



Probeklausur - Informatik (INF)
1100 - Fortgeschrittene Programmierkonzepte(FPK)

Datum: 24.12.2020	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**. Die Seiten sind beidseitig bedruckt.
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**.

MUSTERLÖSUNG gibt es im neuen Jahr!

1. Aufgabe - Allgemeines**6+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 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.
5. Annotationen:
 - ☐ Wird eine Methode überschrieben, so **muss** diese mit **@Override** annotiert werden.
 - ☐ Java-Programme sind nicht compile-fähig ohne Annotationen.
 - ☐ Es ist möglich eigene Annotationen zu definieren.
6. 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.

Name:

Matrikelnr.:

b)

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

1. Nennen Sie zwei syntaktische Alternativen zu einer anonymen inneren Klasse, die nur eine Methode hat (@FunctionalInterface).

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 Punkte

a)

Schreiben Sie eine generische Klasse `Container`, welche Objekte eines 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.

```
// Klasse Container
```

```
-----  
    // Attribute
```

```
-----  
    // Konstruktor
```

```
-----  
    // Methode getContainedClass
```

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.: _____

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?

3. Aufgabe - Generics und Bounds

1+4 Punkte

a)

Kurz und knapp: Was bedeutet das Akronym PECS im Kontext von Generics?

b)

Gegeben ist folgender Code:

```
public class PECS {

    interface Cloneable<T> {
        T copy();
    }

    class Shape implements Cloneable<Shape> { public Shape copy() {return null;}}

    class Circle extends Shape {public Shape copy() {return null;}}

    class Square extends Shape {public Shape copy() {return null;}}

    class Rectangle extends Shape {public Shape copy() {return null;}}

}
```

Sie sollen nun eine generische Methode `copyShapes` schreiben, die aus einer gegebenen Liste von `Shapes` eine neue Liste mit Kopien der `Shapes` erstellt. Hierzu kann natürlich die `copy`-Methode verwendet werden. Diese gibt im Code zwar `null` zurück (Platzgründe!), Sie können aber davon ausgehen, dass diese Methode korrekt implementiert wäre.

In der Aufgabe müssen die Bounds entsprechend gesetzt werden. Die Methode erhält 2 Parameter:

- 1. Parameter ist die originale Liste von Shapes
- 2. Parameter die Liste, in die die Shapes hineinkopiert werden sollen

Name: _____ Matrikelnr.: _____

Name: _____ Matrikelnr.: _____

Wie lautet die Methodendeklaration und wie wird die Methode implementiert?

Martrikelnr.:

5+5+5 Punkte

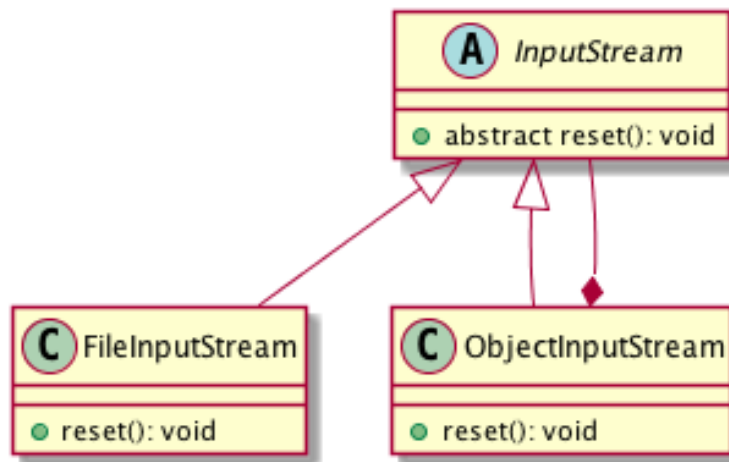
Kurz und knapp: Was ist der Sinn des Strategiemusters (strategy pattern).

Zeichnen Sie das Klassendiagramm des Strategiemusters.

Martrikelnr.:

c)

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



5. Aufgabe - Threads**12 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.

Name:

Matrikelnr.:

```
public class Buffer<T> {  
  
    private Queue<T> queue = new LinkedList<>();  
    private final int maxSize = 10;  
  
    public ----- T get() throws Exception{  
  
        -----  
        while ( ----- )  
  
            -----  
            -----  
  
            T obj = queue.remove();  
  
            -----  
            return obj;  
        }  
    }  
  
    public ----- put(T obj) throws Exception {  
  
        -----  
  
        while ( ----- )  
  
            -----  
            -----  
  
            queue.add(obj);  
  
            -----  
            -----  
        }  
    }  
}
```

6. Aufgabe - Code Smells**12 Punkte****a)**

Gegeben ist der folgende Code. Der ist nicht besonders sauber. Man könnte auch sagen der 'Code smells!'. Stellen Sie sich vor, dass das Switch-Statement komplexer wäre und beim Abspielen der Akkorde gar kompliziertere MIDI-Sequenzen programmiert werden müssten.

```
public static void main(String[] args) {  
    String chords = "C G A C G F C";  
    for (String chord: chords.split(" ")) {  
        switch (chord) {  
            case "C":  
                System.out.printf("c_e_g "); break;  
            case "G":  
                System.out.printf("g_d_d "); break;  
            case "A":  
                System.out.printf("a_e_c "); break;  
            case "F":  
                System.out.printf("f_a_c "); break;  
            case "D":  
                System.out.printf("d_fis_a "); break;  
            default:  
                System.out.printf("---");  
        }  
    }  
}
```

Das Ziel ist nun, das längliche Switch-Statement durch Verwendung des **Strategy Pattern** zu vereinfachen.

Entwerfen und skizzieren Sie unter Verwendung der UML Notation entsprechende Klassen und Methoden. Ihre Lösung soll das Strategy-Pattern umsetzen.

Hierzu sollte eine **play**-Methode definiert werden in einer Klasse, die die entsprechenden Töne (hier simuliert durch die Ausgabe der Töne als System.out) abspielen kann.

Martrikelnr.:

b)

Ersetzen Sie das Switch-Statement durch eine `enum`-Klasse. Hierbei soll das Strategy-Pattern aus Teilaufgabe b) verwendet und als `enum` umgesetzt werden

Am Ende sollten das Switch-Statement durch folgenden Code ersetzbar sein:

```
public static void main(String[] args) {
    List<Chord> chords = Arrays.asList(Chord.C, Chord.G, Chord.A, Chord.F
        , Chord.C, Chord.G, Chord.F, Chord.C);
    for (Chord c:chords) {
        c.play();
    }
}
```

Wie sieht der enum Chord aus?

Name:

Matrikelnr.:

