5.

Java: Pakete

Buch Mark Weiss „Data Structures & Problem Solving Using Java“ siehe:
- 126-131 (Pakete)
- 635-639 (Stack als Liste)
- 122-126 („BigRational class“)

Lernziele Kapitel 5 Pakete in Java

- Die Rolle von Java-Standardpaketen verstehen
- Eigene Pakete aus einer Menge zusammengehöriger Klassen erstellen können
- Zugriffskontrolle und Schutzattribute „private“, „public“ verstehen

Thema / Inhalt

Pakete stellen in Java eine nützliche Strukturierungsmöglichkeit dar, um eine Menge inhaltlich zusammengehöriger Klassen zu einem Modul zusammenzufassen. Durch die Nutzung (das „Importieren“) solcher Module, die von Experten erstellt wurden und deren Implementierung man nicht zu kennen braucht, kann quasi die Funktionalität der Sprache erweitert werden: Nur funktional relevante Teile werden von den Paketentwicklern für die Nutzer zugreifbar („öffentlich“) gemacht, der Rest bleibt verborgen („privat“).

Wir erläutern dies an zwei Beispielen: Zum einen schnüren wir ein Paket, welches einfache Funktionalität rund um eine **verkettete Liste** anbietet, und nutzen dies dann in einer Anwenderrolle, um damit einen potentiell unbegrenzten Stack zu realisieren. Zum andern definieren wir ein Paket, das einem Anwender das **Rechnen mit Brüchen** ermöglicht. Im Unterschied zu rationalen Zahlen hat man dann, wenn Zähler und Nenner eines Bruches explizit als ganze Zahlen gespeichert werden, nicht mit Rundungsproblemen zu kämpfen. Das Bruch-Paket bietet nicht nur die Addition und Multiplikation von Brüchen an, sondern beispielsweise auch

Thema / Inhalt (2)

Möglichkeiten zum Kürzen und zum Vergleich von Brüchen. Dazu müssen wir bei der Definition des Pakets die in der Schule gelernte Vorgehensweise beim Bruchrechnen implementieren – insbesondere den Hauptnenner, den grössten gemeinsamen Teiler (ggT) sowie das kleinste gemeinsame Vielfache (**kgV**) bilden.

In unserem optionalen Teil zu Kontext und Geschichte der Konzepte gehen wir in diesem Kapitel genauer auf den **euklidischen Algorithmus** ein, mit dem sich der ggT bestimmen lässt, ohne die aufwändigere Primfaktorzerlegung bemühen zu müssen. Der euklidische Algorithmus ist einer der ältesten nicht-trivialen Algorithmen, er hat eine faszinierende Geschichte und gleichzeitig auch interessante mathematische Eigenschaften.

Historisch wurde der euklidische Algorithmus nicht algebraisch formuliert, sondern in der Tradition der Pythagoreer geometrisch verstanden – die beiden Parameter a und b stellen dabei zwei Streckenlängen dar, am einfachsten stellt man sich diese als die Länge und Breite eines Rechtecks vor. Man suchte nach dem gemeinsamen Mass der beiden Strecken a und b – eine Länge, die in beiden Streckenlängen ganzzahlig enthalten ist. Der Algorithmus zur Bestimmung dieses gemeinsamen Masses, das dann beide Strecken ohne Rest teilt, beruht auf dem Prinzip der „**Wechselwegnahme**" (Antiphairesis) und lautet im Kern bei Euklid so:

„Nimm immer die kleinere Zahl von der grösseren weg, bis ein Rest kommt, welcher die nächstvorgehende Zahl genau misst. Dieser Rest ist das grösste gemeinschaftliche Mass der beiden gegebenen Zahlen."

Im „schlimmsten Fall" ist das Ergebnis 1, dann sind die beiden Zahlen teilerfremd. Dem Verfahren liegt wieder eine Schleifeninvariante zugrunde: Durch die Subtraktion $a = a - b$ bzw. $b = b - a$ ändert sich die Menge der gemeinsamen Teiler von a und b, also auch der ggT, nicht.

Thema / Inhalt (3)

Man kommt also auf eine immer einfachere Aufgabe zur Bestimmung des ggT, bis man bei der Subtraktion 0 erhalten würde, man also zwei gleiche Zahlen hat, von denen der ggT zu bestimmen wäre – von zwei gleichen Zahlen x ist der ggT(x,x) jedoch x – man ist also fertig und hat aufgrund der Invarianten damit, d.h. mit x, den ggT der beiden ursprünglichen Parameter bestimmt.

Das **kleinste gemeinsame Vielfache (**kgV) steht in einer einfachen Beziehung zum ggT: Man nehme das Produkt von a und b und dividiere es durch ggT(a,b) – dann erhält man das kgV von a und b. Wir lernen aber auch eine leichte Erweiterung des euklidischen Algorithmus kennen, mit der sich ggT und kgV zweier Zahlen simultan berechnen lassen.

