



Technische Universität Dresden, 01062 Dresden



## **Klausur Softwaretechnologie SS 2016**

Prof. Dr.rer.nat.habil.  
Uwe Aßmann

Name:	
Vorname:	
Immatrikulationsnummer:	

<b>Aufgabe</b>	<b>Maximale Punktzahl</b>	<b>Erreichte Punktzahl</b>
<b>1</b>	<b>20</b>	
<b>2</b>	<b>13</b>	
<b>3</b>	<b>57</b>	
<b>Gesamt</b>	<b>90</b>	

### **Hinweise:**

- In der Klausur ist als Hilfsmittel lediglich ein **A4-Blatt, beidseitig beschrieben**, zugelassen.
- Die Klammerung der Aufgabenblätter darf **nicht** entfernt werden.
- Tragen Sie bitte die Lösungen auf den Aufgabenblättern ein!
- Verwenden Sie keine roten, grünen Stifte oder 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.

### **Aufgabe 1: GitHub (20 Punkte)**

GitHub ist eine webbasierte Plattform, die Software-Entwicklungsprojekte bereitstellt und diese einer Versions- und Konfigurationsverwaltung unterstellt. Im Folgenden wird eine vereinfachte Sicht auf GitHub beschrieben.

In GitHub gibt es Nutzer (*User*) und Organisationen (*Organization*).

Innerhalb einer Organisation ist ein Nutzer ein Teilnehmer (*Participant*), der wiederum entweder ein *Owner* und/oder ein *Member* ist.

Eine Organisation kann beliebig viele Repositories (*Repository*) anlegen, die ausschließlich dieser Organisation gehören.

Eine Organisation kann aus beliebig vielen Teams (*Team*) bestehen, die nur innerhalb der Organisation existieren. Ein Team besteht aus einer Menge von Teilnehmern.

Ein Teilnehmer kann gleichzeitig verschiedenen Teams angehören. Ein Team hat Zugriff auf beliebig viele Repositories. Ebenso können beliebig viele Teams auf ein Repository zugreifen. Außer den Teilnehmern der Organisation können weitere Nutzer Zugriff auf beliebige Repositories der Organisation erhalten. Diese Nutzer werden als *Outside\_Collaborator* bezeichnet. Es werden drei Arten von Zugriffsrechten (*Permission*) unterschieden: READ, WRITE, ADMIN, die ein Outside-Collaborator auf ein Repository haben kann.

Organisationen, Repositories, Github Nutzer und Teams haben einen Namen (*Name*). Zusätzlich haben Organisationen, Repositories und Teams eine Sichtbarkeit (*Visibility*).

**Erstellen Sie ein Analyse-Klassendiagramm (Domänenmodell)!**

**Berücksichtigen Sie dabei folgende Hinweise:**

- **Die Modellierung der Beziehungen soll aus der Sicht einer Organisation erfolgen!**
- **Denken Sie an Klassen, Enumerationen, Klassenbeziehungen, Rollennamen und Attribute!**
- **Es sollen keine Operationen modelliert werden!**
- **Alle domänenspezifischen Begriffe, die im obigen Text *kursiv* geschrieben sind, müssen im Modell wiederzufinden sein!**



**Aufgabe 2: Wissensfragen (13 Punkte)**

Antworten Sie jeweils kurz auf die folgenden Fragen!

(1) Nennen Sie die drei wesentlichen Eigenschaften eines Objektes im Kontext des objektorientierten Paradigmas (3 P.)!

-----

(2) Auf welches Modellelement der UML wird eine Verantwortlichkeit einer CRC-Karte abgebildet (1 P.)?

-----

(3) Was bedeutet das Testen von Klassen und Methoden? Kreuzen Sie die richtige Antwort an (1 P.)!

- Validation
- Stichprobenhafte Validation
- Formale Verifikation
- Stichprobenhafte Verifikation

(4) Wie nennt man einen automatisierten Vergleich von Ausgabedaten (gleicher Testfälle) unterschiedlicher Versionen eines Programms (1 P.)?

-----

(5) Ordnen Sie die Merkmalsausprägungen von Datenstrukturen den Collection-Klassen in Java zu! Kreuzen Sie dazu das entsprechende Feld in der Tabelle an (2 P.)!

