SS 2012

Fakultät Angewandte Informatik
Lehrprofessur für Informatik

10.05.2012

Prof. Dr. Robert Lorenz

Übungen zur Vorlesung Informatik II, Blatt 4

Abgabe: **Mittwoch, 17.05, 14.00 Uhr, (Ausnahme!!!)** Postkasten der Veranstaltung und Mail an Tutor
Dieses Übungsblatt muss im Team abgegeben werden.
Bitte Namen und Matrikelnummer aller Teammitglieder angeben.

***** leichte Aufgabe
****** mittelschwere Aufgabe
******* schwere Aufgabe

Aufgabe 13 (Klausuraufgabe Sommersemester 2010, insgesamt 20 Punkte)**
Betrachten Sie die folgende Beschreibung eines Ausschnitts von einem Verwaltungssystems der Angestellten an einer Universität.

Zu jedem Angestellten wird eine Personalnummer, der Nachname und das Geburtsdatum gespeichert. Außerdem wird erfasst, welche Angestellten Vorgesetzte von anderen Angestellten sind. Dabei hat jeder Angestellte höchstens einen Vorgesetzten, und jeder Vorgesetzte mindestens einen Mitarbeiter. Es werden zwei Gruppen von Angestellten unterschieden: wissenschaftliche Angestellte und Verwaltungsangestellte. Wissenschaftliche Angestellte sind einem Lehrstuhl zugeordnet und haben entweder ein Lehrdeputat oder arbeiten in einem Projekt. Verwaltungsangestellte sind einer Abteilung zugeordnet. Es soll möglich sein, eine Liste aller Angestellten mit Personalnummer, Nachname, Geburtsdatum und Vorgesetzten zu erstellen.

a) Modellieren Sie diesen Ausschnitt durch ein Klassendiagramm, das alle Informationen aus dem Text geeignet integriert.

   Attribute, Assoziationen und Generalisierungen modellieren Sie so detailliert wie möglich. Insbesondere wählen Sie den Typ von Attributen so spezifisch wie möglich und nehmen jeden Typ als Klasse oder Datentyp in das Klassendiagramm auf.

   Benutzen Sie, wo sinnvoll, Vererbungsbeziehungen.

   Operationen und Konstruktoren sollen keine angegeben werden. Sie sollen auch keine Informationen „hinzuerfinden“: finden sich im Text zu einer Klasse oder einem Datentyp keine Attribute im Text, so müssen auch keine angegeben werden.

b) Diskutieren Sie, wie man an welchen Stellen Vererbungsbeziehungen durch eine alternative Modellierung ersetzen könnte und inwieweit das Vor- bzw. Nachteile hätte.

Aufgabe 14**
Betrachten Sie die folgende Beschreibung eines Ausschnitts von einem Verwaltungssystems für Fahrzeugzulassungen:

Motorbetriebene Fahrzeuge sind auf Fahrzeughalter zugelassen. Es gibt unterschiedliche Arten von motorbetriebenen Fahrzeugen: Autos, Lastwägen und Motorräder. Für motorbetriebene Fahrzeuge existieren Angaben zur Fahrgestellnummer, zur Motorleistung und zur Art des Motors (Benzin- oder Dieselmotor). In Motorrädern werden ausschließlich Benzin- und in Lastwägen ausschließlich Dieselmotoren verwendet – in Autos gibt es beide Varianten, aber es kann nur eine davon verbaut sein. Die Zulassung auf einen Fahrzeughalter wird an einem bestimmten Datum in einem bestimmten Landkreis durchgeführt. Der Fahrzeughalter hat eine Adresse und kann entweder eine juristische oder eine natürliche Person sein. Bei einer juristischen Person wird der Firmenname und die Rechtsform erfasst und bei einer natürlichen Person der Nachname und das Geburtsdatum. Es soll möglich sein, zu einem bestimmten Fahrzeughalter alle Fahrzeuge und zu einem bestimmten Fahrzeug den aktuellen Fahrzeughalter und alle früheren Fahrzeughalter abzurufen.