Die fortgesetzte Subtraktion ist langwierig, wenn die eine Ausgangszahl a viel grösser als die andere Zahl b ist – dann subtrahiert man recht oft, bis es zu einem Wechsel kommt. Das kann man einfach abkürzen, indem man die eine Zahl durch die andere ganzzahlig dividiert. Dabei sind wir gar nicht am Ganzzahlquotienten interessiert, sondern am Rest, der bleibt, wenn man ganzzahlig dividiert (also subtrahiert hat, sooft es eben ging) – dieser Rest wird typischerweise mit „mod (a,b)" bezeichnet. Damit ergibt sich die modernere und kürzere Form des euklidischen Algorithmus: Es wird fortwährend **(a, b) = (b, mod(a,b))** berechnet, bis b Null wird; dann ist a der gesuchte ggT. Wenn die Ausgangszahlen zwei benachbarte **Fibonacci-Zahlen** sind, dauert es besonders lang, weil sich dann als Rest stets die nächstkleinere Fibonacci-Zahl ergibt – dennoch bleibt die Schrittzahl logarithmisch zur Eingabe: Gabriel Lamé zeigte schon 1844, dass nie mehr Schritte als das Fünffache der Stellenzahl der kleineren der beiden Ausgangszahlen benötigt wird – ein Resultat, das über 100 Jahre vor der Etablierung **Komplexitätstheorie** erzielt wurde, welche die systematische Beschäftigung der Mathematik und Informatik mit dem Aufwand von Algorithmen begründete.

Pakete in Java

- Paket = (zusammengehörige) Menge von Klassen
  - Sowie Unterpakete und Interfaces
- Hierarchischer Aufbau
  - Paket „xyz“ im Paket „java“ → „java.xyz“
- Relevant für Strukturierung und Zugriffskontrolle
  - Klassen sowie deren Methoden und Attribute sind (ohne Angabe von **`public`**) nur im eigenen Paket sichtbar und zugreifbar
- Klassen befinden sich (logisch) immer in Paketen
  - Paketdeklaration direkt am Anfang einer Quelldatei, z.B.

```
package abc;
```

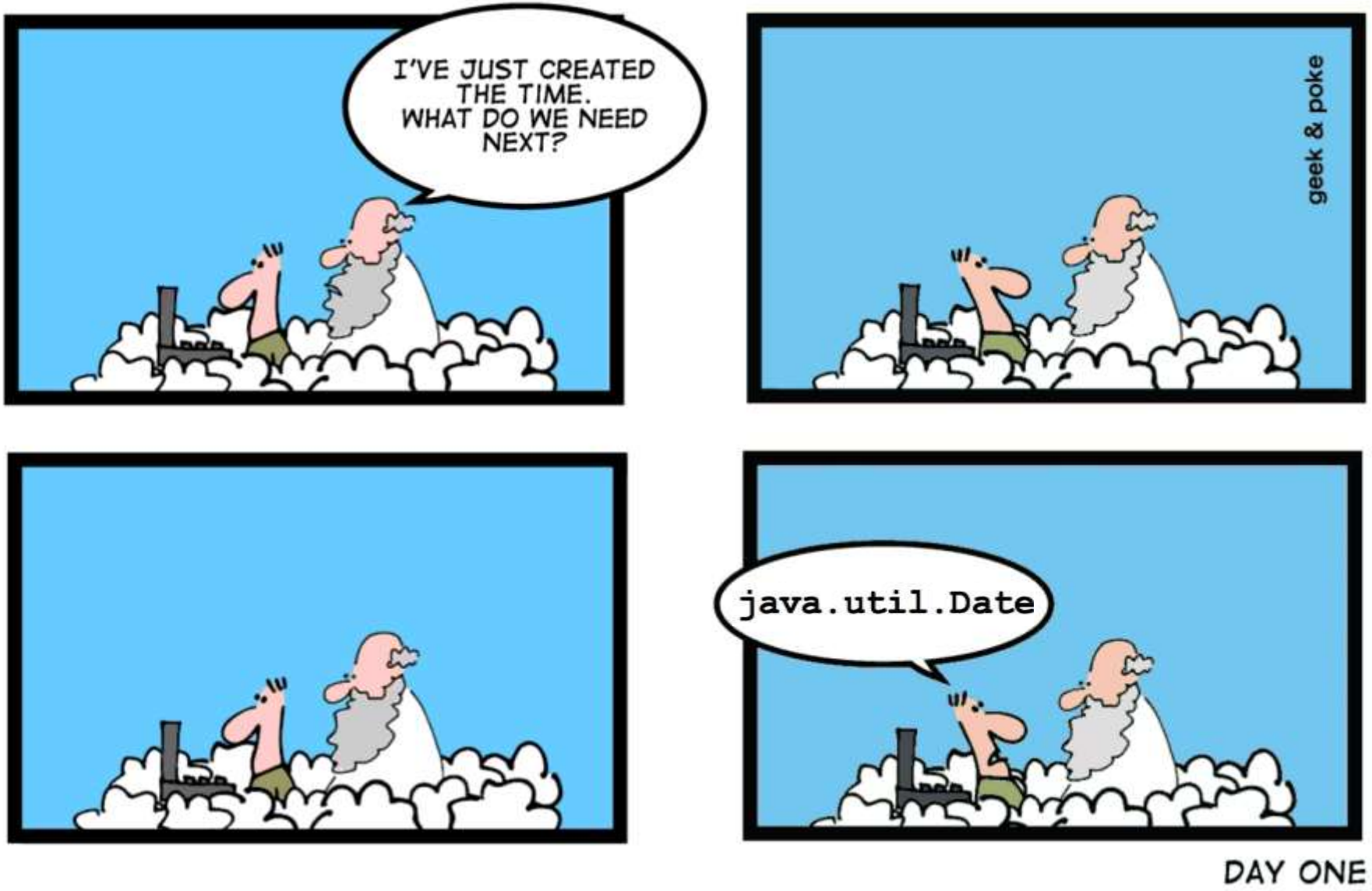
  - Falls package-Deklaration fehlt: „unnamed package“
  - Konvention: kleingeschriebene Namen

