FernUniversität in Hagen

FAKULTÄT FÜR
MATHEMATIK UND INFORMATIK

Lehrgebiet Programmiersysteme
Prof. Dr. Friedrich Steimann

(Name, Vorname)

(Straße, Nr.)

(PLZ) (Wohnort)

(Land, falls außerhalb Deutschlands)

Kurs 01618 WS 2013 / 2014

„Einführung in die objektorientierte Programmierung“

Klausur am 15.03.2014

Dauer: 3 Std., 10 – 13 Uhr

Lesen Sie zuerst die Hinweise auf der folgenden Seite!

Matrikelnummer: | | | | | | | | **Geburtsdatum:** | | . | | . | | |

Klausurort: _____

Aufgabe	1	2	3	4	5	6	7	8	9	10	Summe
habe bearbeitet											
maximal	10	10	10	10	10	10	10	10	10	10	100
erreicht											
Korrektur											

○ Herzlichen Glückwunsch, Sie haben die Klausur bestanden. Note: .....

○ Sie haben die Klausur leider nicht bestanden. Für den nächsten Versuch wünschen wir Ihnen viel Erfolg. Die nächste Klausur findet im Sommersemester 14 statt.

Hagen, den 26.03.2014

Im Auftrag

Musterlösung

© 2014 FernUniversität in Hagen

Hinweise zur Bearbeitung

1. Prüfen Sie die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst auf insgesamt 22 Seiten:
   – 1 Deckblatt
   – Diese Hinweise zur Bearbeitung
   – 10 Aufgaben auf Seite 3-19
   – Zwei zusätzliche Seiten 20 und 21 für weitere Lösungen

2. Füllen Sie jetzt bitte zuerst das Deckblatt aus:
   – Name, Vorname und Adresse
   – Matrikelnummer, Geburtsdatum und Klausurort

3. Schreiben Sie Ihre Lösungen mit Kugelschreiber oder Füllfederhalter (*kein Bleistift*) direkt in den bei den jeweiligen Aufgaben gegebenen, umrahmten Leerraum. Benutzen Sie auf keinen Fall die Rückseiten der Aufgabenblätter. Versuchen Sie, mit dem vorhandenen Platz auszukommen; Sie dürfen auch stichwortartig antworten. Sollten Sie wider Erwarten nicht mit dem vorgegebenen Platz auskommen, benutzen Sie bitte die beiden an dieser Klausur anhängenden Leerseiten. **Es werden nur Aufgaben gewertet, die sich auf dem offiziellen Klausurpapier befinden.** Eigenes Papier ist nur für Ihre persönlichen Notizen erlaubt.

4. Kreuzen Sie die bearbeiteten Aufgaben auf dem Deckblatt an. Schreiben Sie unbedingt *auf jedes Blatt* Ihrer Klausur Ihren Namen und Ihre Matrikelnummer, auf die Zusatzblätter auch die Nummer der Aufgabe.

5. Geben Sie die gesamte Klausur ab. Lösen Sie die Blätter nicht voneinander.

6. Es sind *keine Hilfsmittel* zugelassen.

7. Lesen Sie vor der Bearbeitung einer Aufgabe den *gesamten* Aufgabentext sorgfältig durch.

8. Es sind maximal 100 Punkte erreichbar. Wenn Sie mindestens 50 Punkte erreichen, haben Sie die Klausur bestanden.

9. Sie erhalten die korrigierte Klausur zurück, zusammen mit einer Bescheinigung für das Finanzamt und ggf. dem Übungsschein.

10. Legen Sie jetzt noch Ihren Studierendenausweis und einen amtlichen Lichtbildausweis bereit, dann kann die Arbeit beginnen. Viel Erfolg!

Musterlösung

Aufgabe 1: Objektgeflechte (10 Punkte)

Gegeben sei folgendes Java-Programm:

```
public class FahrzeugDemo {

	Fahrzeug auto;
	Fahrzeug motorrad;
	Fahrzeug fahrrad;

	public static void main(String[] args) {
		FahrzeugDemo fuhrpark = new FahrzeugDemo();
		Fahrzeug meinFahrzeug = new Fahrzeug();

		fuhrpark.auto = new Fahrzeug();
		fuhrpark.auto.gibName("Auto");

		fuhrpark.motorrad = meinFahrzeug;
		fuhrpark.motorrad.gibName("Motorrad");

		fuhrpark.fahrrad = meinFahrzeug;
		fuhrpark.fahrrad.gibName("Fahrrad");

		fuhrpark.auto.beschleunigen();
		fuhrpark.motorrad.beschleunigen();
		fuhrpark.fahrrad.beschleunigen();
	}

}

class Fahrzeug {
	String name;

	public void gibName(String name) {
		this.name = name;
	}

	public void beschleunigen() {
		System.out.println(name + " faehrt schneller...");
	}
}
```

