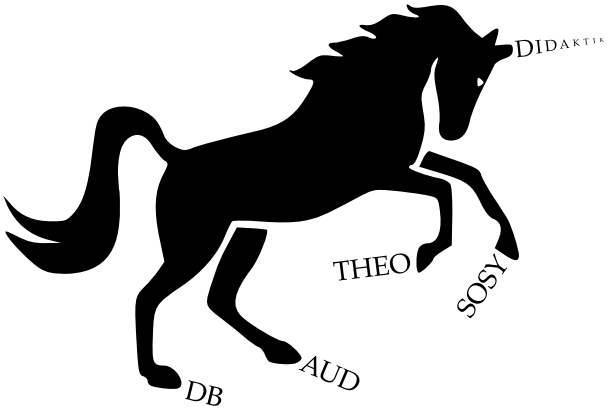
66116
Frühjahr
2018

Datenbanksysteme / Softwaretechnologie (vertieft)

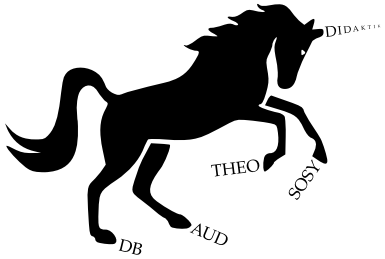
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Thema Nr. 2

Teilaufgabe Nr. 1

Aufgabe 1 [Zirkus]

Das Fremdenverkehrsamt will sich einen besseren Überblick über Zirkusse verschaffen. In einer Datenbank sollen dazu die Zirkusse, die angebotenen Vorstellungen, die einzelnen Darbietungen in einer Vorstellung sowie die zugehörigen Dompteure und Tiere verwaltet werden.

Ein Zirkus wird durch seinen Namen gekennzeichnet und hat einen Besitzer. Vorstellungen haben eine `VorstellungsID` und ein Datum. Darbietungen haben neben der eindeutigen `ProgrammNr` eine Uhrzeit. Ein Dompteur hat eine eindeutige `AngestelltenNr` sowie einen Künstlernamen. Tiere sind eindeutig durch eine TierNr bestimmt und haben außerdem eine Bezeichnung der Tierart.

Ein Zirkus bietet Vorstellungen an und stellt Dompteure an. Eine Darbietung findet in einer Vorstellung statt. Des weiteren trainiert ein Dompteur Tiere. In einer Darbietung tritt ein Dompteur mit Tieren auf.

(a) 1.1

(i) Listen Sie die Entity-Typen und die zugehörigen Attribute auf.

```
* Zirkusse (Zirkus-Nummer, Namen)
  * Besitzer
  * Namen

* Vorstellungen (VorstellungsID)
  * VorstellungsID
  * Datum

* Darbietungen (ProgrammNr)
  * ProgrammNr
  * Datum

* Dompteuere (AngestelltenNr)
  * AngestelltenNr
  * Künstlernamen

* Tiere (TierNr)
  * TierNr
  * Tierart
```

(ii) Bestimmen Sie zu jedem Entity-Typen einen Schlüssel. Fügen Sie, wenn nötig einen künstlichen Schlüssel hinzu.

```
Zirkus (ZID, Besitzer, Name)
Vorstellung (VorstellungsID, Datum, ZID[Zirkus])
Darbietung (ProgrammNr, VorstellungsID[Vorstellung], Uhrzeit)
Dompteur (AngestelltenNr, Kuenstlername, ZID[Zirkus])
Tier (TierNr, Tierart)
```

```
trainiert (AngestelltenNr[Dompteur], TierNr[Tier])
trittAuf (AngestelltenNr[Dompteur], TierNr[Tier], ProgrammNr[Darbietung],
↪ VorstellungsID[Vorstellung])
```

(iii) Erstellen Sie das ER-Diagramm!

