Technische Hochschule Rosenheim

Fakultät für Informatik
Prof. Dr. M. Tilly

Wintersemester 2019

Prüfung - Informatik (INF)
000 - Fortsgeschrittene Programmierkonzepte(FPK)

Datum: 24.12.2019	Dauer: 90 Minuten	Material: Ein Buch mit ISBN-Nr

---

Name: .....

Matr.-Nr.: .....

Viel Erfolg!

Hinweise:

1. Die Heftklammern dürfen nicht gelöst werden. Bitte überprüfen Sie: Die Klausur umfasst 14 Seiten incl. Deckblatt und Arbeitsblätter.
2. Bearbeiten Sie die Fragen direkt in der Angabe. Nutzen Sie ggfs. die Arbeitsblätter und Rückseiten.
3. Sollten Ihrer Meinung nach Widersprüche in den Aufgaben existieren bzw. Angaben fehlen, so machen Sie sinnvolle Annahmen und dokumentieren Sie diese.
4. Die Punkteverteilung dient zur Orientierung, sie ist jedoch unverbindlich.
5. Alle Fragen beziehen sich auf die Programmiersprache Java; Ausnahmen sind gekennzeichnet.
6. Bitte schreiben Sie nicht mit Bleistift, roten oder grünen Stiften und wenn möglich leserlich.

Aufgabe	Punkte	von
1		15
2		14
3		18
4		10
5		10
6		5
Summe		72

Note: .....

.....
(Erstprüfer)

.....
(Zweitprüfer)

Name: Martrikelnr.:

1. Aufgabe - Allgemeines 8+6 Punkte

---

a)

Markieren Sie die richtige Antwort bzw. Aussage; pro Frage ist genau eine Antwort zu markieren.

1. Interfaces und abstrakte Klassen in Java 9 und neuer.
   - ☐ Eine abstrakte Klasse muss mindestens eine abstrakte Methode haben.
   - ☐ Ererbte abstrakte Methoden müssen immer implementiert werden.
   - ✓ Methoden in Interfaces können private sein.
2. Bezüglich innerer Klassen gilt:
   - ☐ Innere Klassen können keine Schnittstellen implementieren.
   - ☐ Innere Klassen müssen immer als static deklariert sein.
   - ✓ Es gibt sowohl innere Klassen als auch innere Interfaces.
3. Welche der folgenden Signaturen ist korrekt und generisch?
   - ✓ abstract <T> void a(T t);
   - ☐ abstract void a(T t);
   - ☐ <T> abstract void a(T t);
4. Bezüglich Ausnahmen (Exception) gilt:
   - ☐ Ungeprüfte Ausnahmen müssen in der Methode behandelt werden, in der sie auftreten.
   - ✓ Geprüfte Ausnahmen müssen lokal behandelt oder mit throws ausgewiesen werden.
   - ☐ Eine Ausnahme muss immer mit try..catch behandelt werden.
5. Bezüglich Sichtbarkeiten gilt:
   - ☐ Interfaces können protected Methoden enthalten.
   - ☐ Innere Klassen ohne Sichtbarkeitsangabe sind öffentlich sichtbar.
   - ✓ Ist eine innere Klasse private, so kann sie in abgeleiteten Klassen nicht instanziiert werden.

6. Bezüglich funktionaler Programmierung gilt:
   - ✓ Endrekursiv bedeutet dass der Rekursionsschritt die letzte Anweisung ist.
   - ☐ Rufen zwei Methoden f und g sich wechselweise gegenseitig auf, so spricht man von kaskadierter Rekursion.
   - ☐ Eine Rekursion kann auch ohne Terminalfall regulär berechnet werden.
7. Bezüglich paralleler (nebenläufiger) Ausführung gilt:
   - ☐ Ein kritischer Abschnitt ist ein Teil einer Methode, welcher besonders kompliziert ist.
   - ✓ Das Java Interface Future wird für asynchrone Programmierung verwendet.
   - ☐ Methoden welche nur von genau einem Thread gleichzeitig ausgeführt werden dürfen, müssen mit @Synchronized annotiert werden.
8. Bezüglich paralleler Verarbeitung gilt:
   - ☐ Java regelt konkurrierenden Zugriff automatisch, wodurch Deadlocks vermieden werden.
   - ☐ Das Gegenstück zu wait() ist signal().
   - ✓ Die Methode notify() kann nur in kritischen Abschnitten und auf dem Lockobjekt verwendet werden.

