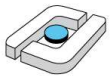
Programmierung 1

Aufgabenblatt 10-12

Prof. Dr.-Ing. Heiko Tapken / Programmier-Team
Abgabe/Testat: vom Dozenten festzulegen
Bestehensgrenze 35 Punkte
Wintersemester 2020/21



Hochschule Osnabrück
University of Applied Sciences

Aufgabe 1 (einfache Vererbung) [4 Punkte]

Eine Klasse RundfunkEmpfangsGeraet sei folgendermaßen definiert:

```
public class RundFunkEmpfangsGeraet {
	int lautstaerke;
	boolean eingeschaltet;

	RundFunkEmpfangsGeraet() {
	}

	/**
	 * verändere Lautstaerke um x nach oben oder unten, je nach
Vorzeichen von x
	 */
	void volume(int x) {
	}

	/** Erhöhe Lautstaerke um 1 */
	void lauter() {
	}

	// getter und setter

	/** Vermindere Lautstaerke um 1. */
	void leiser() {
	}

	/** Schalte ein. */
	void an() {
	}

	/** Schalte aus */
	void aus() {
	}

	/** an oder aus? */
	boolean istAn() {
	}
}
```

a) Implementieren Sie die Methoden dieser Klasse adäquat. Ein neues Objekt der Klasse RundFunkEmpfangsGeraet soll für Lautstaerke den Wert 0 haben und ausgeschaltet sein. Einstellungen an einem Empfangsgerät sollen nur verändert werden können, wenn das Gerät an ist.

b) Leiten Sie von der Klasse RundfunkEmpfangsGeraet zwei Klassen Radio und Fernseher ab. Bei Radios und Fernsehern soll es sich dabei um Empfangsgeräte handeln, die zusätzlich jeweils noch folgende Attribute besitzen bzw. Methoden bereitstellen.

Für die Klasse Radio soll gelten:

- Diese Klasse besitzt ein Attribut double frequenz, das über die Methode void waehleSender(double newFrequenz) verändert werden kann, falls das Radio eingeschaltet ist.
- Ein neues Radio-Objekt ist auf die Frequenz 87.5 MHz eingestellt.

Für die Klasse Fernseher soll gelten:

- Die Klasse Fernseher besitzt ein Attribut int kanal, das über die Methode void waehleKanal(int newKanal) verändert werden kann, falls der Fernseher eingeschaltet ist.
- Ein neues Fernseher-Objekt ist auf den Kanal 1 eingestellt.

Alle(!) Klassen sollen über geeignete toString-Methoden verfügen.

c) Schreiben Sie ein Testprogramm, das die korrekte Funktion Ihres Programmes zeigt.

Aufgabe 2 (Vererbung) [4 Punkte]

Gegeben seien die folgenden drei Klassen:

```
public class Y1{
    public void test(){
        System.out.println("1\n");
    }
}
```

```
public class Y2 extends Y1{
    @Override
    public void test(){
        System.out.println("2\n");
    }

    public void do1(){
        System.out.println("y2:do1 ");
        test();
    }

    public void do2(){
        System.out.println("y2:do2 ");
        super.test();
    }

    public void do3(){
        System.out.println("y2:do3 ");
        do4();
    }

    public void do4(){
        System.out.println("3\n");
    }
}
```

```
public class Y3 extends Y2{
    @Override
    public void do1(){
        System.out.println("y3:do1 ");
        test();
    }

    @Override
    public void do2(){
        System.out.println("y3:do2 ");
        super.do3();
    }

    @Override
    public void do3(){
        System.out.println("y3:do3 ");
        super.do2();
    }

    @Override
    public void do4(){
        System.out.println("y3:do4 ");
        do2();
    }
}
```

In einem anderen Programm werden folgende drei Objekte erzeugt.

```
Y1 x1 = new Y1();
Y2 x2 = new Y2();
Y3 x3 = new Y3();
```

Welche Ausgabe liefern die folgenden einzelnen Zeilen (falls sie keine Ausgabe angeben können, beschreiben Sie, was passiert)?

```
a) x1.test()
b) x2.test()
c) x2.do1()
d) x2.do2()
e) x2.do3()
f) x2.do4()
g) x3.test()
h) x3.do1()
i) x3.do2()
j) x3.do3()
k) x3.do4()
```

Aufgabe 3 [5 Punkte]

Langtons Ameise ist eine Turingmaschine mit einem zweidimensionalen Speicher und wurde 1986 von Christopher Langton entwickelt. Sie ist ein Beispiel dafür, dass ein deterministisches (das heißt nicht zufallsbedingtes) System aus einfachen Regeln sowohl für den Menschen visuell überraschend ungeordnet erscheinende als auch regelmäßig erscheinende Zustände annehmen kann. (Quelle: Wikipedia)

Ausgangspunkt:
Gegeben sei eine quadratische 2-dimensionale Welt mit SIZE * SIZE Zellen. SIZE (hier SIZE = 200) sei die Anzahl der Reihen und Spalten. Die Zellen können zwei Zustände einnehmen (weiß und schwarz). Anfangs sind alle Zellen weiß. In dieser Welt lebt genau eine Ameise. Anfangs befindet sich die Ameise auf der Zelle in der Mitte der Welt und schaut nach Norden. Nun wird STEPS (hier STEPS = 40000) mal folgender „Ameisenalgorithmus“ ausgeführt:

