Prüfung Praktische Informatik 2
Prüfung Softwareentwicklung 2

Aufgabensteller: Prof. Glavina, Prof. Grauschopf, Prof. Hahndel, Prof. Schmidt
Prüfungsdauer: 90 Minuten
Hilfsmittel: keine

Aufgabe 1: (Klassenmodellierung, ca 25%)

Wir betrachten nochmals die Interfaces und Klassen aus Vorführaufgabe 8. Zur Erinnerung: Beide relevante Klassen, `Gerät` und `Bauteil`, haben ein Attribut `name` und implementieren das Interface `Artikel`:

```
public interface Artikel {
  // weitere Methoden ...
  public int getPreis();
}
```

Das Ergebnis von `getPreis()` wird im weiteren als Anschaffungswert interpretiert.

Ein `Gerät` kann aus mehreren `Artikeln`, sprich `Bauteilen` oder auch `Geräten`, bestehen. Es speichert diese Bestandteile in einem Array vom Typ `Artikel[]` ab.

Nun gilt es, folgendes zu modellieren:

a) Ein `Equipment` ist ein `Gerät`, das zusätzlich eine Inventarnummer und ein Anschaffungsjahr hat. Definieren Sie die Klasse `Equipment` mit `private`-Attributen und einem Konstruktor, der alle Attribute mit übergebenen Werten belegt und dazu auch den Konstruktor

```
public Gerät(String name, Artikel[] teile)
```

von `Gerät` nutzt. Weiter soll `Equipment` eine Methode haben, die die Inventarnummer zurückgibt.

b) Erweitern Sie `Equipment` um eine Methode

```
public double abschreibungswert()
```

Der Abschreibungswert ist für ein `Equipment`, das z.B. im Jahr 2001 angeschafft wurde, 2001 noch 100% seines Anschaffungswertes, 2002 noch 75%, 2003 noch 50%, 2004 noch 25% und 2005 und später nur noch 0% davon. Zur Vereinfachung müssen Sie in Ihrer Implementierung nicht zum nächsten Cent-Betrag runden. Das aktuelle Jahr erhält man mit

```
int aktJahr = Calendar.getInstance().get(Calendar.YEAR);
```

c) `Equipment` implementiere außerdem das Interface `java.lang.Comparable`. Ergänzen Sie die Klassendefinition entsprechend, und schreiben Sie die benötigte Methode. Das Vergleichskriterium für zwei Equipments ist wie folgt definiert: Ein Equipment ist „kleiner“ als das andere, wenn es einen geringeren Abschreibungswert hat. Bei gleichen Abschreibungswerten ist das `Equipment` mit der kleineren Inventarnummer kleiner.

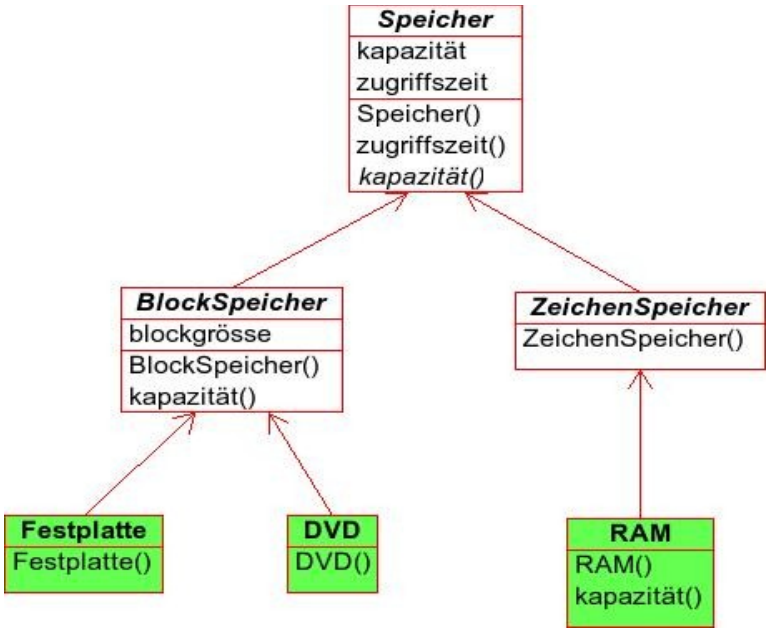
d) Eine Klasse `Bestand` verwaltet die `Equipments` eines Unternehmens in einer parametrisierten `LinkedList` (Hilfestellung: siehe Anhang letzte Seite). Schreiben Sie die Klasse `Bestand` mit dem Default-Konstruktor und einer Methode

```
public boolean addEquipment(Equipment e)
```

die dem Bestand ein `Equipment` hinzufügt, aber nur, wenn die Inventarnummer des `Equipments` im Bestand noch nicht vorkommt. Der Rückgabewert signalisiert, ob erfolgreich hinzugefügt werden konnte oder nicht.