Vorstellungen werden von genau einem Zirkus angeboten. Ein Zirkus bietet mehrere Vorstellungen an und stellt mehrere Dompteure an. Ein Dompteur ist genau bei einem Zirkus angestellt. Eine Darbietung findet in einer bestimmten Vorstellung statt. Des weiteren trainiert ein Dompteur mehrere Tiere, ein Tier kann allerdings auch von mehreren Dompteuren trainiert werden. In einer Darbietung tritt genau ein Dompteur mit mindestens einem Tier auf.

(b) 1.2 Ergänzen Sie die Funktionalitäten im ER-Diagramm.

Vorstellungen werden von genau einem Zirkus angeboten. Ein Zirkus bietet mehrere Vorstellungen an und stellt mehrere Dompteure an. Ein Dompteur ist genau bei einem Zirkus angestellt. Eine Darbietung findet in einer bestimmten Vorstellung statt. Des weiteren trainiert ein Dompteur mehrere Tiere, ein Tier kann allerdings auch von mehreren Dompteuren trainiert werden. In einer Darbietung tritt genau ein Dompteur mit mindestens einem Tieren auf.

(c) 1.3

(i) Was bedeutet „mehrere“?

(ii) Ergänzen Sie die Kardinalitäten in min-max Notation im ER-Diagramm.

Aufgabe 4 [Tupelkalkül bei Dozenten-Datenbank]

Gegeben sei das folgende Datenbank-Schema, das für die Speicherung der Daten einer Universität entworfen wurde, zusammen mit einem Teil seiner Ausprägung. Die Primärschlüssel-Attribute sind jeweils unterstrichen. Die Relation *Dozent* enthält allgemeine Daten zu den Dozentinnen und Dozenten. Dozentinnen und Dozenten halten Vorlesungen, die in der Relation *Vorlesung* abgespeichert sind. Wir gehen davon aus, dass es zu jeder Vorlesung genau einen Dozenten (und nicht mehrere) gibt. Zusätzlich wird in der Relation *Vorlesung* das *Datum* gespeichert, an dem die Klausur stattfindet. In der Relation *Student* werden die Daten der teilnehmenden Studierenden verwaltet, während die Relation *besucht* Auskunft darüber gibt, welche Vorlesung von welchen Studierenden besucht wird.

```
Dozent (DNR, DVorname, DNachname, DTitel)
Vorlesung (VNR, VTitel, Klausurtermin, Dozent)
Student (Matrikelnummer, SVorname, SNachname, Semesterzahl)
besucht (Student, Vorlesung)
```

Formulieren Sie die folgenden Anfragen im Tupelkalkül. Datumsvergleiche können Sie mit $>$, $\geq$, $<$, $\leq$ oder $=$ angeben:

(a) Geben Sie die Vornamen aller Studierenden aus, die die Vorlesung *„Datenbanksysteme“* besuchen oder besucht haben.

Lösungsvorschlag

$$\{s.SVorname | s \in Student \wedge \forall v \in Vorlesung(v.VTitel = \text{'Datenbanksysteme'} \Rightarrow \exists b \in besucht(b.Vorlesung = v.VNR \wedge b.Student = s.Matrikelnummer))\}$$

oder

Lösungsvorschlag

$$\{s.SVorname | s \in Student \wedge s.Matrikelnummer = b.Student \wedge b \in besucht \wedge b.Vorlesung = v.NVR \wedge v.VNR \in Vorlesung \wedge v.VTitel = \text{'Datenbanksysteme'}\}$$

(b) Geben Sie die Matrikelnummern der Studierenden an, die keine Vorlesung mit einem Klausurtermin nach dem 31.12.2017 besuchen oder besucht haben.

Lösungsvorschlag

$$\{s.Matrikelnummer | s \in Student \wedge \forall v \in Vorlesung(v.Klausurtermin > \text{'31.12.2017'} \Rightarrow b \in besucht(b.Vorlesung = v.VNR \wedge b.Student = s.Matrikelnummer))\}$$

(c) Geben Sie die Matrikelnummern der Studierenden aus, die alle Vorlesungen von Prof. Dr. Schulz hören oder gehört haben.