b)

Beantworten Sie folgende Fragen kurz und knapp (je 2 Punkte):

1. Nennen Sie zwei syntaktische Alternativen zu einer anonymen inneren Klasse.

   Lösung:
   Lambdaausdruck, Methodenreferenz

2. Wozu dient die Annotation @Deprecated?

   Lösung:
   Markiert eine Methode oder Klasse, dass diese in zukünftigen Versionen ersetzt wird oder ganz herausfällt.

3. Ordnen Sie die Designpatterns ihrer Kurzbeschreibung zu?

   Lösung:
   Factory ——— Erzeugung von Objekten
   Singleton ——— Eine globale Instanz
   Fliegengewicht —- Reduktion des Speicherbedarfs
   Visitor ——— Traversieren von Datenstrukturen

2. Aufgabe - Generics 6+3+1+4+4 Punkte

---

a)

Schreiben Sie eine generische Klasse Container, welche Objekte von beliebigen (aber festen) Typs speichert. Die Klasse soll weiterhin eine öffentliche Methode besitzen, welche den Laufzeittyp des gespeicherten Elements zurückgibt oder null wenn das Element null ist.

Lösung:

```
1  // Klasse Container
2  public class Container<T> {
3
4     // Attribute
5     private T obj;
6
7     // Konstruktor
8     public Container(T obj) {
9        this.obj = obj;
10    }
11
12    // Methode getContainedClass
13    Class getContainedClass() {
14       if (obj == null)
15          return null;
16       return obj.getClass();
17     }
18
19     public static void main(String[] args) {
20         Container<Integer> c = new Container<>();
21         c.obj = 2;
22         System.out.println(c.getContainedClass());
23    }
24 }
```

b)

Gegeben sei die folgende (nicht-generische) Methodensignatur:

Comparable minimum(Comparable[] feld)

Schreiben sie eine generische Variante dieser Signatur, welche es erlaubt ein Array eines festen Typs unter Verwendung der Methode Comparable.compareTo zu sortieren.

Lösung:

static <T extends Comparable<? super T>> T minimum(T[] feld)

c)

Wie heisst der Mechanismus in Java um den Objektyp zur Laufzeit zu bestimmen?

Lösung:
Reflection.

d)

Kurz und knapp: Was bedeuten die Zeichen ? und & in Zusammenhang mit Generics?

Lösung:
? ist wildcard, beliebige aber feste Klasse (2P)
& Auflistungsoperator für komplexes Bound (2P)

e)

Gegeben ist die folgende generische Signatur um von einer Liste in eine andere zu kopieren. Ergänzen Sie die korrekten Bounds.

Lösung:

```
static <T> void copy(List<? super T> ziel, List<? extends T> quelle)
```

Name: Martrikelnr.:

3. Aufgabe - Design Pattern 5+5+5 Punkte

---

a)

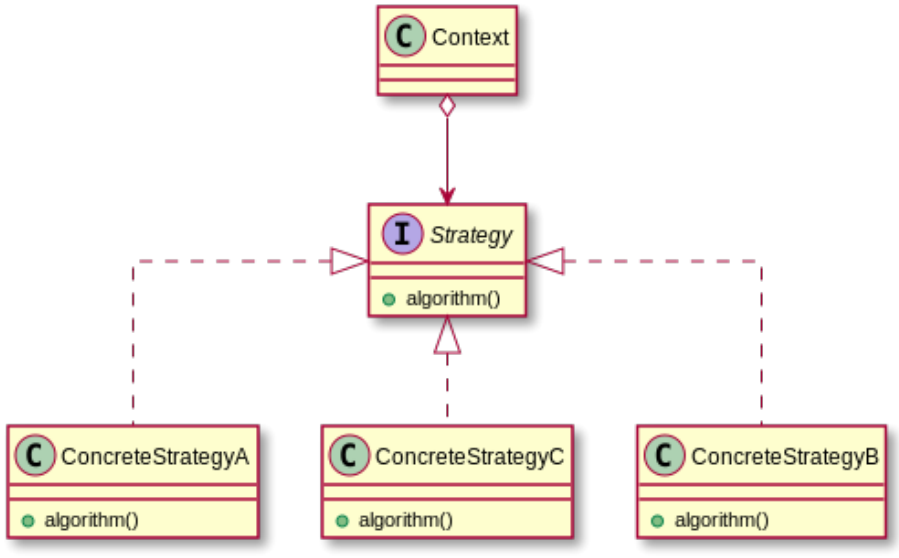
Kurz und knapp: Was ist der Sinn des Strategiemusters (strategy pattern).

