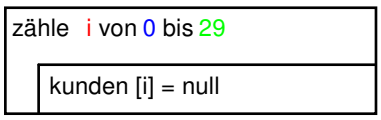
Zugriff auf alle Feldelemente:

```
kunden [0] = null;
kunden [1] = null;
          :
kunden [29] = null;
```

} bringt keinen Vorteil gegenüber Einzelattributen

Die Zählwiederholung löst das Problem (Java-Code):

```
for(int i = 0; i <= 29; i++)
{
    kunden [i] = null;
}
```



i Zählvariable
0 Startwert
i < = 29 Endwert
i++ Erhöhung um 1

Elementanzahl:

Die Klasse Feld hat das Attribut length, mit dem die Anzahl der Feldelemente abgefragt werden kann.

```
for(int i = 0; i < kunden.length; i++)
{
    kunden [i] = null;
}
```

Arbeit am Computer

Die Schüler ersetzen in der Klasse WARTESCHLANGE die Attribute `kunde1` und `kunde2` unter Verwendung des Attributs `kunden[]`. Sie ergänzen den Konstruktor. Bei der Übersetzung werden alle noch vorhandenen Vorkommen der Attribute `kunde1` und `kunde2` auskommentiert, so dass das neue Attribut und der Konstruktor mithilfe des Objektinspektors getestet werden können.

Anschließend werden die Methoden `IstKundeVorhanden()` und `SchlangeAnordnen()` von den Schülern angepasst.

Hausaufgabe

Die Schüler führen Vorüberlegungen zum Einfügen neuer Kunden in die Warteschlange durch: Wie kann erkannt werden, ob noch ein Platz frei ist? Welches Feldelement nimmt den neuen Kunden auf, wenn bereits 29 Kunden in der Schlange vorhanden sind? Wie lässt sich diese Nummer allgemein bestimmen?

Damit die Neuerungen vollständig umgesetzt sind, müssen noch die Methoden `KundeUebergeben()` und `Anfuegen()` der Klasse WARTESCHLANGE angepasst werden.

Die Wiederholung mit Eingangsbedingung

Bei der Anpassung der Methode `KundeUebergeben()` führt die Vorüberlegung mit expliziten Feldelementen sehr schnell auf die Formulierung des Algorithmus mittels Zählwiederholung. Hier kann das Ergebnis sofort in der Syntax der Programmiersprache angegeben werden, da der eigentliche Algorithmus schon entwickelt ist und „nur noch“ eine geeignete Notation gesucht wird.

Für die Methode `Anfuegen()` haben die Schüler in der Hausaufgabe erkannt, dass es notwendig ist, die Feldelemente bei 0 beginnend so lange zu durchlaufen, bis ein freier Platz gefunden ist. Diese Wiederholung wird als Wiederholung mit Eingangsbedingung formuliert; die Zählvariable muss dabei „von Hand“ verwaltet werden. Hier werden die Details am besten schrittweise als Struktogramm ausgebaut und z. B. mit Magnetsymbolen an der Tafel visualisiert. Erst wenn der Algorithmus vollständig verstanden ist, wird er in die Programmiersprache übertragen; dabei wird auch die neu benötigte Syntax (Wiederholung mit Eingangsbedingung) angegeben.

Hefteintrag

KundeUebergeben():

KundeUebergeben () liefert den ersten Kunden in der Warteschlange, alle anderen rücken nach vorn.

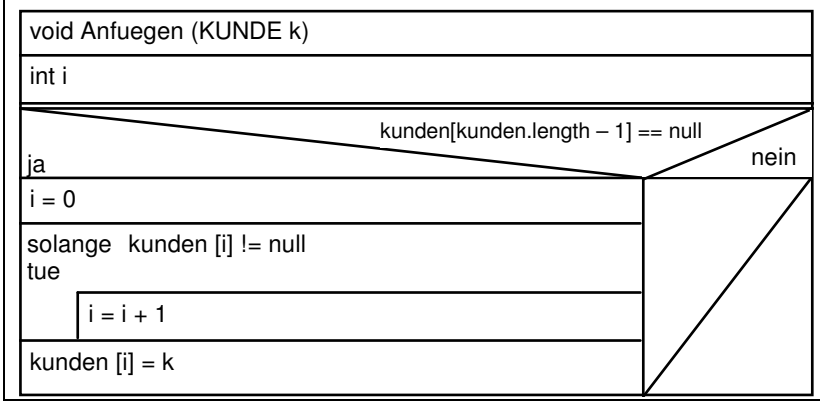
```
res = kunden [0];
kunden [0] = kunden [1];
kunden [1] = kunden [2];
    :
kunden [28] = kunden [29];
kunden [29] = null;
```

kann zusammengefasst werden zu

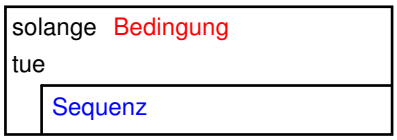
```
res = kunden [0];
for (int i = 0; i < kunden.length - 1; i++)
{
    kunden [i] = kunden [i + 1];
}
kunden [kunden.length - 1] = null;
```

Anfuegen():

Wenn das letzte Element den Wert `null` hat, kann ein neuer Kunde aufgenommen werden. Das erste freie Element wird mit „Wiederholung mit Eingangsbedingung“ gesucht, dort wird der Kunde eingetragen.



Wiederholung mit Eingangsbedingung




```
while (Bedingung)
{
    // zu wiederholende Sequenz
}
```

Arbeit am Computer

Die Schüler ergänzen die fehlenden Methoden der Klasse WARTESCHLANGE und testen diese. Insbesondere überprüfen sie die korrekte Arbeit von `KundeUebergeben()` für eine leere, eine etwas gefüllte und eine volle Warteschlange sowie die Methode `Anfuegen()` ebenfalls für diese drei Fälle. Für diesen Test empfiehlt es sich, die Feldlänge auf z. B. fünf Elemente zu beschränken.

Anschließend testen die Schüler das Zusammenspiel von Kasse und Warteschlange.

Anmerkungen

1. Für die eventuell in der Voreinheit eingeführte Methode `PositionSetzen()` ist keine Anpassung nötig. (Diese Methode ist nicht unbedingt nötig, vgl. Anmerkung 1 in Einheit 6).
2. Auf die Umsetzung der Methode `Anfuegen()` mit der in Java auch möglichen Schreibweise `for (i = 0; kunden[i] != null; i++)` wurde verzichtet, da die Zählwiederholung als eigenständige Struktur und nicht als abkürzende Schreibweise einer besonderen Form der Wiederholung mit Eingangsbedingung aufgefasst wird.
3. Auf die Möglichkeit von Java 5, Felder mittels eines Iterators zu durchlaufen, d. h. `for (KUNDE k: kunden)` wurde verzichtet, da dieses Konstrukt bei Feldern die Zugriffsidee eher verschleiert.
4. Nach dieser Einheit kann eine Doppelstunde zum Thema „collection classes“ eingeschoben werden, die eine allgemeine Implementierung der Enthält-Beziehung erlauben.

7a*. Collection Classes (eingeschobene Doppelstunde)

Die Schüler lernen Sammlungsklassen (engl. collection classes) als nutzbringende Anwendung vorhandener Klassenbibliotheken kennen. Damit steht ihnen ein Mittel zur Verfügung, die Enthält-Beziehung in voller Allgemeinheit umzusetzen. Nach der Erarbeitung der benö-

tigten Semantik muss die Lehrkraft in dieser Stunde die Umsetzung der gewählten Klasse (z. B. ArrayList) mit ihren Detailaspekten weitgehend im Vortragsstil vorstellen.

Hier bietet sich das Durchlaufen mittels Iterator gut an, so dass die Syntax der entsprechenden Wiederholung thematisiert wird (vgl. Einheit 7, Anmerkung 3).

Impuls

Durch die Verwendung von Feldern kann eine Warteschlange zwar sehr viele Kunden verwalten, die Anzahl ist aber trotzdem begrenzt.

Collections

Die Schüler haben erkannt, dass sich mithilfe von Feldern Warteschlangen beliebiger, aber fest vorgegebener Maximalgröße realisieren lassen. Um die Enthält-Beziehung in ihrer allgemeinen Form zu realisieren, wird eine Datenstruktur benötigt, die es wie bei einem Feld erlaubt, einzelne Elemente über ihre Nummerierung anzusprechen, bei der aber die Anzahl der Elemente nicht vorgegeben ist.

Entsprechende Klassen könnten selbst entworfen werden. Da der Bedarf nach dieser Funktionalität sehr hoch ist, sind sie aber in der mitgelieferten Klassenbibliothek jeder gängigen Programmiersprache enthalten. Alle Klassen des angegebenen Funktionsumfangs, mit denen sich die 1:n-Beziehung realisieren lassen, werden mit dem Namen **collection classes** (engl. to collect; sammeln) zusammengefasst.

Es gibt dafür verschiedene Implementierungen, die für unterschiedliche Verwendungszwecke optimiert sind. Manche erlauben ein möglichst schnelles Durchlaufen aller Elemente, manche sind besonders schnell, wenn man nur auf das erste und/oder das letzte Element zugreift. Andere unterstützen ein schnelles Suchen nach einem bestimmten Objekt, wobei es noch einen Unterschied macht, ob man nach dem Suchkriterium sortieren kann oder nicht.

ArrayList

Eine für die Realisierung der Warteschlangen geeignete collection class ist die Klasse ArrayList. Sie erlaubt es, mit den Methoden `get()` und `set()` Objekte wie in einem Feld anzusprechen, kann aber darüber hinaus mit `add()` Objekte einfach anfügen und mit `remove()` entfernen.

Die Lehrkraft stellt die zu verwendende Klasse den Schülern vor. Hier bietet sich eventuell auch ein Arbeitsblatt an, da bei reiner Tafelanschrift der Hefteintrag doch ziemlich umfangreich wird.

Die Klasse ArrayList muss aus dem Paket java. util importiert werden.

`void add (Object obj)`	fügt das angegebene Objekt an der letzten Position an.
`Object get (int index)`	holt das Objekt mit der Indexposition index.
`boolean isEmpty ()`	liefert wahr, wenn kein Objekt gespeichert ist (entspricht Größe 0).
`Object remove (int index)`	entfernt das Objekt an der Position index aus dem Feld und liefert es als Funktionsergebnis. Alle weiter hinten liegenden Objekte rücken automatisch um eine Position nach vorne.
`Object remove (Object obj)`	entfernt das angegebene Objekt aus dem Feld und liefert es als Funktionsergebnis. Alle weiter hinten liegenden Objekte rücken automatisch um eine Position

	nach vorne.
`void set (int index, Object obj)`	fügt das Objekt an der angegebenen Stelle ein; das an dieser Position befindliche Objekt und alle weiter hinten liegenden Objekte rücken automatisch um eine Position nach hinten.
`int size ()`	meldet die Anzahl der gespeicherten Objekte.

Verwendung von collections

Nach dem Vorstellen der Klasse ArrayList (oder einer anderen Collectionklasse) wird nun die Klasse WARTESCHLANGE systematisch überarbeitet. Dabei werden alle Stellen berücksichtigt, an denen das Feld `kunden` verwendet wird, diese Stellen werden umgebaut. Die im Folgenden angeführten Details kommen dabei zur Sprache.

Alle collections können Objekte beliebiger Klassen speichern. Deshalb muss man sowohl beim Vereinbaren des Attributs als auch beim Anlegen des Objekts angeben, von welcher Klasse die gespeicherten Objekte sind. Das geschieht jeweils durch Angabe der Elementklasse hinter dem Namen der collection class in spitzen Klammern.