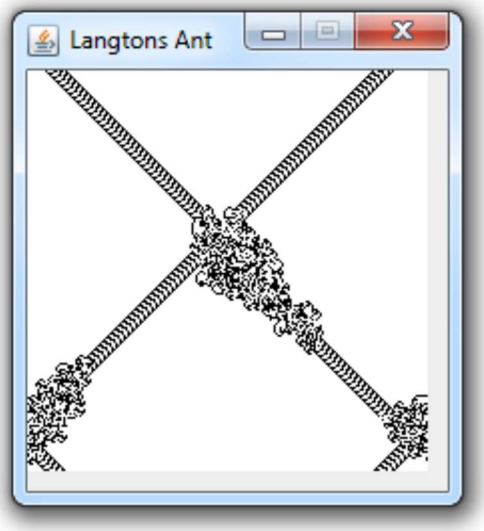
- Befindet sich die Ameise auf einer weißen Zelle, so färbt sie sie schwarz, dreht sich um 90 Grad nach rechts und begibt sich auf die nächste Zelle in der neuen Blickrichtung.
- Befindet sich die Ameise auf einer schwarzen Zelle, so färbt sie sie weiß, dreht sich um 90 Grad nach links und begibt sich auf die nächste Zelle in der neuen Blickrichtung.

Die Welt sei dabei ein Torus, d.h. wenn die Ameise die Welt nach oben verlässt, erscheint sie wieder unten und umgekehrt; wenn sie die Welt nach links verlässt, erscheint sie wieder rechts und umgekehrt.

Aufgabe:

Im OSCA steht ein Programm *LangtonsAnt* zur Verfügung. Implementieren Sie die fehlenden Klassen *World* und *Ant*, die die Welt bzw. die Ameise repräsentieren. Wenn

Sie alles korrekt gemacht haben und das Programm ausführen, sollte innerhalb einiger Sekunden nach und nach das folgende Fenster auf dem Bildschirm erscheinen.



Hinweise:

- Sie brauchen den vorgegebenen Code nicht komplett zu verstehen; schauen Sie sich nur die Stellen an, an denen die Klassen *World* bzw. *Ant* genutzt werden. Hier stehen entsprechende Kommentare. Insbesondere brauchen Sie aus den Klassen *World* und *Ant* nicht auf die gegebenen Klassen zugreifen!
- Sie dürfen die vorgegebenen Klassen nicht ändern!

Aufgabe 4 (Zugriffsrechte) [2 Punkte]

Gegeben sei folgende Klasse:

```
//Verzeichnis: zugriff
//Datei: Zugriffsrechte.java
package zugriff;
public class Zugriffsrechte {
  float attribut1;
  private float attribut2;
  public float attribut3;
  protected float getAttribut2() {
    return this.attribut2;
  }
}
```

und folgendes *Klassen-Gerüst*:

```
//Verzeichnis: <verzeichnis>
//Datei: Zugriffstest.java
package <verzeichnis>;
import zugriff.Zugriffsrechte;
```

```
public class Zugriffstest extends <oberklasse> {
  Zugriffsrechte obj;
  public void testen() {

    this.attribut1 = 3.0F;
    this.attribut2 = 5.6F;
    this.attribut3 = obj.getAttribut2();

    obj.attribut1 = 3.0F;
    obj.attribut2 = 5.6F;
    obj.attribut3 = this.getAttribut2();
  }
}
```

Ersetzen Sie dabei wechselseitig

`<verzeichnis>`	durch (a) `zugriff`	(b) `zugriffstest`
`<oberklasse>`	durch (1) `Zugriffsrechte`	(2) `Object`

und überlegen Sie: Wo liefert der Compiler Fehlermeldungen und wieso?

Aufgabe 5 (Exceptions und Testen) [5 Punkte]

Wir haben in einer früheren Aufgabe einen einfachen Taschenrechner implementiert. Die Methoden und die Variablen haben wir mit static gekennzeichnet und die Funktionalität manuell getestet. Schreiben Sie die Klasse nun so um, dass Sie ohne static auskommt und schreiben Sie eine weitere Klasse TaschenrechnerTest, die nur eine main-Methode enthält, eine Instanz erzeugt und einige Rechenoperationen ausführt.

a) Ergänzen Sie Ihre Lösung der Taschenrechneraufgabe um sinnvolle Ausnahmebehandlung für die Methoden dividiere(int, int) und modulo(int,int).
b) Nur Fortgeschrittene: Ergänzen Sie das Programm um eine grafische Benutzungsoberfläche mit SWING (s. Foliensatz SWING).
c) Das Testen der Methoden einer Klasse kann man sehr schön an ein vorhandenes Tool übergeben. Müssen wir ja nicht selber alles machen. Ein solches Tool ist JUnit (s. auch Foliensatz im OSCA).

   Verwenden Sie das Eclipse-Projekt für den Taschenrechner und erzeugen Sie im default-Package eine TestCase-Klasse. Führen Sie hierzu einen Rechtsklick auf Projektname im Package-Explorer durch und selektieren New -> JUnit Test Case (wenn nicht vorhanden: Other und unter Java -> JUnit -> JUnit Test Case auswählen).

