W

wawi02

Vorführaufgabe 2

Name	Last update
.idea (/pg2/wawi02/tree/master/.idea)	a week ago
lib (/pg2/wawi02/tree/master/lib)	a week ago
src/geschaeftsobjekt (/pg2/wawi02/tree/master/src/geschaeftsobjekt)	a week ago
test/geschaeftsobjekt (/pg2/wawi02/tree/master/test/geschaeftsobjekt)	a week ago
.gitignore (/pg2/wawi02/blob/master/.gitignore)	a week ago
README.md (/pg2/wawi02/blob/master/README.md)	a day ago
wawi02.iml (/pg2/wawi02/blob/master/wawi02.iml)	a week ago

README.md (/pg2/wawi02/blob/master/README.md)

Entwicklung eines Kassensystems

In kleineren Shops und Restaurants haben Sie sicherlich schon einmal beobachtet, dass das Personal die Bestellungen und Abrechnungen mit Smartphones oder Tablets statt mit Registrierkassen oder PCs erledigen. Im Rahmen dieser und noch zwei darauf aufbauender Vorführaufgaben werden Sie ein stark vereinfachtest Kassensystem (auch "Point of Sale"-System, POS-System, genannt) in Java entwickeln.

Vorführaufgabe 2

Behandelte Themen

Package, Vererbung, abstrakte Klassen, Interfaces, Generics, Arrays und Collections, Enumerations

Vorbereitungen

Sofern Sie sich mit dem CVS-Tool *git* auskennen, können Sie dieses Projekt mit folgendem Terminal-Befehl klonen:

`git clone http://im-lamport.oth-regensburg.de:1080/pg2/wawi02.git` . Hierfür müssen Sie das Tool *git* von folgender Website (http://www.git-scm.com/download) downloaden und installieren.

Ansonsten kopieren dieses Projekt auf ihren lokalen Rechner, so haben Sie bereits die entsprechenden JUnit-Testcases im Projekt und können lokal testen.

Aufgabe 1: Neue Klassenstruktur mit Geschäftsobjekten

Vorbemerkung: Wichtige Daten, die innerhalb von Informationssystemen verarbeitet werden, nennt man „Geschäftsobjekte" oder auch „Entities" (Beispiele: „Kunde", „Rechnung", „Artikel", „Lieferschein", „Buchung" usw.). Das zu erstellende Warenwirtschaftssystem enthält ebenfalls Geschäftsobjekte, die als Java-Klassen definiert werden sollen und deren Ausprägungen als Objekte im Speicher liegen.

Legen Sie zunächst ein neues Package namens `geschaeftsobjekt` an und fügen Sie alle hier beschriebenen Klassen diesem Package hinzu.

Für die Verwendung von Geschäftsobjekten in Ihrem Projekt berücksichtigen Sie folgende Punkte:

- Erstellen Sie eine Klasse `Geschaeftsobjekt` .
- Ein Geschäftsobjekt hat eine Nr.
- Ein Geschäftsobjekt kann nicht ohne Nr angelegt werden.
- Die Nr kann später gelesen werden.
- Die Nr darf nach Objekterzeugung nicht mehr überschrieben werden.

Erstellen Sie aus den nachfolgenden Angaben weitere Klassen mit den entsprechenden Ableitungsbeziehungen (achten Sie hierbei auf die aus der Vorlesung bekannten Beziehungen „ist ein" und „hat ein"):

- Ein Artikel ist ein Geschäftsobjekt (kopieren Sie Ihre in Vorführaufgabe 1 erstellte Klasse `Artikel.java` entsprechend in dieses Projekt)
- Ein Kunde ist ein Geschäftsobjekt.
- Ein Kunde hat einen Namen, eine Strasse und einen Ort.
- Ein Kundenobjekt kann nur angelegt werden, wenn alle vier (Vererbung!) Attribute gegeben sind.
- Ein Produkt ist ein Geschäftsobjekt.
- Ein Produkt hat eine Bezeichnung.
- Ein Produkt hat einen Preis.
- Die Firma, für die das Warenwirtschaftssystem entwickelt wird, hat zwei generelle Arten von „Produkten": Artikel und Dienstleistung
    - Ein Artikel ist ein Produkt (Achtung, die Klasse Artikel besteht bereits und muss ggf. entsprechend angepasst werden).
    - Eine Dienstleistung ist ein Produkt.
    - Anders als ein Artikel hat eine Dienstleistung zusätzlich noch eine Einheit, auf die sich ihr Preis bezieht (z. B. „h", „qm", „lfdm").
    - Ein Produkt kann immer nur entweder ein Artikel oder eine Dienstleistung sein; Instanzen von Produkten gibt es nicht!
    - Artikel- und Dienstleistungsobjekte können nur dann angelegt werden, wenn alle drei bzw. vier Attribute (außer Kurzbezeichnung) gegeben sind.