Attributvereinbarung: `ArrayList<KUNDE> kunden;`

Anlegen des Objekts: `kunden = new ArrayList<KUNDE>();`

Der Compiler kann dann absichern, dass alle mit `set()` oder `add()` eingefügten Objekte von der korrekten Klasse sind. Damit ist auch garantiert, dass alle mit `get()` oder `remove()` geholten Objekte die richtige Klasse haben.

Das Durchlaufen aller Elemente eines Collection-Objekts kann mit der schon bekannten Zählwiederholung durchgeführt werden. Einfacher ist es aber, eine in Java 5 neu eingeführte Form der Wiederholung zu verwenden. Die „Für alle“-Wiederholung durchläuft alle Elemente einer Collection.

```
for(KUNDE k: kunden)
{
    //Für jedes Element zu wiederholende Sequenz
}
```

Der folgende Vergleich zeigt die geänderten Methoden in der Klasse WARTESCHLANGE. Die Änderungen sind hervorgehoben.

```
class WARTESCHLANGE
{
    ArrayList<KUNDE> kunden;
    int x;
    int y;

    WARTESCHLANGE ()
    {
        kunden = new ArrayList<KUNDE> ();




        x = 400;
        y = 50;
    }

    boolean IstKundeVorhanden ()
    {
        return ! kunden. isEmpty();
    }

    KUNDE KundeUebergeben ()
    {
        KUNDE res;
        if (kunden. isEmpty ())
        {
            res = null;
        }
        else
        {
            res = kunden. remove (0);
        }
        SchlangeAnordnen ();
        return res;
    }

    void Anfuegen (KUNDE kunde)
    {
        kunden. add (kunde);









        SchlangeAnordnen ();
    }

    void SchlangeAnordnen ()
    {
        for (KUNDE k: kunden)
        {
            k. PositionSetzen
           (x - kunden. indexOf (k) * 50, y);
        }




    }

    void PositionSetzen(int xNeu, int yNeu)
    {
        x = xNeu;
        y = yNeu;
        SchlangeAnordnen ();
    }
}
```

```
class WARTESCHLANGE
{
    KUNDE [] kunden;
    int x;
    int y;

    WARTESCHLANGE ()
    {
        kunden = new KUNDE [30];
        for (int i = 0; i < kunden. length;
                                        i++)
        {
            kunden [i] = null;
        }
        x = 400;
        y = 50;
    }

    boolean IstKundeVorhanden ()
    {
        return kunden [0] != null;
    }

    KUNDE KundeUebergeben ()
    {
        KUNDE res;
        res = kunden [0];
        for (int i = 0;
             i < kunden. length - 1; i++)
        {
            kunden [i] = kunden [i + 1];
        }
        kunden [kunden. length - 1] = null;
        SchlangeAnordnen ();
        return res;
    }

    void Anfuegen (KUNDE kunde)
    {
        int i;
        if (kunden [kunden. length - 1] ==
                                       null)
        {
            i = 0;
            while (kunden [i] != null)
            {
                i = i + 1;
            }
            kunden [i] = kunde;
        }
        SchlangeAnordnen ();
    }

    void SchlangeAnordnen ()
    {
        for (int i = 0; i < kunden. length;
                                        i++)
        {
            KUNDE k = kunden [i];
            if (k != null)
            {
                k. PositionSetzen
                            (x - i * 50, y);
            }
        }
    }

    void PositionSetzen(int xNeu, int yNeu)
    {
        x = xNeu;
        y = yNeu;
        SchlangeAnordnen ();
     }
```

Hefteintrag

Collection Classes

Objekte von **collection classes** implementieren die Enthält-Beziehung vollständig. Auf die enthaltenen Elemente kann mit einer Indexangabe zugegriffen werden. Hinzufügen und Löschen von Elementen ist mit einfach verwendbaren Methoden möglich. Es können beliebig viele Elemente hinzugefügt werden.
Java bietet im Paket `java.util` verschiedene collection classes an.

Aufgaben

1. Die Klasse WARTESCHLANGE

a) Füge den Import für die Klasse ArrayList ein, vereinbare damit das Attribut `kunden` neu und ändere die Vereinbarung von `kunden` im Konstruktor entsprechend ab. Warum kannst du jetzt auf das `null`-Setzen der Feldelemente verzichten?

b) Die Methode `Anfuegen()` wird wesentlich einfacher, da keine Prüfung mehr benötigt wird, ob noch Platz ist. Auch das Einfügen des neuen Kunden ist viel einfacher. Ändere diese Methode ab.

Ähnliche Vereinfachungen sind auch in den anderen Methoden der Klasse WARTESCHLANGE möglich.

Hinweis:
In der Methode `SchlangeAnordnen()` kann man sich für die Positionsberechnung den Index des aktuellen Kunden mit der Methode `indexOf(k)` beschaffen

```
for (KUNDE k: kunden)
{
    k. PositionSetzen(x – kunden. indexOf(k) * 50, y);
}
```

c) Teste die Umbauten.

8. Der Supermarkt (11. und 12. Doppelstunde)

In dieser Einheit üben und vertiefen die Schüler den Umgang mit Feldern. Sie ergänzen die Klasse KUNDE um eine Methode `Anstellen(KASSEN [] kasse)` sowie den dazu nötigen Hilfsmethoden bei KUNDE, KASSE und WARTESCHLANGE und entwerfen eine Klasse SUPERMARKT, die die Kassen und Warteschlagen erzeugt und eine Methode für das Erzeugen neuer Kunden hat.

Am Ende dieser Einheit bietet sich eine kurze Zusammenfassung der verwendeten Kontrollstrukturen (Arbeitsblatt „Kontrollstrukturen“) an.

Um den Schülern den Überblick zu erleichtern, kann zu Beginn der Überlegungen ein Arbeitsblatt mit dem bisherigen Stand des Projekts ausgeteilt werden (Arbeitsblatt „Einheit 8 Anfangsstand“). Alternativ ist auch ein Arbeitsblatt angegeben, auf dem die Schüler zusätzlich den Stundeninhalt fixieren können (Arbeitsblatt „Einheit 8“).

Impuls

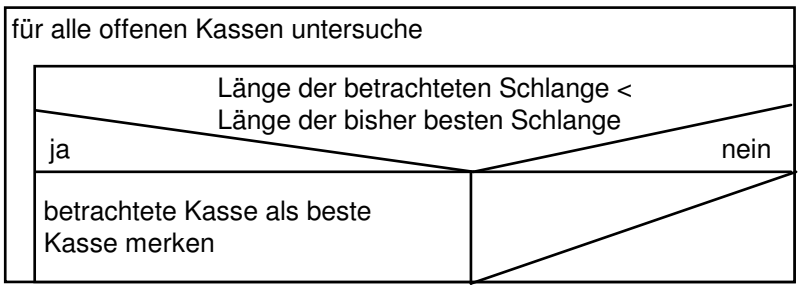
Der Supermarkt hat mehrere Kassen. Wo stellt sich ein Kunde an, der neu in den Kassenbereich kommt?

Die Entscheidungsfindung

Die Schüler wissen aus eigener Erfahrung, dass sich Kunden (trivialer Weise) nur an offenen, nicht aber an geschlossenen oder schließenden Kassen anstellen können und dass es

verschiedene Verhaltensweisen für die Auswahl der Warteschlange gibt: kürzeste Schlange, Schlange mit den wenigsten Artikeln im Warenkorb oder einfach die nächstbeste Schlange. Eine mögliche Strategie ist, sich an der kürzesten Schlange anzustellen.

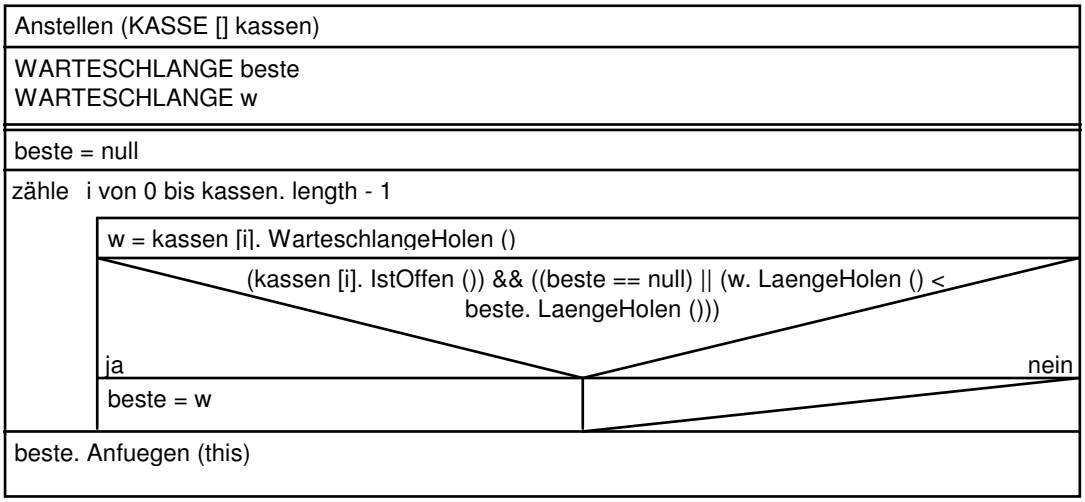
Eine erste, halb informelle Beschreibung einer Methode `Anstellen()` der Klasse KUNDE dient als Gedankenstütze für die weiteren Überlegungen.



Diese Überlegung wird schrittweise bis zur vollständig formalen Beschreibung verfeinert. Die Tatsache, dass es beim Start der Methode noch keine „beste Warteschlange“ gibt, kann durch den Wert null für die beste Schlange dargestellt werden.

Der Kunde muss am Ende „sich selbst“ zum Anfügen an die Warteschlange übergeben. Jede objektorientierte Sprache hat einen reservierten Namen, mit dem ein Objekt eine Referenz auf sich selbst erhält. In Java heißt dieser Bezeichner `this`.

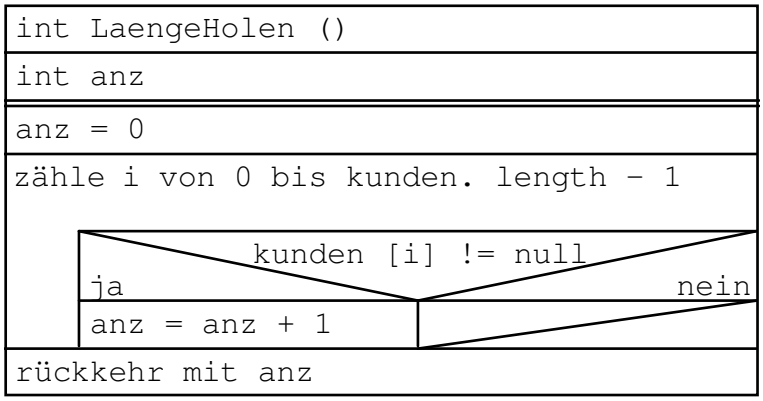
Folgendes Struktogramm gibt ein mögliches Ende der Überlegungen wieder. Mit diesem Struktogramm beginnt der Hefteintrag.



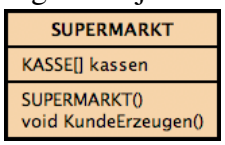
Aus diesem Ansatz ergeben sich eine Reihe noch benötigter Methoden bei verschiedenen Klassen.

- Ein Kunde muss erkennen können, ob eine Kasse offen ist. Die Klasse KASSE benötigt daher eine Abfragemethode `IstOffen()`, die wahr liefert, wenn die Kasse offen ist.
- Die Klasse WARTESCHLANGE benötigt eine Methode `LaengeHolen()`, die die Anzahl der Kunden in der Warteschlange angibt.
- Da der Kunde nur Zugriff auf die Kassen hat, benötigt die Klasse KASSE eine Methode `WarteschlangeHolen()`, mit der der Kunde die zu dieser Kasse gehörende Warteschlange abfragen kann.
- Zur effizienten Untersuchung aller Kassen sollten diese in einem Feld `kassen` organisiert werden.

