Praktikumsaufgaben Paradigmen der Programmierung
Wintersemester 2016/17

Name: _____ Vorname: _____

Matrikelnummer: _____

Testat Aufgabe 1: _____

Testat Aufgabe 2: _____

Testat Aufgabe 3: _____

Testat Aufgabe 4: _____

Für die Aufgabe 2 und 3 finden Sie die Basisdateien im ZIP-Archiv aufgaben16_17.zip in Ilias.

Nutzen Sie die drei Praktikumstermine ihrer Gruppe, um Fragen zu stellen und sich beraten zu lassen!

Eine Abgabe der Aufgaben ist jederzeit während des Praktikums möglich.

Die Arbeit in Teams ist erlaubt. Bei der gemeinsamen Abgabe muss jedes Gruppenmitglied einen Teil **jeder Aufgabe** erklären können.

Das Praktikum soll Ihnen helfen, die Inhalte der Vorlesung zu vertiefen und sich auf die Klausur vorzubereiten. Nutzen Sie daher die Praktikumstermine intensiv.

Das Praktikum findet in Raum 3204 statt.

Hinweise für die Informationssuche:

1. Schauen Sie sich den Screencast und die Vorlesungsfolien an
2. Nutzen Sie das Skript von Prof. Ehses:
   http://www.gm.fh-koeln.de/ehses/paradigmen/pp1.pdf
   http://www.gm.fh-koeln.de/ehses/paradigmen/pp2.pdf
3. Ziehen Sie die Scala API-Dokumentation zu Rate:
   http://www.scala-lang.org/api/current/index.html
4. Ziehen Sie die Java API-Dokumentation zu Rate:
   https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
5. Nutzen Sie das OpenBook „Java ist auch eine Insel“:
   http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_14_006.htm#mj9bf9f420b34b6f02276dafd31a8fd050

Aufgabenblock 1 - Prolog

1.) Schreiben Sie zunächst einfache Prädikate ohne Listen, um folgende Sachverhalte als Fakten auszudrücken:

- Susi hat am Mo, Mi und Do Zeit.
- Horst hat am Di,Mi,Do,Fr Zeit.
- Lars hat am Mo, Mi, Sa Zeit.
- Hanna hat am Do,Fr, Sa Zeit.
- Fridolin hat am Mo, Mi, Fr, Sa, So Zeit.

Schreiben Sie Anfragen und Klauseln, um folgende Informationen abzuleiten:

- Hat Person X am Tag Y Zeit?
- Welche Personen haben wann gemeinsam Zeit?
- Welche Personen können 2/3/4 Tage hintereinander arbeiten?
- Wer hat 3 Tage hintereinander Zeit?

Definieren Sie bei Bedarf zusätzliche Fakten, z.B. um die Reihenfolge der Wochentage festzulegen.

2) Drücken Sie nun die Sachverhalte – falls sinnvoll – als Listen aus. Definieren Sie als Listenprädikate mindestens `enthaelt(X, Liste)` und `summe(X,Liste)` und `anhaengen(L1,L2,L3)`. Das Listenprädikat `anhaengen(…)` hängt die Liste L2 an L1 an und bindet das Ergebnis an L3.

3) Die folgende Fakultätsfunktion arbeitet normalrekursiv:

```
% fakultaet(N, N-Fakultaet)
% ------------------------
fakultaet(0, 1).
fakultaet(N, F):-
N > 0,
N1 is N – 1,
fakultaet(N1, F1),
F is N * F1.
```

Schreiben Sie eine neue Variante, die endrekursiv arbeitet. Die Zwischenergebnisse müssen dabei weitergereicht werden, damit auf dem „Hinweg“ gerechnet wird.

4) Ein Binärbaum sei wie folgt definiert:

```
tree(-). % leerer Baum
tree(n(X,L,R)):- tree(L), tree(R).  % Baumknoten mit Inhalt X
```

Erzeugen sie verschiedene Baumstrukturen mit Buchstaben und Zahlen.

Schreiben Sie folgende Prädikate:

```
count(X,Tree,Res). % Zählt wie oft X in Tree vorhanden ist.
replace(X , Y, TreeIn ,  TreeOut).    % ersetzt alle Vorkommen von X
                                      % durch Y in TreeIn und gibt
                                      % Tree Out zurück
sum(Tree, Sum). % Liefert (für Bäume mit numerischen Knoten)
                % die Summe der Knoten.
```

Aufgabenblock 2: Scala

1.) In der funktionalen Programmierung sind Listen die grundlegende Datenstruktur. Daher sollen Sie hier einmal eine eigene Listenimplementierung schreiben.