   Legen Sie ein JUnit 4 TestCase an. Was ist überhaupt ein TestCase? Eine spezielle Klasse, die von JUnit zum Testen einer anderen Klasse verwendet wird. In dem TestCase werden verschiedene Tests in Form von Methoden zusammengefasst werden (sehen wir gleich;-)).

Als TestCase-Namen verwenden wir „TaschenrechnerTest“ und als Class-under-Test die Klasse „Taschenrechner“. Das war es auch schon. Eclipse generiert den TestCase und wir können anfangen, die Methoden der Klasse Taschenrechner zu verwenden.

Da alle Tests ein Objekt der Klasse Taschenrechner nutzen, legen wir eine Objektvariable für die Klasse TaschenrechnerTest an: Taschenrechner rechner = new Taschenrechner();

Jetzt erstellen wir neue Tests. Aber wie gehen wir vor?
Bei einem Test wollen wir häufig den berechneten Wert gegen ein bekanntes Ergebnis vergleichen. Wir treffen bspw. die Annahme, dass die Addition der Werte 5 und 3 als Ergebnis den Wert 8 liefert. Die Addition soll von der Methode addiere(int a, int b) der Klasse Taschenrechner durchgeführt werden. Als Annahme gilt, dass bei Übergabe der Werte 5 und 3 der Wert 8 geliefert wird.
Zur Information: Annahmen heißt im englischen Assertion (oder kurz assert).
Unser Test sieht jetzt für die Addition wie folgt aus:

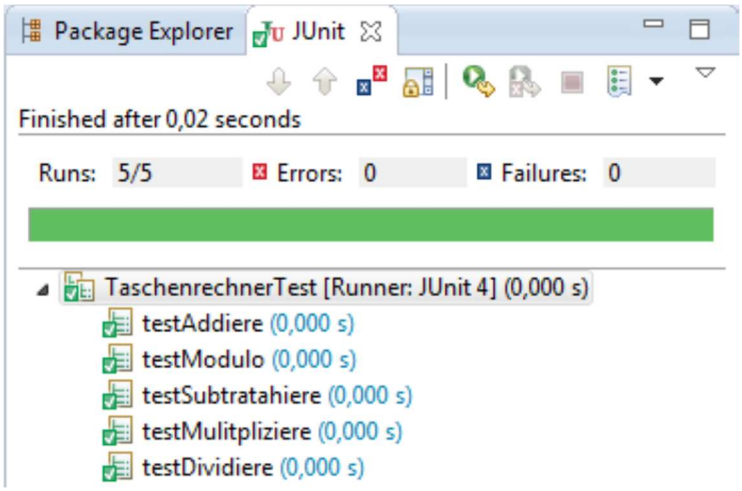
```
@Test
 public void testAddiere() {
   int a=5;
   int b=3;
   assertEquals(8, rechner.addiere(a, b));
}
```

Interessant und wirklich einfach: wir testen die Methode addiere einfach, indem wir die Annahme aufschreiben: assertEquals(8, rechner.addiere(a, b)); Wir nehmen an, dass die Zahl 8 und der Rückgabewert der Methode addiere(5,3) gleich sind (=> assertEquals).

Wir lassen den Test jetzt über JUnit ausführen:

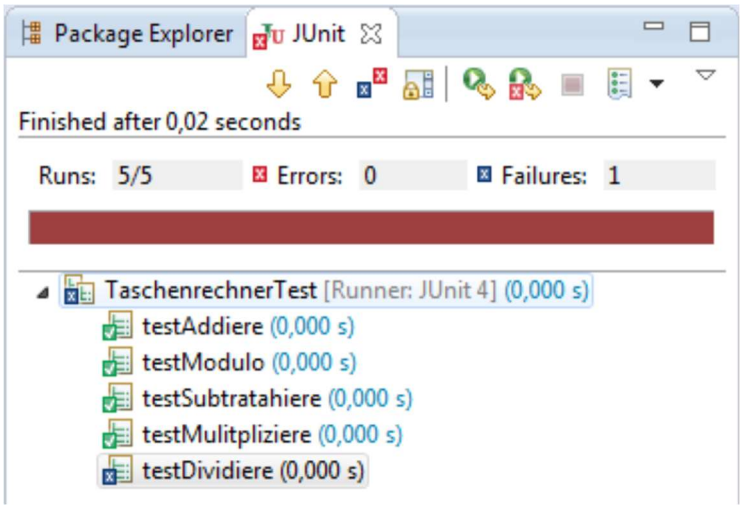
- Im Menü Run und hier den Menüpunkt Run as -> JUnit Test aufrufen
- Auf den Pfeil neben dem klicken und Run As -> JUnit Test verwenden
- Als Shortcut: Alt, Ctrl + Shift + X,T
- Rechtsklick auf das Projekt und RunAs -> JUnit Test auswählen

Nachdem JUnit für den Test gestartet wurde, werden alle Tests ausgeführt. Wenn Sie für jede Methode Tests entwickelt haben, sieht es im besten Fall so aus:



Sie waren erfolgreich!! Alle Tests sind fehlerfrei durchgelaufen. Ist ein Fehler aufgetreten, wird der fehlerhafte Test rot markiert. Doppelklicken Sie auf diesen Test und der Test wird

dargestellt (kann ja sein, dass der Test selber einen Fehler enthält; prüfen Sie dies immer zuerst).



