Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

Logik

Sommersemester 2022
1. Aufgabenblatt

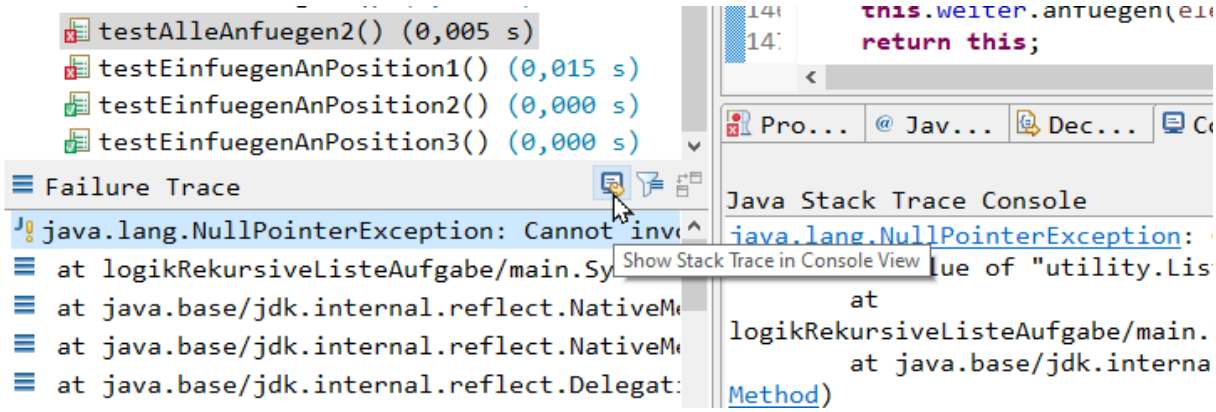
Aufgabe 1 (4 Punkte) [Erinnerung: Listenverarbeitung in Java]

```
public class Recht {
  private String wer = "";
  private String worauf = "";
  private Rechtart wie;
```

Laden Sie von der Veranstaltungsseite das Projekt logikAufgabeRechteanalyse, mit dem Objekte einer stark vereinfachten Klasse Recht (für Zugriffsrechte) verwaltet werden sollen, dabei beschreibt jedes Objekt wer, worauf (also welches Artefakt, wie ein user-Verzeichnis), wie (schreibend, lesend) zugreifen kann.
Implementieren Sie die angefangene Klasse business.Analyse aus, in der Methoden eine Liste von Rechten übergeben bekommen und daraus Ergebnisse in Listenform berechnen. Es geht um folgende Methoden:

darfUserNutzen(): Welche nutzenden Personen (Ergebnis: Liste von wer-Werten) dürfen das Artefakt (worauf) „user“ lesen oder schreiben.
darfLesenUndSchreiben(): Welche nutzenden Personen (Ergebnis: Liste von wer-Werten) haben bei zumindest einem Artefakt gleichzeitig Lese- und Schreibrechte
darfNurLesen(): Welche nutzenden Personen (Ergebnis: Liste von wer-Werten) dürfen auf Artefakte ausschließlich nur lesend (oder gar nicht) zugreifen
kannNurGelesenWerden(): Für welche in der übergebenen Liste bekannten Artefakte (Ergebnis: Liste von worauf-Werten) wurden keine Schreibrechte vergeben
darfAlleArtefakteLesen(): Welche nutzenden Personen dürfen (Ergebnis: Liste von wer-Werten) auf alle bekannten (also in der übergebenen Liste vorhandenen) Artefakte zugreifen

Schauen Sie sich die Klasse main.Systemtest an und verstehen Sie wie die Tests funktionieren und sorgen Sie dafür, dass alle Tests laufen. Das Lesen der Tests kann auch das Verständnis der Aufgabenstellungen erleichtern.
Hinweise: Natürlich dürfen Sie Hilfsmethoden ergänzen. Es wäre eigentlich angebracht, statt List eine Implementierung von Set zu nutzen, da aber in der Veranstaltung oft Algorithmen eine Rolle spielen, die auf Listen basieren, treten diese auch in dieser Aufgabe auf.
Um sich den genaueren StackTrace zu einem von JUnit gefundenen Fehler anzuzeigen, wird auf den linken Button im Balken „Failure Trace“ geklickt, so dass dann rechts Detailinformationen angezeigt werden.



Aufgabe 2 (5 Punkte) [Weiterentwicklung eines rekursiven Typens]

Laden Sie von der Veranstaltungsseite das Projekt logikAufgabeRekursiveListe, in dem sich die leicht erweiterte Klasse Liste aus der Vorlesung befindet. Weiterhin gibt es dort das Interface utitility.Sammlung, das Sie für die Klasse Liste mit rekursiven Methoden umsetzen sollen, so dass zumindest die gegebenen Tests der Klasse main.Systemtest laufen. Für eine

der Methoden macht Rekursion allerdings wenig Sinn, setzen Sie diese ohne um. Die Methoden anfuegen, laenge, isLeer des Interfaces wurden bereits in der Vorlesung realisiert.

`Sammlung<Element>`	`alleAnfuegen(Element... elms)`	Fuegt alle uebergebenen Elemente nacheinander hinten an die Liste an.
`Sammlung<Element>`	`anfuegen(Element e)`	Fuegt das uebergebene Element am Ende in die Liste ein.
`int`	`beinhaltet(Element e)`	Berechnet wie haeufig das Element e in der Liste vorkommt.
`Sammlung<Element>`	`einfuegenAnPosition(int positon, Element e)`	Fuegt das uebergebene Element an der Position position ein, nachfolgende Elemente werden an dieses Element angehaengt
`Sammlung<Element>`	`ersetzeAnPosition(int positon, Element e)`	Ersetzt das Element an der Position position durch das uebergebene Element
`int`	`erstePositionVon(Element e)`	Berechnet die erste Position an der sich ein Element e in der Liste befindet, ist es nicht vorhanden, ist das Ergebnis -1
`Element`	`gib(int position)`	Gibt das Element an der Position position zurueck, gibt es die Position nicht, wird eine IllegalStateException geworfen.
`boolean`	`isLeer()`	Gibt aus, ob die Liste leer ist.
`int`	`laenge()`	Gibt die Laeege der Liste aus.
`Sammlung<Element>`	`loeschen(int position)`	Loescht das Element an der Position position, existiert die Position nicht, wird eine IllegalStateException geworfen.
`Sammlung<Element>`	`loeschen(Element e)`	Loescht das erste Vorkommen des Elements e aus der Liste, existiert das Element nicht, bleibt die Liste unveraendert.
`Sammlung<Element>`	`loeschenAlle(Element e)`	Loescht alle Vorkommen des Elements e aus der Liste, existiert das Element nicht, bleibt die Liste unveraendert.