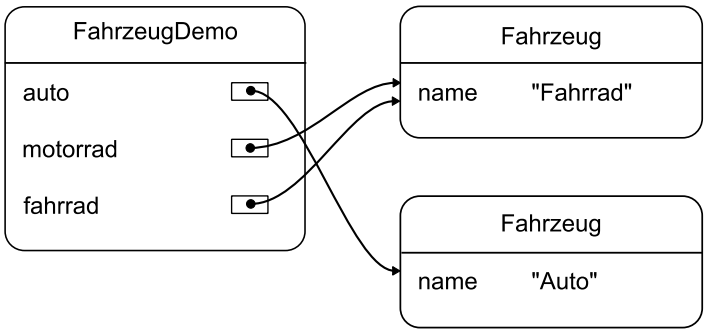
Welche Ausgabe erzeugt die `main`-Methode? (2 Punkte)

```
Auto faehrt schneller...
Fahrrad faehrt schneller...
Fahrrad faehrt schneller...
```

(Fortsetzung der Aufgabe auf folgender Seite)

Musterlösung

Zeichnen Sie das Objektgeflecht, welches bei Beendigung der `main`-Methode besteht. (3 Punkte)



Erklären Sie mit Hilfe Ihrer Zeichnung des Objektgeflechts, wie es zu der (ggf. unerwarteten) Ausgabe kommt. (4 Punkte)

Beim Aufruf der `main`-Methode wird ein neues Objekt der Klasse `Fahrzeug` erzeugt und eine Referenz auf dieses Objekt in der Variable `meinFahrzeug` gespeichert. Eine weitere, neue Instanz von Fahrzeug wird der Variable `auto` zugewiesen und mit „Auto" benannt. Bei der Definition der Variablen `motorrad` und `fahrzeug` wird diesen der Wert der Variable `meinFahrzeug` zugewiesen, sodass diese drei Variablen Referenzen auf das gleiche Objekt, nämlich der zuerst erzeugten Instanz von `Fahrzeug`, zum Wert haben.

Wie wird dieses Phänomen genannt? (1 Punkt)

Dieses Phänomen wird Aliasing genannt, die verschiedenen Referenzen auf das gleiche Objekt sind Aliase (zueinander).

Musterlösung

Aufgabe 2: Überschreiben und Überladen (10 Punkte)

Gegeben sei folgendes Java-Programm:

```
interface A {
	void v();
}

interface B extends A {
	void x(B b);
}

class E implements A {
	public void v() {
		System.out.println("v() in E");
	}

	public void x(G g) {
		System.out.println("x(G g) in E");
	}
}

class F extends E implements B {
	public void x(A a) {
		System.out.println("x(A a) in F");
	}

	public void x(B b) {
		System.out.println("x(B b) in F");
	}
}

class G extends F {
	public void v() {
		System.out.println("v() in G");
	}

	public void x(B b) {
		System.out.println("x(B b) in G");
	}
}

public class Demo {
	public static void main(String[] args) {
		A a = new F();
		B b = new G();

		E e = new E();
		F f = new F();
		G g = new G();

		a.v();
		b.v();

		g.x(e);
		g.x(f);
		g.x(g);
		g.x(b);
	}
}
```

Welche Zeilen werden durch die sechs Methodenaufrufe bei Ausführung des Programms auf der Konsole ausgegeben? Begründen Sie Ihre Antwort!

(Fortsetzung der Aufgabe auf folgender Seite)

Musterlösung

(Fortsetzung von Aufgabe 3)

Aufruf von `a.v()` (1 Punkt)

Ausgabe: `v() in E`
Der deklarierte Typ von `a` ist `A`, ein Interface, welches die Methode `v()` deklariert. Zur Laufzeit referenziert `a` eine Instanz der Klasse `F`, welche das Interface `A` (über die Superklasse `E` und das Interface `B`) implementiert. Da `F` keine eigene Implementierung für `v()` bereitstellt, bindet der Aufruf an die geerbte Implementierung aus `E`.