Die erste Methode `WarteschlangeHolen()` gibt einfach den entsprechenden Attributwert zurück. Die Methode `IstOffen()` gibt das Ergebnis des Vergleichs auf `Zustand.offen` zurück. `WARTESCHLANGE.LaengeHolen()` muss die Warteschlange durchlaufen und dabei die Kunden zählen.



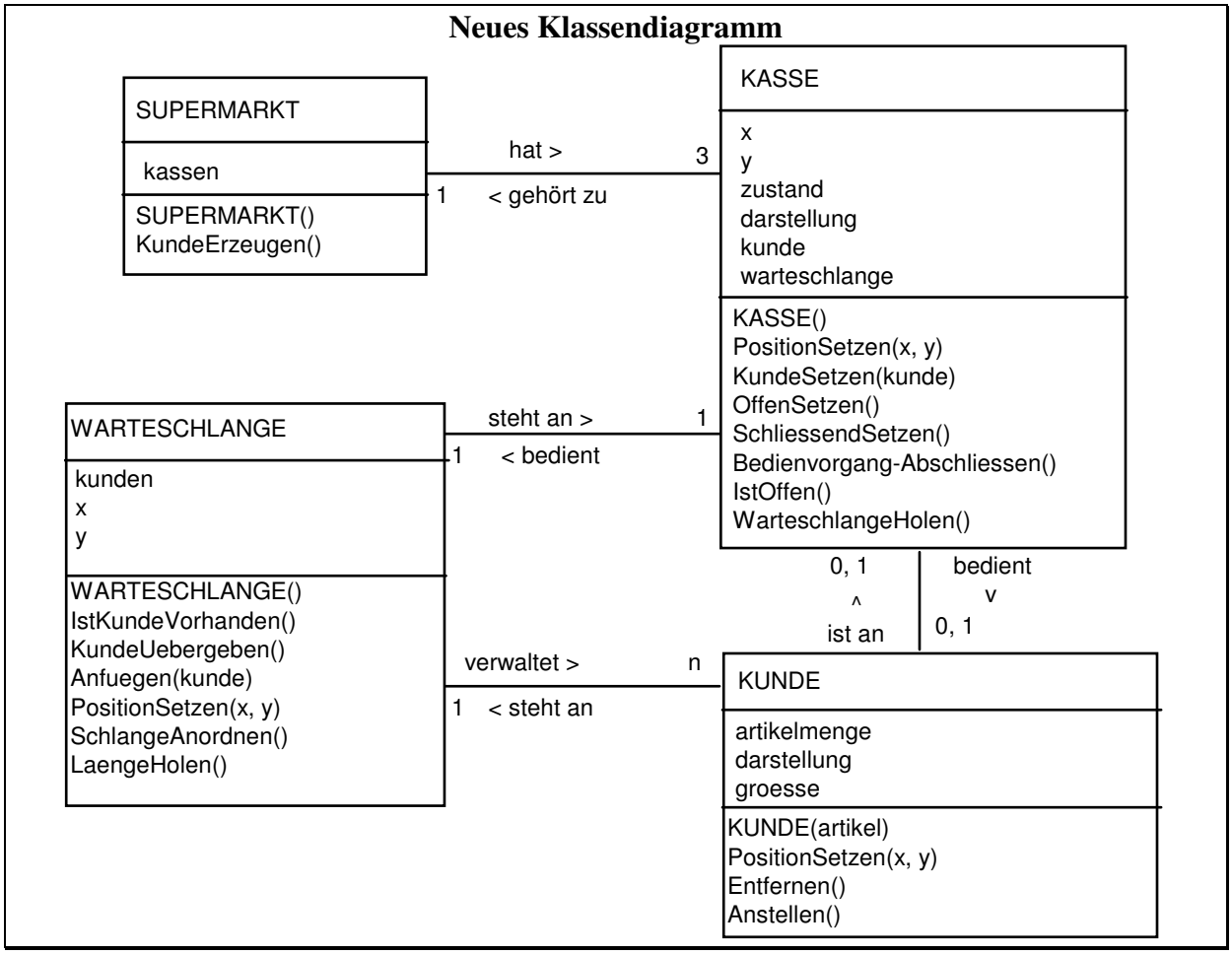
Die Klasse SUPERMARKT



Der Aufbau der einzelnen Kassen und Warteschlangen mit drei einzeln erzeugten Objekten der Klasse KASSE ist mühsam. Die Schüler ordnen die drei benötigten Kassen (und deren Warteschlagen) in der gewünschten Form an und notieren dabei die notwendigen Methodenaufrufe (z. B. im Codepad-Fenster von BlueJ). Anschließend wird daraus die Klasse SUPERMARKT mit dem benötigten Attribut `kassen` entworfen. Im Konstruktor dieser Klasse wird der Markt aufgebaut. Die Klasse SUPERMARKT erhält auch eine Methode `KundeErzeugen()`, die einen neuen Kunden erzeugt und diesen sich anstellen lässt.

Hefteintrag

Der Hefteintrag ergibt sich hier im Wesentlichen aus der Mitschrift der Überlegungen und der Entwicklungsschritte für die neue Funktionalität. Daher wird er die oben genannten Struktogramme und die Angabe der neuen Methoden für die Klassen KUNDE, KASSE und WARTESCHLANGE enthalten. Als Zusammenfassung bietet sich ein Klassendiagramm der neuen Situation an.



Aufgaben

1. Entscheidungsfindung

a) Erweitere die Klasse WARTESCHLANGE um die Methode `LaengeHolen()`.

b) Erzeuge ein Objekt der Klasse WARTESCHLANGE und mehrere Kunden, die dort anstehen. Teste die neue Methode `LaengeHolen()`.

c) Erweitere die Klasse KASSE um die Methode `HoleWarteschlange()`.

d) Ergänze die Klasse KUNDE um die Methode `Anstellen()`.

2. Supermarkt

a) Erzeuge drei Objekte der Klasse KASSE und positioniere sie so, dass die gewünschte Anordnung entsteht. Merke dir die Positionen (z. B. im Objektinspektor anschauen).

b) Entwirf die Klasse SUPERMARKT, die die drei Kassen mit ihren Warteschlangen erzeugt und korrekt positioniert.

c) Ergänze in der Klasse SUPERMARKT die Methode `KundeErzeugen()`.

9. Automatischer Ablauf (13. Doppelstunde)

In dieser Einheit ergänzen die Schüler den Supermarkt um ein automatisches Zeitverhalten. Dazu lernen sie eine Möglichkeit für die automatische Generierung von Zeitimpulsen kennen und entwickeln Verfahren, diese zu nutzen.

Impuls

Die Teile des Supermarkts sind aufgebaut. Jetzt wird ein automatischer Ablauf der Vorgänge benötigt.

Wo wird Zeit benötigt?

Die Schüler untersuchen die vorhandenen Klassen und finden zwei Stellen, an denen Methoden nach einer vorgegebenen Zeit aufgerufen werden müssen.

- In der Klasse SUPERMARKT muss jeweils nach der mittleren Wartezeit ein neuer Kunde erzeugt werden.
- In der Klasse KASSE muss die Methode `BedienvorgangAbschliessen()` aufgerufen werden, wenn die Bedienung des aktuellen Kunden abgeschlossen ist. Die Zeit für das Bedienen eines Kunden setzt sich zusammen aus einem variablen Anteil für das Registrieren der Waren im Korb und einen festen Anteil für das Abkassieren.

Der Taktgeber

In einer anschließenden Diskussion erkennen die Schüler, dass eine individuelle Zeitsteuerung für jedes einzelne Objekt Nachteile hat, wenn die Simulationsgeschwindigkeit variabel gestaltet werden soll (in Realzeit, um Details zu beobachten und im Zeitraffer, um langfristige Vorgänge schneller untersuchen zu können). Für die gesamte Simulation ist eine einheitliche Zeitbasis sinnvoll. Ein Taktgeberobjekt erzeugt für die ganze Simulation einheitliche Taktimpulse, die z. B. einer Sekunde „Realzeit“ entsprechen; der Abstand der Impulse kann von Realzeit bis zum extremen Zeitraffer variiert werden.

Die Klassenbibliotheken der meisten Programmiersprachen bieten Mittel an, um solche Taktgeber zu realisieren. Die Schüler erhalten die Quelle einer entsprechend implementierten Klasse TAKTGEBER mit folgenden Methoden:

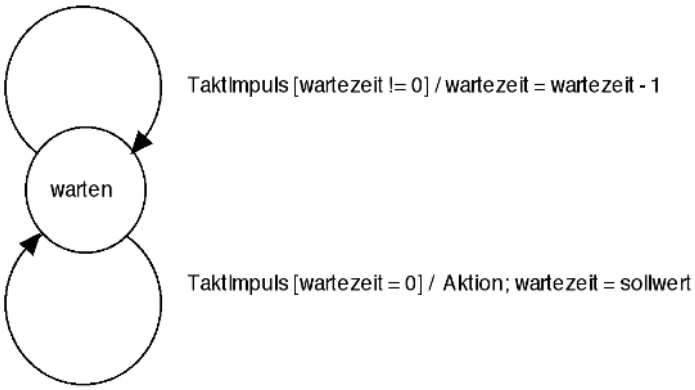
`Starten()`	startet die Folge der Taktimpulse und damit die eigentliche Simulation.
`Anhalten()`	hält die Folge der Taktimpulse an, um eine Situation in Ruhe analysieren zu können.
`DauerSetzen()`	setzt den Abstand der Taktimpulse in Millisekunden.
`Registrieren()`	meldet den Supermarkt für den Empfang der Taktimpulse beim Taktgeber an.

Taktimpulse verwerten

Der Taktgeber informiert den Supermarkt stets, wenn ein neuer Takt schlägt; er will dann jeweils die Botschaft „Nächster Taktimpuls“ an den Supermarkt versenden. Zum Empfang dieser Botschaft benötigt der Supermarkt eine neue Methode, z. B. mit dem Namen `TaktImpulsAusfuehren()`; diese wird vom Taktgeber im Taktabstand aufgerufen. Die Methode muss dafür sorgen, dass sie die gewünschten Ereignisse wie das Erzeugen eines neuen Kunden im richtigen Zeitabstand auslöst.

Ein Kunde muss zum Beispiel alle 50 Sekunden, d. h. nach fünfzig empfangenen Taktimpulsen, erzeugt werden. Hier kommt von den Schülern schnell der Vorschlag, in der Klasse SUPERMARKT ein Attribut bis 50 hoch zu zählen, um dann eine gewünschte Aktion auszulösen. Flexibler ist allerdings der Ansatz, das Attribut von 50 an herunter zu zählen, da dann bei unterschiedlichen Wartezeiten (z. B. bei zufälliger Wartezeit) das „Ereignis-tritt-ein“-Kriterium immer der Attributwert 0 ist.

Allgemein wird das Abwarten einer bestimmten Zeit durch einen einfachen Automaten beschrieben. Aktion steht hier als Platzhalter für alle nach Ablauf der Wartezeit auszuführenden Aktionen.



In der Klasse SUPERMARKT besteht die ausgelöste Aktion im Erzeugen eines neuen Kunden. Die Umsetzung wird zusammen mit den Schülern als Struktogramm vorbereitet (siehe Hefteintrag).

