Fakultät Informatik
Institut für Software- und Multimediatechnik
Lehrgebiet Softwaretechnologie

Prof. Dr. Uwe Assmann
Technische Universität Dresden * 01062 Dresden

TECHNISCHE UNIVERSITÄT DRESDEN

Tel.: (+49)351/463-38464
Fax .: (+49)351/463-38459
e-mail: ua1@inf.tu-dresden.de

Dresden, den 15.07.2005

Klausur "Softwaretechnologie"

Name:	
Vorname:	
Immatrikulationsnummer:	

Aufgabe	**Maximale Punktanzahl**	**Erreichte Punktanzahl**
1	29	
2	16	
3	45	
Gesamt	**90**	

Zum Bestehen der Leistungskontrolle müssen mindestens 40 Punkte (entspricht 45%) erreicht werden.

Hinweise:

- Die Klammerung der Aufgabenblätter darf **nicht** entfernt werden.
- Tragen Sie bitte die Lösungen auf den Aufgabenblättern ein!
- Verwenden Sie keine roten, grünen Stifte und Bleistifte!
- Es ist kein eigenes Papier zu verwenden! Bei Bedarf ist zusätzliches Papier bei der Aufsicht erhältlich. Bitte jedes zusätzliche Blatt mit Name, Vorname und Immatrikulationsnummer beschriften.
- Es sind alle Aufgabenblätter abzugeben!
- Ergänzen Sie das Deckblatt mit Name, Vorname und Immatrikulationsnummer!
- Halten Sie Ihren Studentenausweis und einen Lichtbildausweis zur Identitätsprüfung bereit.
- **Achtung!** Die grau unterlegten Bereiche bedeuten: **Java-Text ist einzufügen!**

Aufgabe l: Mittelständischer Betrieb

Gegeben ist nachfolgende vereinfachte Beschreibung des Anwendungsbereiches „mittelständischer Betrieb“:

Ein Betrieb hat eine Geschäftsführung und besteht aus den Geschäftsbereichen Produktion, Einkauf, Vertrieb und Buchführung. Die Geschäftsbereiche haben Namen.
Die Geschäftsführung besteht aus einem Geschäftsführer, der den Betrieb verantwortlich führt und den Geschäftsbereichsleitern, von denen jeder einen Geschäftsbereich des Betriebes leitet. Die Geschäftsbereichsleiter sind dem Geschäftsführer berichtspflichtig.
Den Geschäftsbereichen gehören Bereichsangehörige als Beschäftigte des Betriebes an.
Die Geschäftsbereiche unterhalten miteinander Beziehungen. So tätigt der Geschäftsbereich Buchführung die Abrechnungen für alle Geschäftsbereiche. Der Geschäftsbereich Vertrieb setzt die Produkte aus dem Geschäftsbereich Produktion ab. Der Geschäftsbereich Einkauf beschafft für alle Geschäftsbereiche Material.
Alle Beschäftigten (einschließlich Geschäftsführer und Geschäftsbereichsleiter) haben einen Namen und eine Beschäftigungsbezeichnung. Jeder Leiter eines Geschäftsbereiches trägt eine Nebenverantwortung.
Jeder Angehörige eines Geschäftsbereiches hat eine Aufgabenbeschreibung.

Entwickeln Sie aus der Anwendungsbeschreibung ein statisches Analyse-Modell in der Notation für UML-Klassendiagramme! Achten Sie dabei insbesondere auf:

- Klassen und deren Attribute
- Assoziationen, Aggregationen und die mögliche Angabe von Multiplizitäten
- Assoziationsnamen (mit Leserichtung) oder Assoziationsenden-Namen (Rollennamen)
- Vererbung

Aufgabe 2: Hashtabellenverwaltung

Eine Hashtabelle ist eine dynamische Datenstruktur. In ihr werden Datenelemente gespeichert. Der Speicherort dieser Datenelemente richtet sich nach einem mitgeführten, eineindeutigen Schlüssel, der jeweils mit in der Tabelle abgelegt wird.
Die Datenstruktur erweitert sich automatisch, wenn ein festgelegter Füllgrad beim Ablegen eines neuen Elementes überschritten werden soll.
Zur Führung einer Hashtabelle stehen folgende Methoden zur Verfügung:

- put(): Anlegen eines Elements in der Hashtabelle. (Mitgelieferte Parameter sind Schlüssel und Element.)
- get(): Liefern eines Elementes aus der Hashtabelle, ohne es zu löschen. (Mitgelieferter Parameter ist der Schlüssel.)
- remove(): Liefern eines Elementes mit gleichzeitigem Löschen von Schlüssel und Element. (Mitgelieferter Parameter ist der Schlüssel.)