Für alle Attribute sind entsprechend passende Getter- und (falls nicht ausgeschlossen) Setter-Methoden zu definieren. Definieren Sie auch Konstruktoren und nutzen Sie in passender Weise Konstruktorenverkettung. Für die Attribute und die Methoden gelten die aus der Vorlesung bekannten Namenskonventionen.

Aufgabe 2: Vergleich von Objekten mittels `equals` und `Comparable<T>`

Überschreiben Sie die `equals` -Methode der Klasse `Produkt` .

Dabei gelten folgende Regeln: Zwei Produkte sind immer dann gleich, wenn...

- ...sie vom selben Typ sind (eine Dienstleistung kann nie gleich einem Artikel sein) und
- ...ihre Nr übereinstimmt.

Die Klasse `Produkt` muss zusätzlich das Interface `Comparable<Produkt>` implementieren, woraus sich die Ordnungsrelation zwischen Produkten ergibt. Folgende Regeln sind hierfür zu berücksichtigen:

- Eine Dienstleistungsobjekt ist grundsätzlich als „kleiner“ anzusehen als ein Artikelobjekt.
- Haben beide den selben Typ (beide sind Dienstleistung oder beide sind Artikel) bestimmt sich die aufsteigend alphabetische Ordnung aus der Bezeichnung der Produkte.
    - Sollte der String einer Bezeichnung nicht gesetzt (null) sein, so darf dies nicht zu einem Fehler führen; null-Werte gelten entsprechend als „kleiner“ als Nicht-null-Werte.
- Haben zwei Produkte den selben Typ und die selbe Bezeichnung so entscheidet der Preis (kleinerer Preis vor größerem Preis).

Aufgabe 3: Rechnungspositionen

Eine Rechnung hat eine oder mehrere Rechnungspositionen. Bevor die Klasse `Rechnung` definiert werden kann (siehe nächste Aufgabe), muss zunächst die Klasse `Rechnungsposition` definiert werden. Hierfür sind folgende Hinweise zu berücksichtigen:

- Eine Rechnungsposition hat ein(e Referenz auf ein) Produkt, auf das sie sich bezieht (je Produkt soll es in einer Rechnung immer nur eine Rechnungsposition geben!).
- Eine Rechnungsposition hat eine ganzzahlige Anzahl des bestellten Produkts.
- Eine neue Rechnungsposition kann nur dann angelegt werden, wenn Anzahl und Produkt gegeben ist.
- Eine Rechnungsposition kann ihren Preis zurückgeben. Dies ist der Wert der Rechnungsposition.
- Die Anzahl kann verändert, d. h. gesetzt, werden.
- Eine Rechnungsposition ist kein Geschäftsobjekt.

Die Klasse `Rechnungsposition` muss die `toString()` -Methode überschreiben, die einen zweizeiligen String zurückgibt, der exakt dem unten angefügten Muster/Beispiel entspricht. Achten Sie hierbei auf die Einhaltung der Hinweise zur Formatierung.

Folgender Code-Ausschnitt...

```
Artikel a = new Artikel(526, "Laminatbodenpack Buche Klick-Fix SuperEasy", 13.99);
System.out.println(new Rechnungsposition(4, a));

Dienstleistung d1 = new Dienstleistung(123, "Parkettmontage", 75.00, "h");
System.out.println(new Rechnungsposition(20, d1));
```

```
Dienstleistung d2 = new Dienstleistung(128, "Montage Sockelleisten", 5.59, "lfdm")
System.out.println(new Rechnungsposition(331, d2));
```

...erzeugt exakt diese Ausgabe:

```
Laminatbodenpack Buche Klick-Fix S
   4     x    13,99 =        55,96
Parkettmontage
  20 h   x    75,00 =      1500,00
Montage Sockelleisten
 131 lfd x     5,59 =       732,29
```

Tipp: Nutzen Sie die Methode `format(String format, Object... args)` aus der Klasse `String`.

Aufgabe 4: Rechnungen

Als letztes Geschäftsobjekt ist die Klasse `Rechnung` zu definieren.

Folgende Hinweise sind dabei zu beachten:

- Eine Rechnung ist ein Geschäftsobjekt.
- Die Nummer (Nr) einer Rechnung muss automatisch vergeben werden.
  - Das erste Rechnungsobjekt erhält die Nr 1, das nächste die 2 usw.
  - Nach dem Neustart des Programms beginnt die erste Nr wieder bei 1.