Teilaufgabe Nr. 2

Aufgabe 1 [UML-Diagramme entspreched Java-Code zeichnen]

Gegeben sei das folgende Java-Programm:

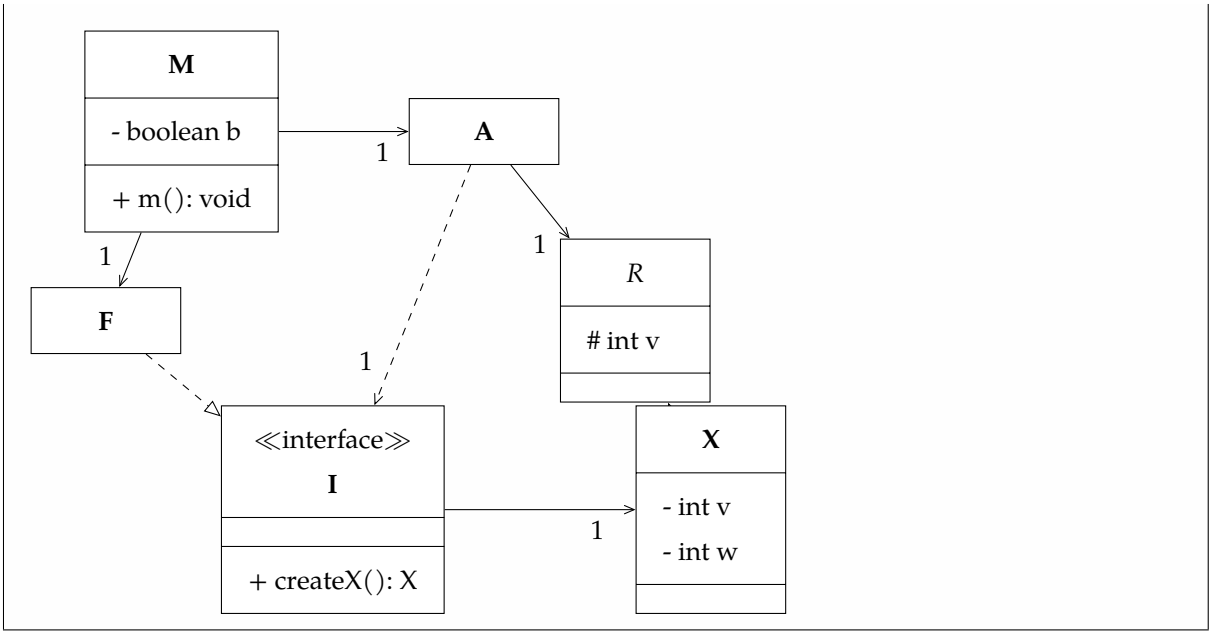
```
class M {
  private boolean b;
  private F f;
  private A a;

  public void m() {
    f = new F();
    a = new A(f);
    b = true;
  }
}

class A {
  private R r;
  public A(I i) {
```

```
    r = i.createX();
  }
}

interface I {
  public X createX();
}

class F implements I {
  public X createX() {
    return new X(0, 0);
  }
}

abstract class R {
  protected int v;
}

class X extends R {
  private int v, w;
  public X(int v, int w) {
    this.v = v;
    this.w = w;
  }
}
```

(a) Das Subtypprinzip der objektorientierten Programmierung wird in obigem Programmcode zweimal ausgenutzt. Erläutern Sie wo und wie dies geschieht.

