universität ulm

1. Klausur zur Vorlesung

Programmierung von Systemen

im Wintersemester 2022

Institut für Softwaretechnik und Programmiersprachen
Prof. Dr. Matthias Tichy

28.02.2022, 14 Uhr **Bearbeitungszeit 90min**

Nachname:	Vorname:	Matrikelnummer:
Studiengang und Abschluss:		Platznummer:

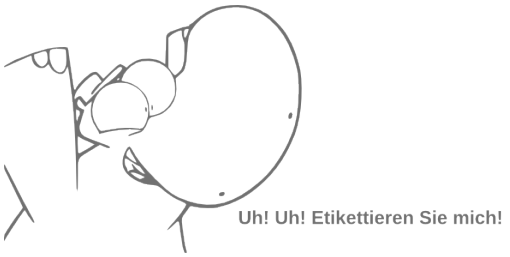
Hiermit erkläre ich, dass ich prüfungsfähig bin.
Sollte ich nicht auf der Liste der angemeldeten Studierenden aufgeführt sein, dann nehme ich hiermit zur Kenntnis, dass diese Prüfung nicht gewertet werden wird.

_____

Datum, Unterschrift des Prüfungsteilnehmenden

Zur allgemeinen Beachtung:

- Füllen Sie das Deckblatt vollständig und korrekt aus.
- Lesen Sie sich zunächst die Klausur sorgfältig durch (sie besteht aus **18** Aufgabenseiten plus Anhang).
- Hilfsmittel sind nicht erlaubt.
- Schreiben Sie ihre Lösungen **lesbar** direkt auf das jeweilige Aufgabenblatt.
- Aufgaben, welche nicht mit einem dokumentenechten Stift in den Farben blau oder schwarz bearbeitet worden sind, werden nicht bewertet.
- Zusätzlich benötigtes Papier wird Ihnen von der Aufsicht zur Verfügung gestellt.



Viel Erfolg !

Punkteverteilung								
1	2	3	4	5	6	7	$\sum$	**Note**
von 14	von 11	von 12	von 12	von 14	von 15	von 12	von 90	
								Korrektur
Einsichtnahme ohne Nachkorrektur ◯				Einsichtnahme mit Nachkorrektur ◯				

Aufgabe 1 - OOP

2 + 4 + 8 = 14 Punkte

Betrachten Sie folgenden Sachverhalt:
In einem Beet werden reihenweise Pflanzen eingesät. Reihen werden mit den Großbuchstaben *'A-Z'* bezeichnet. Dies funktioniert, da kein Beet (Patch) länger als 24 Reihen oder kürzer als 3 Reihen ist.

