5 Komplexbeispiel-Quellen

Object subclass: #Product
instanceVariableNames: 'name availableQuantity unitVolume demandLog '
classVariableNames: ''
poolDictionaries: ''

Product comment: 'Diese Klasse entspricht der Klasse **Produkt** aus der OBA-Analyse.

Instanzvariablen:
name <Symbol>
Die Produktbezeichnung innerhalb des Unternehmenssortiments, 'Name'.
availableQuantity <Number>
Im Lager aktuell verfuegbare Menge des Produktes, 'Bestand'.
unitVolume <Number>
Raumbedarf des Produktes pro Mengeneinheit, 'Volumen'.
demandLog <OrderedCollection>
Menge der Bestellposten fuer das Produkt in der Reihenfolge
des Eingangs, 'Nachfragejournal'.'

Product class publicMethods

```
name: aSymbol
    "Ein neues Produkt wird erstellt."
    | product |
    product := super new initialize. product name: aSymbol.
    ^product

new
    ^self shouldNotImplement
```

Product publicMethods

```
add: aNumber
    "Entspricht der Dienstleistung 'Bestandsaenderung', Teil 1."
    availableQuantity := availableQuantity + aNumber

availableQuantity
    "Zugriff 'Bestand'."
    "Im Lager aktuell verfuegbare Mengen des Produkts."
    ^availableQuantity

initialize
    "Instanzvariablen werden initialisiert."
    name := nil.
    availableQuantity := 0.
    unitVolume := 1.
```

```
    demandLog := SortedCollection sortBlock:
                    [:position1 :position2 | position1 date <= position2 date].

logOrderPosition: anOrderPosition
    "Entspricht der Dienstleistung 'Verbuchung-Nachfrage'."
    "anOrderPosition wird verbucht, um die Nachfragerate berechnen zu koennen."
    demandLog add: anOrderPosition

name
    "Zugriff 'Name'."
    "Die Produktbezeichnung wird zurueckgegeben."
    ^name

name: aSymbol
    "Die Produktbezeichnung wird festgelegt."
    name := aSymbol

printString
    "Gibt die im Lager verfuegbare Menge eines Produktes aus."
    ^'P# ', (self name asString , ': ', self availableQuantity printString) , 'avail.'

release
    "Loescht Datenspeicher eines Objekts."
    name := nil.
    availableQuantity := nil.
    unitVolume := nil.
    demandLog := nil.

remove: aNumber
    "Entspricht der Dienstleistung 'Bestandsaenderung', Teil 2."
    availableQuantity := availableQuantity - aNumber

unitVolume
    "Zugriff 'Einheitdvolumen'."
    "Der Raumbedarf des Produkts wird zurueckgegeben."
    ^unitVolume

unitVolume: aNumber
    "Der Raumbedarf des Produkts wird festgelegt."
    unitVolume := aNumber

Product privateMethods

demandRate
    "Private - Liefert die Nachfragerate nach dem Produkt.
    Es wird der Durchschnittsbedarf ueber alle Bestellungen hinweg verwendet. Der
Bedarf
    der ersten Bestellung ist abzuziehen, da diese lediglich das erste Datum fuer die
    Berechnung des Zeitraummes liefert."
```

```
    | totalDemand totalTime |
    demandLog size <= 1 ifTrue: [^self error: 'Unsufficient data ti determine demand
rate.'].
    totalDemand := demandLog inject: 0 into: [:total :position | total + position
quantity].
    totalDemand := totalDemand - demandLog first quantity.
    totalTime := demandLog last date
                    subtractDate: demandLog first date.
    ^totalDemand / totalTime
```

Object subclass: #OrderPosition
instanceVariableNames: 'product quantity date '
classVariableNames: ''
poolDictionaries: ''

OrderPosition comment: 'Diese Klasse entspricht der Klasse **Bestellposten** aus der OBA-Analyse.

