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 `Repeaterator<A>`. Die Klasse soll die Schnittstelle `Iterator<A>` implementieren.

Die Klasse soll im Konstruktor ein Objekt des Typs `A` und eine ganze Zahl erhalten.

Bei der Methode `next` soll jeweils das im Konstruktor übergebene Objekt zurückgegeben werden.

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

```
class Repeaterator<A> implements Iterator<A>{
    A a;
    int n;
    Repeaterator(A a, int n){
        this.a = a;
        this.n = n;
    }
    @Override
    public A next(){
        n--;
        return a;
    }
    @Override
    public boolean hasNext(){
        return n > 0;
    }
}
```

Name: Matrikelnummer:

b) (6 Punkte)

Gegeben sei folgende Schnittstelle:

F2

```
1 interface F2<A,B,C>{
2   C apply(A a,B b);
3 }
```

Listing 1: F2

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

Sie soll drei Objekte im Konstruktor übergeben bekommen: ein Objekt `f` des Interfaces `F2<A,B,C>` und je ein Objekt `itA` des generischen Typs `Iterator<A>` und `itB` des generischen Typs `Iterator<B>`.

Ein Objekt der Klasse `ZipWith<A,B,C>` soll beim Aufruf von `next` jeweils von beiden übergebenen Iteratoren `itA` und `itB` das nächste `A`- bzw. `B`-Objekt erfragen und mit diesen dann mit der Funktion `f` in ein `C`-Objekt errechnen, welches das Ergebnis des Aufrufs ist.

Die Methode `hasNext` gibt solange `true` zurück, wie beide übergebenen Iteratoren noch weitere Elemente zurückgeben.

```
class ZipWith<A,B,C> implements Iterator<C>{
  F2<A,B,C> f;
  Iterator<A> itA; Iterator<B> itB;
  ZipWith(F2<A,B,C> f2, Iterator<A> a, Iterator<B> b){
    f=f2; itA=a; itB=b;
  }
  @Override
  public C next(){
    return f.apply(itA.next(), itB.next());
  }
  @Override
  public boolean hasNext(){
    return itA.hasNext() && itB.hasNext();
  }
}
```

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:

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

```
class Random implements Iterator<Long> {
    long m, a, b, y1;
    Random(long pm, long pa, long pb, long py1){
        m = pm; a = pa; b = pb; y1 = py1;
    }

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

    @Override
    public boolean hasNext() {
        return true;
    }
}
```

Aufgabe 2 (Faltungen, reduce)

Gegeben Sie die Schnittstelle

BinFunction

```
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);

  }


  }
}
```

Rekursiv

```
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.

```
Li<String> xs = .... ;

String conatString = xs.falte( "", (r,e) -> r+e );
```

c) (3 Punkte)

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

```
Li<String> xs = .... ;

String laengsterString = xs.falte(
"", (r,e) -> r.length > e.length
? r
: e
);
```

Aufgabe 3 (XML)

a) (6 Punkte)

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

Sie soll den ganzen Baum durchlaufen und die Namen er 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++){
    collectTagName(cs.item(i); result);
}
}
```

b) (6 Punkte)

Schreiben Sie mit dem DOM API (Paket `org.w3c.dom`) die Methode `highestChildrenNumber`. Sie soll berechnen, was die höchste Anzahl an Kindknoten ist, die ein Element im Baum hat.

```
static int highestChildrenNumber(Node node){
var r = node.getChildNodes().setLength();
var cs = node.setChildNodes();
for(int i=0; i<cs.getLength(); i++){
  var r2 = highestChildrenNumber(cs.item(i));
  if(r2>r) r=r2;
}
return r;
}
```

Aufgabe 4 (Strings in C)

Gegeben sei die folgende Struktur, die String-Objekte realisiert:

String

```c
#include <stdbool.h>

typedef struct{
  unsigned int length;
  char* data;
} 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 `indexOfFirstOccurrence(String this,char c)`.

Es soll der erste 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.

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

b) (5 Punkte)

Schreiben Sie die Funktion `isPrefixOf(String this,String that)`.

Es soll genau dann true sein, wenn der String `that` den String `this` als Präfix hat, also der String `that` mit dem String `this` startet.

