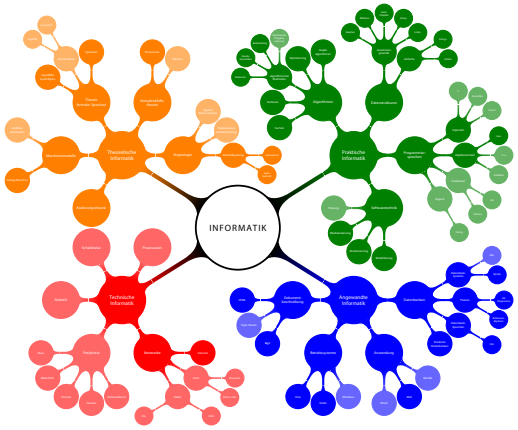
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

# Kapitel 25

# Listen – Konzepte und Erstellung

## Eine Schnitzeljagd im Rechner

Vorlesung Einführung in die Informatik 1 vom 30. Januar 2014 von Till Tantau



# Lernziele von Kapitel 25

1. Die Liste als Datenstruktur verstehen
2. Code zur Erzeugung von Listen erstellen können
3. Den Unterschied zwischen Listen und Arrays erklären können

# Gliederung von Kapitel 25

# Ein Problem aus der Bioinformatik



Beim Shotgun-Sequencing entsteht folgendes Problem:

## Problemstellung

**Eingaben** Sequenzen einer sehr großen Menge kurzer DNS-Stränge

**Ausgabe** Einzelner DNS-Strang, der alle kurzen Stränge möglichst sinnvoll erklärt.

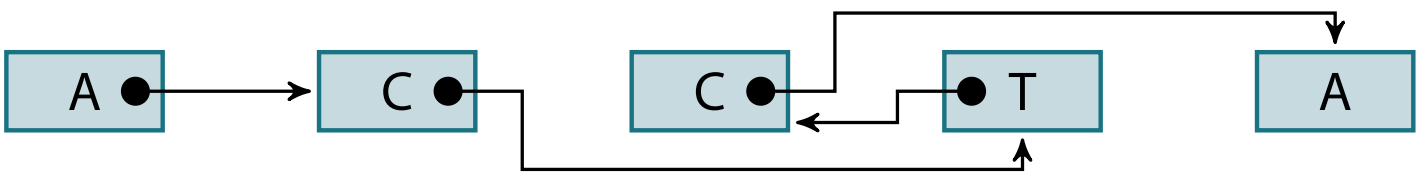
## Zur Diskussion

1. Welche Nachteile haben Strings als Darstellung von langen DNS-Sequenzen, die man »zusammenpuzzeln« möchte?
2. Was wären die idealen Eigenschaften einer Datenstruktur zur Darstellung von DNS-Sequenzen?

# Die Datenstruktur der Liste



- Die *Liste* ist eine Datenstruktur, die nach dem Vorbild eines kafkaesquen Amtes aufgebaut ist.
- Jeder Sachbearbeiter weiß über irgendwas Bescheid, für alles andere wird man zum nächsten Sachbearbeiter geschickt.
- Der nächste Sachbearbeiter sitzt in der Regel nicht nebenan, sondern irgendwo anders.



# Vor- und Nachteile von Listen gegenüber Arrays und Strings.



## Vorteile

Folgende Operationen gehen *sehr leicht* und *sehr schnell*:

+ Einfügen neuer Elemente.
+ Verketten von Listen zu neuen Listen.
+ Löschen vorhandener Elemente.
+ Ausschneiden von Teilen aus einer Liste.

## Nachteile

– Um das *i*-te Element zu finden, muss man »*i* Sachbearbeiter nacheinander aufsuchen«.
– Deshalb ist binäre Suche *nicht möglich* und man muss immer *lineare Suche* durchführen.

# Listen bestehen aus Zellen.



- Eine Liste besteht aus vielen *Zell-Objekten* (»Informatikzellen«, nicht biologische Zellen).
- Eine Zelle speichert:
  1. Einen Wert, wie zum Beispiel eine Base.
  2. Einen Verweis auf die nächste Zelle in der Liste.
- In der letzten Zelle ist der Verweis auf die nächste Zelle `null`.
- Das *Listen-Objekt* speichert lediglich einen Verweis auf die erste Zelle.

