Hochschule Ingolstadt SS 2013

Studiengänge: FFI, WINF

Prüfung
Grundlagen der Programmierung 2

Prüfer: Jobst, Regensburger, Windisch

Prüfungsdauer: 90 Minuten

Hilfsmittel: keine

Studiengang	Dozent	Matrikelnummer	Semester	Raum	Platz

Aufgabe	1	2	3	4	Σ	Note
Punkte						

Bitte beachten:

Tragen Sie Ihre persönlichen Angaben auf dieses Deckblatt ein.

Schreiben Sie Ihre Antworten direkt in die dafür vorgesehenen freien Stellen dieser Prüfung.

Geben Sie alle Blätter wieder ab, auch wenn einzelne Seiten nicht beschrieben sein sollten.

Viel Erfolg!

Aufgabe 1 (Verständnisfragen, Exceptions, Threads, ca. 25 %)

Teilaufgabe a) Sichtbarkeiten von Attributen und Methoden:

Die Einschränkung des Zugriffs (der „Sichtbarkeit“) auf Attribute und Methoden wird durch Schlüsselworte (sog. „Modifier“) angegeben. Nennen Sie diese und geben Sie kurz ihre Bedeutung an?

Teilaufgabe b) Statische Attribute und Methoden:

Geben Sie zwei Unterschiede zwischen statischen und nicht statischen Attributen und Methoden an:

Teilaufgabe c) Exceptions:

Eine Fehlerbehandlung in Java wäre statt mit Exceptions und try-catch auch mit einfachen if-then-Blöcken möglich. Nennen Sie zwei eindeutige Vorteile für das Java-Exception-Handling:

Ein try catch-Block bzw. eine throws-Anweisung ist in der Regel bei allen Exceptions zwingend vom Compiler vorgeschrieben. Bei welcher Art von Exceptions ist dies nicht der Fall?

Welchen Zweck erfüllt der finally-Block bei try-catch?

Geben Sie die Klassen-Deklaration und einen passenden Konstruktor einer eigenen Exception namens `EndOfKnowledgeException` an. Bei der Erzeugung wird eine Fehlermeldung als String übergeben. Sie möchten, dass diese Fehlermeldung immer abgefangen werden muss.

Beispielaufruf:

```
throw new EndOfKnowledgeException("zu wenig gelernt");
```

Teilaufgabe d) Threads:

Vervollständigen Sie die Klasse `MeinThread`, die einen parallel laufenden Thread starten soll. Der Thread soll die Ausgabe `"Geh jetzt schlafen"` auf der Konsole machen, anschließend eine Sekunde „schlafen“, nach dem Aufwachen die Ausgabe `"Bin wieder wach"` machen und sich danach beenden.

Hinweis: Folgende statische Methode der Klasse `Thread` lässt einen Thread `millis` Millisekunden „schlafen“:

```
public static void sleep(long millis) throws InterruptedException;
```

```
public class MeinThread implements Runnable {
    // *** Ergänzen Sie hier die Methode des Threads
    // *** (Ausgabe, "Schlafen", Ausgabe)









    public static void main(String args[]) {
        // *** Erzeugen und starten Sie hier
        // *** einen Thread der Klasse MeinThread



    }
}
```

Aufgabe 2 (Collections, Comparable, Comparator, ca. 25 %)

Für die Verwaltung von Wetterdaten sei folgende Klasse `Wetterdaten` gegeben.

```
public class Wetterdaten extends ArrayList<Messreihe> {
  public static void main(String[] args) {
    Wetterdaten wd = new Wetterdaten();

    wd.add(new Messreihe("London",
        new double[] { 21.2, 26.4, 29.8, 24.2 }));
    wd.add(new Messreihe("Berlin",
        new double[] { 24.0, 32.0, 36.0 }));
    wd.add(new Messreihe("Paris",
        new double[] { 24.1, 28.9 }));
    wd.add(new Messreihe("Rom",
        new double[] { 25.3, 30.4, 34.3, 33.8 }));

    System.out.printf("Aufsteigend\nnach Namen\n");
    Collections.sort(wd);
    for (Messreihe mr : wd) {
      System.out.println(mr);
    }

    System.out.printf("\nAbsteigend\nnach Mittelwerten\n");
    Collections.sort(wd, new MeanComp());
    for (Messreihe mr : wd) {
      System.out.println(mr);
    }
  }
}
```

Die Klasse `Wetterdaten` implementiert eine Liste über dem Typ `Messreihe`. Die Klasse `Messreihe` modelliert die Daten einer Wetterstation, welche aus dem Namen der Station und einem nichtleeren Array von Temperaturdaten bestehen.