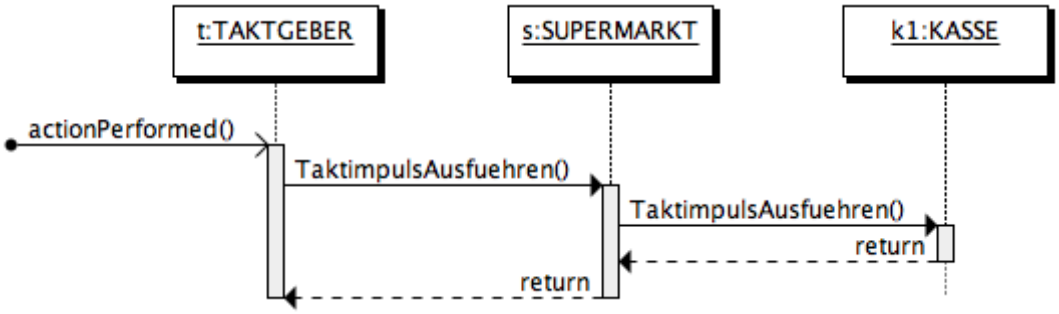
Auch in der Klasse KASSE wird eine gleich aufgebaute Methode `TaktImpulsAusfuehren()` benötigt. Hier ist die ausgelöste Aktion das Holen des nächsten Kunden an die Kasse. Bei der Ermittlung der Wartezeit für das nächste Ereignis muss zusätzlich berücksichtigt werden, ob ein weiterer Kunde aus der Warteschlange geholt werden konnte oder nicht.

– Im ersten Fall beschreibt `wartezeit` die Bedienzeit für den aktuellen Kunden; sie errechnet sich aus einem festen Anteil und der Artikelanzahl, z. B. `wartezeit = 70 + k.ArtikelAnzahlHolen() * 10`. Die Methode `ArtikelAnzahlHolen()` ist eine neue, notwendige Methode des Kunden, damit die Kasse die Artikelanzahl bestimmen kann.

– Falls kein Kunde von der Warteschlange übergeben werden konnte, erkennen die Schüler, dass die Wartezeit den Wert 0 erhalten muss, damit schon beim nächsten Taktimpuls überprüft wird, ob nicht doch ein neuer Kunde eingetroffen ist.

Da nur der Supermarkt die Taktimpulse vom Taktgeber bekommt, muss in der Methode `TaktImpulsAusfuehren()` von SUPERMARKT auch die Methode `TaktImpulsAusfuehren()` aller Kassen aufgerufen werden. Dieser Zusatz wird am Ende der Überlegung im Struktogramm der Methode `TaktImpulsAusfuehren()` von SUPERMARKT ergänzt (genügend Platz lassen!).

Das Sequenzdiagramm macht nochmals deutlich, wie der vom Taktgeber erzeugte Taktimpuls z. B. bei der Kasse 1 ankommt:

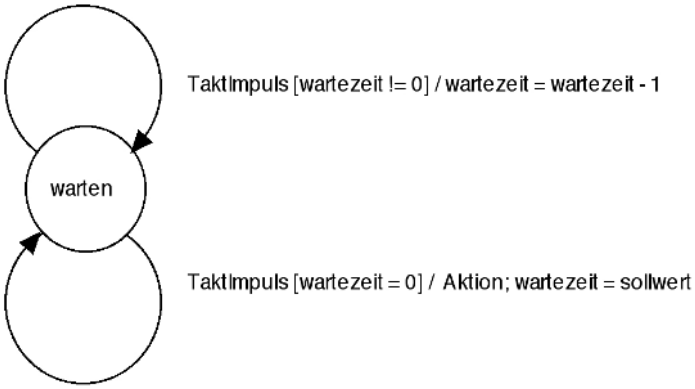


Hinweis: Der Punkt am Beginn der ersten Botschaft signalisiert, dass die Botschaft von einem Objekt außerhalb des betrachteten Ausschnitts abgesetzt wird.

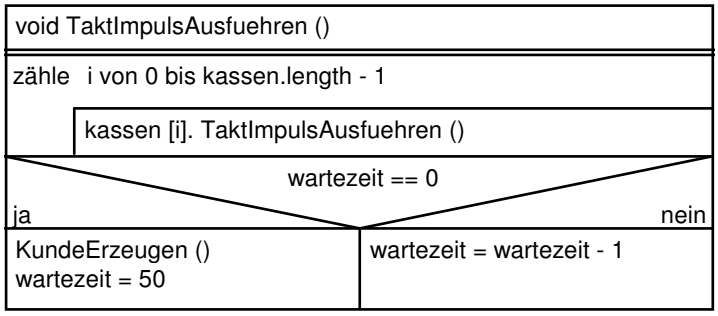
Hefteintrag

Automatischer Ablauf

Das Auswerten von Taktimpulsen durch einen Taktgeber kann durch einen Automaten beschrieben werden:



Die Umsetzung erfolgt in der Methode `TaktImpulsAusfuehren()` der Klasse SUPERMARKT:



Öffnen und Schließen

Mittlerweile muss zum Ausführen der Simulation nur noch je ein Objekt der Klassen SUPERMARKT und TAKTGEBER angelegt werden. Für die Zusammenarbeit muss die Methode `Registrieren()` des Taktgebers aufgerufen werden.

Damit ist es allerdings in der jetzigen Implementierung noch nicht möglich, Kassen zu öffnen oder zu schließen. Die Schüler stellen fest, dass daher die Klasse SUPERMARKT noch um die Methoden `KasseOeffnen(nummer)` und `KasseSchliessen(nummer)` erweitert werden muss.

Die folgenden Aufgaben werden mit den Schülern vorbesprochen. Bei Bedarf werden Teillösungen, z. B. ein Struktogramm, gemeinsam erarbeitet.

Aufgaben:

1. **Die Klasse KUNDE**

   Ergänze die Klasse KUNDE um eine Methode `ArtikelAnzahlHolen()`, die die Anzahl der Artikel des Kunden zurückgibt.

2. **Die Klasse KASSE**

   Ergänze die Klasse KASSE um eine Methode `TaktImpulsAusfuehren()`, die die Zeit bis zum Holen des nächsten Kunden verwaltet und nach Ablauf der Wartezeit einen

neuen Kunden holt. Die Bedienzeit errechnet sich aus einem festen Anteil (Geld wechseln etc.) von z. B. 70 Sekunden und einem von der Artikelzahl abhängigen Anteil (z. B. 10 Sekunden pro Artikel). Falls kein Kunde mehr ansteht, hat die Wartezeit den Wert 0.

3. Die Klasse SUPERMARKT

a) Ergänze die Klasse SUPERMARKT um die Methode `TaktImpulsAusfuehren()`, die die Zeit bis zur Erzeugung des nächsten Kunden verwaltet und die Taktimpulse an die Kassen weitergibt.

b) Ergänze die Klasse SUPERMARKT um die Methoden zum Öffnen und Schließen der Kassen des Supermarkts. Beachte dabei, dass eine Kasse nicht einfach schließen kann, sondern nur über die Methode `SchliessendSetzen()` das Schließen einleiten kann.

c) Teste die Supermarktsimulation.

Hausaufgabe

Die Schüler stellen zusammen, welchen Einfluss die Anzahl der Artikel im Korb auf die Länge der Wartezeit hat. Sie überlegen weiter, ob ein Unterschied entsteht, wenn in der Simulation die Anzahl der Artikel für alle Kunden gleich dem Durchschnittswert ist oder wenn die Anzahl statistisch verteilt um diesen Durchschnittswert schwankt.

10. Zufälligkeiten (14. Doppelstunde)

In dieser Einheit lernen die Schüler Möglichkeiten zur Generierung zufallsabhängiger Werte kennen. Damit vertiefen sie auch ihre Kenntnisse in algorithmischen Beschreibungen.

Impuls

Kunden kommen nicht immer im gleichen zeitlichen Abstand und haben verschiedene Anzahlen von Artikeln im Warenkorb. Diese beiden noch zu bearbeitenden Aspekte der Aufgabenstellung werden von den Schülern nach kurzer Diskussion identifiziert.

Erzeugung zufälliger Zahlenfolgen

Das Arbeiten mit zufällig erzeugten Zahlenfolgen ist den Schülern eventuell schon aus der Informatik (funktionale Modellierung) oder der Mathematik bekannt. Nun muss genauer auf die benötigte Semantik der verfügbaren Zufallsfunktionen eingegangen werden. Die Schüler stellen sofort fest, dass sowohl für variable Abstände zwischen dem Erzeugen der Kunden als auch für unterschiedliche Warenmengen jeweils zufällig erzeugte Werte aus einem bestimmten, ganzzahligen Bereich benötigt werden.

Der Lehrer stellt als Lösungsmöglichkeit die Klasse `Random` aus dem Paket `java.util` mit ihren wichtigsten Methoden vor:

`Random()`	Konstruktor für einen Zufallsgenerator
`Random (startwert)`	wie oben: Die Folge der erzeugten Zufallszahlen ist bei gleichem Startwert immer die gleiche.
`nextFloat()`	liefert eine zufällige Kommazahl aus dem Intervall [0; 1[.
`nextInt(max)`	liefert eine zufällige ganze Zahl aus dem Intervall [0; max[.

Hinweis: Streng genommen spricht man hier von Pseudozufallszahlen, da zwar eine allen Anforderungen der Statistik genügende „zufällige“ Zahlenfolge erzeugt wird, aber bei jedem – z. B. durch `Random(0)` – neu angelegten Objekt immer dieselbe Folge entsteht. Die Methoden `nextFloat()` bzw. `nextInt()` eines Objekts der Klasse `Random` sind im strengen Sinn Funktionen. Um „unerwartete“ Zufallsfolgen zu bekommen, verwendet man

entweder die Form `Random(startwert)` des Konstruktors, wobei als Parameter z. B. die Millisekunden seit Systemstart genommen werden oder die Form `Random()`, die ebenfalls jedes Mal eine andere Folge erzeugt. In Testsituationen ist es dagegen entscheidend, dass immer die gleiche Zufallsfolge entsteht.

Intervalle anpassen

Eine kurze Diskussion über die Warenmengen bzw. die Kundenabstände führt die Schüler zu der Überlegung, dass hier Schwankungen in gewisser Breite um den Durchschnittswert stattfinden. Für die Güte der Simulation ist es wohl nicht entscheidend, hier ein ausgefeilteres Verteilungsmodell anzugeben, das nur durch umfangreiche Erhebungen gewonnen werden könnte. Eine Schwankung mit linearer (gleichmäßiger) Verteilung um den Durchschnittswert, der durch empirische Erhebungen im Supermarkt gewonnen wurde, beschreibt die Situation ausreichend. Das bedeutet aber, dass zu den Ergebnissen des Zufallsgenerators jeweils nur eine Konstante addiert werden muss, um den Mittelwert des Intervalls auf den bekannten Durchschnittswert zu bringen. Mögliche Intervalle sind gegeben durch:

```
Random zufall = new Random();
wartezeit = 25 + zufall.nextInt(50);
artikel = 3 + zufall.nextInt(8);
```

Hefteintrag

Zufälligkeiten

Zur Erzeugung von Zufallszahlenfolgen bietet Java die Klasse `Random` aus dem Paket `java.util` an.
Wichtige Methoden sind:
`nextFloat()` liefert eine zufällige Kommazahl aus dem Intervall [0; 1[,
`nextInt(max)` liefert eine zufällige ganze Zahl aus dem Intervall [0; max[.

