Übungsblatt 1

(11. Oktober 2010)

Aufgabe 1 In dieser Aufgabe sollen Sie noch einmal wiederholen, wie rekursive Funktionen definiert werden:

a) Schreiben Sie eine statische Methode
`static void hello();`
die genau einmal `Hello` auf der Kommandozeile ausgibt.

b) Schreiben Sie eine rekursive Methode
`static void helloWithoutEnd();`
die ohne zu terminieren immer wieder `Hello` auf der Kommandozeile ausgibt.

c) Schreiben Sie eine rekursive Methode
```
static void helloNTimes(int n);
```
die genau n-Mal `Hello` auf der Kommandozeile ausgibt.

Aufgabe 2 (1 Punkt)
In dieser Aufgabe soll die Modellierung der natürlichen Zahlen aus der Vorlesung fort geführt werden.

a) Implementieren Sie die Methode `equals` für die Klasse `SN`.

b) Ergänzen Sie die Schnittstelle `Nat` um eine Methode für die kleiner-gleich Relation und implementieren Sie diese in der Klasse `SN`:
`boolean le(Nat that);`

Dabei ist die kleiner Relation definiert als:
$le(N, S(m)) = true$
$le(S(n), S(m)) = le(n, m)$
$le(S(n), N) = false$
$le(N, N) = true$

c) Ergänzen Sie die Schnittstelle `Nat` um eine Methode für den Modulo-Operator und implementieren Sie diese in der Klasse `SN`:
`Nat mod(Nat that);`

Dabei ist der Modulo Operator definiert als:
$mod(N, m) = N$

$mod(n,m) = mod(sub(n,m),m)$ für $lt(m,n)$
$mod(n,m) = n$ wenn nicht $lt(m,n)$

Ist der rechte Operand N, so soll eine Ausnahme geworfen werden.

d) Ergänzen Sie die Schnittstelle `Nat` um eine Methode für die Division und implementieren Sie diese in der Klasse `SN`:
`Nat div(Nat that);`
Dabei ist die Division definiert als:
$div(N,m) = N$
$div(n,m) = S(div(sub(n,m),m))$ für $lt(m,n)$
$div(n,m) = N$ wenn nicht $lt(m,n)$

Ist der rechte Operand N, so soll eine Ausnahme geworfen werden.

Aufgabe 3 (1 Punkt)
Legen Sie eine Unit-Test Klasse für die Klasse `SN` an, und testen Sie die Methoden für die Grundrechenarten.

Aufgabe 4 In dieser Aufgabe sollen Sie nun wiederholen, rekursive Algorithmen auf einer rekursiven Datenstruktur zu implementieren.

Gegeben sei folgende Klasse für einfach verkettete Listen mit Stringelementen:

StringLi.java
```
1  class StringLi{
2    private String hd=null;
3    private StringLi tl=null;
4    private boolean isEmpt=true;
5
6    public StringLi(){}
7    public StringLi(String h,StringLi t){hd=h;tl=t;isEmpt=false;}
8
9    public String head(){return hd;}
10   public StringLi tail(){return tl;}
11   public boolean isEmpty(){return isEmpt;}
12 }
```

Schreiben Sie die folgenden Methoden für die Klasse StringLi rekursiv:

a) `public int length();` `//Länge der Liste`

b) `public StringLi take(int i);` `//Teilliste aus den ersten i Elemente`

c)
```
public StringLi drop(int i); //Teilliste ohne die ersten i Elemente
```

d)
```
public StringLi sublist(int from,int to);
    //Teilliste von den zwei angegebenen Indexstellen
```

e)
```
public StringLi oddElements();
//Teilliste aus jedem zweiten Listenelement (erste, drittes, fünftes)
```

Aufgabe 5 Jetzt sollen Sie die rekursive Datenstruktur generisch über den Elementtypen halten:

a) Verallgemeinern Sie die Klasse StringLi zu einer generischen Klassen Li, für die der Typ der Listenelemente generisch gehalten ist.

b) Benutzen Sie die generische Klasse Li für verschiedene Elementtypen.

c) Was sind primitive Typen in Java? Was ist zu beachten bei generischen Typen und primitiven Typen.