Dazu ist anzumerken:

- Die Implementierung der Methoden erfolgt in einer Klasse. Die Liste ist also immer der `this`-Parameter.
- Die Methoden entsprechen den Methoden der Klasse `List`.
  Die Implementierung soll jedoch unterschiedlich geschehen (siehe unten).
- Orientieren Sie sich auch an den vorhandenen Funktionen!
- Die Buchstaben A und B bezeichnen Typparameter. Die Angabe ist nur nötig, wenn sonst der Typ nicht klar wäre: z.B. Aufruf der Methode `Lst.empty[A]` oder `Lst.empty[B]`.
- `[B >: A]` bedeutet, B ist ein Obertyp von A.

Sie sollen die folgenden Methoden der Klasse `Lst[A]` implementieren:

- `foldLeft[B] (z: B) (f: (B, A) => B): B`
  Gegenstück zu `foldRight`: fasst linksassoziativ alle Elemente, beginnend mit z, unter der Operation f zusammen („Addition“).
- `map[B](f: A => B): Lst[B]`
  Die Ergebnisliste enthält die Funktionsresultate von f auf den Elemente der Ausgangsliste.
- `flatMap[B](f: A => Lst[B]): Lst[B]`
  Wie `map`, nur, dass `f` Listen liefert, die mit `append` aneinandergehängt werden.
- `filter(p: A => Boolean): Lst[A]`
  Die Ergebnisliste enthält die Elemente der Liste, für die p erfüllt ist.
- `exists(p: A => Boolean): Boolean`
  Es gibt in der Liste ein Element, das p erfüllt
- `apply(n: Int): A`
  Das n-te Element der Liste (`NoSuchElementException` bei leerer Liste)

Gehen Sie bei der Implementierung experimentell vor. D.h. verwenden Sie bei der Implementierung unterschiedliche Methoden:

- Prozedurale Implementierung mit `while`-Schleife.
- Rekursive Implementierung (muss nicht endrekursiv sein)
- Verwendung der Funktion `foldRight`. (z.B. für `map, flatMap, filter, exists`)
- Implementieren Sie `map` oder `filter` mit `flatMap`.

2.) **Testen Sie Ihre Lösungen mit dem Objekt Main.**
Finden Sie heraus, was die `for`-Schleifen bedeuten und was sie mit der Klasse `Lst` zu tun haben

(Hinweis: Eclipse kann Ihnen evtl. helfen, wenn Sie den Cursor über die Elemente der For-Anweisung bewegen. Oder schauen Sie einfach, was passiert, wenn Sie einzelne Methoden auskommentieren (`map, flatMap, filter`). Ignorieren Sie die Warnung (`withFilter`…).

Aufgabenblock 3: Nebenläufigkeit

1) Was ist der Unterschiede zwischen Threads, Runnables, Callables, Futures, und Executors? Erklären Sie die Unterschiede anhand einer selbst angefertigten Skizze.

2) In den nächsten Aufgaben soll die nebenläufige Berechnung von Primzahlen mithilfe des Executor-Frameworks gelöst werden.

Gegeben ist die Klasse `FindPrimeNumbers`. Bei der Objekterzeugung wird dem Konstruktor ein Zahlenbereich übergeben. Die Methode `doCalculations()` findet alle Primzahlen innerhalb des Bereichs.

Die `main()` Methode teilt einen gegebenen Wertebereich auf und erzeugt mehrere `FindPrimeNumber`-Objekte für die aufeinanderfolgenden Segmente. Die Berechnung erfolgt dann mit `doCalculations()` für die einzelnen Objekte. Da es sich noch nicht um eine nebenläufige Lösung handelt, kann nach Beenden der `doCalculations()`-Methode einfach das Objekt als Ergebnis ausgegeben werden, denn `toString()` gibt alle gefundenen Primzahlen des Wertebereichs als einen String zurück.

3) Executors

- Implementieren Sie das `Runnable` Interface für die Klasse `FindPrimeNumbers`

- Wandeln Sie `doCalculations()` in eine private Hilfsmethode, die in `run()` aufgerufen wird.