Hier sieht man, dass der Test testDividiere() nicht durchläuft. Sehen wir uns den Test an (Doppelklicken auf testDividiere):

```
@Test
public void testDividiere() {
    assertEquals(1, rechner.dividiere(a, b));
}
```

Der Test ist korrekt. Markieren Sie die Methode dividiere und klicken Sie F3 (alternativ: Rechtsklick-> Open Declaration). Eclipse zeigt die Methode an. Tatsächlich hier ist der Fehler. Dieser wird behoben und der Test solange durchlaufen, bis alles auf Grün steht.

Aufgabe: Realisieren für jede Methode der Klasse Taschenrechner mindestens einen Test. Lassen Sie die Tests über JUnit durchlaufen und machen Sie einen Screenshot des Ergebnisses.

Aufgabe 6 [4 Punkte]

Kennen Sie das Trinkspiel „Meyern“? Bei der „harmlose“ Variante des Spiels namens „Sieben" geht es darum, dass eine Menge an Spielern reihum Zahlen aufsagen und zwar auf folgende Art und Weise: Begonnen wird mit der Zahl 0. Der jeweils nächste Spieler, der an der Reihe ist, muss dann die nächst höhere Zahl nennen, die keine Ziffer 7 enthält und nicht durch 7 teilbar ist. Ansonsten scheidet er aus. Sieger ist/sind der/die Spieler, wer als letzter Spieler nicht ausgeschieden ist oder beim Erreichen der Zahl 10000 noch nicht ausgeschieden ist. Mit Hilfe des folgenden Programms lässt sich das Spiel "Sieben" am Computer spielen.

```
public class UE18Aufgabe1 {
	// Hauptprogramm
	public static void main(String[] args) {
		// ueber die Argumente werden die Spielernamen uebergeben
		if (args.length < 2) {
			IO.println("Bitte mindestens zwei Spielernamen
uebergeben");
			return;
		}
		// Spieler erzeugen
		SiebenSpieler[] spieler = new SiebenSpieler[args.length];
		for (int i = 0; i < args.length; i++) {
			spieler[i] = new SiebenSpieler(args[i]);
		}
		// Spiel durchfuehren
		spielen(spieler);
	}

	// Durchfuehrung eines kompletten Spiels mit den
	// uebergebenen Spielern
	public static void spielen(SiebenSpieler[] spieler) {
		int anzahlAusgeschieden = 0;
		int zahl = 0; // begonnen wird mit der Zahl 0
		hauptschleife: while (zahl < 10000) {
			// ab 10000 wird noch eine Runde gespielt
			// alle Spieler kommen nacheinander an die Reihe
			for (int i = 0; i < spieler.length; i++) {
				if (!spieler[i].istAusgeschieden()) {
					int ergebnis = spieler[i].naechsteZahl(zahl);
					if (!check(zahl, ergebnis)) {
						spieler[i].ausscheiden();
						IO.println("Falsch! " + spieler[i]
								+ " ist ausgeschieden!");
						anzahlAusgeschieden++;
						if (anzahlAusgeschieden ==
spieler.length - 1) {
							// alle bis auf einen Spieler sind
ausgeschieden
							break hauptschleife; // Spiel
zuende
						}
					} else { // eingegebene Zahl war korrekt
						zahl = ergebnis; // zahl hochsetzen
					}
				}
			}
		}
		gibSiegerBekannt(spieler);
	}
```

```
	// Bekanntgabe der/des Siegers (wer nicht ausgeschieden ist)
	public static void gibSiegerBekannt(SiebenSpieler[] spieler) {
		for (int i = 0; i < spieler.length; i++) {
			if (!spieler[i].istAusgeschieden()) {
				IO.println(spieler[i] + " ist Sieger!");
			}
		}
	}

	/**
	 * Ueberpruefung eines Spielzugs
	 *
	 * @param aktuelleZahl
	 *            die aktuelle Zahl des Spiels
	 * @param gelieferteZahl
	 *            die vom Spieler angegebene Zahl
	 * @return genau dann true, wenn gelieferteZahl die naechst hoehere
Zahl
	 *           nach aktuelleZahl ist, die keine Ziffer 7 enthaelt und
nicht
	 *         durch 7 teilbar ist
	 */
	public static boolean check(int aktuelleZahl, int gelieferteZahl) {

	}
}

class SiebenSpieler {
	private String name;

	private boolean ausgeschieden;

	public SiebenSpieler(String name) {
		this.name = name;
		this.ausgeschieden = false;
	}

	public String toString() {
		return this.name;
	}

	public int naechsteZahl(int vorherigeZahl) {
		return IO.readInt(this + ": Zahl nach " + vorherigeZahl + ":
");
	}

	public void ausscheiden() {
		this.ausgeschieden = true;
	}

	public boolean istAusgeschieden() {
		return this.ausgeschieden;
	}
}
```

Aufgaben:

- Implementieren Sie die Methode *check*
- Leiten Sie von der Klasse *SiebenSpieler* eine Klasse *SiebenProgramm* ab, die die Methode *naechsteZahl* so überschreibt, dass Objekte der Klasse *SiebenProgramm*

immer als Sieger des Spiels hervorgehen würden. Ändern Sie die Methode *main* (und nur die!) so ab, dass immer ein Programm als Spieler mitspielt

Ergänzend zum Spiel Sieben dürfen Sie das Programm „Aufmotzen“, indem Sie das Meyern-Spiel zusätzlich unter Verwendung von Vererbung implementieren (es sind nur wenige Zeilen).