Instanzvariablen:
    product <Product>
        Bestelltes 'Produkt'.
    quantity <Number>
        Bestellte 'Menge'.
    date <Date>
        'Datum' der Bearbeitung.
'

OrderPosition class publicMethods

```
new
    "Instanzvariablen werden initialisiert."
    ^super new initialize
```

```
OrderPosition class categoriesFor: #new are:((OrderedCollection new)
    add: 'Create';
    yourself)
```

OrderPosition publicMethods

```
date
    "Zugriff 'Datum'."
    "Bestimmt das Bestelldatum des bestellten Produkts."
    ^date
```

```
date: aDate
    "Legt das Bestelldatum des bestellten Produkts fest."
    date := aDate
```

```
initialize
    "Anfangswerte werden gesetzt."
    product := nil.
    quantity := 0.
    date := Date today

logOrderPosition
    "Entspricht der Dienstleistung 'Verbuchung-Bestellung'."
    self product logOrderPosition: self!

printString
    "Listet die Bestellung auf."
    ^self date printString , ': ' , self quantity printString ,
        ' units of ' , self product printString

product
    "Zugriff 'Produkt'."
    "Ermittelt das bestellte Produkt."
    ^product

product: aProduct
    "Legt das bestellte Produkt fest."
    product := aProduct

quantity
    "Ermittelt die bestellte Menge des Produkts."
    ^quantity

quantity: aNumber
    "Legt die bestellte Menge des Produkts fest."
    quantity := aNumber

release
    "Objektdaten werden geloescht."
    product := nil.
    quantity := nil.
    date := nil
```

Object subclass: #CustomerOrder
instanceVariableNames: 'orderPositions '
classVariableNames: ''
poolDictionaries: ''

CustomerOrder comment: 'Diese Klasse entspricht der Klasse **Kundenauftrag** aus der OBA-Analyse.

Instanzvariable:
orderPositions <OrderedCollection>
Menge der 'Bestellposten' eines Kundenauftrages.'

CustomerOrder class publicMethods

```
new
    "Instanzvariablen werden initialisiert."
    ^super new initialize
```

CustomerOrder publicMethods

```
addPosition: anOrderPosition
    "Erweiterung des Kundenauftrages."
    orderPositions add: anOrderPosition

initialize
    "Anfangswerte werden gesetzt."
    orderPositions := OrderedCollection new

orderPositions
    "Entspricht der Dienstleistung 'Bestellposten'.
    Ausgabe des Kundenauftrags."
    ^orderPositions

release
    "Objektdaten werden geloescht."
    orderPositions do: [:position | position release].
    orderPositions := nil

removePosition: anOrderPosition
    "Erweiterung des Kundenauftrages."
    orderPositions remove: anOrderPosition ifAbsent:[]
```

```
Object subclass: #TransportationUnit
   instanceVariableNames: 'customerOrder contents '
   classVariableNames: 'Volume '
   poolDictionaries: ''
```

TransportationUnit comment: 'Diese Klasse entspricht der Klasse
Transportbehaelter
aus der OBA-Analyse.

Instanzvariablen:
   customerOrder <CustomerOrder>
      Der 'Kundenauftrag', dessen Produkte im Behaelter abgelegt sind.
   contens <IdentityDictionery>
      'Inhalt' des Transportbehaelters.'

Klassenvariablen:
   Volume <Number>
      Das 'Volumen' eines einzelnen Transportbehaelters.

TransportationUnit class publicMethods

```
initialize
   "TransportUnit initialize."
   Volume := 100

new
   "Instanzvariablen werden initialisiert."
   ^super new initialize

volume
   "Zugriff 'Volumen'."
   "Das Fassungsvermoegen wird zurueckgegeben."
   ^Volume
```