Bei Ausführung erzeugt die main()-Methode die in Abbildung 1 gezeigte Ausgabe.

In den nachfolgenden Teilaufgaben (siehe Folgeseiten) sollen Sie die Klasse `Messreihe` und die zum Sortieren nach Durchschnittswerten benutzte Comparator-Klasse `MeanComp` implementieren.

```
Aufsteigend
nach Namen
   Berlin:30,67
   London:25,40
    Paris:26,50
      Rom:30,95

Absteigend
nach
Mittelwerten
      Rom:30,95
   Berlin:30,67
    Paris:26,50
   London:25,40
```

Abbildung 1

Teilaufgabe a) Klasse `Messreihe`: Konstruktor und Attribute:

Implementieren Sie einen geeigneten Konstruktor für die Klasse `Messreihe` und statten Sie die Klasse nach Bedarf mit Attributen aus. Die main-Methode der Klasse `Wetterdaten` enthält alle zur Auslegung des Konstruktors benötigten Hinweise.
Sie dürfen davon ausgehen, dass das im Konstruktor übergebene Array nicht leer ist.

```
public class Messreihe implements Comparable<Messreihe> {
  // *** Hier bitte bei Bedarf Attribute definieren



  // *** Hier bitte Konstruktor definieren




// Nachfolgend Code aus weiteren Teilaufgaben
```

Teilaufgabe b) Klasse `Messreihe`: Berechnung der Durchschnittswerte

Implementieren Sie die Methode `mittelwert()`, die den Mittelwert aus den Temperaturdaten der Messreihe errechnet und als Resultat zurückgibt.

```
public double mittelwert() {









}
// Nachfolgend Code aus weiteren Teilaufgaben
```

Teilaufgabe c) Klasse `Messreihe`: Methode `compareTo()`

Die Klasse Messreihe implementiert das Interface `Comparable<Messreihe>`.
Programmieren Sie nun die hierfür erforderliche Methode `compareTo()` so, dass der Aufruf `Collections.sort(wd)` in der main-Methode der Klasse `Wetterdaten` die Liste der Messreihen alphabetisch aufsteigend nach den Namen der Wetterstationen sortiert (siehe oberen Teil der Abbildung 1).

```
public int compareTo(Messreihe other) {




}
// Nachfolgend Code aus weiteren Teilaufgaben
```

Teilaufgabe d) Klasse `Messreihe`: Methode `toString()`

Die main-Methode der Klasse `Wetterdaten` gibt die Namen der Wetterstationen zusammen mit einer Durchschnittstemperatur auf der Konsole aus. Dabei wird bei der Ausgabe der einzelnen Messreihen jedesmal implizit die Methode `toString()` der Klasse `Messreihe` genutzt, um eine String-Repräsentation der jeweiligen Messreihe zu erzeugen.

```
Aufsteigend
nach Namen
   Berlin:30,67
   London:25,40
...
```

Abbildung 2

Implementiern Sie nun die Methode `toString()` der Klasse `Messreihe` so, dass für die einzelnen Messreihen die in Abbildung 2 gezeigte formatierte Ausgabe entsteht (fett hervorgehoben).

Hinweis:
`String java.lang.String.format(String format, Object... args).`

```
  public String toString() {



  } // Ende Methode toString()
} // Ende der Klasse Messreihe
```

Teilaufgabe e) Klasse `MeanComp`, Interface `Comparator`

Die Klasse MeanComp implementiert das Interface `Compartor<Messreihe>`. Programmieren Sie nun die hierfür erforderliche Methode compare() so, dass der Aufruf `Collections.sort(wd, new MeanComp())` in der main-Methode der Klasse `Wetterdaten` die Liste der Messreihen absteigend nach den Durchschnittstemperaturen der Wetterstationen sortiert (siehe unteren Teil der Abbildung 1).

```
public class MeanComp implements Comparator<Messreihe> {
  public int compare(Messreihe o1, Messreihe o2) {









  } // Ende der Methode compare()
} // Ende der Klasse MeanComp
```

Aufgabe 3 (Interfaces Iterable und Iterator, ca. 25 %)

Carl, der Organisator des diesjährigen Jungle-Camps, hat 4 weibliche und 3 männliche Models für eine Modenschau eingeladen. Bei der Vorführung sollen die Models jeweils paarweise, ein Girl und ein Boy, auf dem Laufsteg auftreten und jedes Girl soll mit jedem Boy genau einen gemeinsamen Walk machen. Da die Vorbereitungen für die einzelnen Walks zeitaufwändig sind, soll die Abfolge der Paare so gewählt werden, dass jedes Model maximal viel Zeit für die Vorbereitung hat.

