
Modellierung und Programmierung 1

Übungsserie 5

Lösungsvorschläge

1. Klassen

a) und b) Siehe Programm **BanditTest.java** und Ergebnisdatei **BanditTest.out**.

Wahrscheinlichkeiten:

einfach $1 - 0.27 - 0.01 = 0.72$

doppelt $10 * (9 + 9 + 9) / 1000 = 0.27$

dreifach $10 / 1000 = 0.01$

2. Modellierung

zu b)

| Beziehung | Begründung |
|---|---|
| Aggregation Ambulanz → Arzt | Eine Ambulanz hat mindestens einen Arzt (1..*), ein Arzt hat Dienst oder nicht (0..1). |
| Aggregation Ambulanz → Patient | Eine Ambulanz hat Patienten (0..*), ein Patient ist in Behandlung oder nicht (0..1). |
| Komposition Patient → Protokoll | Zu jedem Patienten gibt es genau ein Protokoll (1), ein Protokoll existiert genau für einem Patienten (1). |
| Komposition Protokoll → Protokolleintrag | Ein Protokoll besteht aus mindestens einem Eintrag (1..*), ein Eintrag existiert nur im Protokoll (1). |
| Aggregation Arzt → Patient | Ein Arzt behandelt mehrere Patienten (0..*), ein Patient ist in Behandlung oder nicht (0..1). |
| Komposition Person → Termin | Zwei Termin pro Person: Beginn, Ende der Behandlung (2), ein Termin ist abhängig von der Person (1). |
| Komposition Protokolleintrag → Termin | Termin des Protokolleintrags (1), ein Termin ist abhängig von der Person (1). |
| Komposition Termin → Datum | Ein Termin besteht aus genau einem Datum (1), ein Datum ist abhängig vom Termin (1). |
| Komposition Termin → Uhrzeit | Ein Termin besteht aus genau einer Uhrzeit (1), eine Uhrzeit ist abhängig vom Termin (1). |
| Komposition Person → Termin | Geburtsdatum einer Person (1), ein Geburtsdatum ist abhängig vom Person (1). |

Hinweis:

An der Raute kann bei Aggregation (0..1) und bei Komposition (1) weggelassen werden.

zu e)

| Klasse | Attribut | Collection | Begründung |
|-----------|-----------|------------|--|
| Ambulanz | aerzte | LinkedList | Sortieren nach Namen, Anfangszeit, ..., Löschen. |
| | patienten | LinkedList | Sortieren nach Namen, Einlieferungszeit, ..., Löschen. |
| Arzt | patienten | LinkedList | Sortieren nach Namen, Einlieferungszeit, ..., Löschen. |
| Protokoll | eintraege | ArrayList | kein Umsortieren, überwiegend lesender Zugriff. |

Ergänzung:

Bei dezentralem Datenzugriff sollte in allen Fällen die Collectionklasse Vector verwendet werden, ihre Methoden sind bereits weitgehend synchronisiert.

```

classDiagram
    class Datum {
        - jahr : int
        - monat : int
        - tag : int
        + Datum( tag : int, monat : int, jahr : int ) : 
        + Datum() : 
        + getDate() : Datum
    }
    class Uhrzeit {
        - minuten : int
        - sekunden : int
        - stunde : int
        + Uhrzeit() : 
        + getUhrzeit() : Uhrzeit
    }
    class Termin {
        - datum : Datum
        - uhrzeit : Uhrzeit
        + Termin() : 
        + getTermin() : Termin
    }
    class Person {
        - beginn : Termin
        - ende : Termin
        - name : String
        + Person( name : String ) : 
        + setBeginn( termin : Termin ) : void
        + setEnde( termin : Termin ) : void
    }
    class Patient {
        - KK : String
        - entlassen : boolean
        - geb : Datum
        - protokoll : Protokoll
        + Patient( geb : Datum, KK : String, name : String ) : 
        + entlassen() : Protokoll
        + getProtokoll() : Protokoll
    }
    class Arzt {
        - LANR : int
        - ambulanzen : Ambulanz
        - behandelt : boolean
        - patienten : LinkedList<Patient>
        + Arzt( name : String, LANR : int ) : 
        + behandelt( patient : Patient ) : Protokolleintrag
        + behandeltPatienten() : void
        + entlaesst( patient : Patient ) : Protokolleintrag
        + uebernimmt( patient : Patient ) : boolean
    }
    class Ambulanz {
        - aerzte : LinkedList<Arzt>
        - patienten : LinkedList<Patient>
        + anmelden( patient : Patient ) : void
        + ausscheiden( arzt : Arzt ) : void
        + dienstBeginn( arzt : Arzt ) : void
        + dienstEnde( arzt : Arzt ) : void
        + einstellen( arzt : Arzt ) : void
        + getNextAssistenzarzt() : Assistenzarzt
        + getNextChirurg() : Chirurg
        + getNextInternist() : Internist
        + getNextNotfallarzt() : Notfallarzt
    }
    class Protokoll {
        - eintraege : ArrayList<Protokolleintrag>
        + neuerEintrag( eintrag : Protokolleintrag ) : void
    }
    class Protokolleintrag {
        - arzt : Arzt
        - behandlung : String
        - diagnose : String
        - medikamente : String
        - termin : Termin
        + Protokolleintrag( arzt : Arzt ) : 
        + eintragBehandlung( behandlung : String ) : void
        + eintragDiagnose( diagnose : String ) : void
        + eintragMedikamente( medikamente : String ) : void
        + eintragen() : void
    }
    class Assistenzarzt
    class Notfallarzt
    class Chirurg
    class Internist

    Datum "1" -- "1" Termin
    Uhrzeit "1" -- "1" Termin
    Termin "1" -- "2" Person
    Person "1" -- "0..*" Patient
    Patient "1" -- "0..*" Protokoll
    Protokoll "1" -- "1..*" Protokolleintrag
    Patient "1" -- "0..*" Arzt
    Arzt "1" -- "1..*" Ambulanz
    Arzt "1" -- "0..*" Protokolleintrag
    Protokolleintrag "1" -- "0..*" Arzt
    Protokolleintrag "1" -- "0..*" Ambulanz
    Protokolleintrag "1" -- "0..*" Protokoll
    Assistenzarzt --|> Arzt
    Notfallarzt --|> Arzt
    Chirurg --|> Arzt
    Internist --|> Arzt
    
```

Es ergeben sich folgende Fehler:

- a)
 - i. Die Komposition ist verkehrt herum.
 - ii. Maultier erbt von 2 Klassen. Der Aufgabentext sagt explizit, dass das Programm handeln soll. In Java ist Mehrfachvererbung nicht möglich.
- b)
 - i. Pflanzenfresser fressen Fleischfresser. Die Assoziation ist nicht symmetrisch.
 - ii. Das Programm und der Code erben sich wechselseitig.