Aufgabe 7 [5 Punkte]

Implementieren Sie in Java eine Klasse `Konto`, die ein Bankkonto realisiert. Wählen Sie geeignete Zugriffsrechte für die Attribute und Methoden der Klasse. Ein Konto wird dabei repräsentiert durch einen Kontostand sowie einen eingeräumten Kreditrahmen. Die Klasse soll folgende Methoden zur Verfügung stellen:

- Einen Konstruktor zum Initialisieren eines neuen (leeren) Kontos.
- Einen Copy-Konstruktor zum Initialisieren eines Kontos mit einem bereits existierenden Konto.
- Eine Methode zum Klonieren eines Konto-Objektes.
- Eine Methode zum Überprüfen der Wertegleichheit zweier Konto-Objekte.
- Eine Methode, die den aktuellen Kontostand als String-Objekt zurückliefert.
- Eine Methode zum Einzahlen eines bestimmten Geldbetrages auf ein Konto. Dabei soll gelten: Wenn der Kontostand einmal den Wert von 10000 überschreitet, wird dem Konto im Folgenden ein Kreditrahmen von 3000 eingeräumt.
- Eine Methode zum Abheben eines bestimmten Geldbetrages von einem Konto.
- Eine Methode, die den aktuellen Kontostand als Wert liefert
- Eine Methode zum Überweisen eines bestimmten Geldbetrages von einem Konto auf ein anderes

Schreiben Sie weiterhin ein Programm zum Testen der Klasse.

Aufgabe 8 [5 Punkte]

Implementieren Sie eine ADT-Klasse *Cardinal*, die das Rechnen mit Natürlichen Zahlen (1, 2, 3, ...) ermöglicht!
Die Klasse *Cardinal* soll folgende Methoden besitzen:

- Einen Default-Konstruktor
- einen Konstruktor, der ein Cardinal-Objekt mit einem übergebenen int-Wert initialisiert
- einen Copy-Konstruktor, der ein Cardinal-Objekt mit einem bereits existierenden Cardinal-Objekt initialisiert
- eine Methode clone, die ein Cardinal-Objekt klont
- eine Methode equals, die zwei Cardinal-Objekte auf Wertegleichheit überprüft
- eine Methode toString, die eine String-Repräsentation des Cardinal-Objektes liefert

- eine Klassenmethode sub, die zwei Cardinal-Objekte voneinander subtrahiert
- eine Methode sub, die vom aufgerufenen Cardinal-Objekt ein übergebenes Cardinal-Objekt subtrahiert

Überlegen Sie, wo Fehler auftreten können und setzen Sie Exceptions zur Fehlerbehandlung ein!

Aufgabe 9 [5 Punkte]

