Name:

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.

```
class Twice<A> implements Iterator<A> {
  Iterator<A> xs;
  Twice(Iterator<A> xs){this.xs=xs;}
  boolean first = true;
  A x = null;
  @Override public A next(){
    if(first){
      x = xs.next();
    }
    first = !first;
    return x;
  }
  @Override public boolean hasNext(){
    return !first || xs.hasNext();
  }
}
```

Name:

b) (10 Punkte)

Gegeben sei folgende Record-Klasse

```
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 *Pair* zurück geben.

Wenn der Iterator `xs` über die Objekte

$x_1, x_2, \ldots, x_n$

iteriert, soll `new Pairing<>(xs)` über die Objekte

$\text{Pair}[x_1, x_2]$, $\text{Pair}[x_3, x_4]$, $\text{Pair}[x_5, x_6]$, ..., $\text{Pair}[x_{n-1}, x_n]$ iterieren.

Bei ungeradem n sei das letzte Element $\text{Pair}[x_n, x_n]$.

```
class Pairing<A> implements Iterator<Pair<A>> {
  Iterator<A> xs;
  Pairing(Iterator<A> xs){this.xs = xs;}
  @Override public boolean hasNext() {
      return xs.hasNext();
  }
  @Override public Pair<A> next(){
      A x1 = xs.next();
      if(!xs.hasNext()) return new Pair<>(x1, x1);
      return new Pair<>(x1, xs.next());
  }
}
```

Aufgabe 2 (Faltungen)

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

```
java.util.List<String> xs = ...
String longest = xs.parallelStream().reduce(
...
,(r,x) -> r.length > x.length ? r : x
)
```

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.

```
java.util.List<String> xs = ...
String x = ...
boolean containsX = xs.parallelStream().reduce(
false
,(r,y) -> r || y.equals(x)
,(y,z) -> y || z
)
```

Gegeben sei die folgende Implementierung:

```
package name panitz.util;
import java.util.function.*;
import java.util.NoSuchElementException;

public sealed interface LL<E> permits LL.Nil, LL.Cons{
  static public final record Nil<E>() implements LL<E>{ }
  static public final record Cons<E>(E hd,LL<E> tl) implements LL<E>{
  }

  default boolean isEmpty(){return this instanceof Nil;}

  default E head(){
    if (this instanceof Cons<E> c) return c.hd();
    throw new NoSuchElementException("head on empty list");
  }
  default LL<E> tail(){
    if (this instanceof Cons<E> c) return c.tl();
    throw new NoSuchElementException("tail on empty list");
  }
  static final LL nil = new Nil<>();
  static <E> LL<E> nil(){return nil;}
  static <E> LL<E> cons(E hd,LL<E> tl){return new Cons<>(hd,tl);}
}
```

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){
  if(isEmpty()) return false;
  if(p.test(head())) return true;
  return tail().exists(p);
}
```

b) (7 Punkte)

Schreiben Sie eine Funktion, die eine Liste über die enthaltenen Elemente berechnet, für die das Prädikat p wahr ergibt. Zum Beispiel für eine Liste [2,6,7,8,8,9,10,11,12] soll für ein Prädikat p das für gerade Zahlen wahr ist, die Liste [9,10,11,12] entstehen.

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

```
if (isEmpty()) return this;
if (p.test(head()))
    return tail().dropWhile(p);
return this;
```

}

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){
```

```
return isEmpty() ? nil()
        : cons(f.apply(head())
            ,tail().map(f));
```

}

Name:

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){
```

```
if (node.getNodeType()
        == Node.ELEMENT_NODE)
    result.add(node.getNodeName()

var cs= node.getChildNodes();
for (var i=0; i<cs.getLength(); i++)
    collectTagNames(cs.item(i), result)
```

}

Aufgabe 5 (Bäume

Gegeben sei folgende Struktur für Binarbäume 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){
```

```
if(this==NULL) return 0;
unsigned int lr = maxDepth(this->left);
unsigned int rr = maxDepth(this->right);
return (lr>rr ? lr : rr) + 1;
```

```
}
```

b) (7 Punkte)

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

```
Tree* mkLeaf(int el) {
  Tree* this = (Tree*)malloc(sizeof(Tree));
  this->left = NULL;
  this->right = NULL;
  this->element = el;

  return this;
}
```

c) (7 Punkte)

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

```
void deleteTree(Tree* this){
  if (this->left != NULL)
      deleteTree(this->left);
  if (this->right != NULL)
      deleteTree(this->right);

  free(this);
}
```

```
if(this==NULL) return;
deleteTree(this->left);
deleteTree(this->right);
free(this);
```

Name: Matrikelnummer:

Aufgabe 6 Erklären Sie in kurzen Worten.

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

Stack: bei jedem Aufruf eine Funktion für Parameter + lokale Variablen.

Heap: explizit durch Aufruf der Standardfunktion malloc (und Verwandte).

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

```
#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

→ 8 (Adressgröße)

→ 16 4 int Zahlen