Da Carl ein Hobby-Informatiker ist, schreibt er eine Klasse `JungleWalk`, welche das Interface `Iterable` implementiert und daher bequem in Schleifen verwendet werden kann. Die Klasse enthält folgende main()-Methode:

```
public static void main(String[] args) {
  System.out.printf("Pairs4Walks:\n");
  String[] thegirls = new String[] {
                          "Amber", "Betty", "Cindy", "Diana" };
  String[] theboys  = new String[] { "Kodo", "Luke", "Mike" };
  for (JunglePair p: new JungleWalk(thegirls, theboys)){
    System.out.println(p);
  }
}
```

Bei Ausführung erzeugt die main()-Methode die im nebenstehenden Kasten gezeigte Ausgabe.

```
Pairs4Walks:
(Amber,Kodo)
(Betty,Luke)
(Cindy,Mike)
(Diana,Kodo)
(Amber,Luke)
(Betty,Mike)
(Cindy,Kodo)
(Diana,Luke)
(Amber,Mike)
(Betty,Kodo)
(Cindy,Luke)
(Diana,Mike)
```

Weil die Zahlen 3 und 4 teilerfremd sind, kann die gesamte Folge der unterschiedlichen Paarungen durch mehrfaches zyklisches Durchlaufen der Girls und Boys erfolgen. Die obige Anforderung nach maximaler Vorbereitungszeit kann dadurch erfüllt werden.

In den nachfolgenden Teilaufgabe (siehe Folgeseiten) sollen Sie vorgegebene Code-Fragmente der Klassen `JunglePair` und `JungleWalk` vervollständigen.

Bei der Bearbeitung der Teilaufgaben können Ihnen folgende Hinweise zur Implementierung behilflich sein:

1. Die Klasse `JungleWalk` implementiert das Interface `Iterable<JunglePair>`. Daher muss die Klasse die Methode
   `public Iterator<JunglePair> iterator()`
   implementieren, was am einfachsten über eine innere anonyme Klasse erfolgt. Siehe hierzu das vorgegebene Code-Fragment auf der nächsten Seite.
2. Ein Iterator vom Typ `Iterator<JunglePair>` muss die Methoden
   `public boolean hasNext()` und `public JunglePair next()`
   implementieren. Die dritte zum Interface gehörende Methode `remove()` ist optional und soll hier nicht implementiert werden!
3. Eine innere Klasse hat uneingeschränkten Zugriff auf alle Elemente der sie umschließenden Klasse.
4. Ihr Code soll für eine beliebige Anzahl von Boys und Girls funktionieren. Sie dürfen aber annehmen, dass die Anzahlen der Boys und Girls teilerfremd sind.

Teilaufgabe a) Implementierung der Klasse `JunglePair`:

Die Klasse `JunglePair` modelliert die einzelnen Paarungen. Implementieren Sie einen geeigneten Konstruktor sowie Attribute nach Belieben. Desweiteren soll in der Klasse die Methode `toString()` so überschrieben werden, dass die `main`-Methode der Klasse `JungleWalk` die gewünschte Ausgabe erzeugen kann.

```
public class JunglePair {
  // *** Hier bitte bei Bedarf Attribute einfügen



  // *** Hier bitte den Konstruktor geeignet vervollständigen
  public JunglePair(String girl, String boy) {



  }

  // *** Hier bitte die Methode toString() implementieren



  }
}
```

Teilaufgabe b) Implementierung der Klasse JungleWalk:

Das nachfolgende Code-Fragment enthält ein Gerüst der Klasse `JungleWalk`. Vervollständigen Sie den Code hinsichtlich folgender Aspekte:

1. Konstruktor der Klasse `JungleWalk`
2. In der von der Methode `iterator()` benutzten anonymen inneren Klasse:
   2.1. Attribute nach Bedarf
   2.2. Methode `hasNext()`
   2.3. Methode `next()`

