Beispieltest - PP, 3. Test (2)

1. Aufgabe: Typbeziehung, Vererbung

Plate
Die Klasse `Plate` repräsentiert einen Teller mit einem Durchmesser in cm (`double`) und einer Tiefe in cm (`int`). Jeder Teller wird durch eine eindeutige Seriennummer identifiziert, die vom Konstruktor festgelegt wird und nicht mehr veränderbar ist. Die Seriennummern werden fortlaufend vergeben. Jener Teller, der als erstes erzeugt wird, bekommt die Seriennummer 1. Jeder Teller soll eine `toString`-Methode besitzen, die eine lesbare Repräsentation in der Form Seriennummer, Durchmesser, Tiefe liefert.
Weiters verfügt `Plate` über eine Methode *putOn(Plate)*, die diesen Teller auf den als Parameter angegebenen Teller legt, sodass ein Tellerstapel entsteht. Jedes Teller-Objekt verfügt über eine Teller-Referenz `next`, die auf den darunterliegenden Teller verweist. Die Methode `putOn` muss also dafür sorgen, dass `next` des Tellers entsprechend gesetzt wird.

WoodPlate
Die Klasse `WoodPlate` repräsentiert einen Holzteller. Holzteller sind zwei Zentimeter tief. Schreiben Sie einen entsprechenden Konstruktor. Die Klasse soll eine `toString`-Methode besitzen, die eine lesbare Repräsentation in der Form Seriennummer, Durchmesser, Tiefe liefert.

PorcelainPlate
Die Klasse `PorcelainPlate` repräsentiert einen Porzellanteller. Zusätzlich zu den Eigenschaften eines Tellers ist er spülmaschinenfest (dishwasher safe), d.h. er kann die Information über die maximale Anzahl der Spülgänge (washing cycles) liefern. Die maximale Anzahl der Spülgänge (washing cycles) beträgt bei einem Porzellanteller 1000. Weiters kann ein Porzellanteller einen Goldrand haben oder nicht. Schreiben Sie einen entsprechenden Konstruktor. Die Klasse soll eine `toString`-Methode besitzen, die eine lesbare Repräsentation in der Form Seriennummer, Durchmesser, Tiefe, Spülgänge, Goldrand liefert.

SoupPlate
Die Klasse `SoupPlate` repräsentiert einen speziellen Porzellanteller. Er ist drei Zentimeter tief. Schreiben Sie einen entsprechenden Konstruktor. Die Klasse soll eine `toString`-Methode besitzen, die eine lesbare Repräsentation in der Form Seriennummer, Durchmesser, Tiefe, Spülgänge (washing cycles), Goldrand liefert.

MeatPlate
Die Klasse `MeatPlate` repräsentiert einen speziellen Porzellanteller. Er ist einen Zentimeter tief. Schreiben Sie einen entsprechenden Konstruktor. Die Klasse soll eine `toString`-Methode besitzen, die eine lesbare Repräsentation in der Form Seriennummer, Durchmesser, Tiefe, Spülgänge (washing cycles), Goldrand liefert.

Knife
Die Klasse `Knife` repräsentiert ein Messer. Ein Messer hat eine Länge in mm (`int`) und ist wie der Porzellanteller spülmaschinenfest (dishwasher safe). Die Anzahl der Spülgänge (washing cycles) beträgt bei einem Messer 700. Schreiben Sie einen entsprechenden Konstruktor. Die Klasse soll eine `toString`-Methode besitzen, die eine lesbare Repräsentation in der Form Länge, Spülgänge liefert.

PlateStack
Die Klasse `PlateStack` repräsentiert einen Tellerstapel, der nach der Erzeugung zunächst leer ist. `PlateStack` verfügt über eine ensprechende Objektvariable, die auf das erste Element des Tellerstapels zeigt. Die Methode *push(Plate)* legt den angegebenen Teller auf den obersten Teller des Stapels (nutzen Sie die Methode `putOn` der Klasse `Plate`) oder speichert ihn als ersten Teller, wenn der Tellerstapel leer ist. Auf den obersten Teller wird dann beim nächsten Aufruf von `push` der nächste Teller gelegt, u.s.w.. (Verwenden Sie hier keine Collectionklassen oder Arrays). Die Methode `toString` liefert eine lesbare Repräsentation des Tellerstapels, d.h., aller Teller im Stapel von oben nach unten. Weitere Methoden sind nicht gefragt.