Es soll eine Prüfung durchgeführt werden. Die Prüfung besteht aus einem Quiz, das eine Menge an Prüflingen zu absolvieren haben. Ein Quiz besteht aus einer Menge an Fragen. Bei den Fragen handelt es sich um Fragen unterschiedlichen Typs (Wahr/Falsch, MultipleChoice, ...). Bei einer richtigen Antwort bekommt der Prüfling eine der Frage zugeordneten Punktzahl. Eine Prüfung läuft so ab, dass zunächst das Quiz vorbereitet wird, d.h. es wird eine Menge an Fragen eingegeben. Anschließend wird die Prüfung durchgeführt, d.h. die Prüflinge müssen der Reihe nach die Fragen beantworten. Danach wird eine Rangliste nach den erreichten Punkten erzeugt und die Ergebnisse der Prüfung werden ausgegeben.
Das folgende Programm realisiert eine solche Prüfung:

```
abstract class Frage {

	String text; // Fragetext

	int punkte; // zu erreichende Punktzahl

	Frage(String text, int punkte) {
		this.text = text;
		this.punkte = punkte;
	}

	// Frage auf den Bildschirm ausgeben
	void frageStellen() {
		IO.println(this.text);
	}

	// Frage beantworten durch Prüfling, Antwort auswerten
	// und Punkte vergeben
	abstract void frageBeantworten(Pruefling person);

	int getPunkte() {
		return this.punkte;
	}
}

// Klasse, die Wahr/Falsch-Fragen realisiert
class WahrFalschFrage extends Frage {
	boolean richtig; // richtig oder falsch

	WahrFalschFrage(String text, int punkte, boolean richtig) {
		super(text, punkte);
		this.richtig = richtig;
	}

	// Frage beantworten durch Prüfling, Antwort auswerten
	// und Punkte vergeben
	void frageBeantworten(Pruefling person) {
		boolean ant = IO.readChar("Wahr o. Falsch (w/f)?") == 'w';
		if (ant == this.richtig) {
			IO.println("Richt. Antw.: " + punkte + " Punkte");
```

```
                person.neuePunkte(this.punkte);
            } else {
                IO.println("Falsche Antwort: 0 Punkte");
            }
        }
}

// Klasse, die Multiple-Choice-Fragen realisiert
class MCFrage extends Frage {
        String[] antworten; // moegliche Antworten

        int richtigIndex; // Index der richtigen Antwort

        MCFrage(String text, int punkte, String[] antworten, int
richtigIndex) {
            super(text, punkte);
            this.antworten = antworten;
            this.richtigIndex = richtigIndex;
        }

        // Frage auf den Bildschirm ausgeben
        void frageStellen() {
            super.frageStellen();
            for (int f = 0; f < this.antworten.length; f++) {
                IO.println("(" + f + "): " + this.antworten[f]);
            }
        }

        // Frage beantworten durch Prüfling, Antwort auswerten
        // und Punkte vergeben
        void frageBeantworten(Pruefling person) {
            int antwort = IO.readInt("Auswahl: ");
            if (antwort == this.richtigIndex) {
                IO.println("Richtige Antwort: " + this.punkte + "
Punkte");
                person.neuePunkte(this.punkte);
            } else {
                IO.println("Falsche Antwort: 0 Punkte!");
                IO.println("Richtig Antwort ist " + this.richtigIndex);
            }
        }
}

class Quiz {
        Frage[] fragen; // Menge an Fragen

        String titel; // Titel des Quizes

        int aktuellerIndex; // aktuelle Anzahl an Fragen-1

        int naechsterIndex; // Schleifenvariable

        // Konstruktor
        Quiz(String titel, int maxFragen) {
            this.titel = titel;
            this.aktuellerIndex = -1;
            this.fragen = new Frage[maxFragen];
            for (int i = 0; i < this.fragen.length; i++)
                this.fragen[i] = null;
            this.naechsterIndex = -1;
        }

        String getTitel() {
```

```
            return this.titel;
      }

      // Frage hinzufuegen
      void neueFrage(Frage f) {
            if (this.aktuellerIndex < this.fragen.length - 1)
                  this.fragen[++this.aktuellerIndex] = f;
      }

      // liefert zyklisch die naechste Frage oder null,
      // falls keine (mehr)vorhanden ist
      Frage liefereNaechsteFrage() {
            if (this.fragen.length == 0)
                  return null;
            Frage f = null;
            if (this.naechsterIndex < this.aktuellerIndex)
                  f = this.fragen[++this.naechsterIndex];
            else
                  this.naechsterIndex = -1;
            return f;
      }
}

class Pruefling {
      String name; // Name der Person

      int punkte; // bisher erzielte Punkte

      Pruefling(String name) {
            this.name = name;
            this.punkte = 0;
      }

      String getName() {
            return this.name;
      }

      int getPunkte() {
            return this.punkte;
      }

      void neuePunkte(int anzahl) {
            this.punkte += anzahl;
      }
}

class Pruefung {

      // Hauptprogramm
      public static void main(String[] args) {
            Pruefung klausur = new Pruefung();
            klausur.vorbereiten();
            klausur.durchfuehren();
            klausur.ergebnisseBekanntgeben();
      }

      Quiz pruefung;

      Pruefling[] studenten;

      Pruefung() {
            this.pruefung = null;
            this.studenten = null;
```

```
	}

	void vorbereiten() {
		IO.println("Fragen eingeben");
		IO.println("----------------");
		String titel = IO.readString("Titel des Quizes: ");
		int anzahl = IO.readInt("Anzahl Fragen: ");
		this.pruefung = new Quiz(titel, anzahl);
		// Fragen eingeben
		for (int i = 0; i < anzahl; i++) {
			Frage f = this.frageErzeugen(i + 1);
			this.pruefung.neueFrage(f);
		}
	}

	public Frage frageErzeugen(int nummer) {
		// spaeter Factory-Pattern
		int typ = IO.readInt("Fragetyp: Wahr/Falsch (1), Multiple
Choice (2)?");
		switch (typ) {
		case 1:
			return erzeugeWahrFalschFrage(nummer);
		case 2:
		default:
			return erzeugeMCFrage(nummer);
		}
	}

	private Frage erzeugeWahrFalschFrage(int nummer) {
		String text = IO.readString("Frage " + (nummer) + ": ");
		boolean wahr = IO.readChar("Wahr/falsch(w/f)?") == 'w';
		int punkte = IO.readInt("Erreichbare Punkte: ");
		return new WahrFalschFrage(text, punkte, wahr);
	}

	private Frage erzeugeMCFrage(int nummer) {
		String text = IO.readString("Frage " + (nummer) + ": ");
		int anzahl = IO.readInt("Anzahl an Antworten: ");
		String[] antworten = new String[anzahl];
		for (int i = 0; i < anzahl; i++) {
			antworten[i] = IO.readString("Antwort " + i + ": ");
		}
		int richtigIndex = IO.readInt("Index der richtigen Antwort: ");
		int punkte = IO.readInt("Erreichbare Punkte: ");
		return new MCFrage(text, punkte, antworten, richtigIndex);
	}

	void durchfuehren() {
		IO.println("Pruefung");
		IO.println("--------");
		int anzahl = IO.readInt("Anzahl Prueflinge: ");
		this.studenten = new Pruefling[anzahl];
		// alle Prueflinge abfragen
		for (int i = 0; i < anzahl; i++) {
			IO.println("Pruefling " + (i + 1) + " ist an der Reihe");
			this.studenten[i] = new Pruefling(IO.readString("Name:
"));
			Frage f = null;
			// alle Fragen der Pruefung stellen
			while ((f = this.pruefung.liefereNaechsteFrage()) !=
null) {
				f.frageStellen();
				f.frageBeantworten(this.studenten[i]);
```

```
                }
            }
        }

        void ergebnisseBekanntgeben() {
            this.ranglisteErstellen();
            IO.println("Pruefungsergebnisse");
            IO.println("--------------------");
            IO.println("Quiz: " + this.pruefung.getTitel());
            for (int i = 0; i < this.studenten.length; i++) {
                IO.println("Platz  " + (i + 1) + ":  " +
this.studenten[i].getName()
                        + " mit " + this.studenten[i].getPunkte() + "
Punkten");
            }
        }

        private void ranglisteErstellen() {
            // Bubblesort nach erreichten Punkten
            boolean veraendert = false;
            do {
                veraendert = false;
                for (int i = 0; i < this.studenten.length - 1; i++) {
                    if (this.studenten[i].getPunkte() <
this.studenten[i + 1]
                            .getPunkte()) {
                        Pruefling help = this.studenten[i];
                        this.studenten[i] = this.stu.denten[i + 1];
                        this.studenten[i + 1] = help;
                        veraendert = true;
                    }
                }
            } while (veraendert);
        }
}
```