Pakete in Java (2)

- Attribute bzw. Methoden von Klassen können vollqualifiziert (d.h. mit dem Paketnamen) benannt werden
  - Z.B.: `java.lang.String.substring`

    Paket Klasse Methode
- Importieren von Klassen (als Namensabkürzung) aus Paketen
  - Z.B. `import java.util.Random;`
    (es wird diese Klasse importiert; kann mit Name „Random" benutzt werden)
  - Oder `import java.util.*;`
    (es wird alles aus diesem Paket „java.util" importiert)
- Das Java-System bietet eine Reihe von Standardpaketen mit nützlichen Klassen und Methoden für fast jeden Zweck an
  - Z.B. java.lang, java.io, java.net, java.applet, java.util,...

THE GOD AND THE CODER



Eigene Pakete

Beispiel.java:

```
import myPack.*;
class Beispiel {
 ...main...{
    Stack S;
     ...
    Queue Q;
 }
}
```

Alternativ zu „import“: jew. vollqualifizierte Angabe, also z.B. `myPack.Queue`

Stack.java:

```
package myPack;
public class Stack
{...}
class Hilfselement
{ void setzen()
  {...}
}
```

Queue.java:

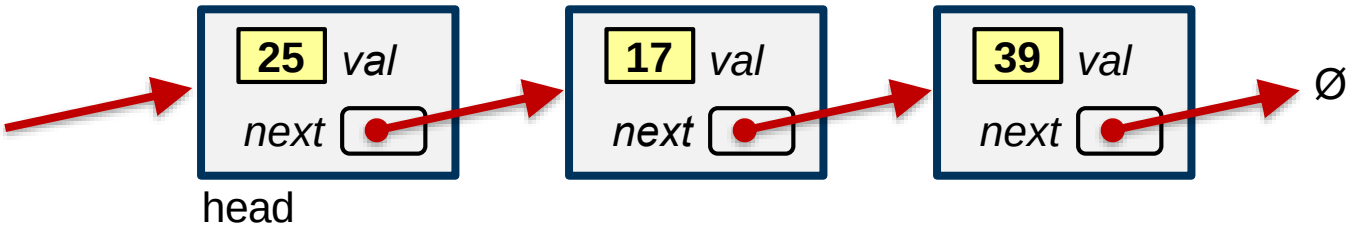
```
package myPack;
public class Queue
{...}
class x
{ private void y()
  {...}
}
```

Gemeinsames Paket (aufgeteilt in getrennte Dateien)

- Die Klasse Stack in der Datei Stack.java (bzw. Queue in Queue.java) muss mit dem Zugriffsmodifikator „public“ qualifiziert werden, da diese Klassen ausserhalb des Paketes MyPack (in Beispiel.java) verwendet werden sollen
  - Entsprechendes würde für öffentl. Methoden / Attribute von Stack und Queue gelten
- Die Methode „setzen“ ist innerhalb von `myPack` (also z.B. von Queue aus) zugreifbar, nicht aber von ausserhalb des Paketes (z.B. von Beispiel aus)
- y ist nur von Methoden der Klasse x aus zugreifbar (da „private“)

Beispiel 1: Ein Paket für int-Listen

- Es geht um verkettete Listen der folgenden Art:



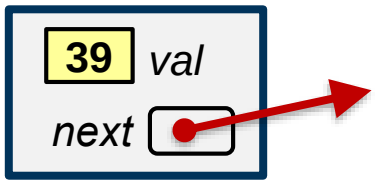
- Nutzer unseres Paketes sollen „Listenelemente“ am Anfang (am Listenkopf: „head“) hinzufügen und entfernen können
  - add_head, remove_head

- Ausserdem sollen sie weitere nützliche Methoden bekommen:
  - Z.B. wie viele Elemente die Liste enthält (size)

Beispiel 1: Ein Paket für int-Listen (2)

```
package listPack;

class ListElem {
  int val;
  ListElem next;

  public ListElem(int i, ListElem e) {
    val = i;
    next = e;
  }
}


public class List {
  ...
}
```

ListElem ist ausserhalb des Paketes nicht sichtbar



Ein Objekt als Instanz der Klasse ListElem

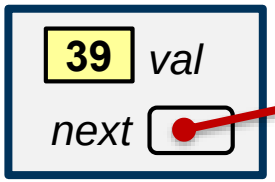
Konstruktor

Hierauf soll das neue Listenelement mit „next" zeigen

Fortsetzung des Paketes „listPack" auf der nächsten Slide →

```
public class List {
  private ListElem first = null;
  private int size = 0;

  public int size() {return size;}

  public void add_head(int i) {
    first = new ListElem(i,first);
    size++;
  }

  public int remove_head() {
    int i = first.val;
    first = first.next;
    size--;
    return i;
  }

  // evtl. weitere
  // Methoden
}
```