TransportationUnit publicMethods

```
customerOrder
   "Entspricht der Dienstleitung 'Kundenauftrag'."
   ^customerOrder

customerOrder: aCustomerOrder
   "Der Kundenauftrag wird dem Transportbehaelter zugeordnet."
   customerOrder := aCustomerOrder

freeSpace
   "Liefert das noch freie Packvolumen im Behaelter."
   | free |
   free := self class volume.
   contents keysAndValuesDo: [:product :amount |
```

```
                                        free := free - (product unitVolume * amount)].
    ^free


initialize
    "Anfangswerte werden gesetzt."
    customerOrder := nil.
    contents := IdentityDictionary new

pack: aProduct amount: aNumber
    "Entspricht der Dienstleistung 'Kommissionieren'."
    "Es wird die Menge zurueckgegeben, die nicht
    in den Behaelter verpackt werden konnte."
    | free actualAmount currentAmount |
    (free := self freeSpace) < (aNumber * aProduct unitVolume)
        ifTrue: [actualAmount := free // aProduct unitVolume]
        ifFalse: [actualAmount := aNumber].
    currentAmount := contents at: aProduct ifAbsent:[0].
    contents at: aProduct put: currentAmount + actualAmount.
    ^aNumber - actualAmount

printString
    "Gibt den Inhalt eines Transportbehaelters aus."
    ^'TransportUnit. Free: ', self freeSpace printString

release
    "Objektdaten werden geloescht."
    customerOrder := nil.
    contents := nil
```

Object subclass: #SupplyPalette
instanceVariableNames: 'product suppliedQuantity '
classVariableNames: ''
poolDictionaries: ''

SupplyPalette comment: 'Diese Klasse entspricht der Klasse **Palette** aus der OBA-Analyse.

Instanzvariablen:
product <Product>
Das auf der Palette liegende 'Produkt'.
suppliedQuantity <Number>
Die 'Liefermenge' des Produktes.'

SupplyPalette class publicMethods

```
new
    "Instanzvariablen werden initialisiert."
```

```
    ^super new initialize

SupplyPalette publicMethods

initialize
    "Anfangswerte werden gesetzt."
    product := nil.
    suppliedQuantity := 0

printString
    "Gibt Produkt und Menge auf der Palette aus."
    ^'Palette with ', self suppliedQuantity printString ,
        ' units of (', self product printString , ')'

product
    "Entspricht der Dienstleistung 'Produkt'."
    ^product

product: aProduct
    "Palette wird mit angegebenes Produkt bestueckt."
    product := aProduct

release
    "Objektdaten werden geloescht."
    product := nil.
    suppliedQuantity := nil

suppliedQuantity
    "Zugriff 'Liefermenge'."
    "Liefermenge des entsprechenden Produkts auf der Palette."
    ^suppliedQuantity!

suppliedQuantity: aNumber
    "Setzt Liefermenge des Produkt auf der Palette fest."
    suppliedQuantity := aNumber
```

Object subclass: #StorageShelf
instanceVariableNames: 'palette storedQuantity '
classVariableNames: ''
poolDictionaries: ''

StorageShelf comment: 'Diese Klasse entspricht der Klasse **Regalfach** aus der OBA-Analyse.

Instanzvariablen:
palette <SupplyPalette> Die im Regalfach liegende 'Palette'.
storedQuantity <Number> Im Regalfach enthaltene 'Fuellmenge' des Produktes.'

StorageShelf class publicMethods

new
"Instanzvariablen werden initialisiert."
^super new initialize

StorageShelf publicMethods

clear
"Entspricht der Dienstleistung 'Leeren'."
palette := nil. storedQuantity := 0

initialize
"Anfangswerte werden gesetzt."
palette := nil. storedQuantity := 0

isEmpty
"Das Regalfach ist leer."
^palette isNil

pick: aNumber
"Entspricht der Dienstleistung 'Entnehmen'."
aNumber <= storedQuantity ifFalse: [^self error: 'No such amount available in shelf.'].
storedQuantity := storedQuantity - aNumber

product
"Zugriff 'Produkt'. Das auf der Palette im Regalfach liegende Produkt."
^palette product

release
"Objektdaten werden geloescht."
palette := nil. storedQuantity := nil

store: aSupplyPalette
"Entspricht der Dienstleistung 'Einlagern'."

```
    palette := aSupplyPalette.
    storedQuantity := aSupplyPalette suppliedQuantity

storedQuantity
    "Zugriff 'Fuellmenge'. Die im Fach liegende Menge des Produkts."
    ^storedQuantity
```

Object subclass: #StorageArea
instanceVariableNames: 'size places '
classVariableNames: ''
poolDictionaries: ''!

