Aufgabe 1 (Iteratoren in Java)

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

a) (6 Punkte)

Schreiben Sie eine generische Klasse `Abwechselnd<A>`. Die Klasse soll die Schnittstelle `Iterator<A>` implementieren.

Die Klasse soll im Konstruktor zwei Objekte vom Typ `A` und eine Zahl erhalten.

Bei der Methode `next` sollen jeweils die im Konstruktor übergebenen Objekte abwechselnd zurückgegeben werden.

Die im Konstruktor übergebene Zahl soll angeben, wie oft die Methode `next` aufgerufen werden kann, bis `hasNext` `false` zurück gibt.

```
class Abwechselnd<A> implements Iterator<A> {
    A a1;   int n;
    A a2;
    Abwechselnd(A a1, A a2, int n) { this.a1=a1; this.a2=a2;
        this.n=n;
    }
    @Override
    public A next() {
        n--;
        return n%2==0 ? a1 : a2;
    }
    @Override
    public boolean hasNext(){
        return n>0;
    }
}
```

b) (6 Punkte)

Gegeben sei folgende Schnittstelle:

```
1 interface F<A,B>{
2   B apply(A a1,A a2);
3 }
```

Listing 1: F

Schreiben eine Klasse `CombineTwo<A,B>`, die `Iterator<B>` implementiert.

Sie soll zwei Objekte im Konstruktor übergeben bekommen, ein Objekt `f` des Interfaces `F<A,B>` und ein Objekt `itA` des generischen Typs `Iterator<A>`.

Ein Objekt der Klasse `CombineTwo<A,B>` soll beim Aufruf von `next` jeweils von dem übergebenen Iterator `itA` die nächsten zwei `A`-Objekt erfragen und mit diesen dann mit der Funktion `f` in ein `B`-Objekt errechnen, welches das Ergebnis des Aufrufs ist.

Die Methode `hasNext` gibt solange `true` zurück, wie der übergebene Iterator noch zwei weitere Elemente zurückgeben kann.

```
class CombineTwo<A,B> implements Iterator<B>{
F<A,B> f; Iterator<A> itA;
A a;
CombineTwo(F<A,B> f, Iterator<A> itA){
    this.f = f; this.itA = itA;
    if(itA.hasNext()) a = itA.next();
    a;
}

@Override
public boolean hasNext(){
    return itA.hasNext();
}
@Override
public B next(){
    var r = f.apply(a, itA.next());
    if(itA.hasNext()) a = it.next();
    return r;
}
```

c) (6 Punkte)

Schreiben Sie eine Klasse Random die über eine unendliche Folge von Pseudozufallszahlen iteriert. Die Klasse soll die Schnittstelle Iterator<Long> implementieren.

Die Klasse soll im Konstruktor vier Zahlen vom Typ long erhalten:

- $m \in \{2, 3, 4, \ldots\}$
- $a \in \{1, \ldots, m-1\}$
- $b \in \{1, \ldots, m-1\}$
- $y_1 \in \{0, \ldots, m-1\}$

(Sie brauchen die Vorbedingungen der Argumente nicht mit asserts zu überprüfen.)

Der Iterator soll über die folgende Folge $y_1, y_2, y_3 \ldots$ iterieren mit

$$y_i = (a \cdot y_{i-1} + b) \bmod m$$

```
class Random implements Iterator<Long> {
    Long m, a, b, y1;
    Random(Long m, Long a, Long b, Long y1) {
        this.m = m; this.a = a; this.b = b; this.y1 = y1;
    }
    @Override
    public boolean hasNext() { return true; }

    @Override
    public Long next() {
        var r = y1;
        y1 = (a * y1 + b) % m;
        return r;
    }
}
```

Aufgabe 2 (Faltungen, reduce)

Gegeben Sie die Schnittstelle

```
interface BinFunction<A,B>{
  A apply(A a,B b);
}
```

Listing 2 BinFunction

a) (6 Punkte)

Schreiben Sie für die Klasse, die einfach verkettete Listen implementiert, eine Funktion **reduce**, die eine Faltung realisiert:

```
class Li<A>{
  final A head;
  final Li<A> tail;
  Li(A h,Li<A> t){head=h; tail=t;}
  Li(){this(null,null);}
  boolean isEmpty(){return head==null&tail==null;}

  <B> B falte(B start, BinFunction<B,A> f){
      if (isEmpty()) return start;
      return tail.falte(f.apply(start,head(),f);
      for (var it=this; !it.isEmpty(); it=it.tail()){
          start = f.apply(start,it.head());
      }
      return start;
  }
}
```

b) (3 Punkte)

Machen Sie einen Beispielaufruf der Funktion **falte**, so dass ein String entsteht, indem alle Strings der Liste mit + aneinander gehängt werden.

```
1  Li<String> xs = ... ;
2
3  String conatString = xs falte("", (r,x) -> r + x );
4
5
6
7
8
9
10
11
12
13
14 );
```

c) (3 Punkte)

Machen Sie einen Aufruf der Funktion **falte**, so dass der längste String der Liste berechnet wird.

```
1  Li<String> xs = ... ;
2
3  String laengsterString = xs falte(
4
5        ""
6     , (r,x) -> r.length() < x.length()
7                 ? x
8                 : r
9
10
11
12
13
14 );
```

Name: Matrikelnummer

Aufgabe 3 (XML)

a) (6 Punkte)

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

Sie soll den ganzen Baum durchlaufen und die Namen der Elementknoten 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 (int i=0; i< cs.getLength(); i++){
        collectTagNames(cs.item(i), result);
    }
```

Name: Matrikelnummer

b) (6 Punkte)

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

Sie soll testen, ob im Dokument ein Elementknoten mit dem übergebenen Tagnamen enthalten ist.