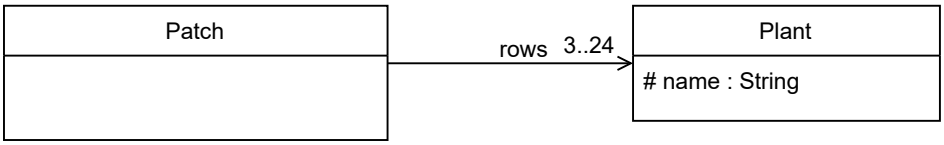


Abb. 1: Klassendiagramm

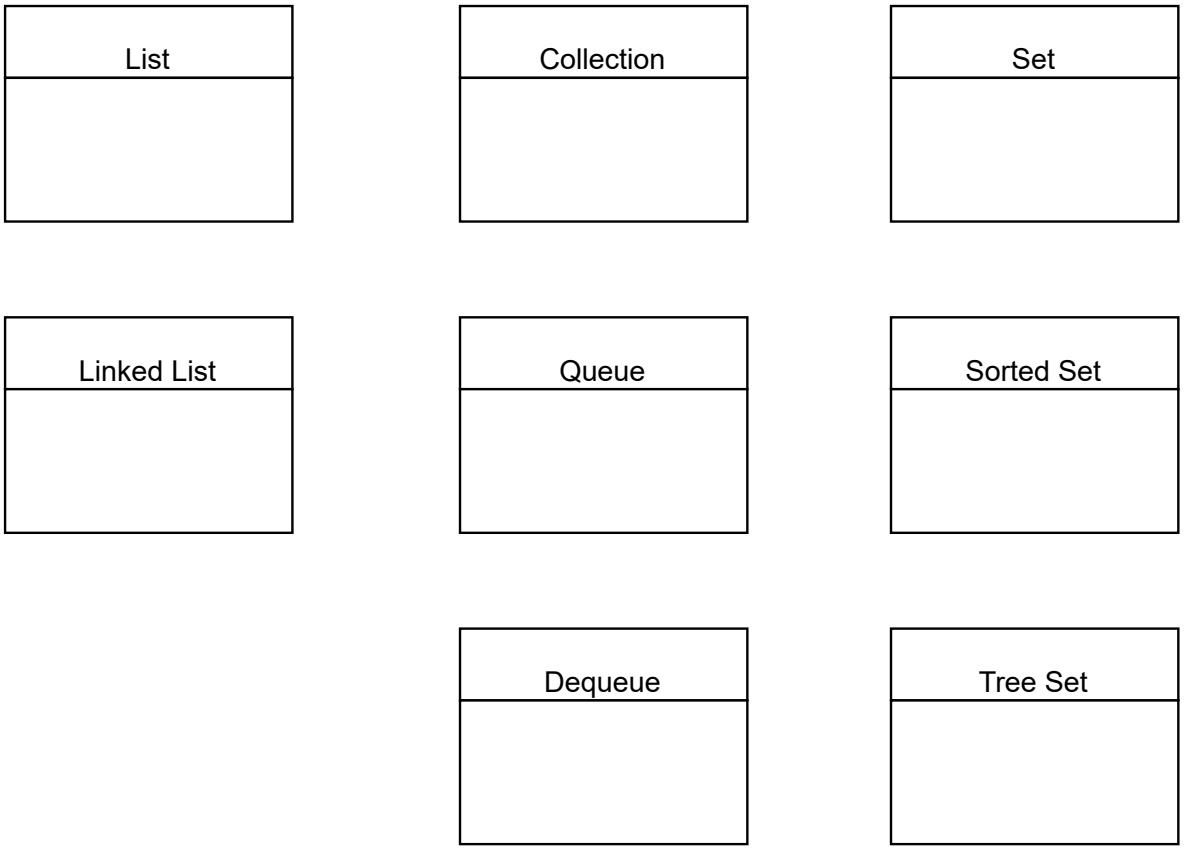
a) Implementieren Sie die Klasse `Patch` und deren Konstruktor `Patch(int size)`, welcher ein Beet der gegebenen Größe, ohne eingepflanzte Pflanzen, anlegt. Stellen Sie in Ihrer Implementierung sicher, dass alle Eigenschaften aus dem obigen Sachverhalt und Klassendiagramm eingehalten werden und behandeln Sie Verstöße sinnvoll.
Hinweis: Sie dürfen gegebenenfalls Hilfsattribute hinzufügen.

b) Implementieren Sie die Methode `addPlantAt` der Klasse `Patch`, welche in einer gegebenen Reihe eine Pflanze pflanzt und `true` zurückliefert wenn dies möglich war. Eine Pflanze kann nicht gepflanzt werden, wenn bereits eine Pflanze in der gegebenen Reihe gepflanzt ist oder die Reihe nicht existiert.

```
public boolean addPlantAt(char row, Plant plant) {
```

c) Ergänzen Sie das folgende Klassendiagramm zur Java Collections-API um Vererbungen und Interface-Implementierungen. Annotieren Sie außerdem Interfaces und Abstrakte Klassen korrekt. Fügen Sie zuletzt die Namen der folgenden Operationen in die Klassen ein, bei denen die jeweilige Operation Sinn ergibt. Es reicht hierbei aus, die Operation in die höchstgelegene Klasse einer Vererbungshierarchie einzutragen.