	<b>LinkedList</b>	<b>HashSet</b>	<b>TreeMap</b>
<b>mit Schlüssel</b>			
<b>geordnet</b>			
<b>sortiert</b>			
<b>mit Duplikate</b>			

**(6) Welche Implementierung des Interfaces `java.util.List` bietet bei welcher vorherrschenden Operation auf der Liste die höhere Effizienz? Kreuzen Sie dazu das entsprechende Feld in der Tabelle an (2 P.)!**

	<b>ArrayList</b>	<b>LinkedList</b>
<b>Lesen eines Objektes</b>		
<b>Iterieren über alle Objekte</b>		
<b>Einfügen eines Objektes</b>		
<b>Löschen eines Objektes</b>		

**(7) Welche der folgenden Aussagen sind in Bezug auf JUnit richtig? Kreuzen Sie die richtigen Antworten an (2 P.)!**

- JUnit kann einen Defect wie z.B. fehlenden oder falschen Code erkennen.**
- Error Guessing wird durch JUnit unterstützt.**
- Ein Error tritt auf, wenn bei der Ausführung des Tests eine Exception bis in die Testmethode gelangt.**
- Ein Failure tritt auf, wenn in einer Testmethode eine falsche Berechnung festgestellt wird (falscher Ausgabewert).**
- Bei einem Bug handelt es sich um einen fehlgeschlagenen Assert-Aufruf.**

**(8) Wie nennt man in Java eine Typumwandlung (1 P.)?**

-----

### **Aufgabe 3: Rechnungen (Invoice) (57 Punkte)**

In einem Programmsystem sollen Rechnungen mit ihren Rechnungsposten implementiert werden. Der Entwurf dazu ist im Folgenden sowohl textuell als auch als Entwurfsklassendiagramm gegeben.

Eine Rechnung (`Invoice`) besteht aus beliebig vielen Rechnungsposten (`LineItem`). Die dazugehörige Datenstruktur (`myLineItems`) wird durch eine `TreeMap<Integer, LineItem>` realisiert, wobei der Schlüssel der Map eine laufende Postennummer (`position` beginnend mit 1) ist. Auf der Rechnung können Rechnungsposten hinzugefügt werden (`addLineItem()`).

Die `toString()`-Methode ist wie bei den anderen Klassen dazu da, die Konsolenausgabe (siehe Seite 8) geeignet zu formatieren. Das Gleiche gilt für die Methode `discountDescription()`.