Aufruf von `b.v()` (1 Punkt)

Ausgabe: `v() in G`
Der deklarierte Typ von `b` ist `B`, ein Interface, welches die Deklaration der Methode `v()` von seinem Superinterface `A` erbt. Zur Laufzeit referenziert `b` eine Instanz der Klasse `G`, sodass der Aufruf an die eigene Implementierung von `v()` in `G` bindet.

Aufruf von `g.x(e)` (2 Punkte)

Ausgabe: `x(A a) in F`
Der deklarierte Typ von `g` ist `G`; der von `e` ist `E`. In der Klasse `G` existieren drei Methoden mit dem Namen ‚x' und einem Parameter: `x(G g)` (von `E` geerbt), `x(A a)` (von `F` geerbt) und `x(B b)`. Da `E` kein Subtyp von `G` und `B` ist, darf der Aufruf nicht an `x(G g)` oder `x(B b)` binden, sondern nur an die geerbte Implementierung von `x(A a)`.

Aufruf von `g.x(f)` (2 Punkte)

Ausgabe: `x(B b) in G`
Der deklarierte Typ von `g` ist `G`; der von `f` ist `F`. Sowohl die geerbte Implementierung von `x(A a)`, als auch `x(B b)` in der Klasse `G` akzeptieren ein Objekt von Typ `F` als Parameter, da `A` und `B` beide Supertypen von `F` sind. Jedoch ist `B` ein Subtyp von `A`, sodass die Implementierung von `x(B b)` die speziellere ist und an sie gebunden wird.

Aufruf von `g.x(g)` (2 Punkte)

Ausgabe: `x(G g) in E`
Der deklarierte Typ von `g` ist `G`. Sowohl die geerbten Implementierungen von `x(G g)` (von `E`) und `x(A a)` (von `F`) als auch die Implementierung von `x(B b)` akzeptieren ein Objekt vom Typ `G` als Parameter. Es ist jedoch `G` ein Subtyp von sowohl `A` als auch `B`, sodass `x(G g)` die speziellste Implementierung ist, an die damit gebunden wird.

Aufruf von `g.x(b)` (2 Punkte)

Ausgabe: `x(B b) in G`
Der deklarierte Typ von `g` ist `G`; der von `b` ist `B`. Sowohl die geerbte Implementierung von `x(A a)`, als auch `x(B b)` in der Klasse `G` akzeptieren ein Objekt von Typ `B` als Parameter, da `B` ein Subtyp von `A` ist. Die Methode `x(B b)` ist daher jedoch auch die speziellere, sodass an sie gebunden wird.

Musterlösung

Aufgabe 3: Kontrollstrukturen (10 Punkte)

Vervollständigen Sie das folgende Java Programm, sodass bei Terminierung der main-Methode die folgenden Zeilen auf der Konsole ausgegeben werden.

```
Alle meine Entchen ...
Alle meine Entchen ...
Alle meine Entchen ...
Alle meine Entchen ...
```

Implementieren Sie dazu vier verschiedene Schleifen, die innerhalb der Methode `schreiben(List<String> liste)` über die Elemente des Parameters `liste` iterieren, und zwar:

- jeweils einmal mit den beiden Varianten einer `for`-Schleife und
- jeweils einmal mit den beiden Varianten einer `while`-Schleife.

Hinweis: Jeweils eine Variante der beiden Schleifenimplementierungen nutzt aus, dass das Interface `List<T>` einen indizierten Zugriff ermöglicht, während die andere Variante ausnutzt, dass das Interface `List<T>` das Interface `Iterable<T>` erweitert (also `Iterable<T>` ein Supertyp von `List<T>` ist.).