StorageArea comment: 'Diese Klasse entspricht der Klasse **Lagerbereich** aus der OBA-Analyse.

Instanzvariablen:
size <Point>
'Groesse' des Lagerbereiches.
places <Dictionary>
'Belegt'e Plaetze bzw Faecher des Lagerbereiches.'

StorageArea class publicMethods

```
new
    "Objekte der Klasse sollten nicht erzeugt werden."
    ^self shouldNotImplement

new: aPoint
    "Erzeugt einen neuen Lagerbereich mit der angegebenen Ausdehnung"
    ^super new initialize: aPoint
```

StorageArea publicMethods

```
at: aPoint
    "Inhalt des Platzes aPoint."
    aPoint <= self size ifFalse: [^self error: 'No such place.'].
    ^places at: aPoint ifAbsent: []

choosePlace
    "Entspricht der Dienstleistung 'Auswahl'."
    1 to: self size x do:
        [:u | 1 to: self size y do:
            [:v | (self isUsedAt: u @ v) ifFalse: [^u @ v]]].
    ^nil

initialize: aPoint
    "Instanzvariablen werden initialisiert."
    size := aPoint.
```

```
    places := Dictionary new

isUsedAt: aPoint
    "Zugriff 'Belegt'."
    "Liefert true, falls Platz aPoint besetzt, sonst false."
    ^(places at: aPoint ifAbsent:[]) notNil

placeOf: anObject
    "Sucht das Objekt anObject und gibt den Platz,
     auf dem es gefunden wurde, zurueck."
    ^places keyAtValue: anObject ifAbsent: []

release
    "Objektdaten werden geloescht."
    size := nil.
    places := nil

size
    "Zugriff 'Groesse'."
    "Gibt die Ausdehnung des Lagerbereiches an."
    ^size

totalSpace
    "Anzahl der verfuegbaren Plaetze"
    ^self size x * self size y

usedPlacesDo: aBinaryBlock
    "Wertet aBinaryBlock fuer jeden belegten Platz aus.
    aBinaryBlock muss genau zwei Argumente haben. Das erste
    Argument ist die Koordinate des Platzes und das zweite
    Argument ist der Platz selbst."
    places keysAndValuesDo:
        [ :point :place | aBinaryBlock value: point value: place]

usedSpace
    "Anzahl der belegten Plaetze"
    ^places keys size
```

StorageArea subclass: #Storage
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''

Storage comment: 'Diese Klasse entspricht der Klasse **Regal** aus der OBA-Analyse.'

Storage publicMethods

```
initialize: aPoint
    "Zusaetzlich muessen noch die Regalfaecher eingerichtet werden."
    super initialize: aPoint.
    1 to: aPoint x do: [:u | 1 to: aPoint y do:
        [:v | places at: u @ v put: StorageShelf new]]

isUsedAt: aPoint
    "Zugriff 'Belegt'."
    "Reimplementierung wegen Belegung der Plaetze
    durch Regalfaecher."
    ^(self at: aPoint) isEmpty not

release
    "Zusaetzlich muessen alle Regalfaecher geloescht werden."
    super usedPlacesDo: [:place | place release].
    super release

usedPlacesDo: aBinaryBlock
    "Reimplementierung wegen Belegung der Plaetze
    durch Regalfaecher."
    super usedPlacesDo: [:point :shelf | shelf isEmpty
        ifFalse: [aBinaryBlock value: point value: shelf]]

usedSpace
    "Anzahl der belegten Plaetze. Reimplementierung, da leere
    Regalfaecher gesucht werden muessen."
    ^places inject: 0 into:
        [:subTotal :next | subTotal +
            (next isEmpty ifTrue: [0] ifFalse: [1])]
```

StorageArea subclass: #WarehouseBuffer
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''

WarehouseBuffer comment: 'Diese Klasse entspricht der Klasse **Bereitstellungsflaeche** aus der OBA-Analyse.'