# Die Listen- und Zell-Klasse



```
// Datei DNASequence.java
class DNASequence
{
  // Attribute
  Cell start;

  // Konstruktoren
  DNASequence () {
    this.start = null;
  }
}

// Datei Cell.java
class Cell
{
  // Die Basen ('A', 'C', 'G', 'T')
  char base;

  // Nächstes Listenelement:
  Cell next;
}
```

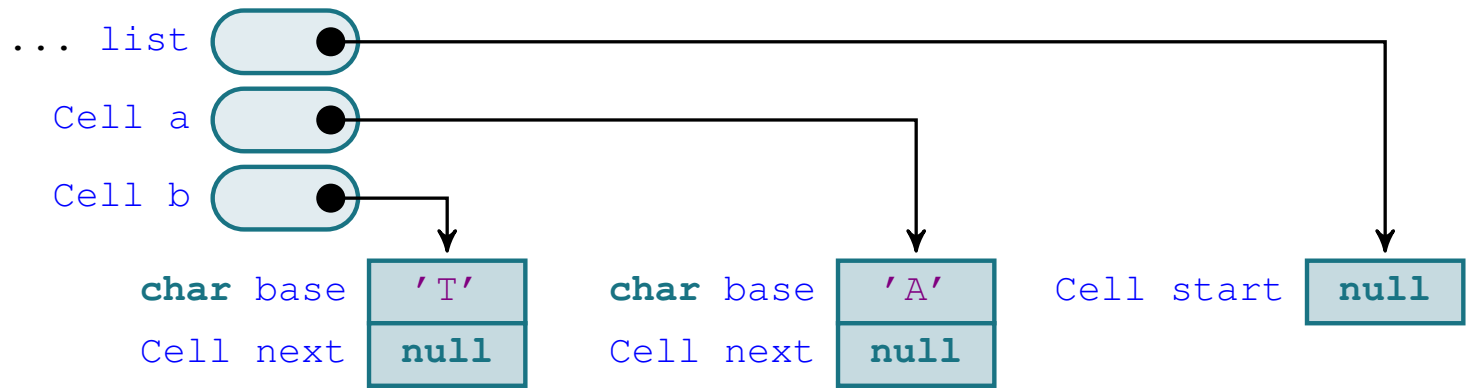
# Erzeugung einer Liste

## Schritt 1



Der folgende Code erzeugt erst ein Listenobjekt und zwei Zellobjekte. Die Zellobjekte haben schon Daten, sind aber noch nicht verkettet.

```
DNASequence list = new DNASequence ();
Cell a = new Cell ();
Cell b = new Cell ();
a.base = 'A';
b.base = 'T';
```