- *add*: fügt ein übergebenes Element ein
- *first*: liefert das erste Element zurück
- *addAt*: fügt ein übergebenes Element an einer bestimmten Stelle ein
- *contains*: liefert zurück ob ein übergebenes Element existiert
- *indexOf*: liefert die Position eines übergebenen Elements zurück
- *pollLast*: liefert das letzte Element zurück und entfernt es



Diese Seite wurde für ein besseres Layout leer gelassen.

Aufgabe 2 - JavaIO

7 + 2 + 2 = 11 Punkte

Betrachten Sie folgende Klasse:

```
1 public interface Stack<T> {

2     public void push(T element);

3     public T pop();
4 }

5 public class StackLockedException extends RuntimeException {//...}
```

a) Implementieren Sie eine Klasse `LockedStack<T>` auf Basis der Klasse `Stack<T>`. Verwenden Sie hierfür das **Decorator-Pattern**. Ein `LockedStack` soll mittels der Methoden `lock` und open geöffnet und geschlossen werden. Ist der `LockedStack` geschlossen, so soll bei *push* oder *pop* Operationen eine `StackLockedException` geworfen werden. Ein geöffneter `LockedStack` verhält sich wie ein regulärer `Stack`.

```
public class LockedStack<T> implements Stack<T> {
```

b) Beschreiben Sie eine Gemeinsamkeit und einen Unterschied zwischen einem `ObjectStream` und einem `DataStream` in ganzen Sätzen.

c) Implementieren Sie die Methode `printIfExist(String... paths)`, welches für jeden übergebenen Pfad, den Pfad ausgibt so wie ein Hinweis dazu ob das Ziel des Pfades ein Ordner ist.
Hinweis: Beachten Sie hierfür auch den JavaNIO Teil des **CheatSheets** am Ende der Klausur.

```
public static void printIfExist(String... paths) throws IOException {
```