WarehouseBuffer publicMethods

```
clearFrom: aTransportationUnitOrSupplyPalette
    "Entspricht der Dienstleistung 'Raeumen'."
    | place |
    place := self placeOf:  aTransportationUnitOrSupplyPalette.
    place isNil ifTrue: [^self error: 'Cannot find.'].
    places removeKey: place

drop:  aTransportationUnitOrSupplyPalette at: aPoint
    "Entspricht der Dienstleistung 'Abstellen'."
    (places at: aPoint ifAbsent: []) isNil
        ifFalse: [^self error: 'Place already used.'].
    places at: aPoint put: aTransportationUnitOrSupplyPalette

hasSufficientSpaceFor: aCollection
    "Entspricht der Dienstleistung 'Frei'."
    ^aCollection size <= (self totalSpace - self usedSpace)!

ship: aTransportationUnit
    "Entspricht der Dienstleistung 'Versenden'."
    "Hier kann der Code noch entsprechend der
    Besonderheiten des Frachtsystemsergaenzt werden."
    Transcript show: 'Order for customer is ready.'; cr
```

Object subclass: #Warehouse
instanceVariableNames: 'products storages buffer '
classVariableNames: ''
poolDictionaries: ''

Warehouse comment: 'Diese Klasse entspricht der Klasse **Lager** aus der OBA-Analyse.

Instanzvariablen:
products <Set>
Die im Lager vorhandenen 'Produkte'.
storages <Dictionary>
Die im Lager vorhandenen 'Regale'.
buffer <WarehouseBuffer>

Die 'Bereitstellungsflaeche' des Lagers.'

```
Warehouse class publicMethods

new
    "Erzeugt ein Lager."
    ^super new initialize

Warehouse publicMethods

addProduct: aProduct
    "Hinzufuegen eines neuen Produkts."
    products add: aProduct

addStorage: aStorage inRow: aNumber
    "Hinzufuegen einer neuen Regalreihe."
    (storages at: aNumber ifAbsent: []) isNil ifFalse: [^self error: 'Row already in
use.'].
    storages at: aNumber put: aStorage

buffer
    "Zugriff 'Bereitstellungsflaeche'."
    ^buffer

canSatisfy: aCustomerOrder
    "Entspricht der Dienstleistung 'Ausfuehrbar'."
    aCustomerOrder orderPositions do: [:position |
        position product availableQuantity < position quantity ifTrue: [^false]].
    ^true

chooseStorage
    "Entspricht der Dienstleistung 'Auswahl Regal'."
    "Es wird immer dasjenige Regal gewaehlt, das die geringste relative Belegung
aufweist."
    | minUsage minStore |
    minUsage := 1.0. minStore := nil.
    storages do:
        [:store | | newMin | (newMin := store usedSpace / store totalSpace) < minUsage
                                        ifTrue: [minUsage := newMin. minStore :=
store]].
    minStore isNil ifTrue: [^self error: 'No space to put item into storage.'].
    ^minStore

initialize
    "Anfangswerte werden gesetzt."
    products := EsIdentitySet new.
    storages := Dictionary new.
    buffer := WarehouseBuffer new: 1 @ 1
```

```
products
    "Zugriff 'Produkte'."
    ^products!

release
    "Objektdaten werden geloescht."
    products := nil.
    storages := nil.
    buffer := nil

shelvesToServe: anOrderPosition
    "Entspricht der Dienstleistung 'Bedienfaecher'."
    "Die Methode liefert ein Dictionary mit Elementen der Form
    Regalnummer -> Fachnummer. Es muessen soviele Faecher bestimmt werden, dass
deren
    summierte Inhalte mindestens die bestellte Menge erreichen. Dabei wird eine
einfache
    Strategie verfolgt: Das erste Fach, das noch etwas von dem bestellten Produkt
enthaelt,
    wird zur Bedienung des Bestellpostens herangezogen."
    | shelves remainingQuantity |
    shelves := Dictionary new.
    remainingQuantity := anOrderPosition quantity.
    (self products includes: anOrderPosition product)
        ifFalse: [^self error: 'Product not im marehouse.'].
    self storagesDo:
        [:row :store | store usedPlacesDo:
            [:point :shelf | shelf product == anOrderPosition product
                ifTrue:
                    [remainingQuantity := remainingQuantity -
                        shelf storedQuantity.
                    shelves at: row put: point.
                    remainingQuantity < 0 ifTrue: [^shelves]]]]

storageInRow: aNumber
    "Zugriff 'Regal'."
    "Liefert das Regal mit der entsprechenden Nummer."
    | temporaries |
    ^storages at: aNumber ifAbsent:[]

storagesDo: aBinaryBlock
    "Wertet aBinaryBlock fuer jedes Regal aus. aBinaryBlock muss genau zwei
Argumente
    haben. Das erste Argument ist die Reihe des Regals und das zweite Argument ist
das Regal
    selbst."
    storages keysAndValuesDo: [:row :store | aBinaryBlock value: row value: store]! !
```

