Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. M. Wirsing, M. Hammer, A. Rauschmayer

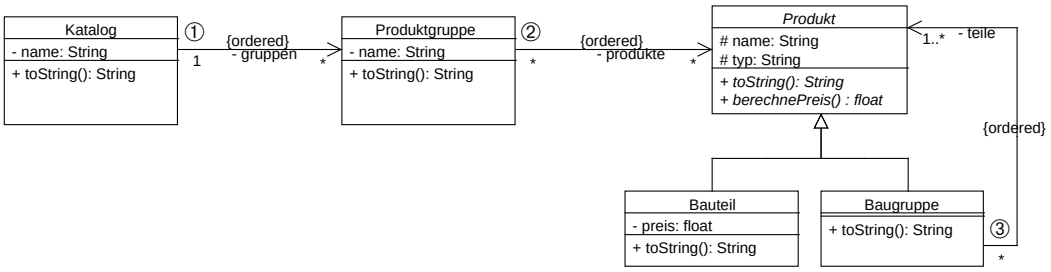
SS 2006
Übungsblatt 9

Informatik II

Zu jeder Aufgabe sind Dateien abzugeben, deren Namen rechts in der Aufgabenüberschrift stehen. Stellen Sie die Dateien in ein extra Verzeichnis (mit beliebigem Namen) und packen Sie dieses zu einem ZIP-Archiv. Geben Sie dieses, wie üblich, per UniWorx ab.

Aufgabe 9-1 Aggregation und Assoziation (10 Punkte, `*.jpg, *.pdf, *.ps, *.java`)

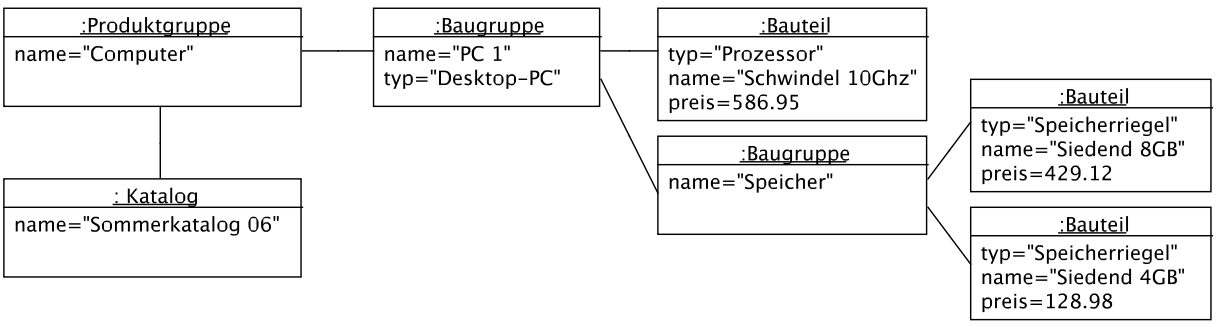
Ein (vereinfachter) Produktkatalog soll abgebildet werden. Ein Katalog besteht aus Produktgruppen, die wiederum Produkte beinhalten. Ein Produkt kann dabei entweder ein einzelnes Bauteil sein, oder eine Baugruppe, die wiederum aus anderen Baugruppen und Bauteilen bestehen kann. Eine UML-Modellierung dieser Beziehungen ist die folgende:



a) Diskutieren Sie, unter welchen Umständen welche Form von Assoziation/Aggregation an den Stellen ① – ③ angebracht ist.

b) Implementieren Sie das UML-Diagramm als Java-Klassen. `berechnePreis` soll dabei für ein Produkt die Summe der Preise seiner Teile angeben. `toString` soll die jeweils assoziierten Objekte mit ausgeben. Verwenden Sie Exceptions (z.B. die `IllegalArgumentException`, oder eine eigene Subklasse), um die korrekten Kardinalitäten sicherzustellen. Es sollte nicht möglich sein, einen inkonsistenten Zustand herzustellen.

c) Ein möglicher Zustand des Objektmodells ist im folgenden Objektdiagramm gegeben:

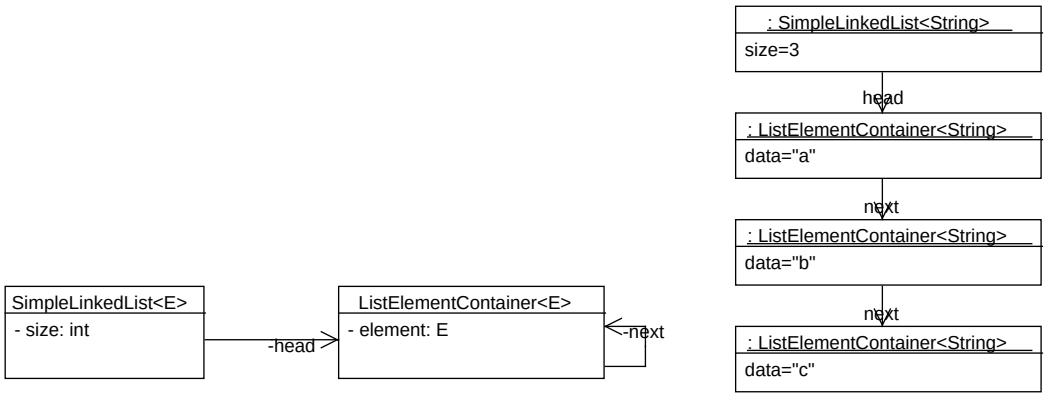


Fügen Sie eine `main`-Methode in die Klasse `Katalog` ein, die den angegebenen Zustand herstellt und auf `System.out` ausgibt.