Ein Objekt als Instanz der Klasse ListElem



Kein Zugriff von aussen

Methode liefert den Wert der gleichnamigen privaten Variablen

Neues Element vorne hinzufügen (einketten)

Referenz auf das bisher 1. Element

Grösse auf privater Variablen mithalten

Ausketten

Wert des (ehemaligen) vordersten Elements zurückliefern

Für Nutzer ist die Komplexität reduziert; sie sehen nicht die verkettete Struktur aus Referenzen und Objekten

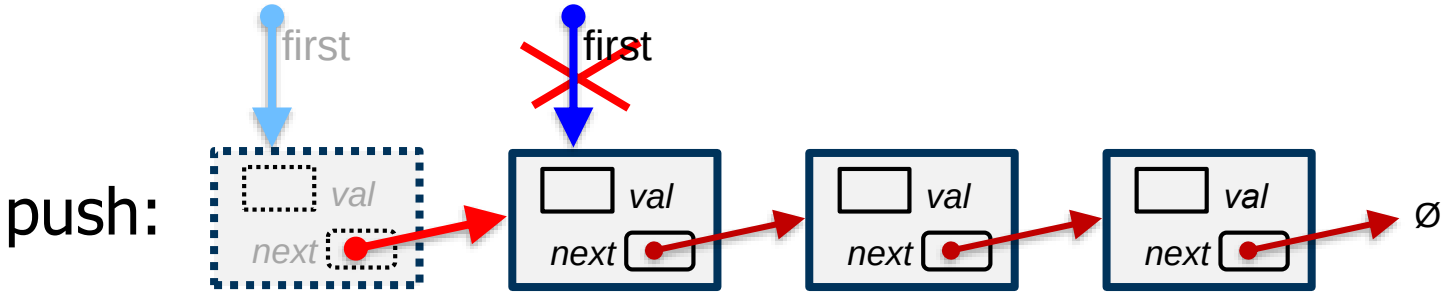
```
public class List

  public int size()
  public void add_head(int i)

  public int remove_head()
```

Für Nutzer ist die Komplexität reduziert; sie sehen nicht die verkettete Struktur aus Referenzen und Objekten

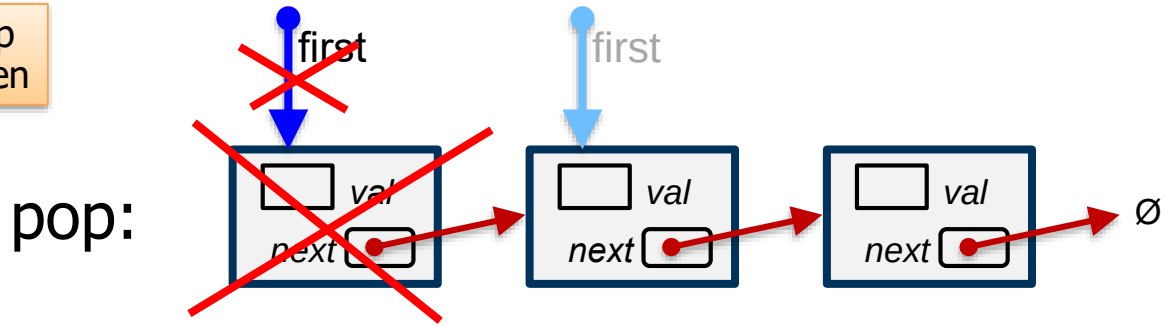
Ein Stack aus einer int-Liste

Prinzip ist aus Teil I der Vorlesung bekannt

- Unter Verwendung realisierter Datentypen (z.B. *List*) können neue Datentypen konstruiert werden
  - z.B. ein Stack mit *push*, *pop* (hier ohne Wertrückgabe) und *top*
- Interne Realisierung von push bzw. pop (nach aussen nicht sichtbar):



Stack kann im Prinzip „beliebig" gross werden



Nutzung des Listen-Pakets

für die Realisierung eine Stacks

```
// Stack aus einer int-Liste
import listPack.List;
public class Stack {
  private List L;
  public Stack() {
    L = new List();
  }
  public boolean empty() {
    return (L.size() == 0);
  }
  public void push(int i) {
    L.add_head(i);
  }
  public void pop() {
    // Test, ob !empty()...
    L.remove_head();
  }
  public int top() {
    // Test, ob !empty()...
    int i = L.remove_head();
    L.add_head(i);
    return i;
  }
}
```

```
public class List {
  public int size()
  public void add_head(int i)
  public int remove_head()
```