Modellieren Sie diesen Ausschnitt durch ein Klassendiagramm, das alle Informationen aus dem Text geeignet integriert.

Attribute, Assoziationen und Generalisierungen modellieren Sie so detailliert wie möglich. Insbesondere wählen Sie den Typ von Attributen so spezifisch wie möglich und nehmen jeden Typ als Klasse oder Datentyp in das Klassendiagramm auf.

Benutzen Sie, wo sinnvoll, Vererbungsbeziehungen.

Operationen und Konstruktoren sollen keine angegeben werden. Sie sollen auch keine Informationen „hinzuerfinden“: finden sich im Text zu einer Klasse oder einem Datentyp keine Attribute im Text, so müssen auch keine angegeben werden. Überlegen Sie sich auch geeignete Datenstruktur-Invarianten.

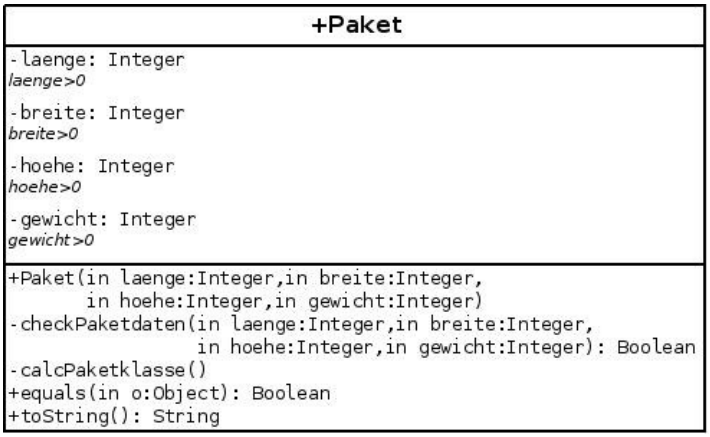
Aufgabe 15*
a) Implementieren Sie für das Computerspiel Pong die Klasse **`Highscore`** mit den Attributen **`name`** (vom Typ **`String`**), **`score`** (vom Typ **`int`**) und **`date`** (vom Typ **`Date`**). Diese drei Attribute sollen nur über den Konstruktor gesetzt werden können, d.h. es sind nur get-Methoden und keine set-Methoden notwendig.

b) Schreiben Sie dann eine Containerklasse **`HighscoreContainer`** im Singletonmuster und sorgen Sie dafür, dass jegliche Verwaltungsoperationen zur Verfügung stehen, um einzelne Highscore-Objekte im Container einzufügen bzw. zu löschen. Neben einer Methode für den Zugriff auf ein bestimmtes **`Highscore`**-Objekt im Container soll auch eine Methode existieren, die alle existierenden Elemente im Container löscht.

c) Testen Sie Ihre beiden implementierten Klassen in einem eigenen Hauptprogramm und prüfen Sie die Funktionalität Ihrer Methoden. Erklären Sie kurz, für was der Zusatz **`<T>`** bei Containerklassen genau verwendet wird.

d) Erweitern Sie Ihren **`HighscoreContainer`** um das Iteratormuster und schreiben Sie ein weiteres Hauptprogramm, in dem Sie verschiedene **`Highscore`**-Objekte anlegen und den nun zur Verfügung stehenden Iterator verwenden, um diese Elemente wieder auszugeben.

Aufgabe 16**



a) Implementieren Sie oben angegebene gegebene UML-Klasse **`Paket`** in Java und sorgen Sie für die nachfolgend gegebene Funktionalität der erwähnten Methoden:

`checkPaketdaten(in laenge:Integer, in breite:Integer, in hoehe:Integer, in gewicht:Integer):Boolean`
überprüft die Datenstruktur-Invarianten und gibt **`true`** zurück, wenn diese erfüllt sind – sonst **`false`**.