Aufgaben

1. Die Klasse SUPERMARKT

a) Erzeuge im Konstruktor der Klasse SUPERMARKT ein Objekt der Klasse `Random`. Hinweis: Vor Beginn der Klassendefinition muss diese Klasse durch `import java.util.Random;` bekanntgemacht werden.

b) Füge bei der Erzeugung neuer Kunden eine zufallsabhängige Artikelmenge ein.

c) Ändere in der Methode `TaktImpulsAusfuehren()` die Wartezeitangabe für den nächsten Kunden auf eine zufällige Angabe ab.

d) Teste die neuen Eigenschaften. Halte dazu nach einer gewissen Zeit den Taktgenerator an und verifiziere, ob die Kunden verschiedene Artikelzahlen mit sich führen.

Hausaufgabe

Die Schüler tragen verschiedene Strategien zusammen, die die Kunden beim Anstellen verfolgen. Sie überlegen, welchen Einfluss diese Strategien auf die Länge der Warteschlangen haben können. Weiter stellen sie Vorüberlegung zur Implementierung der verschiedenen Strategien an.

11. Vererbung und Polymorphismus (15. und 16. Doppelstunde)

In dieser Einheit lernen die Schüler das Konzept der Vererbung und das ergänzende Konzept des Polymorphismus kennen und anzuwenden.

Impuls

Es gibt Kunden mit unterschiedlichem Anstellverhalten. Da die Länge der Schlangen auch von diesen Unterschieden abhängt (ein sich nach dem Zufallsprinzip anstellender Kunde kann auch die längste Schlange noch länger machen), muss diese Tatsache in der Simulation berücksichtigt werden.

Verschiedene Arten von Kunden

Die Schüler diskutieren das unterschiedliche Verhalten der Kunden beim Anstellen an die Schlange. Dabei kristallisieren sich vier Grundstrategien heraus, die in der Simulation verwendet werden sollen. Der Kunde wählt

- die Schlange mit den wenigsten Artikeln in den Warenkörben,
- die kürzeste Schlange,
- zufällig eine der beiden ersten Möglichkeiten oder
- eine zufällige Schlange.

Mechanismus der Vererbung

Bislang wurde in der Supermarktsimulation nur die Strategie der kürzesten Schlange implementiert. Die Schüler erkennen in der Diskussion, dass die drei neuen Alternativen im Anstellverhalten über drei weitere Klassen modelliert werden können. Dieser Ansatz führt aber zu Problemen, da zum einen das Simulationsprogramm sehr aufwändig würde, zum anderen sehr viel Teile der Klassen gleich sind und alle, die Kunden betreffenden, späteren Änderungen vierfach ausgeführt werden müssten.

Die Modelle der jetzt insgesamt vier Kundenklassen sind alle Spezialisierungen einer neuen, allgemeineren Klasse KUNDE. Diese Klasse beschreibt alle gemeinsamen Attribute und Methoden, z. B. `artikelmenge` und `PositionSetzen()`; der Methode `Anstellen()` kann in dieser Klasse keine sinnvolle Bedeutung gegeben werden, der Methodenrumpf bleibt deshalb leer. Bei den Unterklassen ist jedoch im Wesentlichen nur die Methode `Anstellen()` entsprechend der Anstellstrategie ausgeprägt. Eigentlich würde die Klasse KUNDE die Methode `Anstellen()` gar nicht mehr benötigen. Da KUNDE aber beschreibt (vgl. Abschnitt über Polymorphismus), welche Methoden alle Unterklassen zur Verfügung stellen, muss die Methode `Anstellen()` in der Klasse KUNDE aufgeführt sein.

In der objektorientierten Modellierung wurde für diese Standardsituation das Konzept von Generalisierung und Spezialisierung entwickelt. Durch **Vererbung** kann zwischen den Klassen eine hierarchische Beziehungsstruktur abgebildet werden. Dabei erben **Unterklassen** (Sohnklassen) alle Attribute und Methoden einer **Oberklasse** (Vaterklasse). Sie können weitere Attribute und Methoden hinzufügen sowie bestehende Methoden neu definieren (**überschreiben**).

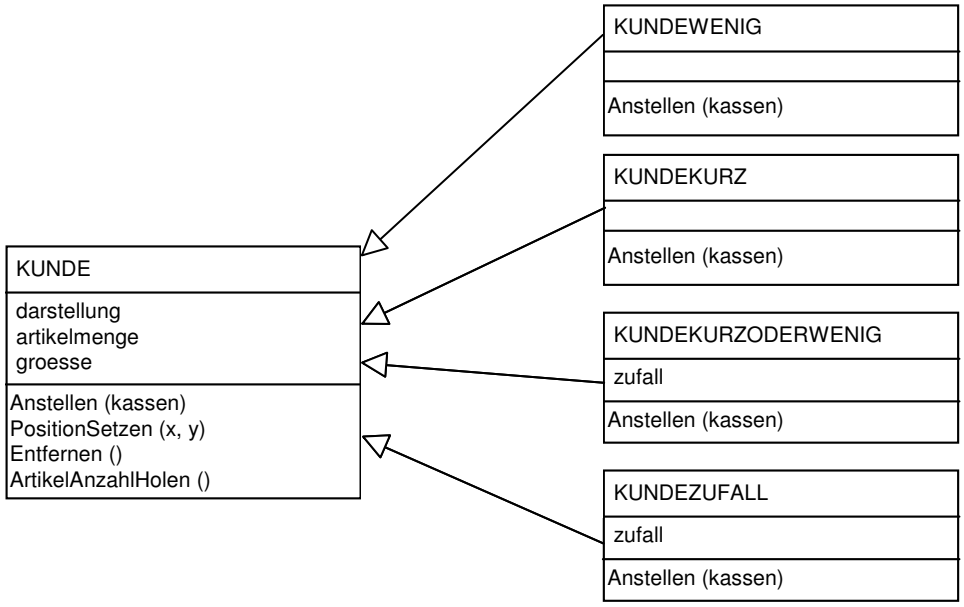
Die Schüler wenden dieses Konzept auf die Supermarktsituation an. Sie identifizieren die Attribute und Methoden der neuen Oberklasse KUNDE sowie die spezifischen Attribute und Methoden der Unterklassen.

Hefteintrag 1

Vererbung und Polymorphismus

Um hierarchische Beziehungen der Form Generalisierung-Spezialisierung zwischen Klassen auszudrücken, bietet die objektorientierte Modellierung das Konzept der Vererbung.
Unterklassen (Sohnklassen)

- erben alle Attribute und Methoden von Oberklassen (Vaterklassen),
- können neue Attribute und Methoden hinzufügen,
- können ererbte Methoden neu definieren (überschreiben).



Hinweis: Der Hefteintrag zum zweiten Thema der Überschrift (Polymorphismus) wird später, bei dessen Behandlung, hinzugefügt.

Vererbung in Java

Die Lehrkraft informiert die Schüler, wie das Konzept der Vererbung in der Sprache Java umgesetzt ist. Die Schüler wenden diese Information direkt auf das Beispiel an.

Hefteintrag 2

In Java erfolgt die Angabe einer Oberklasse durch das Schlüsselwort `extends` hinter dem Klassennamen. Der Konstruktor der Oberklasse kann durch das Schlüsselwort `super` angesprochen werden.

```
class KUNDEKURZ extends KUNDE
{
    KUNDEKURZ (int artikel)
    {
        super (artikel);
    }

    void Anstellen (KASSE [] kassen)
    {
        WARTESCHLANGE beste;
        WARTESCHLANGE w;
        beste = null;
        for (int i = 0; i < kassen. length; i++)
        {
            w = kassen [i]. WarteschlangeHolen ();
            if ((kassen [i]. IstOffen ()) &&
          ((beste == null) || (w. LaengeHolen () < beste. LaengeHolen ())))
            {
                beste = w;
            }
        }
        beste. Anfuegen (this);
    }
}
```

Hinweis: Das Anstellverhalten von KUNDEKURZ entspricht dem der früheren KUNDE-Objekte.

Aufgaben

1. Die Unterklassen von KUNDE

a) Ergänze die Klasse KUNDEKURZ mit der Anstellstrategie der bisherigen Klasse KUNDE in deinem Projekt.

b) Ergänze eine Klasse KUNDEWENIG in deinem Projekt. Die Kunden stellen sich an der Schlange an, in der sich insgesamt die wenigsten Artikel im Warenkorb befinden. Tipp: Ergänze die Klasse WARTESCHLANGE um eine Methode `Artikelanzahl-GesamtHolen()`.

c) Entwirf eine Klasse KUNDEZUFALL, bei der sich der Kunde zufällig an einer offenen Kasse anstellt und implementiere diese Klasse.

d) Entwirf eine Klasse KUNDEKURZODERWENIG, bei der sich der Kunde mit gleicher Wahrscheinlichkeit an der kürzesten Schlange oder der Schlange mit den wenigsten Artikeln anstellt und implementiere diese Klasse.

2. Die Klasse KUNDE

Entferne in der Klasse KUNDE den Inhalt des Rumpfs der Methode `Anstellen()`.

Hausaufgabe

Die Schüler stellen zusammen, wo überall Umbauten nötig wären, wenn die Simulation mit Objekten der Klasse KUNDEZUFALL anstelle von KUNDE arbeiten sollte.

Impuls

Der Mechanismus der Vererbung beseitigt noch nicht das Problem, dass die Warteschlangenverwaltung mit Objekten aus vier Klassen sehr aufwändig würde.

Polymorphismus

Die Lehrkraft bietet als Lösung die Möglichkeit, dass in Attributen vom Typ einer Vaterklasse auch Referenzen auf Objekte einer Unterklasse gespeichert werden können. Die Schüler erkennen, dass dieser Ansatz das Problem prinzipiell löst. Allerdings muss noch die Frage geklärt werden, welche Methode mit den Aufruf `kunde.Anstellen()` in der Methode `KundeErzeugen()` des Supermarkts aufgerufen wird. Ist es die Methode `Anstellen()` der Klasse KUNDE, da `kunde` ein Attribut vom Datentyp KUNDE ist? Oder ist es die Methode `Anstellen()` der Klasse KUNDEZUFALL, wenn ein Objekt dieser Klasse referenziert wird? Im Unterrichtsgespräch wird klar, dass die Methode `Anstellen()` des referenzierten Objekts benötigt wird, nicht die Methode `Anstellen()` der Klasse KUNDE. Glücklicherweise „weiß“ jedes Objekt um seine eigenen Methoden, so dass auch hier das Angebot der Programmiersprache dem Bedarf entspricht.

Hefteintrag

Attribute vom Typ einer Oberklasse können auch Referenzen auf Objekte vom Typ einer Unterklasse speichern. Beim Aufruf werden die Methoden der Klasse des jeweiligen Objekts verwendet. Diese Eigenschaft wird als Polymorphismus bezeichnet.

Aufgaben

1. Viele Kunden im Supermarkt

a) Ergänze die Methode `KundeErzeugen()` der Klasse SUPERMARKT so, dass zufällig Objekte aller vier Klassen von Kunden erzeugt werden.

b) Verifiziere, ob außer in `KundeErzeugen()` an keiner weiteren Stelle der Simulation Information über die tatsächliche Klasse der Kunden benötigt wird.

c) Teste die neue Version deiner Simulation.