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

public class SchleifenDemo {

	public static void main(String[] args) {
		List<String> eineListe = new ArrayList<String>();
		eineListe.add("Alle");
		eineListe.add("meine");
		eineListe.add("Entchen");
		eineListe.add("...");
		schreiben(eineListe);
	}

	static void schreiben(List<String> liste) {


		// for-Schleife mit indiziertem Zugriff auf die Elemente.
```

(2,5 Punkte)

```
		for (int i = 0; i < liste.size(); i++) {
			System.out.print(liste.get(i) + " ");
		}


		System.out.println();
```

(Fortsetzung der Aufgabe auf folgender Seite)

Musterlösung

```
// for-Schleife, die Iterable<String> ausnutzt.
```

(2,5 Punkte)

```
for (String s : liste) {
        System.out.print(s + " ");
}
```

```
System.out.println();

// while-Schleife mit indiziertem Zugriff auf die Elemente.
```

(2,5 Punkte)

```
int i = 0;
while (i < liste.size()) {
        System.out.print(liste.get(i++) + " ");
}
```

```
System.out.println();

// while-Schleife, die Iterable<String> ausnutzt.
```

(2,5 Punkte)

```
Iterator<String> iterator = liste.iterator();
while (iterator.hasNext()) {
        System.out.print(iterator.next() + " ");
}
```

```
    }
}
```

Musterlösung

Aufgabe 4: Aufzählungstypen (10 Punkte)

Die Arbeitstage eines Unternehmens (Montag bis Freitag) sollen als Aufzählungstyp `Arbeitstag` implementiert werden. Die Werte des Aufzählungstypen sollen über die folgenden vier Methoden verfügen:

- `boolean istNach(Arbeitstag tag)`: Liefert `true`, wenn der durch das Empfängerobjekt bezeichnete Arbeitstag *vor* dem Arbeitstag liegt, der durch das Objekt bezeichnet wird, welches durch den Parameter `tag` referenziert wird; ansonsten `false`. (Wobei angenommen wird, dass beide Objekte Tage der gleichen Woche bezeichnen.)
- `boolean istVor(Arbeitstag tag)`: Liefert `true`, wenn der durch das Empfängerobjekt bezeichnete Arbeitstag *nach* dem Arbeitstag liegt, der durch das Objekt bezeichnet wird, welches durch den Parameter `tag` referenziert wird; ansonsten `false`. (Wobei angenommen wird, dass beide Objekte Tage der gleichen Woche bezeichnen.)
- `Arbeitstag naechsterArbeitstag()`: Liefert dasjenige Element der Aufzählung, welches den Arbeitstag beschreibt, der *nach* dem durch das Empfängerobjekt bezeichneten Arbeitstag liegt; wobei dies auch der erste Arbeitstag der folgenden Woche sein kann.
- `Arbeitstag vorherigerArbeitstag()`: Liefert dasjenige Element der Aufzählung, welches den Arbeitstag beschreibt, der *vor* dem durch das Empfängerobjekt bezeichneten Arbeitstag liegt; wobei dies auch der letzte Arbeitstag der vorherigen Woche sein kann.

Die `main`-Methode der Klasse `ArbeitstagDemo` verdeutlicht die Bedeutung der Methoden anhand einiger Beispiele. Der in jeder Zeile hinten angefügte Kommentar gibt dabei jeweils den Wert an, zu dem der Ausdruck dieser Zeile auswerten soll.

```
public class ArbeitstagDemo {
      public static void main(String[] args) {

            Arbeitstag.MITTWOCH.istVor(Arbeitstag.MONTAG);      // false
            Arbeitstag.MITTWOCH.istVor(Arbeitstag.DIENSTAG);    // false
            Arbeitstag.MITTWOCH.istVor(Arbeitstag.MITTWOCH);    // false
            Arbeitstag.MITTWOCH.istVor(Arbeitstag.DONNERSTAG); // true
            Arbeitstag.MITTWOCH.istVor(Arbeitstag.FREITAG);     // true

            Arbeitstag.MITTWOCH.istNach(Arbeitstag.DIENSTAG);   // true
            Arbeitstag.MITTWOCH.istNach(Arbeitstag.MITTWOCH);   // false
            Arbeitstag.MITTWOCH.istNach(Arbeitstag.DONNERSTAG);// false

            Arbeitstag.MITTWOCH.naechsterArbeitstag();          // Arbeitstag.DONNERSTAG
            Arbeitstag.MITTWOCH.vorherigerArbeitstag();         // Arbeitstag.DIENSTAG

            Arbeitstag.FREITAG.naechsterArbeitstag();           // Arbeitstag.MONTAG
            Arbeitstag.MONTAG.vorherigerArbeitstag();           // Arbeitstag.FREITAG
      }
}
```

Aufgrund der guten Auftragslage plant das Unternehmen eine Erweiterung des Arbeitsgeschäfts auf Samstag und Sonntag. Vervollständigen Sie folgende Implementierung der Klasse `Arbeitstag` derart, dass die Methodenimplementierungen bei einer Erweiterung des Aufzählungstyps um die Elemente `SAMSTAG` und `SONNTAG` nicht verändert werden müssen.

(Fortsetzung der Aufgabe auf folgender Seite)

Musterlösung