Oben und in der Vorlesung wurde ein Java-Programm vorgestellt, mit dem eine Prüfung simuliert werden kann. An Fragetypen unterstützt dieses Programm Wahr/Falsch- und MultipleChoice-Fragen. Hierzu wurden entsprechende Klassen von einer abstrakten Klasse `Frage` abgeleitet und die daraus resultierende Polymorphie ausgenutzt.

a) Erweitern Sie das Programm so, dass als weiterer Fragetyp die **Mehrfachauswahl** unterstützt wird, d.h. zu einer Frage werden mehrere Antworten gegeben, von denen keine, eine oder auch mehrere korrekt sind. Leiten Sie eine entsprechende Klasse von der Klasse `Frage` ab und erweitern Sie die Methode `frageErzeugen` der Klasse `Pruefung`, so dass auch Fragen des Typs "Mehrfachauswahl" gestellt werden können.

b) Erweitern Sie das Programm so, dass als weiterer Fragetyp der **Lückenfrage** unterstützt wird, d.h. in einer Aussage fehlt ein Wort, das ein Prüfling zu ergänzen hat. Leiten Sie eine entsprechende Klasse von der Klasse `Frage` ab und erweitern Sie die Methode `frageErzeugen` der Klasse `Pruefung`, so dass auch Fragen des Typs "Lückenfrage" gestellt werden können.

c) Erweitern Sie das Programm so, dass als weiterer Fragetyp **Zahlenfolgen-erweiterungen** unterstützt werden, d.h. es werden *n* Zahlen einer Zahlenfolge bekannt gegeben und der Prüfling muss daraus die zugrunde liegende Zahlenfolge identifizieren und die *n+1*-te Zahl der Zahlenfolge eingeben. Leiten Sie eine entsprechende Klasse von der Klasse `Frage` ab und erweitern Sie die Methode

`frageErzeugen` der Klasse `Pruefung`, so dass auch Fragen des Typs " **Zahlenfolgenerweiterungen** " gestellt werden können.

Beispiel (Eingaben in grün):

```
Frage 1 (Zahlenfolgen ergaenzen)
Anzahl an vorgegebenen Zahlen: 4
Zahl 0: 1
Zahl 1: 2
Zahl 2: 3
Zahl 3: 4
Korrekte Folgezahl: 5
Erreichbare Punkte: 10

...

Identifizieren Sie die Zahlenfolge und geben Sie die nächste Zahl der
Folge ein!
1 2 3 4 5
Richtige Antwort: 10 Punkte
```

d) Erweitern Sie das Programm so, dass als weiterer Fragetyp die **Ordnungsfrage** unterstützt wird. Bei der Ordnungsfrage wird dem Prüfling eine Aufgabe und eine Menge mit n Antworten gegeben, die der Benutzer dann in die korrekte Reihenfolge bringen muss (bekannt durch die Spielerauswahlfrage bei „Wer-wird-Millionär?“). Leiten Sie eine entsprechende Klasse von der Klasse `Frage` ab und erweitern Sie die Methode `frageErzeugen` der Klasse `Pruefung`, so dass auch Fragen des Typs "Ordnungsfrage" gestellt werden können.