(b) Zeichnen Sie ein UML-Klassendiagramm, das die statische Struktur des obigen Programms modelliert. Instanzvariablen mit einem Klassentyp sollen durch gerichtete Assoziationen mit Rollennamen und Multiplizität am gerichteten Assoziationsende modelliert werden. Alle aus dem Programmcode ersichtlichen statischen Informationen (insbesondere Interfaces, abstrakte Klassen, Zugriffsrechte, benutzerdefinierte Konstruktoren und Methoden) sollen in dem Klassendiagramm abgebildet werden.

Lösungsvorschlag



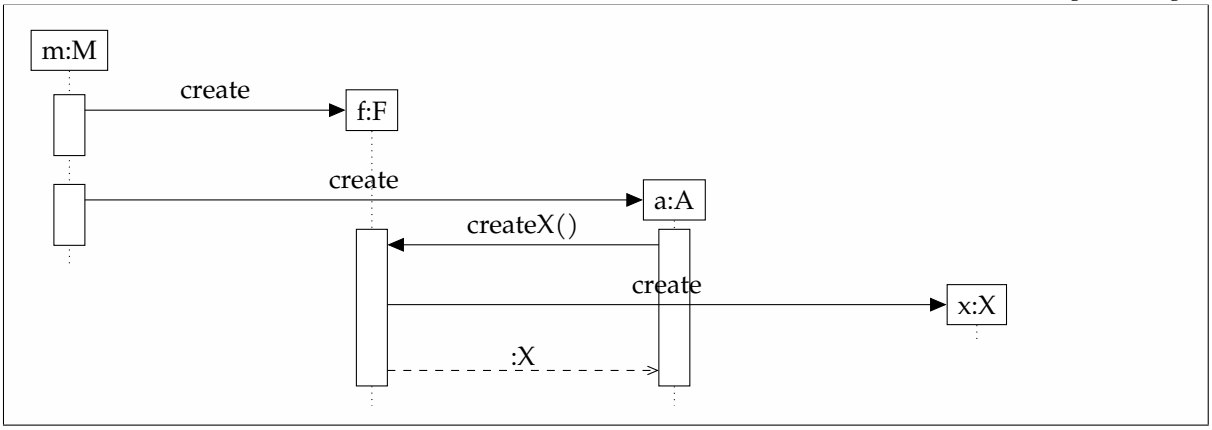
(c) Es wird angenommen, dass ein Objekt der Klasse `M` existiert, für das die Methode `m()` aufgerufen wird. Geben Sie ein Instanzendiagramm (Objektdiagramm) an, das alle nach der Ausführung der Methode `m` existierenden Objekte und deren Verbindungen (Links) zeigt.

Lösungsvorschlag

Diese Aufgabe hat noch keine Lösung. Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.

(d) Wie in Teil c) werde angenommen, dass ein Objekt der Klasse M existiert, für das die Methode `m()` aufgerufen wird. Diese Situation wird in Abb. 1 dargestellt. Zeichnen Sie ein Sequenzdiagramm, das Abb. 1 so ergänzt, dass alle auf den Aufruf der Methode `m()` folgenden Objekterzeugungen und Interaktionen gemäß der im Programmcode angegebenen Konstruktor- und Methodenrümpfe dargestellt werden. Aktivierungsphasen von Objekten sind durch längliche Rechtecke deutlich zu machen.

