JAVA 09: Methoden für einfach verkettete Listen

panitz

Zusammenfassung

Dieser Kurs besteht aus 15 kleinen Programmieraufgaben in der Programmiersprache Java. Es sind Methoden für eine einfache verkettete Listenklasse zu entwickeln, die bis auf wenige Ausnahmen alle am einfachsten rekursiv zu lösen sind.

Frage: Länge einer Listenstruktur

Schreiben Sie für die Listenstruktur `Li` eine Längenfunktion, die die Anzahl der Elemente berechnet.

Java

```
      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;}

   int laenge(){
     return 0;
   }
}
^^I
```

Java

Lösung

Java

```
        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;}

    int laenge(){
      return isEmpty()?0:1+tail.laenge();
    }
  }
  ^^I
```

Java

Erläuterung

Man kann mit dem *Bedingungsoperator* die Fälle unterscheiden. Die leere Liste hat die Länge 0. Ansonsten ist das Ergebnis eins länger als die Länge der `tail`-Liste.

Frage: In Listenstruktur enthalten

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die testet, ob ein bestimmtes Element in der Liste enthalten ist.

Java

```
        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;}

    boolean enthaelt(A a){
      return false;
    }
  }
  ^^I
```

Java

Lösung

Java

```
        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;}

    boolean enthaelt(A a){
      if (isEmpty())return false;
      return head.equals(a)||tail.enthaelt(a);
    }

}
^^I
```

Java

Erläuterung

Fallunterscheidung mit einer *if*-Anweisung. Die leere Liste enthält kein Element. Ansonsten kann das erste Element mit dem gesuchten Element verglichen werden, oder rekursiv weiter in der Restliste gesucht werden.

Frage: Listen Aneinanderhängen

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die eine neue Liste erzeugt, indem zwei Listen aneinander gehängt werden.

Java

```
      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;}

   Li<A> append(Li<A> that){
     return Li<>();
   }
}
^^I
```

Java

Lösung

Java

```
        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;}
    @Override public String toString(){ return "Li("+head+","+tail+")";}

    Li<A> append(Li<A> that){
      if (isEmpty()) return that;
      return new Li<>(head,tail.append(that));
    }
  }
  ^^I
```

Java

Erläuterung

Fallunterscheidung mit einer *if*-Anweisung. Ist die erste Liste leer, so besteht das Ergebnis aus der zweiten Liste. Ansonsten hat das Ergebnis den head der ersten Liste und als tail, das Ergebnis des rekursiven Aufrufs des mit dem tail auf die zweite Liste.

Frage: Funktion Anwenden auf Listenstruktur

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die eine neue Liste erzeugt, indem eine Funktion auf alle Listenelemente angewendet wird.

Java

```
      import java.util.function.Function;
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> Li<B> anwenden(Function<? super A,? extends B> f){
    return Li<>();
  }
}
^^I
```

Java

Lösung

Java

```
        import java.util.function.Function;
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;}
  @Override public String toString(){ return "Li("+head+","+tail+")";}

  <B> Li<B> anwenden(Function<? super A,? extends B> f){
    if (isEmpty()) return new Li<>();
    return new Li<>(f.apply(head),tail.anwenden(f));
  }
}
^^I
```

Java

Erläuterung

Für die leere Liste ist das Ergebnis auch eine leere Liste. Ansonsten ist die als Argument übergebene Funktion auf das erste Element anzuwenden und der rekursive Aufruf auf die *tail*-Liste durchzuführen.

Frage: Letztes Listenelement

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die das letzte Listenelement selektiert. Das Ergebnis sei von der Klasse `Optional` und leer, wenn die Liste kein Element enthält.

Java

```
      import java.util.Optional;
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;}

  Optional<A> letztes(){
    return Optional.empty();
  }
}
^^I
```

Java

Lösung

Java

```
        import java.util.Optional;
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;}
  @Override public String toString(){ return "Li("+head+","+tail+")";}

  Optional<A> letztes(){
    if (isEmpty())return Optional.empty();
    if (tail.isEmpty()) return Optional.of(head);
    return tail.letztes();
  }
}
^^I
```

Java

Erläuterung

Für die leere Liste ist das Ergebnis auch ein leeres Optional. Wenn die Liste genau ein Element hat, so ist das erste auch das letzte Element. Ansonsten rekursiver Aufruf auf die Restliste.

Frage: Ersten Elemente einer Liste

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die die Teilliste aus den ersten `n` Elementen erzeugt. Ist `n` größer als die Länge der Liste, so besteht das Ergebnis aus allen Elementen.