12. Besondere Klassen[1] (17. Doppelstunde)

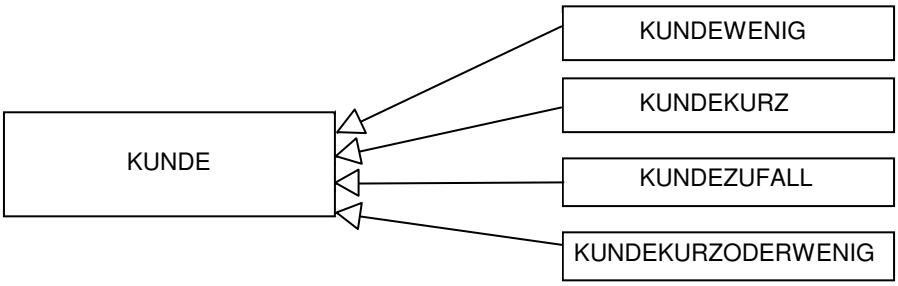
Abstrakte Klassen sind im Lehrplan nicht explizit als Inhalt ausgewiesen. Dennoch sind sie, was die semantische Korrektheit betrifft, ein unverzichtbarer Teil objektorientierter Modellierung. Ähnliches gilt für die Interfaceklassen. Die Beschäftigung mit diesen Konzepten eröffnet darüber hinaus ein konzeptionelles Verständnis für die Struktur vorhandener Klassenbibliotheken, nicht nur der hier angesprochenen Bibliotheken für graphische Oberflächen.

[1] Abstrakte Klassen sollten auf jeden Fall behandelt werden, auf das Konzept der Interfaces (Schnittstellen) kann bei Zeitknappheit verzichtet werden.

In Klassenhierarchien stehen an der Spitze sehr oft Klassen, die „nur“ der Beschreibung der allgemeinen Eigenschaften der gesamten Hierarchie dienen, von denen aber keine Objekte instanziert werden; insbesondere kann in solchen Klassen nicht jede Methode bereits sinnvoll beschrieben werden, weil diese Leistungen erst in den speziellen Unterklasse erbracht werden. Die Schüler lernen die zur Abbildung dieser Situationen entwickelten Konzepte abstrakter Klassen und Interfaces kennen.

Impuls

Die nebenstehende Klassenbeziehung ist korrekt. Sie spiegelt aber nicht wider, dass Objekte der Klasse KUNDE nicht mehr benötigt werden und auch nicht mehr erzeugt werden sollten.



Abstrakte Klassen

Die Oberklasse KUNDE hat zwei Besonderheiten: von dieser Klasse werden keine Objekte instanziert, die Methode `Anstellen()` hat in dieser Klasse keine eigenständige Bedeutung. Der Rumpf der Methode `Anstellen()` könnte leer bleiben. Objekte dieser Klasse könnten dann instanziert werden, würden sich aber nie irgendwo anstellen. Andererseits wird diese Klasse benötigt, da sie das allgemeine Verhalten aller Unterklassen beschreibt; z. B. muss das Feld der Warteschlange Elemente vom Typ KUNDE haben, um Objekte aller Unterklassen referenzieren zu können.

Um diese Problemfelder zu vermeiden, stellt die Lehrkraft das Konzept der abstrakten Klassen vor. Diese beschreiben Klassen, von denen keine Objekte instanziert werden dürfen. Aus diesem Grund dürfen sie auch Methoden ohne Methodenrumpf erhalten. Diese Methoden werden ebenfalls als abstrakt bezeichnet. Im Bild der Klasse als Bauplan für Objekte stellen abstrakte Klassen unvollständige Baupläne dar, die erst in Unterklassen vervollständigt werden. Erst von den Unterklassen können Objekte erzeugt werden.

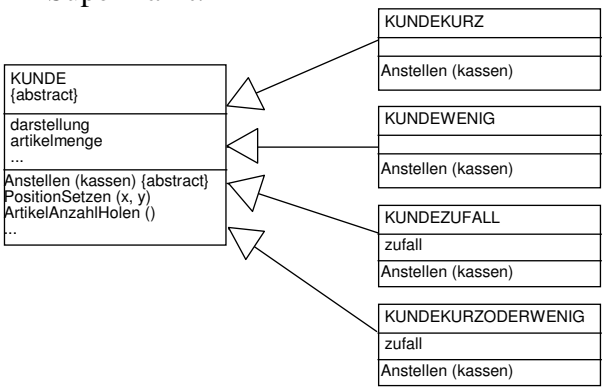
Der folgende Hefteintrag entsteht bei der Vorstellung des Konzepts der abstrakten Klassen und seiner Anwendung auf die Supermarktsimulation.

Hefteintrag 1

Abstrakte Klassen

Abstrakte Klassen beschreiben, welche Botschaften von allen Objekten der Unterklassen verstanden werden; von abstrakten Klassen können keine Objekte erzeugt werden.
Abstrakte Methoden haben keinen Methodenrumpf und können daher nur in abstrakten Klassen vorkommen.

Im Supermarkt:




```
abstract class KUNDE
{
    // Attribute
    // Konstruktor

    abstract void Anstellen (KASSE [] kassen);
    // muss in den Unterklassen
    // überschrieben werden

    // weitere Methoden
}
```

Interfaces

Mit Interfaceklassen wird noch mehr als schon mit den abstrakten Klassen das Thema guten Programmdesigns angesprochen. Das vielen Klassenbibliotheken zugrunde liegende Entwurfsprinzip der losen Kopplung wird hier deutlich. Dabei wird die Information, die eine Klasse über eine andere hat, so weit wie möglich reduziert, um eine bessere Wartbarkeit und Flexibilität zu erreichen.

Der Taktgeber bedient ein Objekt der Klasse SUPERMARKT mit Taktimpulsen. Dabei ist es für den Taktgeber völlig unerheblich, welche Aufgabe ein Objekt der Klasse SUPERMARKT eigentlich hat; für den Taktgeber genügt es zu wissen, dass dieses SUPERMARKT-Objekt eine Methode `TaktImpulsAusfuehren()` besitzt, die der Taktgeber aufrufen kann. Genau so gut könnte er auch Objekte der Klasse KASSE aufrufen, denn auch diese besitzen die Methode `TaktImpulsAusfuehren()`. Mit den bisherigen Mechanismen ist es aber nicht möglich, dass der Taktgeber Objekte beider Klassen bedient, z. B. ist der Parameter der Taktgebermethode `Registrieren()` entweder vom Typ SUPERMARKT oder vom Typ KASSE.

Um diese Problematik zu lösen, existiert das Konzept der Sichten. Eine Sicht beschreibt eine oder mehrere Methoden, die zur Kommunikation zwischen Objekten einer Klasse mit Objekten einer anderen Klasse nötig sind, ohne dass Letztere vollständig bekannt sein muss. Die für den Taktgeber notwendige Sicht, z. B. mit dem Namen TAKTKLIENT, muss nur die Methode `TaktImpulsAusfuehren()` beschreiben. Die Klassen SUPERMARKT und KASSE müssen dann nur zusichern, dass sie die in der Sicht TAKTKLIENT beschriebene Methode implementieren; der Taktgeber stützt sich nur noch auf diese Sicht ab. Die andere Sicht der Klassen SUPERMARKT und KASSE ist die ihrer eigentlichen Aufgabe als Beschreibung des typischen Verhaltens eines Supermarkts und einer Kasse.

In Java werden solche Sichten als so genannte Interfaces beschrieben. Ein Interface beschreibt alle für die Sicht zur Verfügung stehenden Methoden als abstrakte Methoden, Attribute können nicht angegeben werden. Die Klassen, die die Methode(n) wirklich zur Verfügung stellen, implementieren das Interface durch die Angabe `implements Interfacename` hinter einer

eventuellen Vererbungsangabe. In Interfaces angegebene Methoden sind automatisch `abstract`, dieses Schlüsselwort darf daher nicht angegeben werden.

Zugriffsrechte und Datenkapselung

Bisher konnten die verschiedenen Möglichkeiten von Java, das Zugriffsrecht auf Klassen, Methoden und Attributen einzuschränken, außer Acht gelassen werden. Da in Interfaces beschriebene Methoden grundsätzlich das Zugriffsrecht `public` besitzen, müssen diese Angaben nun thematisiert werden. Die Verwendung dieser Möglichkeiten zur Datenkapselung sollte weitgehend genutzt werden. Die Lehrkraft stellt die vier möglichen Zugriffsrechte vor, die Schüler diskutieren deren Einsatzmöglichkeiten. Dabei werden die Angaben `private` bzw. `protected` besonders genau beleuchtet.

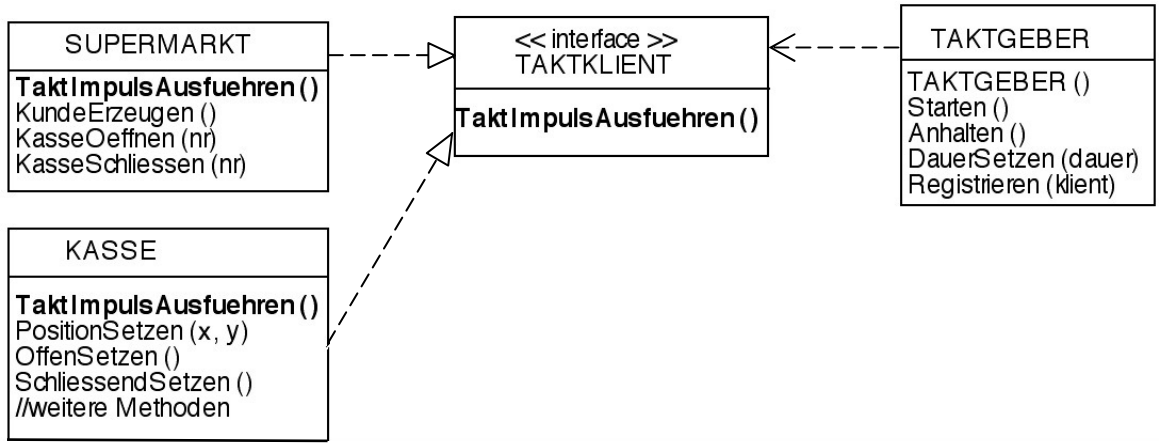
Die Angabe `public` wird nur kurz angesprochen, da das Arbeiten mit den Paketen (packages) von Java nur bei der Erstellung von Klassenbibliotheken Sinn macht. Angegeben werden muss aber, dass die in Interfaces vereinbarten Methoden automatisch das Zugriffsrecht `public` haben; die implementierende Methode muss diese Zugriffsrechte angeben.

Hefteintrag 2

Interfaces und Zugriffsrechte

Ein Interface beschreibt eine bestimmte Sicht auf eine Klasse, d. h. Methoden, die eine Klasse für einen bestimmten Zweck zur Verfügung stellt. Interfaces dienen zur besseren Trennung der verschiedenen logischen Teile eines Programms.

Im Supermarkt:




```
interface TAKTKLIENT
{
    void TaktimpulsAusfuehren ();
}

class SUPERMARKT implements TAKTKLIENT
{
    // Verschiedene Teile
    public void TaktImpulsAusfuehren ();
    {
    // Implementierung der Methode

    ...
}
```

```
class TAKTGEBER
{
    // Attribute

    // Konstruktor

    Registrieren (TAKTKLIENT klient)
    {
        // Inhalt der Methode

    ...
}
```

Mit folgenden Rechten kann der Zugriff auf Klassen, Methoden und Attribute eingeschränkt werden:

`private`	Zugriff nur von innerhalb der Klasse erlaubt
`protected`	Zugriff nur von innerhalb der Klasse oder von Unterklassen aus erlaubt
`keine Angabe`	Zugriff von allen Programmteilen aus erlaubt
`public`	Zugriff auch von außerhalb des Programmpakets möglich

Aufgaben – abstrakte Klassen

1. Die Klasse KUNDE

a) Ändere die Klasse KUNDE in eine abstrakte Klasse mit abstrakter Methode `Anstellen()` ab.

b) Verifiziere, dass die Entwicklungsumgebung BlueJ im Popup-Menü der Klasse KUNDE den Eintrag „new KUNDE ()“ nicht mehr anbietet. Im Codepad kannst du auch überprüfen, dass der Compiler „new KUNDE ()“ nicht mehr akzeptiert.