```
bool isPrefixOf(String this,String that){
    if (this.length > that.length) return false;
    int i;
    for (i=0; i<this.length; i++)
        if (this.data[i] != that.data[i])
            return false;

    return true;
}
```

c) (5 Punkte)

Schreiben Sie die Funktion `removeAt(String this,unsigned int i)`.

Der String soll so modifiziert werden, dass das Zeichen am Index `i` gelöscht wird. Der String wird also ein Zeichen kürzer. Wenn der Index `i` außerhalb des Bereichs ist, dann bleibt der String unverändert.

```
void removeAt(String this,unsigned int i){
    if (i >= this.length) return;
    for ( ; i<this.length-1; i++)
        this.data[i] = this.data[i+1];

    // length kann nicht nach außen sichtbar
    verändert werden.
    Man bräuchte eine Zeiger auf eine Sins
}
```

d) (5 Punkte)

Schreiben Sie die Funktion `copy(String this)`.

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

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

Aufgabe 5 Gegeben sei folgende Datenstruktur für einen Binärbaum.

Expression

```
typedef struct BT{
  struct BT* left;
  int 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* this = malloc(sizeof(BinTree));
  this -> left = left;
  this -> e = e;
  this -> right = right;
  return this;
}
```

b) (6 Punkte)

Schreiben Sie zum Löschen eines Baumes aus dem Speicher:

```
void deleteBinTree(BinTree* this){
if (this -> left != NULL) deleteBinTree(this -> left);
if (this -> right != NULL) deleteBinTree(this -> right);
free(this);
}
```

c) (6 Punkte)

Schreiben Sie eine Funktion, die die Summe der Zahlen in einem Baum berechnet. Ist der Baum eine NULL-Referenz, dann sei das Ergebnis 0.

```
unsigned int sum(BinTree* this) {
if(this == NULL) return 0;
return sum(this -> left)
     + this -> e
     + sum(this -> right);
}
```

Aufgabe 6 Erklären Sie in kurzen Worten.

a) Nach welchem Prinzip arbeitet das SAX API. Was ist der Vorteil und was der Nachteil gegenüber dem DOM API?

SAX benutzt Event-Listener, die auf die unterschiedliche XML-Konstrukte als Events reagieren (Handlerklassen).
Vorteil: es muss nicht das gesamte Dokument auf einmal im Speicher stehen
Nachteil: es kann nicht beliebig durch die Baumstruktur navigiert werden.

b) Es ist angedacht in zukünftigen Versionen von Java die switch-case-Anweisung auf ein volles Pattern-Matching auf reinen Datenhaltungsklassen zu erweitern. Was versteht man darunter?

Eine einfache Fallunterscheidung über die verschiedenen Unterklassen oder Klasse (Code einer Schnittstelle).

c) Welchen Typ kann man in C verwenden, um beliebige Daten auf dem Heap in einem Speicherbereich zu speichern?

Dafür steht der Typ void*, der ein Zeiger auf beliebige Daten ausdrückt.

d) Man spricht von Summen- und von Produkttypen. Mit welchen Konstrukten zur Datendefinition in C werden diese beiden Typarten definiert?

Für Produkttypen kann man in C struct Datentypen definieren. Für Summentypes steht das union-Konstrukt als Datentyp zur Verfügung.

e) Sie wollen die Bearbeitung der Elemente einer Sammlung in Java parallelisieren. Was können Sie nutzen?

Die Standardsammlungsklassen haben die Methode parallelStream(), die einen Stream über die Elemente erzeugt, der parallelisiert iteriert werden kann.