- Schreiben Sie entsprechende Klassen/Interfaces, achten Sie hierbei auf korrekte Typbeziehungen (Vererbung). Vermeiden Sie dabei doppelten Code und nutzen Sie Dateinamen ohne Sonderzeichen. Erstellen Sie die entsprechenden Dateien im Verzeichnis "aufgabe1" und speichern Sie darin Ihre Implementierung. Sie können zum Testen die mitgelieferte ausführbare Klasse nutzen, ihr Inhalt wird nicht bewertet. Sie finden dort bereits eine beispielhafte Nutzung der geforderten Klassen.

2. Aufgabe: Rekursion, Typbeziehung

Ein `PersonNode` ist ein Knoten in einer verketteten Liste. Er speichert einen einzelnen Personeneintrag (Klasse `Person`). Beachten Sie, dass ein Personeneintrag auch ein Studierendeneintrag sein kann, da die Klasse `Student` von `Person` abgeleitet wird. An einen Knoten kann ein weiterer Knoten angehängt werden (next), wodurch eine Liste von Personen entsteht. Jeder Knoten ist der Startknoten der Personenliste, die aus "seinem" Personeneintrag besteht, und der Restliste, die durch next referenziert wird (rekursive Datenstruktur).

Ein Knoten soll folgende Methoden haben:

- Die Methode `getNumberOfNonStudents()` liefert die Anzahl aller Personen in der verketteten Liste (deren Startknoten dieser Knoten ist), die nicht Studierende (also nicht Instanz von Student) sind. Diese Methode kann – muss aber nicht – rekursiv implementiert werden.
- Die Methode `int lastIndexOf(String name)` liefert den Index des letzten Vorkommnisses einer Person mit dem spezifizierten Namen. Der Startknoten hat dabei den Index 0, der direkte Nachfolger den Index 1, usw.. Wenn es also mehrere Personen mit diesem Namen in der Liste gibt, wird die entsprechende Position mit dem höchsten Index geliefert. Wird keine Person in der gesamten verketteten Liste mit entsprechendem Namen gefunden, wird -1 zurückgeliefert. Diese Methode soll rekursiv implementiert werden.

- Implementieren Sie die genannten Methoden in der Klasse `PersonNode`. Die Methode `getNumberOfNonStudents` kann, aber muss nicht rekursiv implementiert werden. Die Methode `lastIndexOf` soll **rekursiv implementiert** werden. Vervollständigen Sie die vorgegebene Klasse im Verzeichnis "aufgabe2". Nutzen Sie die mitgelieferte ausführbare Klasse um Ihre Implementierung zu testen.

3. Aufgabe: Exceptions, Fehlersuche

`Clipper` ist die ausführbare Klasse, die `ClipString` benutzt. `ClipString` repräsentiert einen String, der am Anfang und am Ende um dieselbe Anzahl von Zeichen gekürzt werden kann (siehe Klasse `ClipString`). Das Programm soll mit Hilfe von Exceptionhandling so verbessert werden, dass es nur dann beendet wird, wenn bei der Verarbeitung das Ende der Liste erreicht wird, nicht jedoch wenn die Operation im Schleifenrumpf fehlschlägt. Es soll auch eine entsprechende Schleifenbedingung benutzt werden.

Wenn ein `ClipString` `null` enthält, soll das Programm `"no string"` ausgeben, den Eintrag aus der Liste löschen und danach mit der Verarbeitung des nächsten `ClipStrings` in der Liste fortsetzen. Wenn ein `ClipString` zu wenige Zeichen enthält, um zu kürzen, soll das Programm `"short string"` ausgeben, der ClipString unverändert bleiben und mit der Verarbeitung des nächsten `ClipStrings` in der Liste fortgesetzt werden. In beiden Fällen sollen die Methoden `clip()` und `clip(int)` entsprechende Exceptions werfen. Diese können vom Hauptprogramm gefangen und behandelt werden.

- Erstellen Sie entsprechende Exception-Klassen und ändern Sie das Programm, wie oben beschrieben. Modifizieren Sie die im Verzeichnis "aufgabe3" bereitgestellten Dateien entsprechend der Aufgabenstellung und erzeugen Sie bei Bedarf weitere Klassen. Stellen Sie sicher, dass Ihr Programm nach den Änderungen wieder kompiliert.