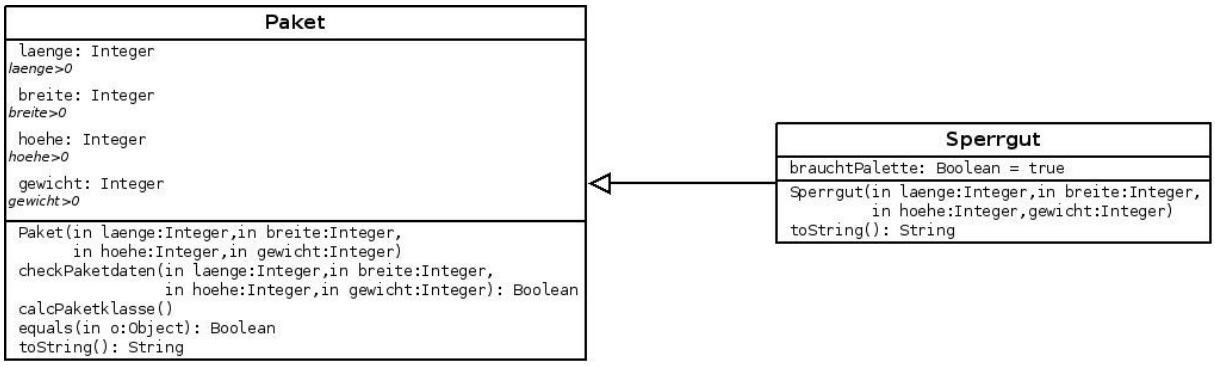
`calcPaketklasse():Integer`
berechnet die Paketklasse eines Paket für ein Gewicht unter **`25`**. Wenn das Gewicht höher ist, soll als (ungültige) Paketklasse **`0`** zurückgegeben werden. Wenn die längste Seite und die kürzeste Seite zusammen weniger als **`50`** sind, dann soll **`1`** als Paketklasse zurückgegeben werden, bei einer Summe von weniger als **`100`** die Paketklasse **`2`** und bei einer Summe von weniger als **`150`** die Paketklasse **`3`**. Wenn das Paket größer ist, dann soll die (ungültige) Paketklasse **`4`** zurückgegeben werden.

`equals(o:Object):Boolean`
überprüft, ob das übergebene Objekt ein Paket ist und gibt **`true`** zurück, wenn das übergebene Objekt ein Paket ist und die gleiche Paketklasse hat wie das vorliegende Objekt – sonst **`false`**.

`toString():String`
gibt je nach vorliegender Paketklasse entweder die Zeichenkette „**`Paketklasse: x`**“ (wobei **`1<=x<=3`**), „**`Sperrgut: Paket zu gross`**“ oder „**`Sperrgut: Gewicht zu hoch`**“ aus.

Der Konstruktor **`Paket(in laenge:Integer, in breite:Integer, in hoehe:Integer, in gewicht:Integer)`** soll die gleichnamigen Attribute nur setzen, wenn die übergebenen Daten gültig sind.

b) Testen Sie Ihre Methoden in einem von Ihnen zu schreibenden Hauptprogramm und überprüfen Sie die verschiedenen Fälle mit unterschiedlichen **`Paket`**-Objekten.



c) Erweitern Sie nun Ihr Programm um die im obigen UML-Diagramm dargestellten Zusammenhänge und passen Sie die Sichtbarkeiten dementsprechend an, so dass die geforderte Funktionalität ohne zusätzliche Attribute und Methoden erfüllt und das Geheimhaltungsprinzip gewahrt wird. Die Methode **`toString():String`** soll hier die genauen Maße und das Gewicht des Sperrguts in der Form „**`Sperrgut (Laenge, Breite, Hoehe, Gewicht): l, b, h, g auf Palette`**“ (wobei **`l`**, **`b`**, **`h`** und **`g`** positive ganze Zahlen sind) ausgeben, wenn **`brauchtPalette`** auf **`true`** gesetzt ist. Begründen Sie Ihre Wahl der Sichtbarkeiten!

d) Schreiben Sie zum Testen der neuen Klasse ein eigenes Hauptprogramm und überprüfen Sie die geforderte Funktionalität.