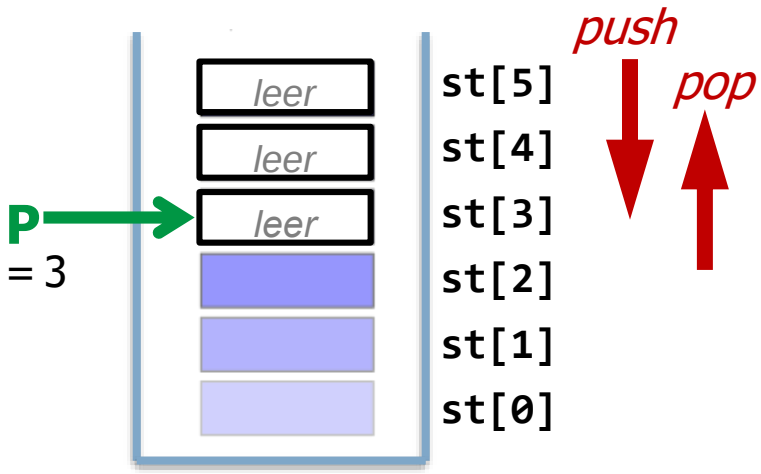
Im Konstruktor des Stacks wird eine (leere) Liste erzeugt

Der zurückgelieferte Wert wird einfach ignoriert

Da List keinen direkten Zugriff auf die Listenelemente gestattet und die Werte nicht direkt ermittelt werden können, verwenden wir diesen Trick

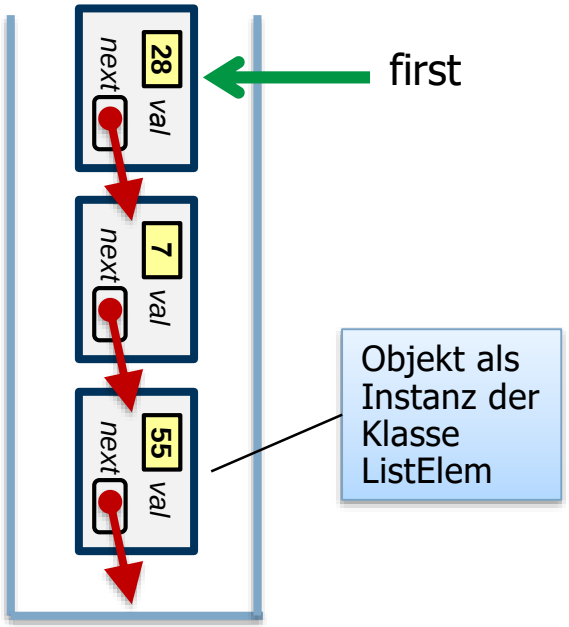
Stack: Implementierungsmöglichkeiten

Wir wissen bereits von früher, wie ein Stack mit einem Array implementiert werden kann:



Array-Implementierung: effizient, aber der Stack ist a priori begrenzt.

Dagegen Implementierung mit verketteter Liste wie eben:



Der Stack ist so potentiell unbegrenzt, wegen „new" jedoch etwas aufwendiger (langsamer) und hat Speicher-Overhead durch die next-Referenzen.

Austausch des Dienstleistungsanbieters

- Eine Klasse als Dienstleistungsanbieter kann „rücksichtslos“ gegen eine mit gleicher Schnittstelle (public-Variablen und -Methoden) und gleicher externer Wirkung ausgetauscht werden
  - Aber kann ein Array-basierter Stack (feste Maximalgrösse!) wirklich exakt das gleiche Verhalten aufweisen wie ein listenbasierter Stack?
  - → Was genau versteht man unter (funktionalem) „Verhalten“?

Noch ein Beispiel für eigene Pakete:

Paket „Bruchrechnen“

$$\text{Bruch} = \frac{\text{Zähler}}{\text{Nenner}}$$

- Mit Zähler $\in \mathbb{Z}$ und Nenner $\in \mathbb{Z}$ ist Bruch $\in \mathbb{Q}$
  - Strukturell ist ein Bruch allerdings $\in \mathbb{Z} \times \mathbb{Z}$
  - Wir beschränken uns hier auf Werte ≥ 0

- Wir greifen nachfolgend aus „Informatik I“ das Beispiel „Rechnen mit rationalen Zahlen“ (modifiziert) wieder auf
  - Beachte aber: In Java gibt es im Unterschied zu C++ keine Operator-Überladung; man schreibt daher etwas ungewohnter

    `c = a.mult(b)` statt der Infix-Notation `c = a * b` oder z.B.:

    ```
    c = a.plus(new Bruch(1,6))
    ```

„Ein Bruch ist ein Vielfaches von einem Teile der Einheit. Zu einem Bruche braucht man daher immer zwei Zahlen, die eine Zahl, die andeutet in wie viele Teile die Einheit geteilt wurde, und nach welcher man diese Teile benennet, und welche Zahl daher Nenner heisst, und die zweite Zahl, welche angibt, wie viele solche Teile zusammengezählt werden, daher Zähler heisst, und schreibt einen Bruch auf die Art, dass man den Zähler über den Nenner anbringt, und beide durch einen Querstrich trennet.“ [Leopold Carl Schulz von Straßnicki: Handbuch der besonderen und allgemeinen Arithmetik, 1844]

lat.: numerator

lat.: denominator

...ober der zeler, unter der neñer

Bruch = Zähler / Nenner

Aus dem ersten gedruckten deutschsprachigen Lehrbuch zur Algebra *Behend unnd Hübsch Rechnung durch die kunstreichen regeln Algebre, so gemeincklich die Coss* genennt werden* von Christoff Rudolff (Strassburg, 1525):

Prüch ſein zweierley / etlich heiſſen ſchlechte prüch / etlich prüch von prüch. Ein ſchlechter pruch würt geſchriben mit zweien zalen vnnd mit einem ſtrichlen darzwiſchen / heiſt die ober der zeler / die vnter der neñer. In außſprechung müß man zum erſten beſtimme den zeler / darnach den nenner mit zůſetzung des wörtlens teil / als $\frac{2}{5}$ iſt der zeler 2 / der nenner 5. würt außgeſprochen zwei fünfftteil. dann prüch ſein nichts dañ teil eins andern dings / Nemlich der nenner zeigt an in wieuil teil das gantz brochen ſei / derſelben etlich zelt die ober zal.