```
enum Arbeitstag {
        MONTAG, DIENSTAG, MITTWOCH, DONNERSTAG, FREITAG;
```

(1 Punkt)

```
        boolean istNach(Arbeitstag tag) {

                return this.ordinal() > tag.ordinal();

        }
```

(1 Punkt)

```
        boolean istVor(Arbeitstag tag) {

                return this.ordinal() < tag.ordinal();

        }
```

(4 Punkte)

```
        Arbeitstag naechsterArbeitstag() {

                int morgenIndex = this.ordinal() + 1;
                if (morgenIndex == Arbeitstag.values().length)
                        morgenIndex = 0;
                return Arbeitstag.values()[morgenIndex];

        }
```

(4 Punkte)

```
        Arbeitstag vorherigerArbeitstag() {

                int gesternIndex = this.ordinal() - 1;
                if (gesternIndex < 0)
                        gesternIndex = Arbeitstag.values().length - 1;
                return Arbeitstag.values()[gesternIndex];

        }
}
```

Musterlösung

Aufgabe 5: Exception Handling (10 Punkte)

Es soll ein Simulator-Programm für einen CD-Spieler in Java implementiert werden. Eine Fehlbedienung besteht dann, wenn die Methode `wiedergeben()` aufgerufen wird und das Feld `cd` des Empfängerobjektes der Klasse `CdPlayer` `null` ist; d.h., wenn der Benutzer in der Simulation zuvor keine CD eingelegt hat. Diese Fehlbedienung soll durch das Auslösen einer Ausnahme vom Typ `CdFachLeerException` angezeigt werden. Tritt eine solche Ausnahme auf, soll an der *Aufrufstelle* von `wiedergeben()` die Ausführung abgebrochen und die Meldung „`Keine CD eingelegt.`" auf der Konsole ausgegeben werden.

Ergänzen Sie das folgende Programm an den vorgesehenen Stellen um Code für eine solche Ausnahmebehandlung. Beachten Sie, dass nicht an allen Stellen eine Ergänzung notwendig ist. Tragen Sie an den Stellen, an denen kein weiterer Code eingefügt werden muss, „`/* k. E. n. */`" (für „keine Ergänzung notwendig") ein.

```
class CdFachLeerException extends Exception { }
class Cd /* k. E. n. */ { }
class CdPlayer /* k. E. n. */ {
    Cd cd;
    void cdEinlegen(Cd cd) /* k. E. n. */ {
        /* k. E. n. */
        this.cd = cd;
        /* k. E. n. */
    }
    void cdAuswerfen() /* k. E. n. */ {
        /* k. E. n. */
        this.cd = null;
        /* k. E. n. */
    }
```

Musterlösung

(Fortsetzung der Aufgabe auf folgender Seite)

```
        void wiedergeben() throws CdFachLeerException {

                if (this.cd == null)
                        throw new CdFachLeerException();

                /* CD lesen und gebe Audio-Strom ausgeben...
                 * (Ist nicht Teil der Aufgabenstellung,
                 * muss also nicht implementiert werden.)     */

                /* k. E. n. */

        }
}

public class CdPlayerDemo {

        public static void main(String[] args) {
                CdPlayer player = new CdPlayer();

                Cd cd = new Cd();
                player.cdEinlegen(cd);

                try {
                player.wiedergeben();
                } catch (CdFachLeerException e) {
                        System.out.println("Keine CD eingelegt.");
                }