- Implementieren Sie eine neue Klasse, die das `Executor` Interface implementiert. Diese Implementierung soll zunächst als SingleThread-Lösung umgesetzt werden.

Hinweis: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html

Implementieren Sie nun zusätzlich das Executor Interface als ThreadPerTask, d.h. für jede Aufgabe wird ein neuer Thread erzeugt.

Warum liefert die Zeile

```
System.out.println(pn);
```

nicht mehr wie vorher den gewünschten Effekt? Wie kann man dies lösen?

4) Implementieren Sie nun das `Executor` Interface als Thread-Pool, d.h. Sie verteilen die Aufgaben auf eine bestimmte Anzahl Threads. Die Anzahl soll als Konstante in der Implementierung festgelegt werden, z.B.

```
final int NTHREADS = 20;
```

Tipps für die Threadpool-Umsetzung:

- Die Threads können gleich beim Erzeugen der Thread-Pool-Lösung angelegt werden.
- Verwenden Sie eine vorhandene `BlockingQueue` Implementierung, um neue Arbeitsaufträge in einer Warteschlange zu puffern.
- Jeder Thread läuft in einer Endlosschleife und holt sich den nächsten Arbeitsauftrag aus der Warteschlange

Hinweis: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html

5) Die Schleife in `main()` teilt die Berechnung von Primzahlen in gleich große Abschnitte ab. Warum ist dies für die parallele Berechnung gerade von Primzahlen nicht die beste Aufteilung?

6) Erstellen Sie eine Klasse `PrimeResults`, in der Primzahlen gespeichert werden können. Intern kann hierzu eine beliebige Containerklasse verwendet werden. Verschiedene Threads können nun Primzahlen an `PrimeResults` melden.

- `PrimeResults` muss threadsicher implementiert sein

- Erstellen Sie eine Funktion `Iterator <Integer> getNPrimes(int n)`,

die n Primzahlen als `Iterator` zurückgibt

- Wenn n größer ist als die Anzahl der bereits berechneten Primzahlen ist, dann soll `getNPrimes` solange blockieren bis entsprechend viele Primzahlen vorhanden sind.

- Verwenden Sie z.B. eine Semaphore, um zu gewährleisten, dass erst auf n Primzahlen zugegriffen wird sobald so viele auch vorhanden sind.

Hinweis: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Semaphore.html

Aufgabenblock 4: Entwurfsmuster

Gegeben sei das folgende Interface:

```
interface Ausgabe {
    public void print(String s);
}
```

Setzen Sie das **Dekorierer-Muster** um und implementieren Sie verschiedene konkrete Komponenten und Dekorierer.

1) Implementieren Sie folgende **konkrete Komponenten**:

- Die Klasse `Konsolenausgabe` soll in der `print`-Methode den String einfach auf `System.out` ausgeben.
- Die Klasse `AusgabeAggregator` soll den `String s` an einen `StringBuffer` (als private Eigenschaft) anhängen. Die `toString()`-Methode vom `AusgabeAggregator` liefert die aggregierte Zeichenkette.

2) Implementieren Sie folgende **Dekorierer**:

- Die Klasse `UpperCase` wandelt die Zeichenkette s in Großbuchstaben um.
- Die Klasse `WordCounter` soll speichern, wie häufig ein bestimmtes Wort in allen an `print` gesendeten Zeichenketten vorhanden ist. Zum Beispiel würde ein mit `WordCounter ("hallo" , new Konsolenausgabe())` erzeugtes Objekt zählen, wie oft das Wort "hallo" an ein Ausgabe-Objekt (hier die Konsolenausgabe) geschickt wird. Die Klasse `WordCounter` soll eine getter-Methode haben, um die gezählte Wortzahl abzufragen.
- Die Klasse `LetterCounter` soll die Summe aller an `print` gesendeten Zeichen speichern, also die Aufrufe von `s.length()` kummulieren.
- Denken Sie sich einen weiteren sinnvollen Dekorierer aus, den Sie implementieren.

3)Erzeugen sie beispielhaft ein paar dekorierte Komponenten und verwenden Sie diese.

4) Was sind die Vor- und Nachteile dieses Entwurfsmusters? Geben Sie diese anhand des konkreten Beispiels an.