*) „Die Coss“: Bei al-Chwarizmi und anderen arabischen Mathematikern wurde die Unbekannte (also die „Variable x“) als „die Sache“ bezeichnet; die Kunst des Auflösens nach x hiess im Italienischen seinerzeit dann „l'arte della cosa“.

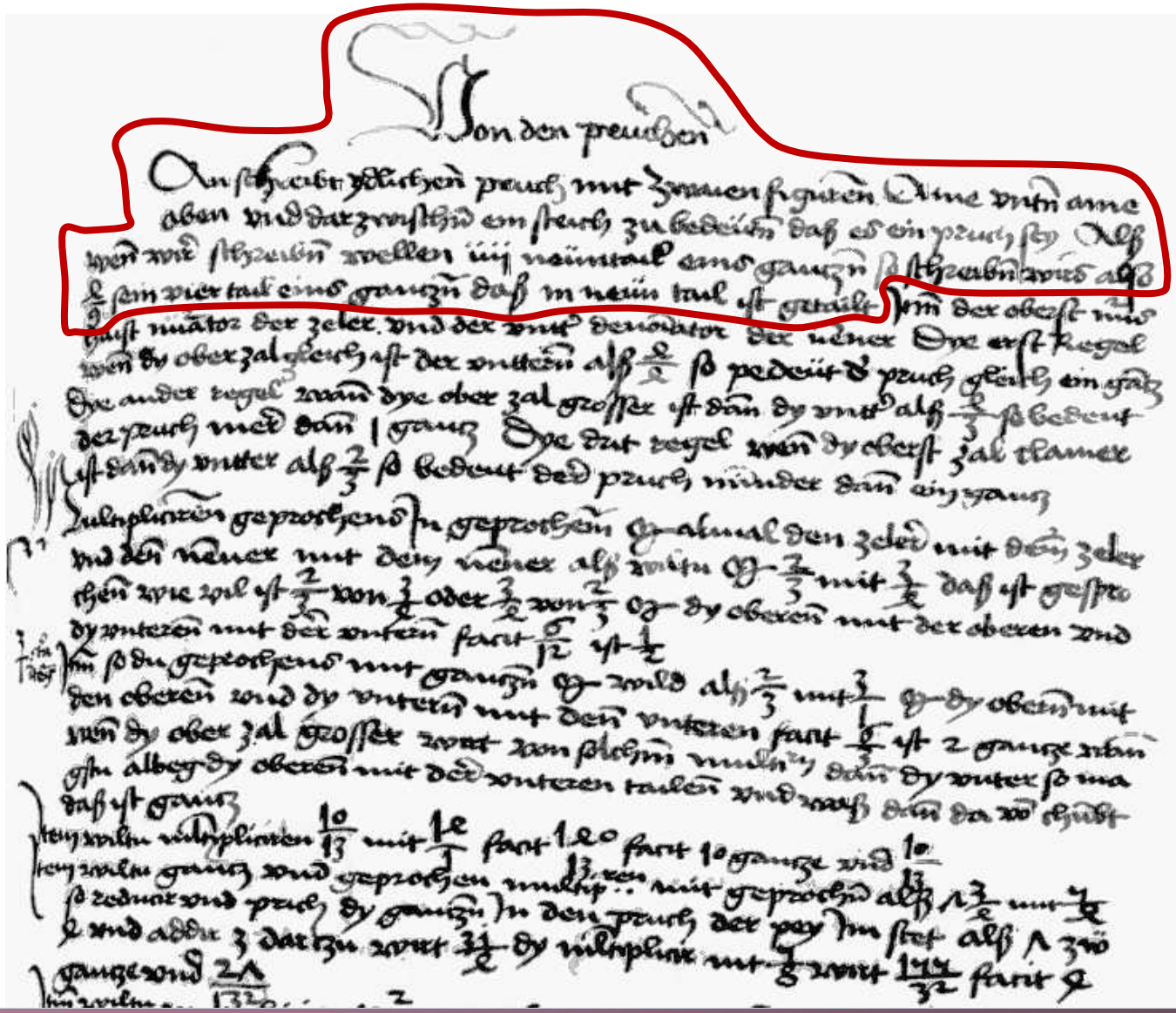
Christoff Rudolff

„Es hat Christoff Rudolff vom Jawer (löblicher ge= dechtnis) anno 1524 die wunderbarliche und gantz Philosophische Kunst dess rechnens, genennet Die Coss, in deutsche sprach durch den Truck gebracht, so gantz getrewlich und so klar und deutlich, das ich die selbige Kunst ohn allen mündtlichen underricht verstanden hab (mit Gottes hülff) und gelernet.“ [Michael Stifel, 1553]