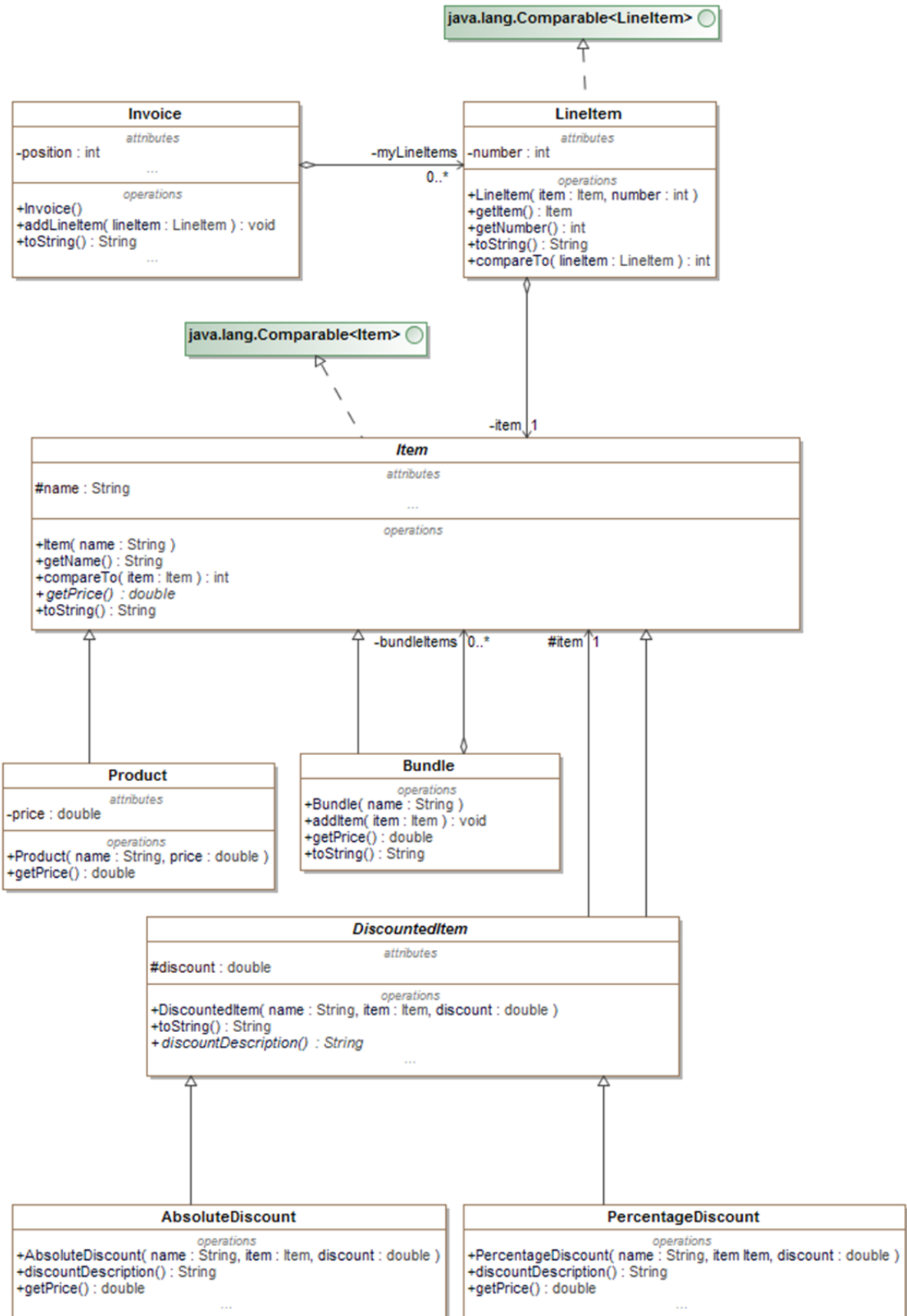
Ein Rechnungsposten kennt sein zu verkaufendes Objekt (`Item`) und dessen Stückzahl (`number`). Es gibt verschiedene Arten (Klassen) von Objekten. Alle diese Objekte haben einen Namen (`name`).

- Bei einzelnen Produkten (`Product`) wird ein fester Rechnungspreis (`price`) im Konstruktor des Objektes übergeben.
- Bei Packen (`Bundle`) von beliebigen Objekten wird der Rechnungspreis als Summe der Preise der in einem Packen enthaltenen Objekte berechnet (`getPrice()`). Ein Packen entsteht, indem diesen Packen mit `addItem()` beliebige Objekte hinzugefügt werden. Achtung, ein Objekt kann mehrmals hinzugefügt werden.
- Bei preisreduzierten Objekten (`DiscountedItem`), die zusätzlich einen Discount (`discount`) kennen, wird der Rechnungspreis (`getPrice()`) durch Anwendung des Discounts (`discount`) berechnet. Dabei gibt es zwei Varianten von preisreduzierten Objekten:
  - Anwendung eines absoluten Discounts, bei dem der Discount in Euro angegeben wird (`AbsoluteDiscount`).
  - Anwendung eines prozentualen Discounts, bei dem der Discount in Prozent angegeben wird (`PercentageDiscount`).

Objekte der Klasse `LineItem` werden anhand ihrer Objekte der Klasse `Item` verglichen. Objekte der Klasse `Item` werden anhand ihres Namens (`name`) verglichen.

#### **Lösen Sie folgende Teilaufgaben:**

- I. Machen Sie sich die Struktur eines Rechnungsobjektes anhand eines Objektdiagramms klar. Dazu ist auf Seite 8 der Ausschnitt aus einer Client-Klasse mit der Erzeugung einer Rechnung als auch die dazugehörige Konsolenausgabe gegeben. Stellen Sie die erzeugten Objekte (Objektname reicht) und deren Links in einem Objektdiagramm dar!**
- II. Vervollständigen Sie die Implementierung der Java-Klassen des gegebenen Codegerüsts! Parameter in Methoden (übergebene Objekte) müssen nicht auf null getestet werden!**
- III. Welche Entwurfsmuster sind im Entwurf enthalten? Finden Sie mindestens zwei Entwurfsmuster!**
  - Achtung, raten Sie nicht! Für falsch angegebene Entwurfsmuster gibt es Punktabzug.
  - Tragen Sie die gefundenen Entwurfsmuster als UML-Kollaboration in das Entwurfsklassendiagramm ein (nächste Seite)!