Die relative Adresse der Elemente in der Tabelle wird durch den mitgelieferten Schlüssel bestimmt. Bei der Berechnung dieser relativen Adresse kann es bei der Methode put() zu Kollisionen kommen, wenn die ermittelte Adresse bereits belegt ist und die Schlüssel sich unterscheiden. In diesem Falle werden Element und Schlüssel an die erste freie Stelle in einen bereitstehenden Überlaufbereich geschrieben.
Sollten der Schlüssel des neuen und der eines bereits vorhandenen Elementes gleich sein, so überschreibt die Methode put() das Element.
Die Methoden get() und remove() berechnen über den Schlüssel die Lage des jeweils gesuchten Elements und prüfen über Schlüsselgleichheit, ob das Element gefunden wurde oder im Überlaufbereich gesucht werden muss.
Wird durch remove() ein Element von einem Tabellenplatz entfernt, für den es im Überlauf weitere Anwärter gibt, nimmt einer dieser Anwärter diesen Platz ein.

Leiten Sie aus der Beschreibung des dynamischen Verhaltens einer Hashtabelle das Zustandsmodell als Protokollautomaten ab! Stellen Sie das Modell als Zustandsdiagramm in UML-Notation dar!

Aufgabe 3:

Gegeben ist folgender Quellcodeausschnitt eines Java-Programmes zur Verwaltung von Inventarlisten (Klasse `InventoryCollection`, siehe auch UML-Klassendiagramm):

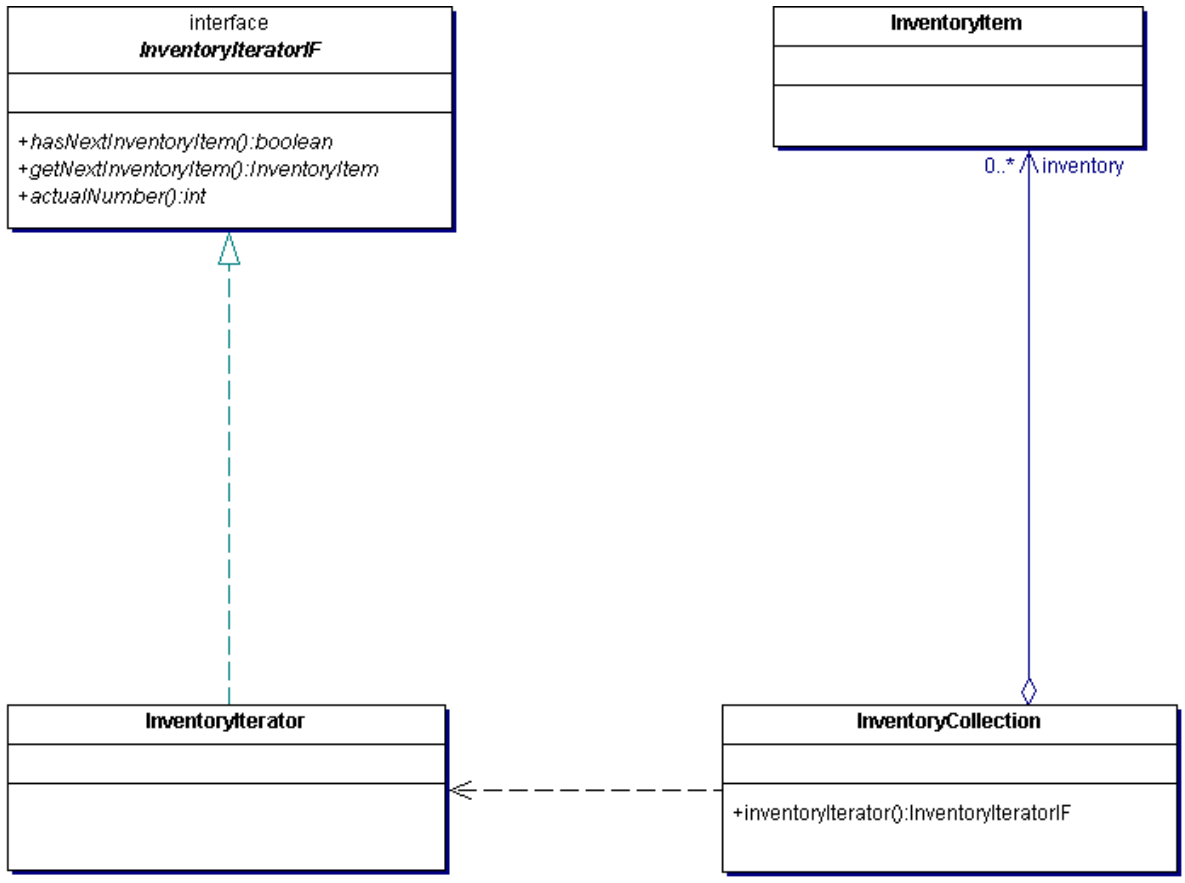
```
import java.util.*;

/**
 * Inventory als Menge von InventoryItem-Objekten.
 * Verwenden Sie dazu java.util.Set
 */
public class InventoryCollection {



   public InventoryIteratorIF inventoryIterator() {



   }
   // Rest nicht relevant
}

/**
 * Iterator für Inventarlisten
 */
public interface InventoryIteratorIF {

   public boolean hasNextInventoryItem();

   public InventoryItem getNextInventoryItem();

   /**
    * Liefert die Anzahl der bisher durch getNextInventoryItem()
    * zurückgegebenen Einträge.
    */
   public int actualNumber();
}

public class InventoryIterator implements InventoryIteratorIF {



   public InventoryIterator(                              ) {



   }
   public boolean hasNextInventoryItem() {



   }
   public InventoryItem getNextInventoryItem() {



   }
   public int actualNumber() {



   }
}
```

```
public class InventoryItem {
   // nicht relevant
}
```