Java

```
        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;}

    Li<A> nimm(int n){
      return new Li<>();
    }
  }
  ^^I
```

Java

Lösung

Java

```
        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;}
   @Override public String toString(){ return "Li("+head+","+tail+")";}

   Li<A> nimm(int n){
     if (isEmpty() || n==0)return new Li<>();
     return new Li<>(head,tail.nimm(n-1));
   }
}
^^I
```

Java

Erläuterung

Zwei Abbruchbedingungen lassen sich definieren. Der Fall wenn `n` gleich 0 ist, also das Ergebnis keine ELement enthalten soll und als zweite Abbruchbedingung, dass die Liste kein Element enthält. Ansonsten wird eine Liste mit dem ersten Element und dem rekursiven Aufruf erzeugt.

Frage: Ohne die ersten Elemente einer Liste

Java

```
        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;}

    Li<A> fallenlassen(int n){
      return new Li<>();
    }
  }
  ^^I
```

Java

Lösung

Java

```
        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;}
    @Override public String toString(){ return "Li("+head+","+tail+")";}

    Li<A> fallenlassen(int n){
      if (isEmpty() || n==0) return this;
      return tail.fallenlassen(n-1);
    }
  }
  ^^I
```

Java

Erläuterung

Zwei Abbruchbedingungen lassen sich definieren. Der Fall wenn `n` gleich 0 ist, also das Ergebnis die gesamte Liste sein soll und als zweite Abbruchbedingung, dass die Liste kein Element enthält. Ansonsten wird eine Liste mit dem rekursiven Aufruf erzeugt.

Frage: Ist Präfix einer Liste

Java

```
        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;}

    boolean isPrefixOf(Li<A> that){
      return false;
    }
  }
  ^^I
```

Java

Lösung

Java

```
        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;}
    @Override public String toString(){ return "Li("+head+","+tail+")";}

    boolean isPrefixOf(Li<A> that){
      if (isEmpty())return true;
      if (that.isEmpty())return false;
      return this.head.equals(that.head) && tail.isPrefixOf(that.tail);
    }
  }
  ^^I
```

Java

Erläuterung

Ist das erste Argument die leere Liste, so ist die Aussage wahr. Ist ansonsten das zweite Argument die leere Liste, so ist die Aussage falsch. Ansonsten müssen die jeweils ersten Elemete gleich sein und der rekursive Aufruf auf die beiden Restlisten wahr ergeben.

Frage: Faltung einer Liste

Java

```
        import java.util.function.BiFunction;
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, BiFunction<? super B, ? super A,? extends B> f){
    return start;
  }
}
^^I
```

Java

Lösung

Java

```
        import java.util.function.BiFunction;
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;}
  @Override public String toString(){ return "Li("+head+","+tail+")";}

  <B> B falte(B start,BiFunction<? super B, ? super A,? extends B> f){
    if (isEmpty())return start;
    return tail.falte(f.apply(start,head),f);
  }
}
^^I
```

Java

Erläuterung

Für eine leere Liste wird der Startwert als Ergebnis genommen. Ansonsten wird der Startwert durch die übergebene Operatorfunktion mit dem ersten Element verknüpft und das Ergebnis als Startwert für den rekursiven Aufruf auf die Restliste genommen.

Frage: Löschen aus einer Liste

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die eine Liste für eine Liste erzeugt, in der alle Auftreten eines bestimmten Elements gelöscht sind.

Java

```
        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;}

    Li<A> loesche(A a){
      return new Li<>();
    }
  }
  ^^I
```

Java

Lösung

Java

```
        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;}
    @Override public String toString(){ return "Li("+head+","+tail+")";}

    Li<A> loesche(A a){
      if (isEmpty())return new Li<>();
      if (head.equals(a)) return tail.loesche(a);
      return new Li<>(head,tail.loesche(a));
    }
  }
  ^^I
```

Java

Erläuterung

Aus einer leeren Liste ist nichts mehr zu löschen. Ansonsten wird das erste Element geprüft, ob es gleich dem zu löschenden Element ist. Wenn ja wird das Ergebnis aus dem rekursiven Aufruf auf die Restliste ermittelt. Wenn nein, dann kommt das erste Element noch zur Ergebnisliste hinzu.

Frage: Einfügen in eine Liste

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die eine neue Liste durch Einfügen eines Elements an einer bestimmten Position erzeugt. Die Position 0 ist das erste Element der neuen Liste. Ist die Position größer als die Länge der Liste, so wird das Element das letzte Element der neuen Ergebnisliste.

Java

```
        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;}

    Li<A> einfuegen(int i, A a){
      return new Li<>();
    }
  }
  ^^I
```

Java

Lösung

Java

```
        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;}
    @Override public String toString(){ return "Li("+head+","+tail+")";}

    Li<A> einfuegen(int i, A a){
      if (i==0|| isEmpty()) return new Li<>(a,this);
      return new Li<>(head,tail.einfuegen(i-1,a));
    }
  }
  ^^I
```

Java

Erläuterung

Terminierende Fälle sind die Position 0 und die leere Liste. Beide Fälle sind durch eigene Gleichungen in der *if*-Anweisung definiert. Ansonsten wird das erste Element für die Ergebnisliste übernommen und ein rekursiver Aufruf mit $n-1$ und der Restliste definiert.