Eine Client-Klasse erzeugt eine Rechnung wie folgt:

```

Invoice invoice = new Invoice();
Product product1 = new Product("p1", 20);
Product product2 = new Product("p2", 10);
Product product3 = new Product("p3", 1);
Bundle bundle1 = new Bundle("b1");
bundle1.addItem(product1);
bundle1.addItem(product1);
bundle1.addItem(product3);
DiscountedItem discountedItem1 = new PercentageDiscount("d1", product2,10);
DiscountedItem discountedItem2 = new AbsoluteDiscount("d2",product2,2);

LineItem lineItem1 = new LineItem(product1,2);
LineItem lineItem2 = new LineItem(product2,1);
LineItem lineItem3 = new LineItem(bundle1,3);
LineItem lineItem4 = new LineItem(discountedItem1, 1);
LineItem lineItem5= new LineItem(discountedItem2,1);

invoice.addLineItem(lineItem1);
invoice.addLineItem(lineItem2);
invoice.addLineItem(lineItem3);
invoice.addLineItem(lineItem4);
invoice.addLineItem(lineItem5);

```

Der Aufruf der Methode

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

gibt auf der Konsole Folgendes aus:

```

LineItem 1: (Product p1, Preis: 20.0 EUR, Anzahl: 2, Postenpreis: 40.0)
LineItem 2: (Product p2, Preis: 10.0 EUR, Anzahl: 1, Postenpreis: 10.0)
LineItem 3: (Bundle b1 bestehend aus
[Product p1, Preis: 20.0 EUR, Product p1, Preis: 20.0 EUR, Product p3, Preis: 1.0
EUR]
Bundle-Preis: 41.0, Anzahl: 3, Postenpreis: 123.0)
LineItem 4: (PercentageDiscount p2, Preis: 9.0 EUR (Discount 10.0%, Originalpreis:
10.0 EUR), Anzahl: 1, Postenpreis: 9.0)
LineItem 5: (AbsoluteDiscount p2, Preis: 8.0 EUR (Discount 2.0 EUR, Originalpreis:
10.0 EUR), Anzahl: 1, Postenpreis: 8.0)

```



**Zeichnen Sie das Objektdiagramm (Teilaufgabe I)!**

**Vervollständigen Sie den folgenden Java-Code (Teilaufgabe II)!**

```
import java.util.*; // gilt für alle Klassen

public class Invoice {

    public Invoice() {

    }

    public void addLineItem (LineItem lineItem) {

    }

    public String toString(){
        String out ="";
        Set<Map.Entry<Integer, LineItem>> set = myLineItems.entrySet();
        for (Map.Entry<Integer, LineItem> entry : set){
            out = out + "LineItem " + entry.getKey() + ": " + entry.getValue();
        }
        return out;
    }
}
-----
public class LineItem {

    public String toString(){
        return "(" + item + ", Anzahl: " + number + ", Postenpreis: " +
            number*item.getPrice() + ") \n" ;
    }
}
```

```

public          class Item                                     {

    public String toString() {
        return getClass().getName() + " " + name +
            ", Preis: " + getPrice() + " EUR";
    }
}
-----
public class Bundle                                         {

    public Bundle(String name) {

    }

    public void addItem (Item item){

    }

    public String toString(){
        return getClass().getName() + " " + name + " bestehend aus \n" +
            bundleItems.toString() + " \n Bundle-Preis: " + getPrice();
    }

    public double getPrice() {

    }
}

```

```

public          class DiscountedItem          {

    public DiscountedItem(String name, Item item, double discount) {

    }

    public String toString() {
        return getClass().getName() + " " + name + ",
            Preis: " + this.getPrice() + " EUR" +
            discountDescription()+", Originalpreis: " +item.getPrice()+" EUR";
    }
}
-----
public class AbsoluteDiscount          {

    public AbsoluteDiscount(String name, Item item, double discount) {

    }

    public String discountDescription() {
        return " (Discount " + discount + " EUR";
    }
}
-----
public class PercentageDiscount          {

    public PercentageDiscount(String name, Item item, double discount) {

    }

    public String discountDescription() {
        return " (Discount " + discount + "%";
    }
}

```