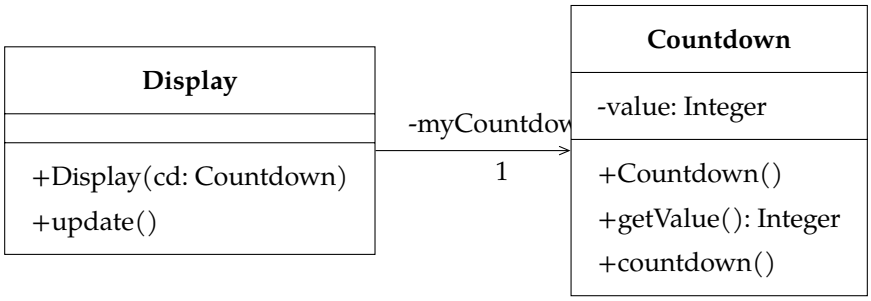
Lösungsvorschlag



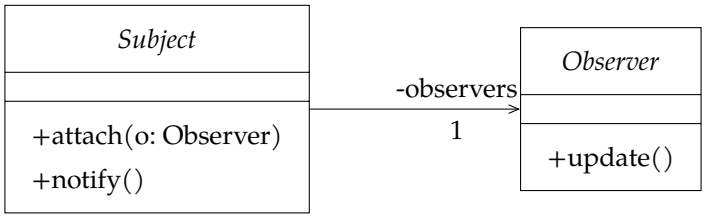
Aufgabe 2 [Countdown und Observer]

Es soll eine (kleine) Anwendung entwickelt werden, in der ein Zähler in 1-er Schritten von 5000 bis 0 herunterzählt. Der Zähler soll als Objekt der Klasse `Countdown` realisiert werden, die in UML-Notation dargestellt ist. Das Attribut value soll den aktuellen Zählerstand speichern, der mit dem Konstruktor zu initialisieren ist. Die Methode `getValue` soll den aktuellen Zählerstand liefern und die Methode `countdown` soll den Zähler von 5000 bis 0 herunterzählen.

Der jeweilige Zählerstand soll von einem Objekt der in untenstehender Abbildung angegebenen Klasse `Display` am Bildschirm ausgegeben werden. Bei der Konstruktion eines `Display`-Objekts soll es mit einem `Countdown`-Objekt verbunden werden, indem dessen Referenz unter `myCountdown` abgespeichert wird. Die Methode `update` soll den aktuellen Zählerstand vom `Countdown`-Objekt holen und mit `System.out.println` am Bildschirm ausgeben. Dies soll zu Beginn des Zählprozesses und nach jeder Änderung des Zählerstands erfolgen.



Damit das `Display`-Objekt über Zählerstände des `Countdown`-Objekts informiert wird, soll das Observer-Pattern angewendet werden. Untenstehende Abbildung zeigt die im Observer-Pattern vorkommenden abstrakten Klassen. (Kursivschreibweise bedeutet abstrakte Klasse bzw. abstrakte Methode.)



(a) Welche Wirkung haben die Methoden `attach` und `notify` gemäß der Idee des Observer-Patterns?

Lösungsvorschlag

> Das beobachtete Objekt bietet mit der Methode `attach` einen Mechanismus, um Beobachter anzumelden und diese über Änderungen zu informieren.
>
> Mit der Methode `notify` werden alle Beobachter benachrichtigt, wenn sich das beobachtete Objekt ändert.

(b) Welche der beiden Klassen `Display` und `Countdown` aus obenstehender Abbildung spielt die Rolle eines `Subject` und welche die Rolle eines `Observer`?