        }

}
```

Musterlösung

Aufgabe 6: Interfaces (10 Punkte)

Aus welchen Elementen kann eine Interface-Deklaration bestehen? (6 Punkte)

- (teilweise optionale) Zugriffsmodifikatoren;
- Name des Interfaces;
- eine optionale Liste von Typparametern, ggf. mit Schranken;
- eine optionale Liste von Namen von Interfaces (als Supertypen), die dieses Interfaces erweitert;
- benannte Konstanten, d.h. final deklarierte statische Felder;
- erweiterte Methodensignaturen, die durch Klassen, die dieses Interface als Supertyp deklarieren, implementiert werden müssen.

Einer Ihrer Kollegen sagt Ihnen: „Interfaces in Java sind überflüssig. Statt ihnen könnte man in Java genauso abstrakte Klassen verwenden." Was sagen Sie dazu? Begründen Sie! (4 Punkte)

Zwar ist es möglich, abstrakte Klassen zur Deklaration der öffentlichen Schnittstelle von Objekten zu verwenden; allerdings erlaubt Java aufgrund der fehlenden Mehrfachvererbung nicht, zu einer Klasse mehrere (abstrakte) Superklassen zu spezifizieren. In der Konsequenz könnte jede Klasse nur einen direkten Supertyp haben. Dahingegen können zu einer Klasse mehrere Interfaces als Supertypen spezifiziert werden, sodass eine Klasse auch mehrere direkte Supertypen haben kann.

Musterlösung

Aufgabe 7: (Beschränkt) Parametrische Polymorphie(10 Punkte)

Ein Hersteller von HiFi-Komponenten möchte eine universelle Klasse `Wiedergabegeraet` für die Wiedergabegeräte verschiedener Medien implementieren. Alle Wiedergabegeräte haben gemein, dass sie über die folgenden zwei Methoden verfügen:

- Eine Methode mit dem Namen `mediumEinlegen`, welche eine Referenz auf die Objektrepräsentation eines Mediums als Parameter erhält und eine weitere Referenz auf dieses Objekt als Alias in einem Feld `medium` speichert
- und eine Methode mit dem Namen `mediumEntnehmen`, welche eine Referenz auf die Objektrepräsentation, die auch das Feld `medium` referenziert, zurückgibt und den Alias des Feldes löscht.

Daher sollen diese Methoden unter Verwendung des parametrischen Polymorphismus *typsicher* in der gemeinsamen Basisklasse implementiert werden. Die folgenden Implementierungen der Klassen `WiedergabegeraetDemo` und `Cd` veranschaulichen die Verwendung dieser Klasse.

```
public class WiedergabegeraetDemo {

	public static void main(String[] args) {
		Wiedergabegeraet<Cd> player = new Wiedergabegeraet<Cd>();
		Cd cd = new Cd();
		player.mediumEinlegen(cd);
		Cd medium = player.mediumEntnehmen();
	}

}
class Cd {}
```

Geben Sie eine Implementierung der Klasse `Wiedergabegeraet` an. (5 Punkte)

```
class Wiedergabegeraet<T> {

	T medium;

	T mediumEntnehmen() {
		T altesMedium = medium;
		this.medium = null;
		return altesMedium;
	}

	void mediumEinlegen(T medium) {
		this.medium = medium;
	}

}
```

(Fortsetzung der Aufgabe auf folgender Seite)

Musterlösung

Nachdem der Hersteller sein Produktportfolio neu ausgerichtet hat, soll die Klasse `Wiedergabegeraet` nun so implementiert werden, dass sie ausschließlich Musikmedien unterstützt. Hierbei wird angenommen, dass die diese Medien repräsentierenden Klassen das folgende Interface `Hoerbar` implementieren:

```
interface Hoerbar{
	void liedSpielen(int liedNr);
	int anzahlLieder();
}
```

Neben den beiden oben genannten Methoden soll die Klasse `Wiedergabegeraet` nun über

- eine weitere Methode mit dem Namen `alleSpielen` verfügen, die unter Verwendung der Methode `liedSpielen(int liedNr)` und `anzahlLieder()` nacheinander alle Musikstücke des Mediums nacheinander wiedergibt.

Die folgenden Implementierungen der Klasse `WiedergabegeraetDemo2` und `Cd` veranschaulichen die Verwendung der gemeinsamen Basisklasse.

```
public class WiedergabegeraetDemo2 {

	public static void main(String[] args) {
		Wiedergabegeraet<Cd> player = new Wiedergabegeraet<Cd>();
		Cd cd = new Cd();
		player.mediumEinlegen(cd);
		player.alleSpielen();
		Cd medium = player.mediumEntnehmen();
	}

}
class Cd implements Hoerbar {
	public void liedSpielen(int liedNr)	{ /*...*/ 		}
	public int anzahlLieder()		{ /*...*/	return 42;	}
}
```

Geben Sie eine neue Implementierung der Klasse `Wiedergabegeraet` an. (5 Punkte)

```
class Wiedergabegeraet<T extends Hoerbar> {

	T medium;

	T mediumEntnehmen() {
		T altesMedium = medium;
		this.medium = null;
		return altesMedium;
	}

	void mediumEinlegen(T medium) {
		this.medium = medium;
	}

	void alleSpielen(){
		for (int i = 0 ; i < medium.anzahlLieder() ; i++){
			medium.liedSpielen(i);
		}
	}
}
```

Musterlösung

Musterlösung

Aufgabe 8: Threads (10 Punkte)

Gegeben sei folgendes Java-Programm.