Lösung:
Definiert eine Familie von Algorithmen und kapselt diese. Daruch sind die einzelnen Stratgeien austauschbar. Das Strategiemuster ermöglicht es, den Algorithmus unabhängig von nutzenden Klienten zu variieren.

b)

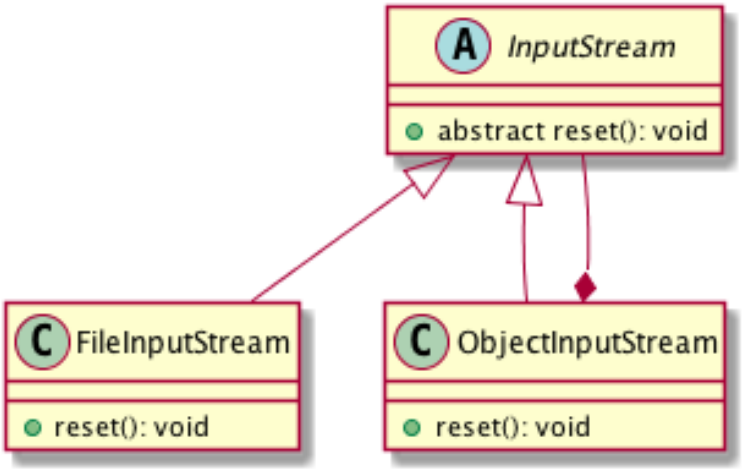
Zeichnen Sie das Klassendiagram des Strategiemusters.

Lösung:



c)

Benennen Sie das folgende Pattern und erläutern Sie kurz einen Anwendungsfall.



Lösung:
Decorator:
Das Decorator Design Pattern ermöglicht das dynamische Hinzufügen von Fähigkeiten zu einer Klasse. Dazu wird die Klasse, dessen Verhalten wir erweitern möchten, mit anderen Klassen (Decorator, Dekorierer) dekoriert. Das heißt der Decorator umschließt (enthält) die Component. Der Decorator ist vom selben Typ wie das zudekorierende Objekt, hat somit die gleiche Schnittstelle und kann an der selben Stelle wie die Component benutzt werden. Er delegiert Methodenaufrufe an seine Component weiter und führt sein eigenes Verhalten davor oder danach aus.
Anwendungsfälle: Bei der Speicherung von Textdaten können zusätzliche Features, wie das Maskieren von Umlauten oder ein Komprimierungsalgorithmus hinzugeschaltet werden.
Wenn Funktionalitätserweiterung mittels Vererbung impraktikabel ist: Dies ist zum einen der Fall bei voneinander unabhängigen Erweiterungen, wenn unter Beachtung jeder möglichen Erweiterungskombination eine schier unüberschaubare Anzahl von Klassen entstehen würde.
Das klassische Kaffeebeispiel: Es sind 3 Kaffeegrundsorten (Espresso, Edelröstung, Cappuccino, Latte Macchiato) und 4 optionale Zusätze (Milch, Schoko, Zucker, Sahne) gegeben. Das ergibt 16 Subklassen! Unüberschaubar, unwartbar und unflexibel.