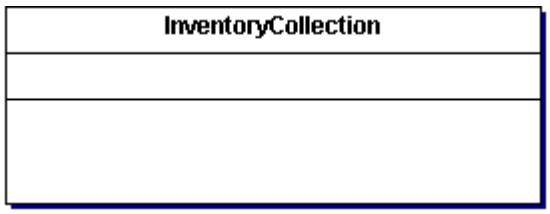
Lösen Sie folgende Aufgaben:

(a) Implementieren Sie die Klassen `InventoryCollection` und `InventoryIterator` unter Verwendung von `java.util.Iterator`! Fügen Sie den dazu nötigen Quellcode in die gegebenen Quellcodeausschnitte an den markierten Stellen ein! Verwenden Sie *Generics* für die typsichere Realisierung der internen Datenstrukturen.

(b) Welche (zwei) Entwurfsmuster erkennen Sie im Entwurfsmodell? Ergänzen Sie das folgende Klassendiagramm um eine Darstellung der Entwurfsmuster in UML-Notation und um evtl. weitere notwendige Modellelemente!



(c) Erweitern Sie diesen Entwurf so, dass auch hierarchisch strukturierte `InventoryItem`s möglich werden! Ein hierarchisch strukturiertes `InventoryItem` ist dabei ein `InventoryItem`, das selbst wieder andere `InventoryItem`s enthalten kann. Überlegen Sie, welches Entwurfsmuster Sie dazu einsetzen können. Zeichnen Sie in das folgende Klassendiagramm zusätzlich zu der Klasse `InventoryCollection` alle benötigten Klassen/Interfaces für hierarchisch strukturierte `InventoryItem`s und das verwendete Entwurfsmuster in UML-Notation ein!



(d) Erweitern Sie Ihre Implementierung aus Teilaufgabe (a) um einen Iterator (`AtomicInventoryIterator`), der nur die atomaren (nicht strukturierten) InventoryItems innerhalb einer InventoryCollection zurückliefert! Ergänzen Sie dazu das untenstehende Codegerüst! Sie können dabei davon ausgehen, dass ein strukturiertes `InventoryItem` immer mindestens ein Unter-`InventoryItem` enthält. Verwenden Sie auch hier *Generics* für die typsichere Realisierung der internen Datenstrukturen.

```
public class InventoryCollection                                    {
   // siehe oben

   public InventoryIteratorIF atomicInventoryIterator() {
                                                                    
                                                                    
                                                                    
   }
}

public class AtomicInventoryIterator implements InventoryIteratorIF {
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
   public AtomicInventoryIterator(                                  ) {
                                                                    
                                                                    
                                                                    
                                                                    
   }
   public boolean hasNextInventoryItem() {
                                                                    
                                                                    
                                                                    
                                                                    
   }

   public InventoryItem getNextInventoryItem() {
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
// Falls der Platz nicht reicht, bitte Rückseite verwenden!
   }
}
```