Frage: Umdrehen einer Liste

Schreiben Sie für die Listenstruktur `Li` eine Methode, die eine neue Liste mit den Elementen der Liste in umgekehrter Reihenfolge erzeugt.

Java

```
      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;}

   Li<A> umdrehen(){
     return new Li<>();
   }
}
^^I
```

Java

Lösung

Java

```
        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;}
    @Override public String toString(){ return "Li("+head+","+tail+")";}

    Li<A> umdrehen(){
      var result = new Li<A>();
      for (var it=this;!it.isEmpty();it=it.tail)
        result = new Li<>(it.head,result);
      return result;
    }
  }
  ^^I
```

Java

Erläuterung

Es wird über die Elemente der Lister iteriert und jedes vorne an die Ergebnisliste angehängt.

Frage: Filter auf einer Listenstruktur

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die die Liste aller Elemente erzeugt, für die ein bestimmtes Prädikat wahr ergibt.

Java

```
      import java.util.function.Predicate;
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;}

  Li<A> filter(Predicate<? super A> p){
    return new Li<>();
  }
}
^^I
```

Java

Lösung

Java

```
        import java.util.function.Predicate;
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;}
  @Override public String toString(){ return "Li("+head+","+tail+")";}

  Li<A> filter(Predicate<? super A> p){
    if (isEmpty())return new Li<>();
    if (p.test(head)) return new Li<>(head,tail.filter(p));
    return tail.filter(p);
  }

}
^^I
```

Java

Erläuterung

Für eine leere Liste ist auch das Ergebnis leer. Ansonsten wird mit der Prädikatfunktion getestet, ob das erste Element in die Ergebnisliste kommt oder nicht.

Frage: Aufsplitten einer Listenstruktur nach einer Eigenschaft

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die ein Paar aus zwei Ergebnisliste erstellt. In der erste Listes des Paares sind alle Elemente der Argumentliste, für die eine Prädikatsfunktion wahr ergibt, in der zweiten alle die, für die die Prädikatsfunktion nicht wahr ergibt.

Java

```
      import java.util.function.Predicate;
interface Splitten{
 static class Pair<A,B>{
  final A e1; final B e2;
  Pair(A a,B b){e1=a;e2=b;}
 }
 static 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;}

  Pair<Li<A>,Li<A>> teilen(Predicate<? super A> p){
    return new Li<>();
  }
 }
}
^^I
```

Java

Lösung

Java

```
        import java.util.function.Predicate;
interface Splitten{
 static class Pair<A,B>{
  final A e1; final B e2; Pair(A a,B b){e1=a;e2=b;}
 }
 static 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;}

  Pair<Li<A>,Li<A>> teilen(Predicate<? super A> p){
    if (isEmpty())return new Pair<>(new Li<>(),new Li<>());
    var posneg = tail.teilen(p);
    if (p.test(head)) return new Pair<>(new Li<>(head,posneg.e1),posneg.e2);
    return new Pair<>(posneg.e1,new Li<>(head,posneg.e2));
  }
 }
}
^^I
```

Java

Erläuterung

Für die leere Liste wird das Paar von zwei leeren Listen erzeugt. Ansonsten wird der rekursive Aufruf auf die Restliste vorgenommen und geschaut, an welche der beiden Listen des Ergebnisses das erste Element vorne anzuhängen ist.

Frage: Insertion Sort

Schreiben Sie für die Listenstruktur `Li` eine Funktion, die Sortierung per Einfügen in eine Ergebnisliste vor nimmt. Schreiben Sie hierzu eine Hilfsfunktion mit akkumulierenden Ergebnisargument und eine Funktion zum Einfügen von Elementen.

Java

```
      class Li<A extends Comparable<? extends 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;}

  Li<A> sort(){
    return new Li<>();
  }
}
^^I
```

Java

Lösung

Java

```
        class Li<A extends Comparable<? super 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;}
    @Override public String toString(){ return "Li("+head+","+tail+")";}

    Li<A> sort(){
      var result = new Li<A>();
      for (Li<A> it= this;!it.isEmpty();it=it.tail)
        result = result.insert(it.head);
      return result;
    }
    private Li<A> insert(A a){
      if (isEmpty()||a.compareTo(head)<=0)
        return new Li<A>(a,this);
      return new Li<>(head,tail.insert(a));
    }
  }
  ^^I
```

Java

Erläuterung

Die Sortierfunktion iteriert über alle Listenelemente und fügt per privater Einfügefunktion die Elemente in die Ergebnisliste ein.