Aufgabe 2: (Programm verstehen und ergänzen, ca. 25%)

Bei Speichern unterscheidet man zwischen blockorientierten (z.B. Festplatte und DVD) und zeichenorientierten Speichern (z.B. Hauptspeicher wie RAM). Gegeben sei im folgenden die Implementierung der im Klassendiagramm gezeigten Klassenhierarchie, bestehend aus den drei abstrakten Klassen `Speicher`, `BlockSpeicher` (blockorientiert) und `ZeichenSpeicher` (zeichenorientiert) sowie den drei konkreten Klassen `Festplatte`, `DVD` und `RAM`.



Erläuterung zum Klassendiagramm: die grau dargestellten Klassen sind konkrete Klassen.

Hilfestellung: Es ist zur Bearbeitung der Aufgabe nicht nötig, den gegebenen Programmcode im Detail durchzulesen. Es genügt in jeder Teilaufgabe, das richtige Programmfragment zu identifizieren, um die Frage beantworten zu können!

```
// Implementierung der Klasse Speicher
public abstract class Speicher {
    protected int kapazität; // in Mbyte  oder Anzahl Blöcken   //(*)
    protected double zugriffszeit;  // in ms

    public Speicher(int kapazität, double zugriffszeit) {
           this.kapazität = kapazität;                          //(**)
           this.zugriffszeit = zugriffszeit;
    }

    public double zugriffszeit() { return zugriffszeit; }
    public abstract int kapazität(); // liefert Kapazität in MByte
}

// Implementierung der Klasse BlockSpeicher
public abstract class BlockSpeicher extends Speicher {

    protected int blockgrösse;  // Blockgrösse in Bytes

    public BlockSpeicher(int kapazität, double zugriffszeit,
                         int blockgrösse){
           super(kapazität, zugriffszeit);
           this.blockgrösse = blockgrösse;
      }

    public int kapazität(){
           return (blockgrösse * kapazität / 1024);
    }
}

// Implementierung von ZeichenSpeicher
public abstract class ZeichenSpeicher extends Speicher {

      public ZeichenSpeicher(int kapazität, double zugriffszeit) {
           super(kapazität, zugriffszeit);
      }
}

// Implementierung von Festplatte
public class Festplatte extends BlockSpeicher {

      public Festplatte(int kapazität, double zugriffszeit,
                        int blockgrösse) {
           super(kapazität, zugriffszeit, blockgrösse);
      }
}

// Implementierung von DVD
public class DVD extends BlockSpeicher {

      public DVD(boolean doublelayer){
           super(2350,2048,120);              // 2350 Blöcke
           if (doublelayer) kapazität = 4250; // 4250 Blöcke
      }
}

// Implementierung von RAM
public class RAM extends ZeichenSpeicher {

      public RAM(int kapazität, double zugriffszeit) {
           super(kapazität, zugriffszeit);                //(***)
      }

      public int kapazität() {
           return kapazität/1024;
      }
}
```

Beantworten Sie anhand des Programmcodes folgende Fragen:
(Hinweis: Die Antworten zählen nur mit einer kurzen verständlichen Begründung! Ja oder Nein als Antwort genügt nicht!)

a) Könnte man die Klassen `BlockSpeicher` und `ZeichenSpeicher` hier auch einfach als nicht abstrakte Klassen (= konkrete Klassen) definieren ?

- Falls Ihre Antwort nein lautet, erläutern Sie warum das nicht geht!
- Falls Ihre Antwort ja lautet, begründen Sie, warum es hier Sinn macht, diese Klassen hier trotzdem abstrakt zu definieren.

b) Wäre es hier möglich, statt der abstrakten Klasse `Speicher` ein Interface `Speicher` zu verwenden ?

c) Könnte man in der abstrakten Klasse `Speicher` die Attribute `kapazität` und `zugriffszeit` bei (*) auch mit dem Modifier **`private`** statt **`protected`** versehen?

d) Warum ist an der Stelle (**) im Konstruktor die Verwendung von **`this`** nötig? Ginge das auch anders?

e) Könnte man bei (***) in der Klasse `RAM` den Konstruktor weglassen ?

