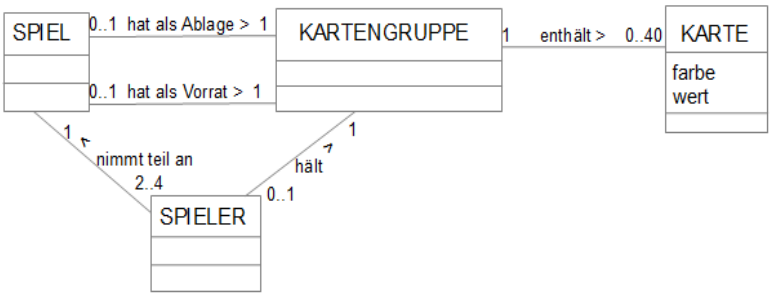
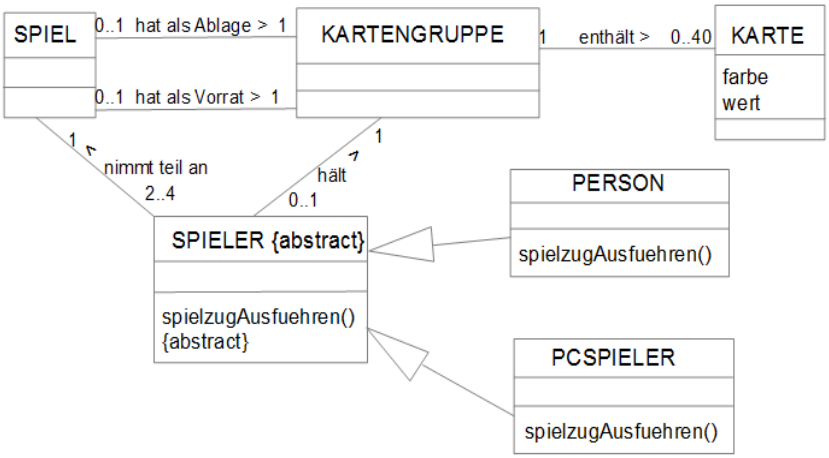
Informatik Abitur Bayern 2015 / I - Beispiellösung

Autor:
Gaisbauer

1a 

10

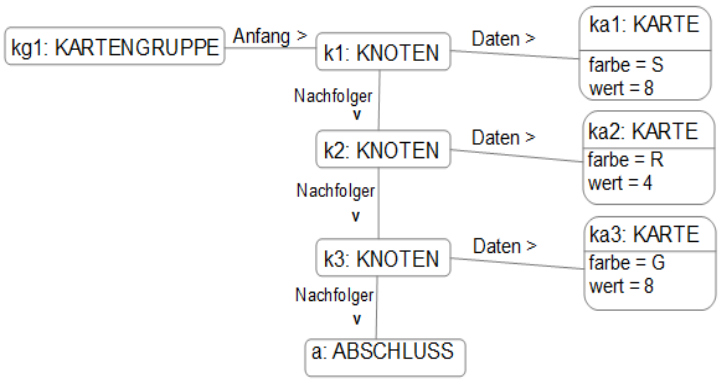
1b 

4

1c Die Beziehung zwischen den Klassen SPIEL und SPIELER kann bidirektional implementiert werden. Ein Objekt der Klasse SPIELER muss auf den Vorrat zugreifen können, was über ein Objekt der Klasse SPIEL erfolgen kann. Ein Objekt der Klasse SPIEL muss die Objekte der Klasse SPIELER kennen, um sie beispielsweise aufzufordern, einen Spielzug auszuführen. 4

1d Vorteile Liste: Variable Größe, daher auch wenig Speicherbedarf; einfaches Einfügen von Objekten. 3

Vorteile Feld: schnellerer Zugriff auf Elemente; statischer Aufbau.

1e 

5

1f Die Methode kann zum Mischen der Karten eingesetzt werden. Es wird hierbei eine zufällig ausgewählte Karte an den Anfang der Liste gesetzt. Dies wird 10 000-mal wiederholt. 4

1g 16

```
public abstract class LISTENELEMENT
{
    public abstract KARTE KarteGeben(int position);
}
```

```
public class KNOTEN extends LISTENELEMENT
{
    private KARTE karte;
    private LISTENELEMENT nachfolger;

    public KNOTEN(KARTE karteNeu, LISTENELEMENT nachfolgerNeu) {
        karte = karteNeu;
        nachfolger = nachfolgerNeu;
    }

    public KARTE KarteGeben(int position) {
        if (position == 0) {
            return karte;
        } else {
            return nachfolger.KarteGeben(position - 1);
        }
    }
}


public class ABSCHLUSS extends LISTENELEMENT
{
    public KARTE KarteGeben(int position) {
        return null;
    }
}



public class KARTENGRUPPE
{
    private LISTENELEMENT erstes;

    public void VorneEinfuegen(KARTE karteNeu) {
        erstes = new KNOTEN(karteNeu, erstes);
    }

    public KARTE KarteGeben(int position) {
        return erstes.KarteGeben(position);
    }
}
```

2a

4

Der Graph ist unzusammenhängend, ungerichtet, ungewichtet und zyklisch.

2b Dies bedeutet, dass die Karte mit dem Index i auf die Karte mit dem Index j gelegt werden darf (und umgekehrt). 3

2c z.B. Breitensuche oder Tiefensuche. Hier Tiefensuche: 8

```
boolean besucht[];
besucht = new boolean[40];
int erreichbareKnoten;

int BeginneZaehlen(int startKnoten){
      erreichbareKnoten = 0;

      for(int i = 0; i < 40; i = i+1) {
            besucht[i] = false;
      }
      ZaehleAb(startKnoten);
      return erreichbareKnoten;
```

```
}

void ZaehleAb(int knotenNr) {
      besucht[knotenNr] = true;

      for (int i=0; i < 40; i++) {

            if (matrix[knotenNr][i] == true && !besucht[i]) {
                  erreichbareKnoten = erreichbareKnoten+1;
                  ZaehleAb(i);
            }
      }
}
```

2d Die beschriebene Strategie gibt für die Karten S8 und S9 je die gleiche Erreichbarkeit an. Es wird also zufällig eine der beiden Karten gewählt. 5
Wird S8 ausgewählt, können anschließend S9 und R9 angelegt werden.
Wird jedoch S9 ausgewählt, so kann entweder S8 oder R9 angelegt werden. Eine Karte bleibt jedoch in jedem Fall übrig.

3a 3 Ebenen: Erste Ebene: 1 Knoten; Zweite Ebene: 3 Knoten; Dritte Ebene: 9 Knoten. 3
Oder: 1 * 3 * 3 = 9 Spieler.

n Ebenen: Ab der zweiten Ebene gibt es in jeder folgenden Ebene dreimal so viele Knoten wie zuvor. Also: 3^{n-1} Spieler.

3b 3 Ebenen: In der dritten Ebene sind es drei Spiele, in der zweiten Ebene ein Spiel. 4
Gesamt: 1 + 3 = 4 Spiele.

n Ebenen: In der zweiten Ebene gibt es ein Spiel, in der dritten Ebene drei Spiele, in der vierten Ebene neun Spiele, usw. Ab Ebene drei werden es also pro Ebene dreimal so viele Spiele. Dies ist also in n-2 Ebenen der Fall (Nicht in Ebene eins und zwei).
Gesamt: $1 + 3 + 3^2 + ... + 3^{n-2}$ Spiele.

3c 7