```
import java.util.LinkedList;

public class ThreadDemo {

	public static void main(String[] args) {
		Lager lager = new Lager();
		lager.einlagern("Gegenstand 1");
		lager.einlagern("Gegenstand 2");
		lager.einlagern("Gegenstand 3");
		lager.einlagern("Gegenstand 4");

		Verbraucher v1 = new Verbraucher(lager);
		Verbraucher v2 = new Verbraucher(lager);
		Verbraucher v3 = new Verbraucher(lager);
		Verbraucher v4 = new Verbraucher(lager);

		v1.start();
		v2.start();
		v3.start();
		v4.start();
	}

}

class Lager {

	LinkedList<Object> gegenstaende = new LinkedList<Object>();

	public Object entnehmen() {
		Object letztes = gegenstaende.getLast();
		gegenstaende.removeLast();
		return letztes;
	}

	public void einlagern(Object gegenstand) {
		gegenstaende.addLast(gegenstand);
	}

}

class Verbraucher extends Thread {

	Lager lager;

	Verbraucher(Lager lager){
		this.lager = lager;
	}

	public void run() {
		Object gegenstand = lager.entnehmen();
		System.out.println(gegenstand);
	}

}
```

(Fortsetzung der Aufgabe auf folgender Seite)

Musterlösung

Nach einer Ausführung des Programms, erscheinen die folgenden Zeilen auf der Konsole.

```
Gegenstand 4
Gegenstand 4
Gegenstand 2
Gegenstand 1
```

Erläutern Sie, wie es hierzu gekommen ist.
Hinweis: Beachten Sie, dass in der Ausgabe `Gegenstand 4` zwei Mal, `Gegenstand 3` hingegen gar nicht ausgegeben wird. (5 Punkte)

Zunächst nimmt einer der vier Threads t seine Arbeit auf. Durch die erste Anweisung in der `run()`-Methode der Klasse `Verbraucher` wird die Methode `entnehmen()` auf dem durch das Feld `lager` referenzierten Objekt der Klasse `Lager` aufgerufen, in welcher zunächst eine Referenz auf das letzte Element (`Gegenstand 4`) der List `gegenstaende` geholt und in der lokalen Variable `letztes` gespeichert wird. Noch bevor dieses letzte Element in der folgenden Anweisung aus der Liste `gegenstaende` entfernt wird, wird nun die Ausführung des Threads durch den Scheduler unterbrochen und die Ausführung eines anderen Threads t' begonnen (oder fortgesetzt). Erreicht dieser Thread nun ebenfalls die erste Zeile der Methode `entnehmen()`, wird eine weitere Referenz auf `Gegenstand 4` geholt und anschließend durch t' `Gegenstand 4` aus der Liste `gegenstaende` des Lagers entfernt.
Nun wird die Ausführung von t forgesetzt. In der folgenden Anweisung wird das letzte Element der Liste entfernt; welches nun jedoch `Gegenstand 3` ist, obwohl die lokale Variable `letztes` den `Gegenstand 4` referenziert. Während `Gegenstad 4` also durch zwei Threads behandelt wird, wird `Gegenstand 3` aus der Liste `gegenstaende` entfernt, ohne jemals eine Behandlung erfahren zu haben.

An welcher Stelle im Programmcode liegt das Problem? (3 Punkte)

In der Methode `entnehmen` findet ein nicht synchronisierter Zugriff auf die gemeinsam genutzte Ressource (die Liste `gegenstaende`) statt. Mindestens die Aufrufe `lager.getLast()` und `lager.removeLast()` müssen in einem synchronisierten Code-Block liegen, was aber nicht der Fall ist.

Geben Sie eine alternative Implementierung an, die dieses Problem vermeidet. (2 Punkte)