Aufgabe 9-2 Verkettete Liste (10 Punkte, `SimpleLinkedList.java, LinkedList.java, LinkedListTest.java`)

Gegebene Dateien: `AbstractLinkedList.java`, `ListIteratorImpl`, `ListElementContainer`.

Eine Möglichkeit, Listen zu implementieren, ist als sogenannte *verkettete Liste* (engl. *linked list*). Jedes der eigentlichen Listenelementen wird dabei in einem speziellen Objekt gespeichert, das wir für diese Aufgabe *Container* nennen. Die Container sind untereinander über Referenzen verlinkt. Jeder Container zeigt dabei über eine Assoziation auf einen Nachfolger, der die Rolle `next` hat. Die untenstehenden Abbildungen zeigen das Klassendiagram einer verketteten Liste und beispielhaft das Objektdiagramm der Liste `["a", "b", "c"]`.



a) Einfache iterierbare Liste: Erstellen Sie eine Klasse `SimpleLinkedList` als Unterklasse der gegebenen Klasse `AbstractLinkedList`. Der Iterator ist ebenfalls gegeben, als Klasse `ListIteratorImpl`. Folgende Methoden in `AbstractLinkedList` sind abstrakt und müssen folglich in `SimpleLinkedList` implementiert werden:

   - Methoden `size, get, set, add, remove`: Diese Methoden stellen eine minimale Schnittstelle für Listen dar und werden von `ListIteratorImpl` verwendet.
   - Methode `getContainer`: Diese Methode soll Ihnen bei der Implementierung von `SimpleLinkedList` helfen. Beschreibung siehe JavaDoc.

   Es soll nur einen nullstelligen Konstruktor geben. Hinweise:

   - `AbstractLinkedList` implementiert die Schnittstelle `Iterable`, so dass Sie mit der vereinfachten `for`-Schleife über die Listenelemente iterieren können. Weiterhin gibt es die Instanzmethode `equals` zum Vergleichen einer Instanz mit einer anderen Liste, die Instanzmethode `toString` zum Darstellen des Inhalts und die statische Methode `safeEquals` zum sicheren Vergleichen zweier Objekte (selbst wenn eines davon `null` ist).
   - Eclipse bietet Fehlerbehebungen an, bei denen automatisch die Rümpfe von zu implementierenden abstrakten Methoden der Oberklasse erzeugt werden.

b) Vollständige Implementierung der Schnittstelle `java.util.List`: Schreiben Sie eine Implementierung `LinkedList`. Es soll zwei Konstruktoren geben: `LinkedList()` und `LinkedList(Collection<E>)`. Letzterer initialisiert die Liste mit den Elementen einer bestehenden Collection. Nicht implementieren müssen Sie die Methoden `toArray(T[])` (`toArray()` schon!), `subList(int,int)` und `retainAll(Collection<?>)`. Dort können Sie im Methoden-Rumpf einfach eine `UnsupportedOperationException` werfen; das ist eine übliche Technik, um nicht unterstützte Funktionalität zu kennzeichnen. Hinweis:

   - Nicht verwirren lassen: Das Interface `List` ist nicht komplett generisch (Beispiele: `remove(Object), contains(Object)`. Leider müssen wir uns daran halten, um mit der Java-API kompatibel zu bleiben.

c) Implementieren Sie folgende Unit-Tests:

- `testSimpleLinkedList`: Testen Sie die Methoden `get(int)`, `set(int, E)`, `add(int, E)`, `remove(int)`, `size()`. Wird bei einem falschen Zugriffsindex wirklich eine `IndexOutOfBoundsException` geworfen?
- `testLinkedList`: Testen Sie die Methoden `addAll(int, Collection<? extends E>)`, `indexOf(Object)`, `contains(Object)`, `remove(Object)`, `toArray()`.
- `testIterator`: Testen Sie, ob der Iterator korrekt funktioniert (Methoden `hasNext`, `next`, `add`, `remove`, `previous`, `hasPrevious`).
- `testNulls`: Kommt Ihre Implementierung auch mit `null`-Elementen klar? Schreiben Sie einen Test dafür, bei dem insbesondere `remove`, `contains`, `indexOf` berücksichtigt werden.

Hinweise:

- Wer (a) und (b) nicht geschafft hat, kann bei (c) trotzdem mitmachen, denn hier kann man auch eine beliebige `List`-Implementierung testen (z.B. `java.util.ArrayList`). Natürlich wird dann die Freude über gefundene Fehler weitgehend ausbleiben.
- Die Tests müssen nicht erschöpfend sein (gerade im JavaDoc zu `ListIterator` stehen sehr viele Bedingungen), sondern sollen ein paar grundlegende Dinge überprüfen.
- `testSimpleLinkedList`: Da eine SimpleLinkedList nicht das Interface `List` implementiert, kann man eine normale List-Implementierung nicht mit SimpleLinkedList vergleichen (umgekehrt schon). Nun ist es aber leider gerade so, dass bei einem Test `assertEquals(erwartet, tatsaechlich)` die `equals`-Methode des ersten Arguments eingesetzt wird (wo das zweite Argument in den Tests eine `SimpleLinkedList` sein wird). Abhilfe: verwenden Sie `assertTrue(tatsaechlich.equals(erwartet))`
- Arrays vergleichen: Leider scheitert `assertEquals` von zwei gleichen Arrays, wenn sie nicht identisch sind (also die selbe Instanz). Man kann sich behelfen mit `assertTrue(Arrays.equals(...))`.
- `java.util.Arrays.asList`: Diese Methode kann beliebig viele Argumente erhalten und erzeugt aus diesen eine Liste. Das hilft einem besonders bei Unit-Tests. Beispielaufruf:

```
Arrays.asList("a", "b", "c");
```

Abgabe: Per UniWorx, bis spätestens Montag, den 10.7.2006 um 9:00 Uhr.