Warehouse class publicMethods

example1
```
    "Ankunft einer Lieferung aus der Produktion.
    Warehouse example1"
    | warehouse supply store place | warehouse := Warehouse new.

    "Bereitstellflaeche wird eingerichtet."
    warehouse buffer initialize: 10 @ 4.

    "Regale werden installiert."
    1 to: 5 do: [:i | warehouse addStorage: (Storage new: 5 @ 4) inRow: i].

    "Naechste Lieferung wird erzeugt."
    supply := Set new.
    supply add: ((SupplyPalette new) product: (Product name: #'00001');
        suppliedQuantity: 100).
    supply add: ((SupplyPalette new) product: (Product name: #'00002');
        suppliedQuantity: 200).
    supply add: ((SupplyPalette new) product: (Product name: #'00003');
        suppliedQuantity: 300).
    supply add: ((SupplyPalette new) product: (Product name: #'00004');
        suppliedQuantity: 400).
    supply add: ((SupplyPalette new) product: (Product name: #'00005');
        suppliedQuantity: 500).

    "Pruefe auf Bereitstellungsflaeche, ob genug Stellflaeche vorhanden ist."
    (warehouse buffer hasSufficientSpaceFor: supply)
        ifFalse: [^self error: 'Not enough space to eccapt supply.'].

    "Lieferung wird entgegengenommen."
    supply do:
        [:palette |
            place := warehouse buffer choosePlace.
            warehouse buffer drop: palette at: place].

    "Zustand des Lagers wird mittels Inspektor gezeigt."
    warehouse inspect.

    "Lieferung wird in die Regale eingeordnet."
    supply do:
        [:palette |
            "Regal und Fach wird festgelegt und Palette abgelegt."
            store := warehouse chooseStorage.
            place := store choosePlace.
            (store at: place) store: palette.
            "Produkt und Menge auf Palette wird registriert."
            warehouse addProduct: palette product.
            palette product add: palette suppliedQuantity.
```

```
            "Platz auf Bereitstellungsflaeche wird wieder freigegeben."
            warehouse buffer clearFrom: palette].
    ^warehouse


example2
    "Bearbeiten eines Kundenauftrags.
    Warehouse example2"
    | warehouse order randomGenerator place currentUnit
        transUnitsToShip remainingQuantity shelf shelves |
    warehouse := Warehouse example1.
    randomGenerator := EsRandom new.
    order := CustomerOrder new.

    "Kunderauftrag wird zufaellig, eintsprechend der zur Verfuegung stehenden
Produkte
    zusammengestellt."
    warehouse products do:
        [:product | |position|
            position := OrderPosition new.
            position product: product;
                quantity: (randomGenerator next * 100) rounded.
        order addPosition: position].

    "Lieferfaehigkeit wird ueberprueft."
    (warehouse canSatisfy: order)
        ifFalse: [^self error: 'Cannot deliver complete demand.'].

    "Kundenauftrag wird mittels Inspektor gezeigt."
    order inspect.

    "Transportbehaelter wird fuer den Kundenauftrag zur Verfuegung gestellt."
    currentUnit := TransportationUnit new.
    currentUnit customerOrder: order.

    "Platz auf der Bereistellungsflaeche wird gesucht."
    (place := warehouse buffer choosePlace) isNil
        ifTrue: [^self error: 'You have to wait for a place.'].

    "Transportbehaelter wird dort abgestellt."
    warehouse buffer drop: currentUnit at: place.

    "Transport wird zusammengestellt."