Aufgaben – Schnittstellen

2. Das Interface TAKTKLIENT

a) Vereinbare das Interface TAKTKLIENT mit der Methode `TaktimpulsAusfuehren()`.

b) Ändere SUPERMARKT so ab, dass das Interface implementiert wird.

c) Ersetze im Taktgeber die Verwendung von SUPERMARKT durch die Verwendung von TAKTKLIENT.

3. Kassen sind auch Klienten des Taktgebers

Auch die Klasse KASSE kann das Interface TAKTKLIENT implementieren. Jedes Objekt der Klasse KASSE kann sich damit selbst beim Taktgeber registrieren; die Methode `TaktImpulsAusfuehren()` des Supermarkts muss den Taktimpuls dann nicht mehr verteilen.

a) Implementiere in der Klasse KASSE das Interface TAKTKLIENT. Ergänze den Konstruktor um den Parameter TAKTGEBER t und registriere die Kasse im Konstruktor beim Taktgeber.

b) Ergänze auch den Konstruktor der Klasse SUPERMARKT um den Taktgeber als Parameter. Registriere den Supermarkt beim Taktgeber. Entferne in der Methode `TaktImpulsAusfuehren()` die Weitergabe des Taktimpulses an die Kassen.

c) Teste den Umbau.

13. Graphische Benutzeroberflächen (18. und 19. Doppelstunde)

In dieser Einheit lernen die Schüler die Grundkonzepte und -strukturen typischer graphischer Oberflächenpakte kennen und auf ihre Situation anzuwenden. Dabei vertiefen sie die Konzepte von Generalisierung und Spezialisierung durch Analysieren und zielgerichtetes Nutzen bestehender Generalisierungshierarchien.

Impuls

Um das Simulationsprogramm auch von Laien verwendbar zu machen, wird noch eine einfache Bedienoberfläche benötigt. Heutzutage kommen dafür graphische Oberflächen zum Einsatz.

Grundelemente graphischer Oberflächen

Die Grundelemente graphischer Oberflächen sind den Schülern aus ihrem Umgang mit dem Computer vertraut; diese Erfahrungen werden gebündelt und strukturiert. Dabei werden direkte Bedienelemente (Menüs, Knöpfe, Radioknöpfe, Eingabefelder usw.) von zusammen-

gesetzten Objekten wie Fenstern oder Dialogen unterschieden, die selbst wieder (Unter-)Elemente enthalten. Anschließend wird in eine konkrete Klassenbibliothek für eine graphische Oberfläche eingeführt.

Java wird standardmäßig mit zwei Oberflächenbibliotheken ausgeliefert: AWT und Swing. AWT (Abstract Window Toolkit) ist nicht ganz so mächtig wie Swing, dafür aber einfacher zu verwenden. Deshalb sind die folgenden Beispiele in AWT ausgeführt.

Zur Einarbeitung in die Erzeugung und Handhabung der Java-AWT-Objekte empfiehlt es sich, vor der Erstellung der Oberfläche für den Supermarkt eine einfache, vom Supermarkt unabhängige Oberfläche zu erstellen. Java bietet hier zusammen mit der Entwicklungsumgebung BlueJ eine sehr schöne Möglichkeit, einzelne Oberflächenklassen interaktiv zu untersuchen. Vereinbart man eine Unterklasse der zu untersuchenden Klasse (Beispiele siehe unten und Arbeitsblatt „Einheit 13 Teil 1“) so kann man Objekte dieser Unterklasse instanzieren und deren Methoden aufrufen. Dabei können auch die ererbten Methoden direkt aufgerufen und deren Wirkung untersucht werden.

Begonnen wird am besten mit der Klasse Frame (Fenster). Die Schüler erstellen eine Unterklasse MEINFENSTER, in der nur der Standardkonstruktor vorhanden ist.

```
import java.awt.*;
class MEINFENSTER extends Frame
{
    MEINFENSTER()
    {
        super();
    }
}
```

Instanzieren eines Objekts dieser Klasse bringt noch kein sichtbares Resultat. Die Schüler erkunden über das Kontextmenü des Objekts, welche Methoden die Klasse Frame anzubieten hat und welche Methoden der Oberklassen von Frame gegebenenfalls noch nützlich sein könnten. Folgende Aufrufe (Reihenfolge beliebig) machen das Fenster sichtbar:

```
setVisible(true)     // Anzeigen des Fensters
setSize(600, 400)    // Größe des Fensters in Pixeln
setLocation(50, 30) // Position der linken, oberen Fensterecke auf dem
                     // Bildschirm
```

Durch den Aufruf `setTitle("Name")` kann dem Fenster ein Titel gegeben werden. Eine wichtige Information muss von der Lehrkraft kommen. Fenster der Bibliotheken AWT und Swing besitzen so genannte Layoutmanager, d. h., sie versuchen, die angezeigten Oberflächenelemente nach vorgegebenen Regeln anzuordnen. Dies ist insbesondere nötig, wenn die Fenster in der Größe veränderbar sind, z. B. innerhalb eines Webbrowsers. Da die sinnvolle Nutzung der automatischen Anordnung sehr viel Erfahrung benötigt, sollte die Anordnungsautomatik mit `setLayout(null)` unterbunden werden.

Im nächsten Schritt wird das Anzeigen einfacher Texte in einem Fenster untersucht. Dazu wird eine Unterklasse MEINTEXT zur zuständigen Klasse Label erstellt und ein Objekt dieser Klasse instanziert (z. B. mit dem Namen `text`). Obwohl alle notwendigen Methoden (`setVisible(), setSize(), setLocation()` und die neue Methode `setText()`) des Textobjekts aufgerufen wurden, erscheint der Text noch nicht auf dem Fenster. Erst wenn auch noch die Methode `add(text)` des Fensters aufgerufen wird, erscheint der Text im Fenster. Das macht den Schülern deutlich, dass die Oberflächenobjekte in einer Baumstruktur eingebunden werden müssen. Dieser Baum wird vom Anzeigemechanismus abgearbeitet; Blätter werden zuletzt gezeichnet, sind also „im Vordergrund“. Hilfreich für die Vorstellung der Schüler ist das Zeichnen des Objektdiagramms der entstehenden Mini-Oberfläche.

Mit den weiteren, für den Supermarkt benötigten Oberflächenklassen (Button, TextField) kann die Voruntersuchung abgeschlossen werden. Sollen noch andere Klassen untersucht werden, kann diese Arbeit auf einzelne Schülergruppen übertragen werden; die Ergebnisse der Gruppenarbeit werden dann im Plenum allen Mitschülern zur Verfügung gestellt.

Die Schüler erstellen nun eine Klasse EINFACHEOBERFLAECHE, in der die bisher untersuchten Klassen zusammen verwendet werden.

Aufgaben

1. EINFACHEOBERFLAECHE

a) Erstelle eine Klasse EINFACHEOBERFLAECHE, die im Konstruktor ein Fenster etwa der Größe 300 x 200 erzeugt.

b) Erzeuge in der Mitte oben des Fensters einen Anzeigetext (Label) mit dem Wert „Vorbesetzung“ und unter diesem Text ein Feld zur Texteingabe.

c) Füge unterhalb des Eingabefeldes drei Knöpfe hinzu, mit denen später der Anzeigetext beeinflusst werden kann.

Umgang mit Ereignissen

Noch bewirkt das Drücken eines Knopfs keine Aktion. Die Schüler müssen sich klarmachen, dass es prinzipiell möglich sein muss, auf äußere Ereignisse (Maustaste gedrückt, Maustaste losgelassen, Taste gedrückt, Taste losgelassen) zu reagieren. Der Umgang mit diesen grundlegenden Ereignissen ist aber sehr aufwändig und im Fall des Knopfs eigentlich gar nicht interessant. Die Semantik eines Knopfdrucks wird im Wesentlichen beschrieben durch „Die Maustaste wurde innerhalb der gekennzeichneten Schaltfläche gedrückt und auch innerhalb der gekennzeichneten Schaltfläche wieder losgelassen“. Nur dieses abstrakte Ereignis ist für das Programm relevant; moderne Oberflächenpakete bieten daher die Möglichkeit, nur über solche höherwertigen Ereignisse zu informieren.

Für die Information über eintretende Ereignisse gibt es feststehende Konzepte. Im objektorientierten Umfeld besitzt das auslösende Element (hier z. B. der Knopf) eine Registriermethode (hier: `addActionListener()`), mit der dem auslösenden Element ein „Lauscher“ übergeben wird, der auf das Eintreten des auslösenden Ereignisses (z. B. ein Mausklick) wartet. Das „Lauscherobjekt“ besitzt hierfür eine vereinbarte Methode (hier: `actionPerformed()`), die beim Eintritt des Ereignisses aufgerufen wird.

Dieses Konzept entspricht vollständig dem Vorgehen beim Taktgeber, bei dem sich die Klienten Supermarkt und Kassen registrieren müssen, um vom Eintreten des Ereignisses `TaktImpulsAusfuehren()` informiert zu werden. Und genau wie beim Taktgeber muss auch bei der Ereignisbehandlung das Objekt nicht von einer fest vorgegebenen Klasse sein; das Objekt muss zu einer Klasse gehören, die das Interface `ActionListener` implementiert und damit die Existenz der Methode `actionPerformed()` zusichert.

Eine gute Möglichkeit bietet die Verwendung sog. anonymer Klassen, d. h. Klassen, die hinter dem Konstruktoraufruf ohne eigenen Namen (daher anonym) vereinbart werden. Sie können eine Unterklasse zu einer bestehenden Klasse sein oder aber, wie hier ausgenützt, ein Interface implementieren. Für den Konstruktor wird der Name des Interface verwendet, die Implementierung der Methode erfolgt direkt hinter dem Konstruktor:

```
knopf.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(ActionEvent e) {
                text.setText("Der Knopf wurde gedrückt.");
            }
        });
```

Die Schüler ergänzen nun ihre Oberfläche um die Ereignisbehandlung für die Knöpfe. In gleicher Weise kann auch die Behandlung des Ereignisses „Fenster schließen“ mit der Semantik Programm beenden“ thematisiert werden.

Aufgaben

2. EINFACHEOBERFLAECHE (Fortsetzung)

a) Ergänze die Klasse EINFACHEOBERFLAECHE um die Ereignisbehandlung der Knöpfe.

b) Füge die Ereignisbehandlung für das Fenster hinzu. Bei Klick auf die Schaltfläche „Schließen“ das Fensters soll das Programm mit `System.exit(1);` beendet werden.

Oberfläche für den Supermarkt

Impuls

Die Grundlagen sind gelegt, jetzt kann die Oberfläche für das Simulationsprogramm gestaltet werden.

Vorbemerkung

Ein Hefteintrag ist hier nicht vorgesehen; für die neuen Konzepte ist ein Arbeitsblatt vorgeschlagen (Arbeitsblatt „Einheit 13 Teil 2“).

Benötigte Oberflächenelemente

In einem kurzen Brainstorming werden die für das Simulationsprogramm benötigten Oberflächenelemente zusammengetragen. Dabei kann durchaus zwischen „unbedingt nötigen“ und „sinnvoll ergänzenden“ Elementen unterschieden werden.

Es empfiehlt sich, in dieser Phase auch die Anordnung der Elemente innerhalb des Fensters anzusprechen. Dabei können durchaus verschiedene Alternativen entstehen. Erst dann werden in einem ersten Schritt die Bedienelemente in die Klasse OBERFLAECHE eingebaut.