Name: Martrikelnr.:

4. Aufgabe - Threads 10 Punkte

Der folgende Codeausschnitt soll einen threadsicheren Buffer für ein Consumer-Producer-Problem implementieren. Ergänzen Sie den Quelltext an den mit Platzhaltern (_____) markierten Stellen, so dass der Buffer sich wie erwartet verhält, und zwar…

- in get() wartet, bis mindestens 1 Element im Buffer verfügbar ist.
- in put() wartet, bis mindestens 1 Element im Buffer frei ist
- eine Verklemmung (deadlock) vermeidet.

Hinweise: Es gibt verschiedene Varianten der Implementierung, es müssen daher nicht alle Leerstellen befüllt werden; catch Blöcke bei Ausnahmebehandlung sollen leer sein.
Lösung:

```
1  public class Buffer<T> {
2
3      private Queue<T> queue = new LinkedList<>();
4      private final int maxSize = 10;
5
6      public  T get() throws Exception{
7          synchronized (this) {
8              while (queue.size() == 0)
9                  wait();
10
11             T obj = queue.remove();
12             notify();
13             return obj;
14         }
15     }
16
17     public void put(T obj) throws Exception {
18         synchronized (this) {
19             while (queue.size() >= maxSize)
20                     wait();
21             queue.add(obj);
22             notify();
23         }
24     }
25 }
```

Name: Martrikelnr.:

5. Aufgabe - Functional Interfaces 5+5+3 Punkte

Gegeben ist das Interface BinaryOperator<T extends Comparable> mit der apply-Methode:

```
1
2     @FunctionalInterface
3     interface BinaryOperator<T extends Comparable> {
4        T apply(T a, T b);
5     }
```

Die apply-Methode bekommt 2 Parameter vom Typ T und gibt einen Wert vom Typ T zurück.

a)

Schreiben Sie eine Methode reduce, die die Methode apply auf jedes Element einer übergebenen Liste vom Typ T anwendet. Stellen Sie sich vor, dass sie eine Liste von Zahlen das Maximum ermitteln wollen. Der Code dazu könnte wie folgt aussehen:

```
1
2  interface BinaryOperator<T extends Comparable> {
3     T apply(T a, T b);
4  }
5
6  static <T extends Comparable> T reduce(List<T> list , BinaryOperator<T> func
       ) {}
7
8  public static void main(String[] args) {
9
10
11    java.util.List<Integer> l1 = Arrays.asList(29, 19, 20, 21, 25, 22, 23);
12
13    Integer max = reduce( l1 , new BinaryOperator<Integer>() {
14       @Override
15       public Integer apply(Integer a, Integer b) {
16          return Integer.max(a,b);
17       }
18    });
19    System.out.println(max);
20 }
```

In dem Code-Beispiel würde die reduce-Methode nun 29 zurückgeben.
Implementieren Sie die vorgegebene reduce-Methode nun so, dass das möglich ist unter Verwendung des BinaryOperators. Sie können davon ausgehen, dass die übergebene Liste mindestens 1 Element enthält!

Lösung:

```
1
2     iterativ:
3     static <T extends Comparable> T reduce(List<T> l, BinaryOperator<T> f) {
4        T result = l.get(0);
5        for (T elem: l) {
6           result = f.apply(elem, result);
7        }
8        return result;
9     }
10
11    rekursiv:
12    static <T extends Comparable> T reduceR(List<T> l, BinaryOperator<T> f)
         {
13       if (l.size() == 1) return l.get(0);
14       return f.apply(l.get(0), reduceR(l.subList(1, l.size()),f));
15    }
```

b)

Gehen Sie davon aus, dass die reduce-Methode existiert. Nun sollen Sie das Minimum in einer Liste von Strings bestimmen. Hierzu bietet es sich nun an, die reduce-Methode zuverwenden.

Schreiben sie also eine BinaryOperator-Implementierung (ähnlich der Implementierung und a) unter Verwendung der Boundry T extends Comparable als anonyme innere Klasse.