    transUnitsToShip := OrderedCollection with: currentUnit.

    "Jede einzelne Bestellung des Kundenauftrages wird bearbeitet."
    order orderPositions
        do: [:position |
```

```
        "Nachfrage nach dem Produkt wird verbucht."
        position product logOrderPosition: position.
        "Die Menge des Produkts der Bestellung wird ermittelt."
        remainingQuantity := position quantity.

        "Regale und Faecher, in denen das bestellten Produkt lagert, werden
ermittelt."
        shelves := warehouse shelvesToServe: position.
        "Die Produktmenge wird entsprchend der Bestellung aus den Faechern
        herausgenommen und in Transportbehaelter abgelegt."
        shelves
            keysAndValuesDo:
                [:row :point | | shippingQuantity |

                "Das Fach aktuelle wird festgelegt."
                shelf := (warehouse storageInRow: row) at: point.
                "Die Menge des Produktes, die aus dem Fach genommen wird, wird
bestimmt
                und entnommen."
                shippingQuantity := shelf storedQuantity
                                                min: remainingQuantity.
                shelf pick: shippingQuantity.
                "Ist die Palette leer, so wird sie aus dem Fach genommen und damit ist
das
                Fach leer."
                shelf storedQuantity = 0 ifTrue: [shelf clear].

                "Entnommene Menge wird aus der Registratur des Lagers und von der
                Bestellung gestrichen."
                position product remove: shippingQuantity.
                remainingQuantity := remainingQuantity -
                                                shippingQuantity.

                "Das Produkt wird in den Transportbehaelter abgelegt und der noch zur
                Verfuegung stehende  Platz im Behaelter bestimmt. Ist der Behaelter
voll,
                so wird ein neuer zur Verfuegung gestellt."
                [shippingQuantity :=
                    currentUnit pack: position product
                                mount: shippingQuantity.
                shippingQuantity > 0]
                    whileTrue:
                        "Bereitstellen eines neuen Behaelters."
                        [Transcript show: 'Need a new unit.'; cr.
                        currentUnit := TransportationUnit new.
                        "Registrierung eines neuen Behaelters zum Kundenauftrag."
                        transUnitsToShip add: currentUnit.
                        "Registrierung der zu ladenen Bestellung."
                        currentUnit customerOrder: order.
```

```
                    "Lagerung des Behaelters auf der Bereitstellungsflaeche der
Lagers."
                    (place := warehouse buffer choosePlace) isNil
                        ifTrue:
                        [^self error:'You have to wait for a place.'].
                    warehouse buffer drop: currentUnit at: place]]].

    "Transportbehaelter sehen auf der Bereitstellungsflaeche des Lagers bereit und
koennen
    mittels Inspektor ueberprueft werden."
    transUnitsToShip inspect.

    "Transportbehaelter werden versendet. Die Bereitstellungsflaeche wird wieder
leer."
    transUnitsToShip
        do: [:trans |
            warehouse buffer ship: trans.
            warehouse buffer clearFrom: trans].
    ^warehouse
```