Statische Methoden und globale Variable (fakultativer Lehrplaninhalt)

Der Konstruktor der Klasse OBERFLAECHE hat das Zugriffsrecht `private` und kann damit nicht von außen verwendet werden. Beim Aufruf der Methode `OBERFLAECHE.FensterGeben()` wird ein Objekt der Klasse OBERFLAECHE automatisch erzeugt. Durch dieses Konstrukt wird sichergestellt, dass es im ganzen Programm nur ein Objekt der Klasse OBERFLAECHE und somit nur ein Anzeigefenster geben kann.

Die Methode `FensterGeben()` ist eine besondere Methode. Sie gehört zu keinem Objekt und kann verwendet werden, ohne dass ein Objekt der Klasse OBERFLAECHE existiert. Solche Methoden nennt man Klassenmethoden oder statische Methoden. In Java werden sie durch das Schlüsselwort `static` vor dem Rückgabetyp der Methode gekennzeichnet. Nur mithilfe einer solchen Methode können auch die Klassen KREIS und RECHTECK ohne Probleme „das eine“ Fenster benutzen.

Die Methode `FensterGeben()` erzeugt das Objekt der Klasse OBERFLAECHE, wenn sie zum ersten Mal verwendet wird. Die Referenz darauf speichert sie in einem ebenfalls von

Objekten unabhängigen statischen Attribut, aus historischen Gründen oft auch globale Variable genannt. Zur Vereinbarung statischer Attribute wird in Java ebenfalls das Schlüsselwort `static` verwendet.

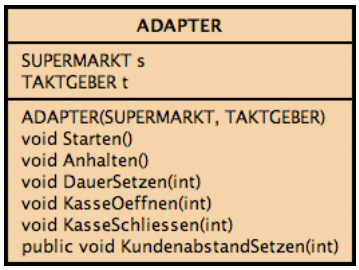
```
private static OBERFLAECHE o = null;
    ...
static Frame FensterGeben()
{
    if (o == null)
    {
        o = new OBERFLAECHE();
    }
    return o.fenster;
}
```

Für die Behandlung dieses Themenbereichs im Unterricht bietet es sich an, den Quelltext der Klassen OBERFLAECHE, KREIS und RECHTECK zu analysieren und so sowohl die Semantik zu erschließen als auch die syntaktischen Mittel kennenzulernen. Dabei können der Nutzen globaler Variablen aber auch die durch unkontrollierten Zugriff entstehenden Gefahren thematisiert werden, wie z. B. die Gefahr einer unerwünschten Änderung durch unmittelbare Zugriffsmöglichkeiten auch von anderen Klassen aus.

Weitergeben der Ereignisse

In diesem (optionalen) Teilkapitel wird dem übergeordneten Prinzip der losen Kopplung eigenständiger Programmteile mehr Raum gegeben als es der Lehrplan bzw. der sichere Umgang mit objektorientierter Modellierung fordert. Es gibt einen ersten Einblick in die Ansätze von Softwaremustern, die in der folgenden Jahrgangsstufe ausführlicher thematisiert werden. Gerade im Bereich der Oberflächenanbindung wird aber aus vordergründiger Bequemlichkeit („geht doch kürzer“) oft sehr schlampig gearbeitet, so dass sich hier ein sehr guter Aufhänger für Diskussionen über Programmentwurf ergibt. Will man diese Diskussion z. B. aus Zeitgründen oder anderer Schwerpunktsetzung nicht führen, können die Ereignisse natürlich direkt an den Supermarkt bzw. den Taktgeber weitergereicht werden.

Die Knöpfe der Oberfläche müssen Botschaften an den Supermarkt und an den Taktgeber senden. Auch hier kann mit den Schülern wieder gut das Thema der losen Kopplung der Klassen angesprochen werden: diejenigen Klassen, die das eigentlich Programm darstellen (Logikklassen, im Englischen oft model) werden möglichst weit von den Klassen der Oberfläche getrennt. Das hat z. B. den Vorteil, dass die Oberfläche jederzeit ausgetauscht werden kann, dass Veränderungen im Programmaufbau die Oberfläche nicht beeinflussen oder dass die Oberfläche auf einem andere Computer laufen könnte als das eigentliche Programm.



Diese Trennung ist hier sehr leicht umzusetzen: Zwischen Oberfläche und Programmlogik wird eine Hilfsklasse ADAPTER gesetzt. Diese Klasse empfängt alle Botschaften der Oberfläche und verteilt sie an die eigentlichen Adressaten, den Taktgeber und den Supermarkt. Die Adressaten des Adapters werden ihm im Konstruktor mitgegeben.

Da aus den oben angeführten Gründen nicht direkt auf das Oberflächenobjekt zugegriffen werden kann, muss der Adapter in der Oberfläche über eine statische Methode bekanntgemacht werden. Diese Methode `AdapterSetzen()` erzeugt gegebenenfalls das Oberflächenobjekt (analog zu `FensterGeben()`) und besetzt dann ein geeignetes Attribut.

Für die Botschaften zum Setzen des mittleren Kundenabstands oder des Taktimpulsabstands müssen die Schüler noch erfahren, wie sie den eingegebenen Text in eine Zahl konvertieren

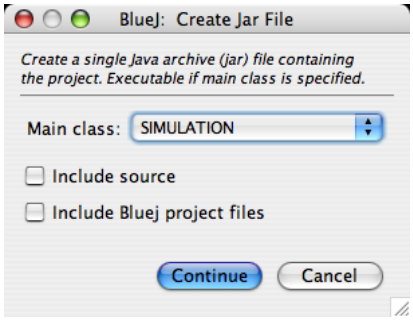
können. Da Java hier aus Sicherheitsgründen die Behandlung einer möglichen Ausnahmesituation („Text kann nicht konvertiert werden“) erzwingt, empfiehlt es sich, den Schülern das entsprechende Programmstück zur Verfügung zu stellen.

```
try
{
    a. DauerSetzen (Integer. parseInt (eingabe2. getText ()));
}
catch (Exception ex)
{
    (Toolkit. getDefaultToolkit ()). beep ();
}
```

Wenn die Methode `parseInt()` den Text nicht konvertieren kann, erzeugt sie eine Ausnahmesituation. Diese wird dann im zweiten Teil (ab dem Schlüsselwort `catch`) aufgegriffen. Die dort angegebenen Methodenaufrufe erzeugen einen Piepston, um dem Benutzer den Fehler zu signalisieren.

Eigenständiges Programm

Noch müssen die zentralen Objekte Supermarkt, Taktgeber und Oberfläche „von Hand“ erzeugt und verknüpft werden. Diese Aufgabe kann einer eigenen Klasse SIMULATION zugewiesen werden. Im Konstruktor dieser Klasse werden die entsprechenden Anweisungen festgelegt. Damit genügt es, ein Objekt der Klasse SIMULATION zu erzeugen, um das Programm zu starten.



Der jetzt erreichte Stand ist bereits sehr einfach zu verwenden, für den Nicht-Informatiker aber immer noch nicht geeignet. Er benötigt eine ausführbare Datei, d. h., das Programm muss durch Doppelklick auf diese Datei zur Ausführung gebracht werden können. Wie bei den meisten Programmiersprachen ist das auch bei Java möglich. Dazu muss in einer von der Sprache festgelegten Art angegeben werden, an welcher Stelle mit der Programmausführung begonnen wird. In Java wird dazu die Klassenmethode `main()` verwendet. Diese wird sinnvoller Weise ebenfalls in der Klasse SIMULATION vereinbart.

```
class SIMULATION
{
    // Konstruktor
    private Simulation()
    {
        ...
    }
    // Programmstart
    public static void main (String [] args)
    {
        new SIMULATION ();
    }
}
```

Um die ausführbare Datei zu erzeugen, wird in BlueJ der Menüpunkt „Project → Create Jar File …“ aufgerufen. Zunächst wird gefragt, in welcher Klasse sich die Methode `main()` befindet. Anschließend werden Dateiname und Speicherort angegeben.

Hefteintrag

In dieser Einheit empfiehlt sich die Ausgabe eines Arbeitsblattes anstelle des Hefteintrags.

1. Benutzerschnittstelle

a) Ergänze die Klasse OBERFLAECHE um die notwendigen Bedienelemente; die Ereignisbehandlung wird momentan noch weggelassen.

b) Erzeuge ein Objekt dieser Klasse, um die korrekte Anordnung zu überprüfen.

2. Die Klasse ADAPTER

a) Ergänze die Klasse SUPERMARKT um eine Methode `KundenabstandSetzen(int zeit)`, mit der die durchschnittliche Wartezeit zwischen zwei Kunden festgelegt werden kann. Die durchschnittliche Wartezeit wird in einem Attribut `mittlereWartezeit` gespeichert, im Konstruktor mit dem bisherigen Wert 50 vorbesetzt und in der Methode `TaktImpulsAusfuehren()` verwendet.

b) Implementiere die Klasse ADAPTER gemäß des angegebenen Klassendiagramms.

c) Ergänze die Klasse OBERFLAECHE um eine statische Methode `AdapterSetzen()`, die das neue Attribut für den Adapter setzt. Beachte, dass auch diese Methode gegebenenfalls das Oberflächenobjekt erzeugen muss.

d) Füge die Ereignisbehandlung für die Knöpfe hinzu.

e) Teste durch Anlegen der benötigten Objekte.

3. Das fertige Programm mit Benutzerschnittstelle

a) Ergänze die Klasse SIMULATION zunächst ohne die Methode `main()`.

b) Teste das fertige Programm.

c) Ergänze die Methode `main()`. Den Konstruktor der Klasse SIMULATION kannst du jetzt als `private` festlegen. Warum geht das? Welche Vorteile bringt das?

d) Exportiere das fertige Programm als JAR-Datei und teste sie ohne BlueJ.

Arbeitsmaterial

Für die Durchführung dieser Einheit ist eine Reihe von Arbeitsmaterialien vorhanden, die vollständig auf der Begleit-CD vorhanden sind. Dazu gehören:

1. Das BlueJ-Projekt zum Einstieg und je ein BlueJ-Projekt als Beispiellösung am Ende jeder Einheit

BlueJ-Projekt	Zuordnung zur Unterrichtssequenz
Supermarkt1	Startprojekt (Einheit 2)
Supermarkt2	Stand nach Einheit 3
Supermarkt3	Stand nach Einheit 4
Supermarkt4	Stand nach Einheit 5 Teil 1
Supermarkt5	Stand nach Einheit 5
Supermarkt6	Stand nach Einheit 6
Supermarkt7	Stand nach Einheit 7
Supermarkt8	Lösung für Einheit 7a
Supermarkt9	Stand nach Einheit 8
Supermarkt10	Stand nach Einheit 9
Supermarkt11	Stand nach Einheit 10
Supermarkt12	Stand nach Einheit 11
Supermarkt13	Stand nach Einheit 12
Supermarkt14	Stand nach Einheit 13

2. Kurze, ausbaufähige Präsentationen für Impulse oder Veranschaulichungen

   Die Präsentationen sind sowohl im Open-Dokument-Format (Endung .odp) als auch im propriäteren Microsoft-Format (Endung .ppt) vorhanden.

   a) Modellierung.odp / Modellierung.ppt
      Präsentation zum Modellierungsbegriff

   b) Supermarkt.odp / Supermarkt.ppt
      Präsentation mit dem „groben“ Bild des Kassenbereichs eines großen Supermarkts

3. Arbeitsblätter

   a) Arbeitsblatt für Einheit 2

   b) Arbeitsblatt für Einheit 8 Anfangsstand

   c) Arbeitsblatt für Einheit 8

   d) Arbeitsblatt für Einheit 13 Teil 1

   e) Arbeitsblatt für Einheit 13 Teil 2

   f) Arbeitsblatt für die Zusammenfassung der Kontrollstrukturen