- Eine Rechnung hat beliebig viele Rechnungspositionen.
- Eine Rechnung hat einen Kunden (auf den sie ausgestellt wird/ist).
- Eine Rechnung hat einen Rechnungsstatus.
- Ein Rechnungsstatus kann nur einen der folgenden Status annehmen: `IN_ANLAGE`, `GEBUCHT`, `CANCELLED` (Dies ist entsprechend als Aufzählungstyp zu definieren!)
- Eine neu erstellte Rechnung ist zunächst `IN_ANLAGE`.
- Der Rechnungsstatus kann gelesen, von außen aber nicht gesetzt werden.
- Eine neue Rechnung kann entweder ohne weitere Daten erstellt werden oder mit Übergabe eines Kundenobjekts.
- Ein nachträgliches Setzen des Kunden ist nicht möglich.

Der Konstruktor sowie die Getter- und Setter sind entsprechend zu implementieren.

Zusätzlich stellt eine Rechnung noch folgende Methoden zur Verfügung:

- `public Rechnungsposition addRechnungsposition(int anzahl, Produkt p)`
  - Der Rechnung muss eine Rechnungsposition für das übergebene Produkt mit der übergebenen Anzahl hinzugefügt werden.
  - Besteht für dieses Produkt bereits eine Rechnungsposition, so ist in dieser die Anzahl entsprechend zu erhöhen.
  - Die Methode gibt eine Referenz auf die neue/geänderte Rechnungsposition zurück.
  - Ist der Lagerbestand eines Artikels hierfür nicht ausreichend, so wird keine Veränderung bzw. Neuanlage vorgenommen; als Referenz wird null bzw. die unveränderte Rechnungsposition

zurückgegeben.
    - der Lagerbestand eines Artikels darf in dieser Methode nicht verändert werden!
    - Eine Rechnungsposition kann nur im Rechnungsstatus IN_ANLAGE hinzugefügt werden (ansonsten wird null zurückgegeben).

- `public Rechnungsposition getRechnungsposition(Produkt p)`
    - Gibt die Rechnungsposition zu einem übergebenen Produkt zurück; Hinweis: zum Finden des Produkts nutzen Sie die equals-Methode, die in Aufgabe 2 überschrieben werden musste.

- `public int getAnzahlRechnungspositionen()`
    - Liefert die Anzahl der Rechnungspositionen in der Rechnung zurück.

- `public double getGesamtpreis()`
    - Berechnet die Gesamtsumme aus allen Rechnungspositionen in der Rechnung und gibt diesen Wert zurück; Mehrwertsteuer oder ähnliches ist dabei zu ignorieren!

- `public List<Rechnungsposition> getRechnungspositionen()`
    - Liefert eine Liste über den Typ Rechnungsposition, in der alle Rechnungspositionen der Rechnung enthalten sein müssen.

- `public void buchen()`
    - „Verbucht“ die Rechnung. Das heißt in diesem Fall, dass
    - alle Lagerbestände der Artikel verändert werden müssen, für die es eine Rechnungsposition gibt (nutzen Sie hierzu die bereits bestehende Methode auslagern aus Vorführaufgabe 1) und
    - sich der Rechnungsstatus entsprechend verändert.
    - Eine Rechnung kann nur im Rechnungsstatus IN_ANLAGE verbucht werden, in anderen Fällen verändert der Aufruf keine Daten.
    - Das Stornieren („Canceln“) einer Rechnung muss nicht implementiert werden.

Die Klasse `Rechnung` muss die `toString()` -Methode überschreiben, die einen mehrzeiligen String zurückgibt, der exakt dem unten angefügten Muster/Beispiel entspricht. Achten Sie hierbei auf die Einhaltung der Hinweise zur Formatierung.

Beispiel 1: Folgender Code-Ausschnitt...

```
Kunde k = new Kunde(42, "Max Muster", "Galgenbergstr. 32", "93053 Regensburg");
Rechnung re = new Rechnung(k);

Artikel a = new Artikel(526, "Laminatbodenpack Buche Klick-Fix SuperEasy", 13.99);
a.einlagern(10);
Dienstleistung d1 = new Dienstleistung(123, "Parkettmontage", 75.00, "h");
Dienstleistung d2 = new Dienstleistung(128, "Montage Sockelleisten", 5.59, "lfdm")

re.addRechnungsposition(4, a);
re.addRechnungsposition(20, d1);
re.addRechnungsposition(131, d2);

System.out.println(re);
```

