



Prüfung - Informatik (INF) 000 - Fortgeschrittene 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 **15 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		14
2		18
3		15
4		10
5		13
6		10
7		5
Summe		85

Note:

.....
(Erstprüfer)

.....
(Zweitprüfer)

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 wechselseitig 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.

2. Wozu dient die Annotation `@Deprecated`?

3. Ordnen Sie die Designpatterns ihrer Kurzbeschreibung zu?

Factory	Eine globale Instanz
Singleton	Traversieren von Datenstrukturen
Fliegengewicht	Reduktion des Speicherbedarfs
Visitor	Erzeugung von Objekten

Name:

Matrikelnr.:

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.

```
1 // Klasse Container
2
3 _____
4
5 // Attribute
6
7 _____
8
9 // Methode getContainedClass
10
11 _____
12
13     _____
14
15         _____
16
17     _____
18
19
20 _____
```

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.

Name:

Matrikelnr.:

c)

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

d)

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

e)

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

```
static <_____> void copy(List<_____> ziel, List<  
_____> quelle)
```

Name:

Matrikelnr.:

3. Aufgabe - Design Pattern

5+5+5 Punkte

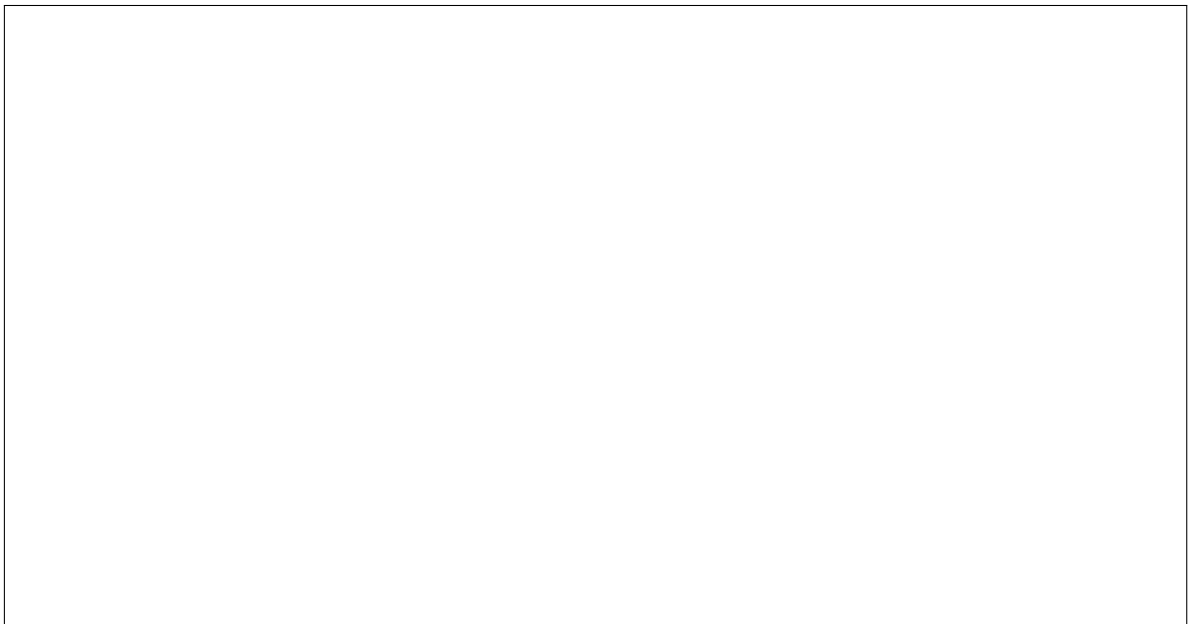
a)

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



b)

Zeichnen Sie das Klassendiagramm des Strategiemusters.

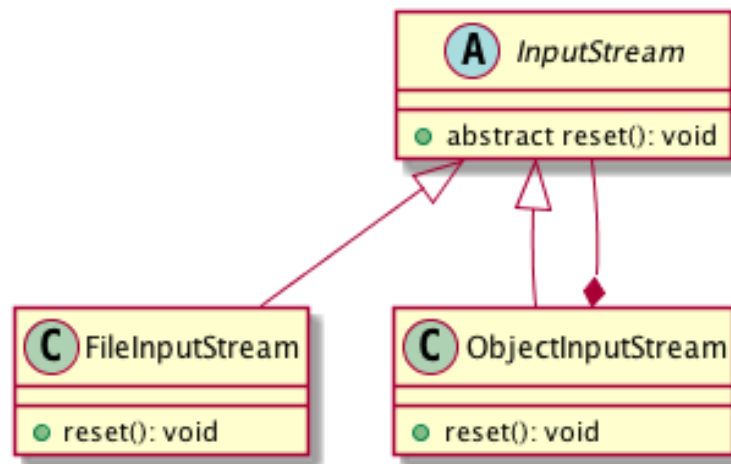


Name:

Matrikelnr.:

c)

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



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.

```
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
8         _____
9
10        while ( _____)
11
12            _____
13
14            _____
15
16
17        T obj = queue.remove();
18
19        _____
20
21        _____
22        return obj;
23    }
24 }
25
26 public _____ put(T obj) throws Exception {
27
28     _____
29
30     _____
31
32     while ( _____)
33
34         _____
35
36         _____
```


Name:

Matrikelnr.:

36

37

38

39

40

41 `queue.add(obj);`

42

43

44

45

46 `}`

47 `}`

48 `}`

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

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!

Name:

Matrikelnr.:

```
1
2  static <T extends Comparable> T reduce(List<T> l, BinaryOperator<T> f) {
```

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 zu verwenden.

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:

```
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
8
9
10
11
12
13
14
15          });
16      System.out.println(ms);
17  }
18 }
```

Name:

Matrikelnr.:

c)

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

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 }
```

```
1
2 // funktional:
3 double[] f = new double [] {-3.0, -1.0, 0.0, 1.0, 2.0 };
4 List<Double> li = new LinkedList<>();
```

Name:

Matrikelnr.:

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 }
```

```
1 // funktional
2 public BigDecimal expB(Double[] f, int n) {
3
4     return _____
5
6     _____
7
8     _____
9
10    _____
11
12    _____
13
14    _____
15
16    _____
17
18 }
```

Name:

Matrikelnr.:

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`.