Zwaien figurenn – Aine vntnn aine oben

Vor der Erfindung des Buchrucks durch Johannes Gutenberg (um 1450) gab es Schriften nur als Manuskripte. Eine erste weitgehend auf Deutsch gehaltene Sammlung mathematischer Handschriften verfasste zwischen 1450 und 1465 Fridericus Amann, ein Mönch aus Regensburg. Der Teil zum Bruchrechnen beginnt so:

Von den pruchen
Man schreibt ydlichenn pruch mit Zwaien figurenn Aine vntnn aine oben vnd darzwischnn ein strich zu bedeutnn daß es ein pruch sey Alß wenn wirr schrei= benn wellen iiii neüntail eins gantznn so schreibnn wirs also $^4/_9$ sein vier tail eins gantznn dass in neün tail ist getailt“



Paket „Bruchrechnen“

$$\text{Bruch} = \frac{\text{Zähler}}{\text{Nenner}}$$

- „Tools“ für das Rechnen mit rationalen Zahlen in Bruchform
  - Vermeidet so gegenüber Gleitpunktzahlen mögliche Rundungsfehler
  - Hier: *long* `statt` *int* `und` *double* statt *float* → grösserer Wertebereich

```
package bruchPak;

public class Bruch {
  private long Zaehler = 0;
  private long Nenner  = 1;

  public void setzen(long z, long n) {
    if (z < 0 || n <= 0)
      System.out.println("*** Fehler Wertebereich");
    else {
      Zaehler = z; Nenner = n; kuerzen();
    }
  }

  public Bruch(long z, long n) {setzen(z,n);}
  public Bruch(long z) {setzen(z,1);}
  public Bruch() {setzen(0,1);}

  // Fortsetzung class Bruch nächste Seite
```

Lokale Daten sind 'private' und geeignet vorbesetzt

Professionellere Fehlerbehand-lung mit excep-tions später

Gleich in gekürzter Form abspeichern – Methode „kuerzen()“ kommt gleich

Drei überladene Konstruktoren

Könnte man sich das nicht sparen?

Paket „Bruchrechnen“ (2)

Bruch = Zähler / Nenner

```
public double ratio() {
  return (double)Zaehler / (double)Nenner;
}

public void ausgeben() {
  System.out.println(Zaehler + "/" + Nenner +
                     " = " + ratio());
}

public boolean kleiner(Bruch y) {
  return (Zaehler*y.Nenner < y.Zaehler*Nenner);
}

public boolean identisch(Bruch y) {
  return (!this.kleiner(y) && !y.kleiner(this));
  // Bei gekuerzten Bruechen geht auch ein jeweili-
  // ger Vergleich mit "==" von Zaehler und Nenner
}

private static long ggT(long u, long v) {...}

private static long kgV(long u, long v) {...}
```

Explizite Typumwandlung ("type cast")

Z.B.: 3/4 = 0.75

Vorsicht bzgl. Überschreitung des Wertebereichs

Denkübung: Wie wäre es mit ratio() < y.ratio() ?

Algorithmen für ggT bzw. kgV folgen (→ euklidischer Algor.)

ggt (grösster gemeinsamer Teiler) und kgV (kleinstes gemeinsames Vielfaches) sind hier in Form einer klassenbezogenen Funktionalität („static“) realisiert

Paket „Bruchrechnen“ (3)

Bruch = Zähler / Nenner

```
  public Bruch plus(Bruch y) {
    long n = kgV(Nenner, y.Nenner); // Hauptnenner
    long z = (n/Nenner)*Zaehler + (n/y.Nenner)*y.Zaehler;
    return new Bruch(z,n); // Wird vom Konstruktor gekürzt
  }

  public Bruch mult(Bruch y) {
   return new Bruch(Zaehler*y.Zaehler,Nenner*y.Nenner);
  }

  private void kuerzen() {
    if (Zaehler == 0)
      Nenner = 1;
    else {
      long g = ggT(Zaehler,Nenner);
      if (g != 1) {
         Zaehler = Zaehler/g;
         Nenner = Nenner/g;
      }
    }
  }
} // Ende der Klasse Bruch
```

Das Additionsergebnis ist ein *neuer* Bruch

Durch die Multiplikationen kann es leicht zu einem Überlauf kommen → besser: Faktoren frühzeitig kürzen, um das Problem zu minimieren.

„kuerzen“ ist eine von aussen nicht zugängliche Methode, die den Bruch in eine Normalform bringt

```
import bruchPak; // Anwendung:
  Bruch a = new Bruch();
  Bruch b = new Bruch(1,2);
  Bruch c = a.plus(b.mult(a));
  c.ausgeben();
  (a.plus(b)).ausgeben();
  c = c.plus(new Bruch(1,8));
```

Nutzung des Pakets zum Bruchrechnen

Als Übung

```
import bruchPak;
...
  Bruch a = new Bruch();
  Bruch b = new Bruch(1,2);
  Bruch c = new Bruch(5);

  a.ausgeben();
  a.setzen(2,3);
  a.ausgeben();
  b.ausgeben();
  c.ausgeben();

  c = a.mult(b);
  c.ausgeben();

  (a.plus(b)).ausgeben();

  c = a.plus(b.plus(a));
  c.ausgeben();

  c = c.plus
      (new Bruch(1,8));
  c.ausgeben();
```