Beispiel (Eingaben in grün):

```
Frage 1: Ordnen Sie die folgenden deutschen Städte von Nord nach Süd!
Anzahl an Antworten: 4
Antwort 0: Oldenburg
Antwort 1: Kiel
Antwort 2: München
Antwort 3: Frankfurt
1. Index: 1
2. Index: 0
3. Index: 3
4. Index: 2
Erreichbare Punkte: 20

...

Ordnen Sie die folgenden deutschen Städte von Nord nach Süd!
(0): Oldenburg
(1): Kiel
(2): München
(3): Frankfurt
1. : 1
2. : 0
3. : 3
4. : 2
Richtige Reihenfolge: 20 Punkte
```

Aufgabe 10 [5 Punkte]

Schreiben Sie Klassen, die Eisenbahnzüge repräsentieren. Ein Eisenbahnzug besteht aus einer Lokomotive und einer beliebigen Anzahl Wagen, möglicherweise auch überhaupt keinen. Lokomotiven und Wagen haben die folgenden Eigenschaften (alle ganzzahlig):

Lokomotive (locomotive)

- Länge (Meter)
- Typ (irgendeine Zahl)

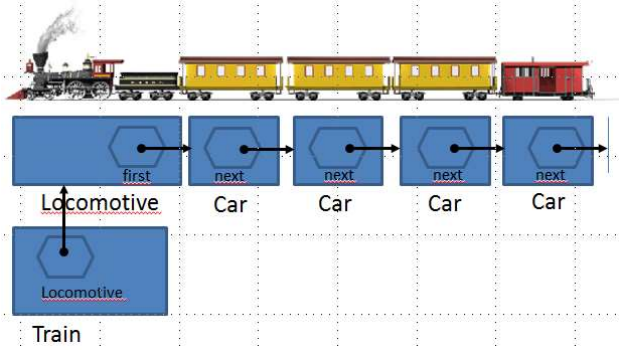
Wagen (car)

- Länge (Meter)
- Passagierkapazität (Anzahl Personen)

Definieren Sie die Klassen Locomotive und Car, jeweils mit sinnvollen Methoden. Die oben genannten Eigenschaften sind unveränderlich. Das interessante Problem ist das Zusammenstellen eines Zuges aus den Einzelteilen. Der erste Wagen hängt direkt an der Lokomotive. Geben Sie der Klasse Locomotive deshalb ein Element first vom Typ Car, dazu eine Getter und eine Setter-Methode.

An jedem Wagen hängt der jeweils nächste Wagen oder gar nichts beim letzten Wagen. Definieren Sie in der Klasse Car ein Element next des gleichen Typs Car, wieder mit Gettern und Settern. Diese Objektvariable speichert ein anderes Objekt derselben Klasse oder null beim letzten Wagen.

Die folgende Abbildung zeigt die Idee.



Definieren Sie schließlich eine Klasse Train, die den ganzen Zug repräsentiert. Ein Train-Objekt speichert "seine" Lokomotive, aber nicht die Wagen. Diese können, einer nach dem anderen, auf dem Weg über die Lokomotive erreicht werden. Die Klasse Train bietet die folgenden Methoden:

- Konstruktor der Train-Konstruktor erwartet eine Lokomotive und baut einen ziemlich kurzen Zug, der nur aus der Lokomotive, noch ohne Wagen besteht.
- add hängt in diesen Zug einen gegebenen Wagen ein. Es spielt keine Rolle, wo im Zug der Wagen eingefügt wird.
- print gibt eine Liste dieses Zuges mit allen Bestandteilen aus.
- getPassengers liefert die gesamte Passagierkapazität dieses Zuges, das heißt die Summe der Passagierkapazitäten aller Wagen.
- getLength liefert die Gesamtlänge dieses Zuges, d. h. die Summe der Längen der Lokomotive und aller Wagen.
- removeFirst hängt den ersten Wagen aus diesem Zug aus und liefert den ausgehängten Wagen als Ergebnis zurück. Die restlichen Wagen rücken nach vorne. Falls es keinen Wagen gibt, ist das Ergebnis null.

- relink akzeptiert als Parameter einen anderen Zug und hängt alle Wagen des anderen Zuges in der gleichen Reihenfolge an diesen Zug an. Im anderen Zug bleibt nur die Lokomotive zurück. Nutzen Sie für diese Methode geschickt die vorher definierten Methoden.
- revert dreht die Abfolge der Wagen in diesem Zug um, das heißt, der vorher letzte Wagen wird zum ersten und umgekehrt.

Schreiben Sie schließlich eine Anwendung, die Folgendes abwickelt:

1. Eine Lokomotive "Big Chief" mit der Nummer 5311 und der Länge 23m wird erzeugt.
2. Ein Zug namens "Santa Fe"mit der Lokomotive "Big Chief" wird erzeugt.
3. An "Santa Fe" werden drei Wagen mit den Längen 12m, 15m, 20m und den Passagierkapazitäten 50, 75, 100 Personen angehängt.
4. Eine Lokomotive "Steel Horse" mit der Nummer 5409 und der Länge 21m wird erzeugt.
5. Ein Zug namens "Rio Grande Express" mit der Lokomotive "Steel Horse" wird erzeugt.
6. An den "Rio Grande Express" werden zwei Wagen mit den Längen 13m und 18m sowie den Passagierkapazitäten 60 und 80 Personen angehängt.
7. Alle Wagen von "Santa Fe" werden in den "Rio Grande Express" übernommen.
8. Die Wagenreihenfolge im "Rio Grande Express" wird umgedreht.