Aufgabe 3 - XML, JSON, Versionierung **3 + 6 + 3 = 12 Punkte**

Betrachten Sie die folgende *pom.xml* (auch zu finden am Ende der Klausur).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0">
3      <groupId>de.uulm.sp.pvs</groupId>
4      <build>
5          <pluginManagement>
6              <plugins>
7                  <plugin>
8                      <artifactId>maven-clean-plugin</artifactId>
9                      <version>3.1.0</version>
10                 </plugin>
11                 <plugin>
12                     <artifactId>maven-reports-plugin
13                       </artifactId>
14                     <version>3.0.0</version>
15                 </plugin>
16             </plugins>
17         </pluginManagement>
18     </build>
19 </project>
```

a) Sie sollen mittels eines Java-Programms dieses XML-Dokument verarbeiten. Nennen Sie *2* Ansätze hierfür. Erläutern Sie zudem, in ganzen Sätzen, **2** Punkte die geklärt werden müssten, um sich auf einen der beiden Ansätze festlegen zu können.

b) Definieren Sie ein JSON Dokument, welches äquivalent zum gegebenen XML Dokument ist.

c) Nennen und beschreiben Sie **3** Vorteile, die sich durch die Verwendung eines Versionierungsystems wie z.B git oder svn ergeben.

Aufgabe 4 - ER-Modellierung

5 + 1 + 4 + 2 = 12 Punkte

Betrachten Sie folgenden Sachverhalt (dieser steht in **keinem** Zusammenhang zu dem aus Aufgabe **1** bekannten):

Ein Beet besteht aus zwischen 3 und 24 Reihen. Pro Reihe können zwischen 10 und 30 Pflanzen gepflanzt werden, hierbei ist die Größe der Pflanzen ausschlaggebend. Ein Beet und dessen zugehörige Reihen werden von Personen bewirtschaftet. Hierbei gilt, dass jedes Beet von mindestens einer Person bewirtschaftet werden muss, während eine Person jeweils nur maximal 3 Beete bewirtschaften kann.

a) Modellieren Sie den beschriebenen Sachverhalt als **E-R-Diagramm**. Verwenden Sie UML-Notation für die Angabe von Kardinalitäten.

b) Nennen Sie **alle** Sachverhalte aus der Beschreibung, die in Ihrem **E-R-Diagramm** nicht modelliert werden können.

c) Erstellen Sie zu dem E-R-Diagramm das dazugehörige **relationale Datenbankschema**. Achten Sie hierbei auf sinnvoll gewählte **Primärschlüssel** und stellen Sie sicher, dass das Schema mindestens in 3. Normalform ist.

d) Beschreiben Sie welche Eigenschaften eine Relation R besitzen muss um in 3. Normalform zu sein.

Aufgabe 5 - SQL

0.5 + 3.5 + 4 + 4 + 2 = 14 Punkte

Gegeben seien die folgenden Relationenschemata (auch zu finden auf dem beiliegenden Extrablatt):

Pferd		
PID	Name	**HID**

Auktion	
AID	Datum

Gebot		
PID	**AID**	Preis

Halter	
HID	HName

Primärschlüssel sind unterstrichen, Fremdschlüssel sind **fett** dargestellt. Formulieren Sie, insofern nicht anders spezifiziert, die **SQL-Statements** zur Lösung folgender Teilaufgaben:

a) Löschen Sie die Tabelle *Auktion*.
Hinweis: Dies geschieht nur im Kontext dieser Teilaufgabe und hat keine Auswirkungen auf andere Teilaufgaben.

b) Fügen Sie die Tabelle *Stall* mit den Attributen *Name*, *Capacity* mit sinnvollen Datentypen und einem Primärschlüssel ein.
Hinweis: Dies geschieht nur im Kontext dieser Teilaufgabe und hat keine Auswirkungen auf andere Teilaufgaben.

c) Erhöhen Sie den Preis aller Gebote, für das Pferd mit dem Namen *Max* um 100.

d) Geben Sie das höchste Gebot für alle Pferde, die an der Auktion am 31.12.1991 teilgenommen haben und mehr als 3 Gebote erhalten haben, aus.

e) Was sind im Kontext von SQL in Java die sogenannten *PreparedStatement*s, wozu dienen Sie, und wie funktionieren Sie?
Hinweis: Sie können Ihre Erklärung durch ein Codebeispiel unterstützen, müssen dies aber nicht.

Aufgabe 6 - GUI

2 + 5 + 4 + 4 = 15 Punkte

Betrachten Sie folgenden Scene Graph:

```
<BorderPane prefHeight="500.0" prefWidth="700.0"
  xmlns="http://javafx.com/javafx/8.0.111"
  xmlns:fx="http://javafx.com/fxml/1">
    <center>
        <BorderPane>
            <top>
                <ToolBar>
                    <items>
                        <Button background-color="blue">
                            <graphic>
                                <ImageView pickOnBounds="true"
                                  preserveRatio="true">
                                    <image>
                                        <Image url="@/open.png" />
                                    </image>
                                </ImageView>
                            </graphic>
                        </Button>
                    </items>
                </ToolBar>
            </top>
            <center>
                <SplitPane orientation="VERTICAL">
                    <items>
                        <AnchorPane minHeight="0.0" minWidth="0.0">
                            <children>
                                <Label text="no file loaded..."/>
                                <ScrollPane fx:id="imageScrollPane">
                                    <content>
                                        <ImageView fx:id="imageView"/>
                                    </content>
                                </ScrollPane>
                            </children>
                        </AnchorPane>
                    </items>
                </SplitPane>
            </center>
        </BorderPane>
    </center>
</BorderPane>
```

a) Nennen Sie **2** Stellen im SceneGraph die Beispiele für eine Anwendung des Composite-Patterns sind und erklären Sie, in ganzen Sätzen, auf Basis derer, was dieses Pattern ist.

b) Erklären Sie, in ganzen Sätzen, was das Observer-Pattern ist und wie es funktioniert. Beschreiben Sie zusätzlich **2** Nachteile des Patterns und wieso diese aufgrund des Aufbaus des Patterns existieren.

c) Erklären Sie, in ganzen Sätzen, wozu Model-View-Controller Architektur dient. Beschreiben Sie zusätzlich die Aufgaben **aller 3** Komponenten.

d) Ergänzen Sie folgenden Code so, dass ein Fenster mit Button angezeigt wird und beim Mausklick des Buttons der Text *Ouch!* auf der Console ausgegeben wird.