Im Prinzip sollte folgendes Programm funktionieren:

Lösung:

```
1
2     public static void main(String[] args) {
3
4        List<String> l2 = Arrays.asList("das", "ist", "ein", "test");
5
6        String ms = reduce(l2, new BinaryOperator<String>() {
7           @Override
8           public String apply(String a, String b) {
9              return (a.compareTo(b) < 0 ? a : b);
10          }
11       });
12       System.out.println(ms);
13    }
14 }
```

c)

Schreiben Sie die anonymen inneren Klasse aus Teilaufgabe b als Lambda-Ausdruck:

Lösung:

```
1
2       public static void main(String[] args) {
3
4         java.util.List<Integer> l1 = Arrays.asList(29, 19, 20, 21, 25, 22,
              23);
5
6         Integer max = reduce( l1, (a,b) -> Integer.max(a,b));
7
8         System.out.println(max);
9       }
```

Name: Martrikelnr.:

6. Aufgabe - Funktionale Programmierung 5+5 Punkte

---

a)

Implementieren Sie den folgenden imperativen Codeabschnitt funktional, unter der Verwendung von Streams; die Variablen f und li können übernommen werden.

```
1
2     // imperativ:
3     double[] f = new double [] {-3.0, -1.0, 0.0, 1.0, 2.0 };
4     List<Double> li = new LinkedList<>();
5     for (int i = 0; i < f.length && i < 4; i++) {
6        if (f[i] >= 0)
7           li.add(f[i]);
8     }
```

Lösung:

```
1
2        // funktional:
3        double[] f = new double [] {-3.0, -1.0, 0.0, 1.0, 2.0 };
4        List<Double> li = new LinkedList<>();
5           Arrays.stream(f)
6            .limit(4)
7            .filter(e -> e >= 0)
8            .forEach(e -> li.add(e));
9
10        // oder
11        Arrays.stream(f)
12           .limit(4)
13           .filter(e -> e >= 0)
14           .forEach(li::add);
15       // oder
16         li = Arrays.stream(f)
17           .limit(4)
18           .filter(e -> e >= 0)
19           .boxed()
20           .collect(Collectors.toList());
```

b)

Implementieren Sie den folgenden imperativen Codeabschnitt funktional, unter der Verwendung von Streams. Hinweis: Es wird der Wrappertyp Double verwendet!

```
1
2     // imperativ
3     public BigDecimal expA(Double[] f, int n) {
4        BigDecimal ergebnis = new BigDecimal(1.0);
5        for (int i = 0; i < f.length; i++) {
6           BigDecimal hilf = new BigDecimal(Math.exp(f[i]));
7           ergebnis = ergebnis.multiply(hilf);
8           if ((i + 1) == n)
9              break;
10       }
11       return ergebnis;
12    }
```

Lösung:

```
1
2     // funktional
3     public BigDecimal expB(Double[] f, int n) {
4        return Arrays.stream(f)
5           .limit(n)
6           .map(x -> new BigDecimal(Math.exp(x)))
7           .reduce((x, y) ->     x.multiply(y))
8           .get();
9     }
```

Name: Martrikelnr.:

7. Aufgabe - Versionierung mit Git 5 Punkte

Ihr Kollege hat eine Änderung der Datei File.java vorgenommen und als Commit in den master Branch des Remote Repository eingebracht. Sie haben ebenso Änderungen an dieser Datei vorgenommen, jedoch auf Ihrem lokalen Branch feat. Sie möchten nun die Änderungen des Kollegen in Ihren lokalen Branch einbringen, ohne Ihre Veränderungen zu verwerfen.
Geben Sie die dazu nötigen git Operationen in der richtigen Reihenfolge an.
Hinweise:

- Geben Sie nach Möglichkeit konkrete git Befehle an, oder bleiben Sie möglichst nahe an den git Vorgängen – es kommt aufs Prinzip an, nicht auf die Syntax.
- Das Repo wurde vor der Änderung des Kollegen geklont, sie befinden sich im feat Branch, die Datei File.java wurde modifiziert.
- Es gibt zwei Lösungen: Mit explizitem Commit oder ohne via stash.

Lösung:
via stash
git stash – 0.5P
git checkout master
git pull
git checkout feat
git merge master
git stash apply – 0.5P

oder commit
git add File.java – 0.5P
git commit -m Meine Änderungen- 0.5P
git checkout master
git pull
git checkout feat
git merge master