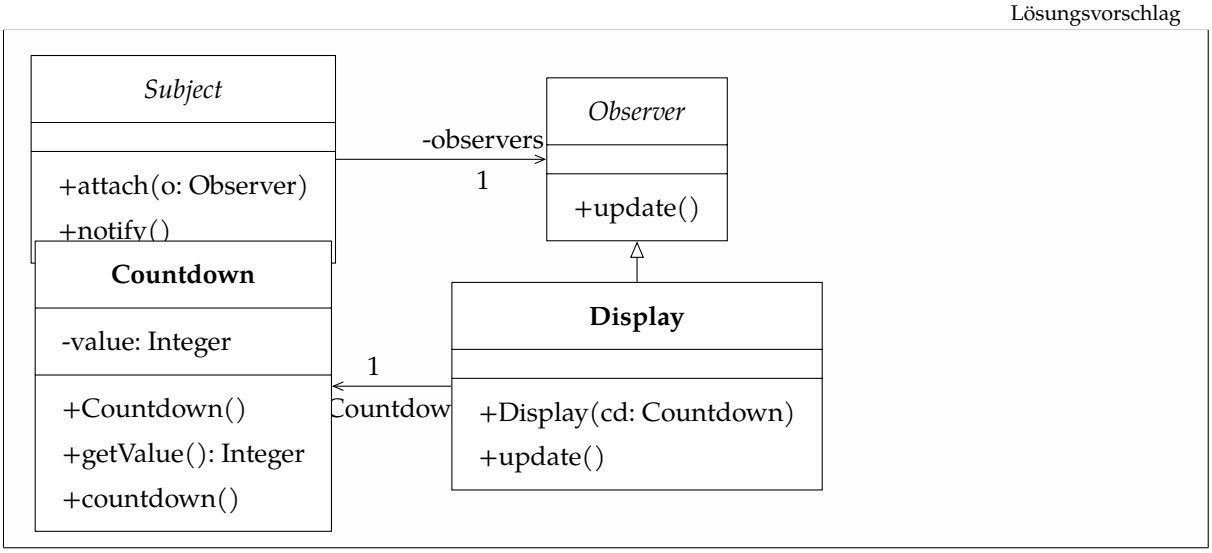
Lösungsvorschlag

Die Klasse `Countdown` spielt die Rolle des `Subject`s, also des Gegenstands, der beobachtet wird.

Die Klasse `Display` spielt die Rolle eines `Observer`, also die Rolle eines Beobachters.

(c) Erstellen Sie ein Klassendiagramm, das die beiden obenstehenden gegebenen Diagramme in geeigneter Weise, ðentsprechend der Idee des Observer-Patterns, zusammenfügt. Es reicht die Klassen und deren Beziehungen anzugeben. Eine nochmalige Nennung der Attribute und Methoden ist nicht notwendig.

Lösungsvorschlag



(d) Unsere Anwendung soll nun in einer objektorientierten Programmiersprache Ihrer Wahl (z. B. Java oder C++) implementiert werden. Dabei soll von folgenden Annahmen ausgegangen werden:

- Das Programm wird mit einer main-Methode gestartet, die folgenden Rumpf hat:

```
public static void main(String[] args){
  Countdown cd = new Countdown();
  new Display(cd);
  cd.countdown();
}
```

- Die beiden Klassen Subject und Observer sind bereits gemäß der Idee des Observer-Patterns implementiert.

Geben Sie auf dieser Grundlage eine Implementierung der beiden Klassen `Display` und `Countdown` an, so dass das gewünschte Verhalten, ðAnzeige der Zählerstände und Herunterzählen des Zählers, realisiert wird. Die Methoden der Klassen `Subject` und `Observer` sind dabei auf geeignete Weise zu verwenden bzw. zu implementieren. Geben Sie die verwendete Programmiersprache an.

```java
public class Client {
  public static void main(String[] args){
    Countdown cd = new Countdown();
    new Display(cd);
    cd.countdown();
    cd.countdown();
    cd.countdown();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Client.java`

```java
import java.util.ArrayList;
import java.util.List;

public abstract class Subject {
  private final List<Observer> observers = new ArrayList<Observer>();

  public void attach(Observer o) {
    observers.add(o);
  }

  public void notifyObservers() {
    for (Observer o : observers) {
      o.update();
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Subject.java`

```java
public abstract class Observer {
  public abstract void update();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Observer.java`

```java
public class Countdown extends Subject {

  private int value;

  public Countdown() {
    value = 5000;
  }

  public int getValue() {
    return value;
  }

  public void countdown() {
    if (value > 0) {
      notifyObservers();
```

```
      value--;
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Countdown.java`

```
public class Display extends Observer {
  Countdown myCountdown;
  public Display(Countdown cd) {
    myCountdown = cd;
    myCountdown.attach(this);
  }

  public void update() {
    System.out.println(myCountdown.getValue());
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Display.java`