```
public synchronized Object entnehmen() {
        Object letztes = gegenstaende.getLast();
        gegenstaende.removeLast();
        return letztes;
}
```

Musterlösung

Aufgabe 9: Verteilte Systeme (10 Punkte)

Im Internet werden Daten in Paketen endlicher Länge übertragen, während zur Kommunikation zwischen Prozessen jedoch häufig ein kontinuierlicher Datenstrom angemessener ist. Welches Konzept bietet eine Abstraktion von den technischen Verwaltungsaufgaben in der Kommunikation über das Netzwerk, sodass diese Diskrepanz überwunden wird? (1 Punkt)

(vgl. Kapitel 7.2.1 im Kurstext) Durch Sockets, die im Rahmen von Berkeley Unix entwickelt wurden, wird eine Abstraktionsschicht zur Kommunikation zwischen Prozessen bereitgestellt, die den Programmierer von folgenden Aufgaben befreit:

Welche Tätigkeiten fallen hier an? (9 Punkte)

- Zerteilen der Daten in Pakete;
- Verpacken der Daten in Pakete mit Adress- und Verwaltungsinformationen sowie dem Dateninhalt
- Auspacken der ankommenden Pakete (insbes. Entfernen der Adress- und Verwaltungsinformationen);
- Ordnen der Pakete in der richtigen Reihenfolge;
- Anfordern von fehlenden (im unzuverlässigen Internet ggf. verloren gegangenen) Paketen;
- Zusammensetzen der Daten zu einem kontinuierlichen Datenstrom.

Musterlösung

Aufgabe 10: MVC-Architektur (10 Punkte)

Geben Sie an, ob die folgenden Aussagen wahr oder falsch sind.

Achtung! Falsche Antworten geben Minuspunkte, eine gesamt negative Punktzahl bei dieser Aufgabe wird Ihnen aber auf null Punkte aufgerundet.

(vgl. Kapitel 5.3.1 im Kurstext)

Die MVC-Architektur dient zur Koordination der Kommunikation zwischen entfernten Prozessen.	☐ Wahr	☑ Falsch
Das ‚V' in ‚MVC' steht für ‚Visibility' und verweist damit auf die Aufgabe, die Elemente einer Benutzerschnittstelle sichtbar bzw. unsichtbar zu machen.	☐ Wahr	☑ Falsch
Das ‚C' in ‚MVC' steht für ‚Controller' und verweist damit auf die Aufgabe, die Benutzereingaben entgegenzunehmen und die Interaktion zwischen Anwendungsschnittstelle und Präsentation zu koordinieren.	☑ Wahr	☐ Falsch
Das ‚M' in ‚MVC' steht für ‚Model' und verweist damit auf die Aufgabe, die Daten einer Anwendung persistent auf dem Sekundärspeicher eines Computers zu speichern.	☐ Wahr	☑ Falsch
Obwohl es in einer Anwendung notwendig sein kann, mehrere verschiedene Darstellungen sowie die zugehörige Steuerung vorzusehen, existiert immer nur eine Schnittstelle zur gesteuerten Anwendung.	☑ Wahr	☐ Falsch
Der Controller ist nur dann involviert, wenn durch eine Benutzereingabe eine Operation in der Anwendung ausgelöst wird.	☐ Wahr	☑ Falsch
Das Model hat die Aufgabe, Änderungen (z.B. Änderungen von Daten oder Status) sowohl an die Steuerung als auch an die Präsentation zu melden.	☑ Wahr	☐ Falsch
Die Darstellungskomponenten haben keinen direkten Zugriff auf das Modell, sondern greifen ausschließlich über die Steuerung auf das Modell zu.	☐ Wahr	☑ Falsch
Obwohl es ein Programmgerüst ist, bietet die MVC-Architektur viele Interpretations- und Anpassungsmöglichkeiten.	☑ Wahr	☐ Falsch
Da die MVC-Architektur die Entwicklung des AWT beeinflusst hat, bietet das AWT eine gute Basis, um Benutzerschnittstellen gemäß MVC-Architektur zu implementieren.	☑ Wahr	☐ Falsch

Musterlösung

Zusätzlicher Platz für Ihre Lösungen

Ergänzung zu Aufgabe Nr.

Ergänzung zu Aufgabe Nr.

Musterlösung

Zusätzlicher Platz für Ihre Lösungen

Ergänzung zu Aufgabe Nr.

Ergänzung zu Aufgabe Nr.

Musterlösung