...erzeugt exakt diese Ausgabe:

```
Rechnung: 1
Kunde: 42
Max Muster
Galgenbergstr. 32
93053 Regensburg

Laminatbodenpack Buche Klick-Fix S
   4     x    13,99 =        55,96
Parkettmontage
  20 h   x    75,00 =      1500,00
Montage Sockelleisten
 131 lfd x     5,59 =       732,29
---------------------------------
                           2288,25
```

Beispiel 2: Folgender Code-Ausschnitt...

```
Rechnung re1 = new Rechnung();   // kein Kunde gesetzt, wird "Barverkauf"
Rechnung re2 = new Rechnung();   // die Nr der Rechnung erhöht sich automatisch

Artikel a = new Artikel(526, "Laminatbodenpack Buche Klick-Fix SuperEasy", 13.99);
a.einlagern(3);
Dienstleistung d1 = new Dienstleistung(123, "Parkettmontage", 75.00, "h");

re1.addRechnungsposition(4, a);   // Lagerbestand nicht ausreichend, wird ignorier
re1.addRechnungsposition(20, d1);
re1.addRechnungsposition(1, d1);  // muss der Parkettmontage noch eine h hinzufueg

re2.addRechnungsposition(3, a);   // Lagerbestand diesmal ausreichend!

System.out.println(re1);
System.out.println();             // leere Zeile ausgeben
System.out.println(re2);
```

...erzeugt exakt diese Ausgabe:

```
Rechnung: 1
Barverkauf

Parkettmontage
  21 h   x    75,00 =      1575,00
---------------------------------
                           1575,00

Rechnung: 2
Barverkauf
```

```
Laminatbodenpack Buche Klick-Fix S
   3     x    13,99 =        41,97
---------------------------------
                              41,97
```

Aufgabe 5: Serialisierung und Deserialisierung von Objekten

Bei der Serialisierung werden die Objektstruktur und der Zustand eines Objekts (seine Attribute mit den aktuellen Werten) in ein binäres Format gesichert, um sie beispielsweise auf Festplatte zu speichern oder über das Netzwerk versenden zu können. Die Umwandlung in das binäre Format übernimmt dabei die Java-Laufzeitumgebung (JVM).

Im Rahmen des Praktikums wird die Serialisierung genutzt, um das Warenwirtschaftssystem beim Systemstart mit bereits bestehenden Artikel zu initialisieren.

Hierfür sind nur zwei geringe Vorbereitungen Ihrerseits notwendig:

1. Geschäftsobjekte müssen für Java als „serialisierbar“ gekennzeichnet werden. Dies geschieht ganz einfach dadurch, dass die Klasse Geschaeftsobjekt das Interface Serializable aus dem Paket java.io implementiert. (Serializable ist ein sog. „kennzeichnendes Interface“, es enthält keinerlei abstrakte Methoden; es muss nichts weiter implementiert werden!)

2. Fügen Sie der Klasse Produkt folgende statische Methode hinzu:

```
public static List<Produkt> loadProducts(String SERIALIZATION_PATH){
    List<Produkt> produkte = new LinkedList<>();
    try {
        // oeffnet die Datei "data/Produkt.ser" zum Lesen
        InputStream ins  = new FileInputStream(SERIALIZATION_PATH);
        ObjectInputStream objin = new ObjectInputStream(ins);
        // als erstes Objekt wird ein Integer-Objekt gelesen,
        // das angibt, wie viele Produkt-Objekte kommen werden
        int i=objin.readInt();
        int z=0;
        while(z++ < i){
            // lese das nächste Produkt-Objekt aus der Datei
            Produkt p = (Produkt) objin.readObject();
            produkte.add(p);  // und füge es der Liste hinzu
        }
        // die Datei muss wieder geschlossen/freigegeben werden
        objin.close();
        ins.close();
    } catch (Exception e) {
        e.printStackTrace(); // im Fehlerfall Fehlermeldung ausgeben
    }
    return produkte; // Liste der Produkte zurueckgeben
}
```

Tests

Dieses Projekt enthält mehrere Testklassen, die (analog der Tests im Praktomat) Sie selbst lokal ausführen können. Mit einem rechten Mausklick auf das Verzeichnis `test` können Sie alle Test entsprechend über das Kontextmenü starten. Die Testergebnisse (grün oder rot) sehen Sie in einem entsprechenden Fenster der Entwicklungsumgebung.

Abgabe

Die Abgabe und automatische Testierung erfolgt über den *Praktomat* (http://praktomat3.oth-regensburg.de), indem Sie dort all Ihre entwickelten Klasse (inkl. `Artikel.java`) einzeln oder als ZIP-Datei hochladen.