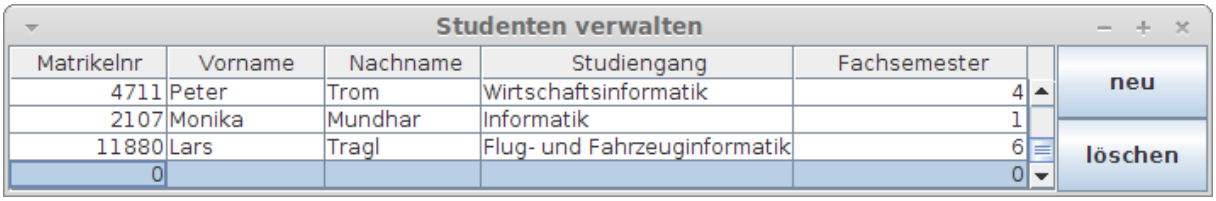
Nutzen Sie den nachfolgenden Kasten, welcher sich auch über die nächste Seite erstreckt. Anfang und Ende der einzelnen Abschnitte sind jeweils vorgegeben.

```
public class JungleWalk implements Iterable<JunglePair> {
  private String[] girls;
  private String[] boys;

  // *** Hier bitte Konstruktor der Klasse JungleWalk angeben




  } // Ende des Konstruktors der Klasse JungleWalk
```

```
  @Override
  public Iterator<JunglePair> iterator() {
    return new Iterator<JunglePair>() {
      // *** Hier bitte Attribute der anonymen Klasse einfügen




      @Override
      public boolean hasNext() {
        // *** Hier bitte Methode hasNext() vervollständigen




      } // Ende der Methode hasNext()

      @Override
      public JunglePair next() {
        // *** Hier bitte Methode next() vervollständigen
        if (




        // ab hier ist der Rest des Codes vorgegeben
        } else {
          throw new NoSuchElementException(
                          "No more pairs to generate");
        }
      } // Ende Methode next()
    }; // Ende der anonymen inneren Klasse
  }// Ende der Methode iterator()

  public static void main(String[ ] args) {
    // siehe Angabe …
  }
} // Ende der Klasse JungleWalk
```

Aufgabe 4 (GUI und Events, ca. 25 %)

Folgende Abbildung zeigt einen Dialog zur Verwaltung einer Liste von Studierenden. Der Button „neu“ bewirkt, dass eine „leere“ Studentenbeschreibung (vgl. 4. Zeile in der Tabelle) nach der selektierten Zeile eingefügt wird. Ist keine Zeile selektiert, so wird an das Ende der Liste angehängt. Der Button „löschen“ bewirkt, dass die gerade selektierte Zeile aus der Liste gelöscht wird.



Vervollständigen Sie im unten gezeigten Codefragment in Teilaufgabe a) den Konstruktor und in Teilaufgabe b) die Callback-Methode unter Beachtung folgender Hinweise zu vorgegebenen Klassen und deren Methoden:

1. Die Klassen `StudentTable` und `StudentTableModel` sind komplett vorgegeben und müssen von Ihnen nicht implementiert werden.
2. Die Klasse `StudentTable` ist eine Unterklasse von `JTable`. Über deren Konstruktor wird das Tabellenmodell `model` bekannt gemacht, welches ein Objekt der Klasse `StudentTableModel` ist. Das eigentliche Datenmodell `liste` speisen wir über den Konstruktor der Klasse `StudentTableModel` ein.
3. Die Methode `getSelectedRow()` der Klasse `StudentTable` liefert den Index der aktuell selektierten Zeile oder -1, falls keine Zeile selektiert wurde.
4. Die ebenfalls vorgegebene Klasse `Studentenbeschreibung` enthält die in der Tabelle zu sehenden Attribute einschliesslich eines passenden Konstruktors zur Initialisierung dieser Attribute.
5. Das Hinzufügen einer Tabellenzeile erfolgt über die überladene Methode `addRow` der Klasse `StudentTableModel`:
   - Anfügen am Ende der Tabelle:
     `void addRow(Studentenbeschreibung rowData)`
   - Einfügen an Indexposition `index`:
     `void addRow(int index, Studentenbeschreibung rowData)`
6. Das Löschen einer Zeile erreichen Sie durch Aufruf der Methode removeRow der Klasse `StudentTableModel`:
   - Löschen der Zeile an Indexposition `index`:
     `void removeRow(int index)`

```
public class StudentenVerwaltenDialog extends JFrame
                                       implements ActionListener {
    private StudentTable table;
    private StudentTableModel model;
    private JButton newButton, deleteButton;

    public StudentenVerwaltenDialog(
                         List<Studentenbeschreibung> liste) {

        model = new StudentTableModel(liste);
        table = new StudentTable(model);
```

```
        // *** ab hier bitte Konstruktor ergänzen (Teilaufgabe a)
    } // Ende Konstruktor StudentenVerwaltenDialog

    public void actionPerformed(ActionEvent event) {
        // *** ab hier bitte ergänzen (Teilaufgabe b)
    } // Ende actionPerformed
} // Ende Klasse StudentenVerwaltenDialog
```