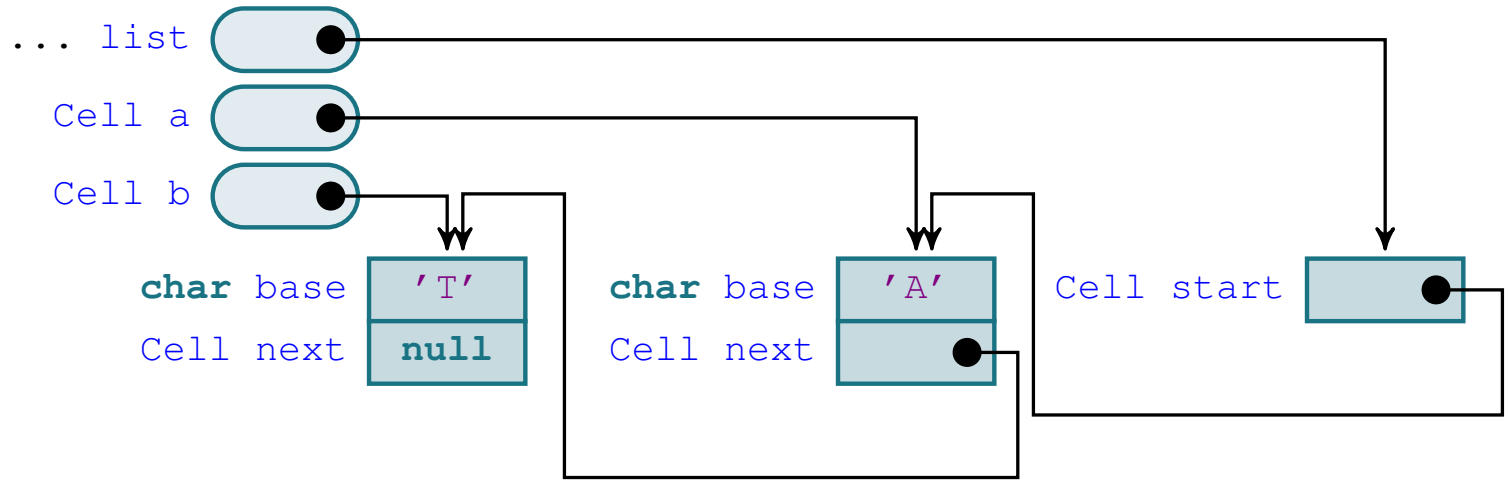
# Erzeugung einer Liste

## Schritt 2



Nun werden die Objekte verkettet.

```
...
list.start = a;
a.next = b;
```



# 5-Minuten-Aufgabe



## Zur Übung

1. Geben Sie den Code der Klassen `Studentenliste` und `Studentenzelle` zur Verwaltung einer Liste von Studenten.
2. Geben Sie weiter den Code zur Erzeugung einer Studentliste mit drei Studenten an.

# Einfügen eines neuen Elements am Anfang



## Problemstellung

Am Anfang der DNA-Sequenz soll eine neue Base angefügt werden.

## Algorithmus

1. Erzeuge eine *neue Zelle* für die neue Base.
2. Der *Nachfolger* dieser neuen Zelle ist der *Start der Liste*.
3. Setze den *Start der Liste* auf die *neue Zelle*.

Beachte:

- Dieser Algorithmus ist eine *Fähigkeit der Listen-Klasse*: Man kann eine Listen-Klasse durch ein Nachricht »bitten«, ein Element hinzuzufügen.
- Deshalb fügen wir ihn als Methode zur Klasse `DNASequence` hinzu.

# Code der Methode.



```
class DNASequence {
  // Attribute
  Cell start;

  // Konstruktor
  DNASequence () {
    this.start = null;
  }

  // Methoden
  void addBaseAtFront (char b) {
    Cell new_front = new Cell ();

    new_front.base = b;
    new_front.next = this.start;

    this.start = new_front;
  }
}
```

# 5-Minuten-Aufgabe



## Zur Übung

Visualisieren Sie wie zuvor graphisch alle Objekte und Variablen, die folgender Code erzeugt:

```
DNASequence list = new DNASequence ();

list.addBaseAtFront ('A');
list.addBaseAtFront ('C');
list.addBaseAtFront ('T');
```

# Löschen des Elements am Anfang



## Problemstellung

Das Element am Anfang der Liste soll gelöscht werden.

## Algorithmus

Ersetze `start` durch den Nachfolger von `start`.

```
class DNASequence {
  ...
  void deleteFirst () {
    this.start = this.start.next;
  }
}
```

# 5-Minuten-Aufgabe



## Zur Übung

Visualisieren Sie wie zuvor graphisch alle Objekte und Variablen, die folgender Code erzeugt:

```
DNASequence list = new DNASequence ();

list.addBaseAtFront ('A');
list.addBaseAtFront ('C');
list.addBaseAtFront ('T');
list.deleteFirst ();
list.deleteFirst ();
```

# Zusammenfassung



## Listen versus Arrays

- In einem Array werden Elemente schön »hinteinander weg« gespeichert (Modell Reihenhaus). Der Zugriff auf das *i*te Element ist schnell, Löschen und Einfügen sind langsam.
- In einer Liste wissen Elemente immer nur, wo das nächste Element ist (Modell Zeltplatz). Der Zugriff auf das *i*te Element ist langsam, Löschen und Einfügen sind schnell.

## Zwei Klassen pro Listenart

- Man benötigt *zwei Klassen* um Listen einer Art zu modellieren:
- Zunächst braucht man »Zellen«, die die eigentlichen Daten enthalten und immer einen Verweis auf die nächste Zelle.
- Als Zweites braucht man die »eigentliche Listenklasse«, die einen Verweis auf den Anfang speichert. *Dieser* Klasse schickt man alle Nachrichten.