```
public void start(Stage primaryStage) {
var sp = new ScrollPane();
var button = new Button();
```

Diese Seite wurde für ein besseres Layout leer gelassen.

Aufgabe 7 - Threads

3 + 7 + 2 = 12 Punkte

a) Was besagt *Ahmdal's law* und welche Schlussfolgerung kann daraus für die Parallelisierung von Programmen gezogen werden?

b) Stellen Sie die verschiedenen Zustände, die ein Thread annehmen kann und wie diese ineinander übergehen, mittels eines (hierarchischen) Zustandsautomaten dar.

c) Betrachten Sie folgenden Code:

```
1  public class Locks extends Thread {

2      Object l1;
3      Object l2;

4      public static void main(String[] args) {
5          var aLock = new Object();
6          var anotherLock = new Object();
7          var locks = new Locks(aLock, anotherLock);
8          var blocks = new Locks(locks, aLock);

9          blocks.doOtherStuff();
10         locks.doStuff();
11     }

12     public Locks(Object a, Object b) {
13         l1 = a;
14         l2 = b;
15     }

16     public void doOtherStuff() {
17         synchronized (l1) {
18             synchronized (l2) {
19                 //...
20             }
21         }
22         synchronized (this) {
23             //...
24         }
25     }

26     public synchronized int doStuff() {
27         return 1337;
28     }
29 }
```

Geben Sie die Namen der Objekte in der Reihenfolge an, in der sie durch die Aufrufe in den Zeilen *9* und *10* verwendet werden, um den Programmablauf zu synchronisieren.

Falls Sie noch Platz benötigen, so können Sie dieses Blatt benutzen. Weitere Blätter erhalten Sie von der Aufsicht. Machen Sie **eindeutig kenntlich**, welche Aufgaben hier bearbeitet bzw. fortgesetzt werden und **streichen** Sie den Lösungsversuch eindeutig durch, den sie nicht bewertet haben wollen.

Diese Seite wurde für ein besseres Layout leer gelassen.

Aufgabe 3 - XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">
    <groupId>de.uulm.sp.pvs</groupId>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <artifactId>maven-clean-plugin</artifactId>
                    <version>3.1.0</version>
                </plugin>
                <plugin>
                    <artifactId>maven-reports-plugin
                      </artifactId>
                    <version>3.0.0</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

Aufgabe 5 - SQL

Pferd		
PID	Name	**HID**

Auktion	
AID	Datum

Gebot		
PID	**AID**	Preis

Halter	
HID	HName

JavaNIO

Files

Modifier and Type	Method	Description
static `Stream<Path>`	`find(Path start, int maxDepth, BiPredicate<Path,BasicFileAttributes> matcher, FileVisitOption... options)`	Return a `Stream` that is lazily populated with `Path` by searching for files in a file tree rooted at a given starting file.
static `Object`	`getAttribute(Path path, String attribute, LinkOption... options)`	Reads the value of a file attribute.
static `<V extends FileAttributeView> V`	`getFileAttributeView(Path path, Class<V> type, LinkOption... options)`	Returns a file attribute view of a given type.
static `FileStore`	`getFileStore(Path path)`	Returns the `FileStore` representing the file store where a file is located.
static `FileTime`	`getLastModifiedTime(Path path, LinkOption... options)`	Returns a file's last modified time.
static `UserPrincipal`	`getOwner(Path path, LinkOption... options)`	Returns the owner of a file.
static `Set<PosixFilePermission>`	`getPosixFilePermissions(Path path, LinkOption... options)`	Returns a file's POSIX file permissions.
static boolean	`isDirectory(Path path, LinkOption... options)`	Tests whether a file is a directory.
static boolean	`isExecutable(Path path)`	Tests whether a file is executable.
static boolean	`isHidden(Path path)`	Tells whether or not a file is considered *hidden*.
static boolean	`isReadable(Path path)`	Tests whether a file is readable.
