Name: Matrikelnummer:

Bachelorprüfung: Programmiermethoden und Techniken

WS 21/22

Erlaubte Hilfsmittel: keine

Jede Griff zu einem elektronischen Gerät (z.B. Smartphone) wird als Täuschungsversuch gewertet.

Lösung ist auf den Klausurbögen anzufertigen. (eventuell Rückseiten nehmen)

Bitte legen Sie den Studentenausweis auf den Tisch.

Wird die Heftung der Klausur gelöst, ist auf jedem Blatt der Name einzutragen, ansonsten reicht der Name auf dem Deckblatt.

Bearbeitungszeit: 60 Minuten

Unterschrift

Benotung

Aufgabe:	1	2	3	4	5	6		Gesamt	Note
Punkte:	20	20	21	8	21	10		100	
erreicht:									

Aufgabe 1 (Iteratoren in Java)

In dieser Aufgabe sollen Sie Klassen schreiben, die die Schnittstelle `Iterator` implementieren.

a) (10 Punkte)

Schreiben Sie eine Iteratorklasse `Twice<A>`, die im Konstruktor einen Iterator `xs` übergeben bekommt. Wenn der Iterator `xs` über die Objekte $x_1, x_2, \ldots x_n$ iteriert, dann soll `new Twice<>(xs)` über die Objekte $x_1, x_1, x_2, x_2, \ldots x_n, x_n$ iterieren.

b) (10 Punkte)

Gegeben sei folgende Record-Klasse:

```
1 record Pair<A>(A fst, A snd){}
```

Listing 1: Pair

Schreiben eine Klasse `Pairing<A>`, die `Iterator<Pair<A>>` implementiert.

Sie soll im Konstruktor ein Objekt `xs` des Typs `Iterator<A>` übergeben bekommen.

Ein Objekt der Klasse `Pairing<A>` soll beim Aufruf von `next` jeweils vom übergebenen Iteratoren `xs` die nächsten zwei Elemente holen und als ein Paar zurück geben.

Wenn der Iterator `xs` über die Objekte
$x_1, x_2, \ldots, x_n$
iteriert, soll `new Pairing<>(xs)` über die Objekte
Pair$[x_1, x_2]$, Pair$[x_3, x_4]$, Pair$[x_5, x_6]\ldots]$, Pair$[x_{n-1}, x_n]$ iterieren.

Bei ungeradem n sei das letzte Element: Pair$[x_n, x_n]$.

Aufgabe 2 (Faltungen)

a) Verwenden Sie die Methode `reduce` auf Java `java.util.stream.Stream`-Objekten, um den längsten String aus einem Stream zu selektieren.

```
1  java.util.List<String> xs = ....;
2  String longest = xs.parallelStream().reduce(
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 );
```

b) Verwenden Sie die Methode `reduce` auf Java `java.util.stream.Stream`-Objekten, um zu testen, ob ein bestimmter String `x` in dem Stream enthalten ist.

```
1  java.util.List<String> xs = ....;
2  String x = ...;
3  boolean containsX = xs.parallelStream().reduce(
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 );
```

Aufgabe 3 (Verkettete Listen)

Gegeben sei die folgende Implementierung einfach verketteter Listen:

```
1  package name.panitz.util;
2  import java.util.function.*;
3  import java.util.NoSuchElementException;
4
5  public sealed interface LL<E> permits LL.Nil, LL.Cons{
6    static public final record Nil<E>() implements LL<E>{ }
7    static public final record Cons<E>(E hd,LL<E> tl) implements LL<E>{
       }
8
9    default boolean isEmpty(){return this instanceof Nil;}
10
11   default E head(){
12     if (this instanceof Cons<E> c) return c.hd();
13     throw new NoSuchElementException("head on empty list");
14   }
15   default LL<E> tail(){
16     if (this instanceof Cons<E> c) return c.tl();
17     throw new NoSuchElementException("tail on empty list");
18   }
19   static final LL nil = new Nil<>();
20   static <E> LL<E> nil(){return nil;}
21   static <E> LL<E> cons(E hd,LL<E> tl){return new Cons<>(hd,tl);}
22 }
```

Listing 2: LL.java

Schreiben Sie für diese Schnittstelle die folgenden default-Methoden:

a) (7 Punkte)

Schreiben Sie eine Funktion, die testet, ob für mindestens ein Element in der Liste das übergebene Prädikat wahr ergibt.

```
default boolean exists(Predicate<? super E> p){




}
```

b) (7 Punkte)

Schreiben Sie eine Funktion, die eine Liste ohne die ersten Elemente berechnet, für die das Prädikat p wahr ergibt. Zum Beispiel für eine Liste [2,6,78,80,9,10,11,12] soll für ein Prädikat p, das für gerade Zahlen wahr ist, die Liste [9,10,11,12] enstehen.

```
default LL<E> dropWhile(Predicate<? super E> p){




}
```

c) (7 Punkte)

Schreiben Sie eine Funktion, die eine Liste erzeugt durch Anwendung der übergebenden Funktion auf alle Elemente der Liste.

```
default <R> LL<R> map(Function<? super E,? extends R> f){




}
```

Aufgabe 4 (XML)

a) (8 Punkte)

Schreiben Sie mit dem DOM API (Paket `org.w3c.dom`) die Methode `collectTagNames`.

Sie soll den ganzen Baum durchlaufen und alle Tagnamen der Menge `java.util.Set<String> result` zufügen.

```
static void collectTagNames(Node node, Set<String> result){




}
```

Aufgabe 5 (Bäume in C)

Gegeben sei folgende Struktur für Binärbäume von Zahlen:

```
#include <stdbool.h>
typedef struct TreeNode {
  struct TreeNode* left;
  int element;
  struct TreeNode* right;}
 Tree;
```

Listing 3: String.c

a) (7 Punkte)

Schreiben Sie die Funktion, die die maximale Tiefe im Baum berechnet. Damit ist die größte Anzahl von Knoten auf einem Pfad von der Wurzel bis zu einem Blatt gemeint.

```
unsigned int maxDepth(Tree* this){




}
```

b) (7 Punkte)

Schreiben Sie die Funktion `mkLeaf`. Es soll ein Baum erzeugt werden, der nur aus einem Wurzelknoten besteht.

```
Tree* mkLeaf(int el)
```

```
}
```

c) (7 Punkte)

Schreiben Sie eine Destruktor-Funktion, die den Baum komplett aus dem Speicher löscht.

```
void deleteTree(Tree* this){
```

```
}
```

Aufgabe 6 Erklären Sie in kurzen Worten.

a) Wann und wie wird Speicher in C auf dem Stack und auf dem Heap allokiert.

b) Was gibt folgendes Programm aus? Wie kommt es dazu?

```c
#include <stdio.h>
void print(int xs []){
  printf("%lu\n",sizeof(xs));
}
int main(){
  int xs [] = {1,2,3,4};
  print(xs);
  printf("%lu\n",sizeof(xs));
  return 0;
}
```

Listing 4: A.c