```
static boolean containsTagWithName(Node node, String tagname){
if (node.getNodeType() == Node.ELEMENT_NODE
    && tagname.equals(node.getNodeName()))
    return true;
var cs = node.getChildNodes();
For (var i=0; i<cs.getLength(); i++){
  if (containsTagWithName(cs.item(i), tagname))
      return true;
}
return false;
```

Name: Matrikelnummer:

Aufgabe 4 (Strings in C)

Gegeben sei die folgende Struktur, die String Objekte realisiert

```
1 #include <stdbool.h>
2
3 typedef struct{
4   unsigned int length;
5   char* data;
6 } String;
```

Listing 3: String.c

Schreiben Sie folgende Funktionen in C ohne Verwendung der Funktionen aus der Standardbibliothek `string.h`

a) (5 Punkte)

Schreiben Sie die Funktion `indexOfLastOccurrence(String this,char c)`. Es soll der letzte Index zurück gegeben werden, an dem der Parameter c im String zu finden ist. Ist c nicht im String enthalten, so soll −1 zurück gegeben werden

```
int indexOfLastOccurrence(String this,char c){
    int i;
    for (i = this.length; i > 0; i--){
        if ( c == this.data[i-1])
            return i-1;
    }
    return -1;
```

Name: Matrikelnummer:

d) (5 Punkte)

Schreiben Sie die Funktion `reverse(String this)`

Es soll ein neuer String erzeugt werden, der aus denselben Zeichen wie der übergebene String in umgekehrter Reihenfolge existiert.

```
String reverse(String this){
String r;
r.length = this.length;
r.data = malloc(sizeof(char)*(r.length+1));
r.data[r.length] = '\0';
int i;
for(i=0; i<r.length; i++){
  r.data[i] = this.data[r.length-1-i];
}
return r;
```

Name: Matrikelnummer

Aufgabe 5 Gegeben sei folgende Datenstruktur für einen Binärbaum.

```
typedef struct BT{
  struct BT* left;
  ... element;
  struct BT* right;
} BinTree;
```

Listing 4: Expression.h

Die linken und rechten Kinder können auch NULL sein. Wenn beide NULL sind, liegt ein Blatt vor.

Schreiben Sie folgende Funktionen für diese Datenstruktur.

a) (6 Punkte)

Schreiben Sie eine Konstruktorfunktion zum Erzeugen eines Baumes

```
BinTree* newBinTree(BinTree* left, int e,BinTree* right){
  BinTree* result = malloc(sizeof(BinTree));
  result->left = left;
  result->right = right;
  result->element = element;

  return result;
```

Name: Matrikelnummer

b) (6 Punkte)

Schreiben Sie eine Funktion die testet, ob ein bestimmtes Element e in einem Baum gespeichert ist:

```
bool contains(BinTree* this, int e){
    if (this == NULL) return false;
    return this -> element == e
        || contains(this -> left, e)
        || contains(this -> right, e);
}
```

c) (6 Punkte)

Schreiben Sie jetzt die Funktion **contains** unter der Annahme, dass der übergabene Baum **this** ein binärer Suchbaum ist, d.h. für jeden Knoten alle im linken Kind gespeicherten Elemente kleiner und alle im rechten größer als das Wurzelelement sind.

```
bool contains(BinTree* this, int e){
    if (this == NULL) return false;
    if (this -> element == e) return true;
    if (this -> element < e) return contains(this->left, e);
    return contains(this -> right, e);
}
```

Name: Matrikelnummer:

Aufgabe 6 Erklären Sie in kurzen Worten.

a) Wie kann man im SAX API formulieren, dass man für Textknoten etwas bestimmtes programmieren will. Was ist dabei insbesondere zu beachten, wenn man mit dem kompletten Text eines Textknotens arbeiten will.

Man überschreibt die Methode characters-Methode der Handlerklasse. Diese kann für einen Textknoten mehrfach mit Textfragmenten aufgerufen werden.

b) Die Methode reduce für Faltungen auf Streams gibt es überladen mit zwei Parametern, die von einer funktionalen Schnittstelle sind. Einem Parameter `BiFunction<U, ? super T, U> accumulator` und einem Parameter `BinaryOperator<U> combiner`. Wofür wird der Parameter `combiner` benötigt?

Für die Parallelisierung, um Teilergebnisse vom Typ U zu einem Gesamtergebnis zusammenzufügen.

Name: Matrikelnummer:

c) Was ist an folgenden Code in C problematisch?

```
1 int* f(int* x){
2   int xs[] = {1,2,3};
3   int* ys = xs;
4   ys[2] = *x;
5   return ys;
6 }
```

Listing 5: a.c

Es wird ein Zeiger auf den Stack zurück gegeben. Die dort gespeicherte lokale Variable existiert nach dem return nicht mehr.

d) Warum verwendet man bei der Definition von Datentypen mit einem `struct` zumeist auch gleich ein `typedef` in C?

weil sonst das Wort struct Teil der Typname ist, und das umständlich ist.

Name: Matrikelnummer

e) Wofür werden Objekte des Typs `Spliterator` in Java verwendet?

Um Streams über Streams überhaupt und parallel zu
iterieren.