static boolean	`isRegularFile(Path path, LinkOption... options)`	Tests whether a file is a regular file with opaque content.
static boolean	`isSameFile(Path path, Path path2)`	Tests if two paths locate the same file.
static boolean	`isSymbolicLink(Path path)`	Tests whether a file is a symbolic link.
static boolean	`isWritable(Path path)`	Tests whether a file is writable.
static `Stream<String>`	`lines(Path path)`	Read all lines from a file as a `Stream`.
static long	`size(Path path)`	Returns the size of a file (in bytes).
static `Stream<Path>`	`walk(Path start, int maxDepth, FileVisitOption... options)`	Return a `Stream` that is lazily populated with `Path` by walking the file tree rooted at a given starting file.
static `Stream<Path>`	`walk(Path start)`	Return a `Stream` that is lazily populated with `Path` by walking the file tree rooted at a given starting file.
static `Path`	`walkFileTree(Path start, FileVisitor<? super Path> visitor)`	Walks a file tree.

There are two hard things in computer science: cache invalidation, naming things, and off-by-one errors.

JavaNIO

Path

Modifier and Type	Method	Description
FileSystem	getFileSystem()	Returns the file system that created this object.
Path	getName(int index)	Returns a name element of this path as a Path object.
int	getNameCount()	Returns the number of name elements in the path.
Path	getParent()	Returns the *parent path*, or null if this path does not have a parent.
Path	getRoot()	Returns the root component of this path as a Path object, or null if this path does not have a root component.
int	hashCode()	Computes a hash code for this path.
boolean	isAbsolute()	Tells whether or not this path is absolute.
default Iterator<Path>	iterator()	Returns an iterator over the name elements of this path.
Path	normalize()	Returns a path that is this path with redundant name elements eliminated.
static Path	of(String first)	Returns a Path by converting a path string, or a sequence of strings that when joined form a path string.
static Path	of(URI uri)	Returns a Path by converting a URI.
default WatchKey	register(WatchService watcher, WatchEvent.Kind<?>... events)	Registers the file located by this path with a watch service.
WatchKey	register(WatchService watcher, WatchEvent.Kind<?>[] events, WatchEvent.Modifier... modifiers)	Registers the file located by this path with a watch service.
Path	relativize(Path other)	Constructs a relative path between this path and a given path.
default Path	resolve(String other)	Converts a given path string to a Path and resolves it against this Path in exactly the manner specified by the resolve method.
Path	resolve(Path other)	Resolve the given path against this path.
default Path	resolveSibling(String other)	Converts a given path string to a Path and resolves it against this path's parent path in exactly the manner specified by the resolveSibling method.
default Path	resolveSibling(Path other)	Resolves the given path against this path's parent path.
default boolean	startsWith(String other)	Tests if this path starts with a Path, constructed by converting the given path string, in exactly the manner specified by the startsWith(Path) method.
boolean	startsWith(Path other)	Tests if this path starts with the given path.
Path	subpath(int beginIndex, int endIndex)	Returns a relative Path that is a subsequence of the name elements of this path.
Path	toAbsolutePath()	Returns a Path object representing the absolute path of this path.
default File	toFile()	Returns a File object representing this path.
Path	toRealPath(LinkOption... options)	Returns the *real* path of an existing file.
String	toString()	Returns the string representation of this path.
URI	toUri()	Returns a URI to represent this path.

There are two hard things in computer science: cache invalidation, naming things, and off-by-one errors.