Implementierung der Anwendungsklasse:

f) Schreiben Sie eine Anwendungsklasse `SpeicherApp`, die unter Verwendung der gegebenen Klassen folgende Schritte in `main` durchführt:

- Anlegen einer Reihung vom Typ `Speicher` mit den folgenden vier Elementen, die ebenfalls erzeugt werden müssen:
  - ein `RAM` mit Kapazität 2000MB und 0,008ms Zugriffszeit
  - eine Single-Layer-DVD (`doublelayer = false`!) mit 4,7GB
  - eine Festplatte mit 6 Millionen Blöcken, 10ms Zugriffszeit und 512 Bytes Blockgröße
  - eine Festplatte mit 3,4 Millionen Blöcken, 14ms Zugriffszeit und 512 Bytes Blockgröße.

- Anschliessend soll in einer Schleife die Gesamtkapazität (= Summe der Einzelkapazitäten) aller Datenträger berechnet und dann als Ergebnis ausgegeben werden.

Aufgabe 3 (Collections, ca. 25%)

Wir können die Fußballweltmeisterschaft durch folgende Klassen modellieren:

Spieler

`String name;   // Name des Spielers`
`int tore;      // Zahl der von ihm erzielten Tore`

`Spieler(String name)`
`void setTore(int tore)`
`int getTore()`
`String getName()`

Team

`String name;                    // Name des Landes`
`LinkedList<Spieler> mannschaft; // Die Spieler des Teams`

`Team(String name)`
`void addSpieler(Spieler spieler)`
`LinkedList<Spieler> getMannschaft()`
`Spieler getBesterSpieler()    // der mit den meisten Toren`

FußballWM

`Team[] teams = new Team[32];  // die WM-Mannschaften`
`HashMap<Team, String> wohnorte; // wo die Teams wohnen`

a) Implementieren Sie die angegebenen Methoden der Klasse `Team`.

b) Ergänzen Sie die Klasse `FußballWM` um die Methode `getTore`, welche die Summe aller erzielten Tore ermittelt und zurückgibt.

c) Ergänzen Sie die Klasse `FußballWM` um die Methode `getSpielerWohnort`, welche für einen gegebenen Spielernamen den Wohnort seines Teams zurückgibt.

d) Ergänzen Sie die Klasse `FußballWM` die Methode `dreamTeam`, welche ein neues `Team` zusammenstellt und zurückgibt. In dieses `dreamTeam` soll jede der teilnehmenden Mannschaften genau einen Spieler entsenden, und zwar den mit den meisten Toren (bei mehreren torgleichen Spielern kann ein beliebiger Spieler aus dieser Gruppe entsandt werden).

(Ein Auszug aus der Collections-Framework-API mit den wichtigsten Methoden ist im Anhang beigefügt.)

Aufgabe 4 (JAVA-GUI):

Bei der Anmeldung von Benutzern soll eine interaktive Oberfläche gemäß folgender Abbildung zum Einsatz kommen.

Der Benutzer gibt Name und Kennwort ein und drückt dann „OK“; für unsere Zwecke soll hier statt einer Weitergabe der Daten ans Betriebssystem nur der eingegebene Name und das Kennwort an der Konsole ausgedruckt werden (etwa „Anmeldung für Hacker/Ich_will_hier_rein“.

Ähnlich soll bei „Herunterfahren ...“ ein entsprechender Text gedruckt werden; bei „Abbrechen“ wird das Anmeldefenster geschlossen und das Programm beendet.

Schreiben Sie ein vollständiges Java-Programm, so dass das vom Programm erzeugte Fenster der obigen Abbildung entspricht und auch das Verhalten des Programms mit der obigen Beschreibung übereinstimmt.

Anhang

Auszug aus der Collections-Framework-API

```
public class LinkedList<E>
```

`public LinkedList()`
Konstruktor

`public boolean add(E o)`
fügt das übergebene Element an das Ende der Liste an

`public boolean contains(Object elem)`
`true`, falls die Liste das übergebene Element enthält

`public E get(int index)`
gibt das Element an der angegebenen Position zurück

`public boolean isEmpty()`
`true`, falls die Liste leer ist

`public int size()`
gibt die Anzahl der Listenelemente zurück

```
public class HashMap<K, V>
```

`public HashMap()`
Konstruktor

`public V get(Object key)`
gibt den Wert für diesen Schlüssel zurück (`null`, falls `key` nicht vorhanden)

`public boolean isEmpty()`
`true`, falls keine Schlüssel-Werte-Paare vorhanden

`public V put(K key, V value)`
fügt das Schlüssel-Wert-Paar (K, V) hinzu

`public int size()`
gibt die Anzahl der Schlüssel-Werte-Paare zurück

Auszug aus der AWT-API

```
public class TextField
```

`void setText(String s)`
setzt den Text

`String getText()`
gibt den Text als String zurück

```
public class ActionEvent
```

`String getActionCommand()`
gibt die Bezeichnung (String) der auslösenden Ereignisquelle bzw. Aktion zurück