Zur Erinnerung – die Konstruktoren:

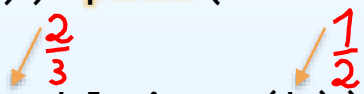
```
public Bruch(long z, long n)
   { setzen(z,n); }
public Bruch(long z)
   { setzen(z,1); }
public Bruch() { setzen(0,1); }
```

```
0/1 = 0

2/3 = 0.666667
1/2 = 0.5
5/1 = 5

1/3 = 0.333333
7/6 = 1.16667

11/6 = 1.83333

47/24 = 1.95833
```

Nutzung des Pakets zum Bruchrechnen (2)

Als Übung

```
c = (new Bruch(7,12)).plus(new Bruch(1,6));
c.ausgeben();

System.out.println(a.kleiner(b));
System.out.println(b.kleiner(a));
c = a;
System.out.println(a.identisch(c));
System.out.println(a.identisch(b));

a.setzen(0,3);
b.setzen(0,4);
c = a.plus(b);
c.ausgeben();
c.setzen(3,0);
```




```
3/4 = 0.75

false
true

true
false

0/1 = 0
*** Fehler Wertebereich
```

- Die interne Repräsentation von Zähler und Nenner durch „long“ ist beschränkt; man könnte sich andere Realisierungen vorstellen; z.B. bei der die einzelnen Ziffern von Zähler und Nenner in (potentiell beliebig langen) verketteten Listen gehalten werden.
- Analoge Pakete auch für andere mathem. Objekte, z.B. Polynome, Matrizen...

Euklidischer Algorithmus für ggT (und kgV)

Einer der ältesten bekannten nicht-trivialen Algorithmen; wurde schon *vor* Euklid (~ 300 v. Chr.) verwendet

```
// Iterative Version mit
// fortgesetzter Subtraktion
public int ggT(int a, int b)
{ while (a != b)
  { if (a > b) a = a - b;
    else       b = b - a;
  }
  return a;
}
```

```
// Rekursive Version mittels
// Division mit Rest, spart
// viele Subtraktionen
public int ggT(int a, int b)
{ if (b == 0) return a;
  else return ggT(b, a%b);
} // aus Informatik I bekannt
```

Wechselwegnahme

„Nimm immer die kleinere von der grösseren weg, bis ein Rest kommt, welcher die nächstvorgehende Zahl genau misst. Dieser Rest ist das grösste gemeinschaftliche Mass der beiden gegebenen Zahlen.“ [Euklid]

Mit dem ggT eng verwandt ist das kgV; es gilt die Beziehung **a × b = ggT(a,b) × kgV(a,b)**.

Frage zur fortgesetzten Subtraktion bei einer Prüfung zum General Certificate of Secondary Education:
Q: How many times can you subtract 7 from 83, and what is left afterwards?
A: I can subtract it as many times as I want, and it leaves 76 every time.

Euklidischer Algorithmus für ggT

Berechnung des ggT nach dem euklidischen Algorithmus – hierbei werden keine aufwändig zu berechnende Primfaktorzerlegungen (Schulmethode zu Bestimmung des ggT!) benötigt; es geht mit dem Algorithmus viel einfacher! Das Verfahren beruht im Wesentlichen darauf, dass die Menge der Teiler zweier Zahlen a, b (mit $a \geq b > 0$) identisch zur Menge der Teiler der beiden Zahlen a, a – b ist. Dies ist zugleich die Idee für die Schleifeninvariante zur Verifikation der Implementierungen oben. Die iterative Version (Prinzip des wechselseitigen Subtrahierens, sogen. Wechselwegnahme) ist über 2000 Jahre alt. Die modernere Divisionsversion ist effizienter („modus brevis", erstmals beschrieben 1494 von Luca Pacioli, sie erspart viele Subtraktionen a = a – b im Falle a >> b): Da a%b < a/2 (Beweis?), werden die Zahlen schrittweise exponentiell kleiner. Der modus brevis beruht auf der mathematischen Tatsache ggT(a, b) = ggT(b, a – *k* b) für jede ganze Zahl *k*.

Die Schleifenabbruchbedingung „a != b" lässt sich auch zu „$b \neq 0$" oder „$b > 0$" abändern (man beachte, dass wir hier keine negativen Zahlen betrachten); der Kern des Algorithmus wird daher in Form von mathematischem Pseudocode oft auch so notiert:

while $b > 0$ **do**:
  $(a, b) = \left(b, a - \left\lfloor \frac{a}{b} \right\rfloor b\right)$
return a

Euklidischer Algorithmus für ggT – Beispiel

```
(1)   121393 % 75025 = 46368
(2)   75025 % 46368  = 28657
(3)   46368 % 28657  = 17711
(4)   28657 % 17711  = 10946
(5)   17711 % 10946  = 6765
(6)   10946 % 6765   = 4181
(7)   6765 % 4181    = 2584
(8)   4181 % 2584    = 1597
(9)   2584 % 1597    = 987
(10)  1597 % 987     = 610
(11)  987 % 610      = 377
(12)  610 % 377      = 233
(13)  377 % 233      = 144
(14)  233 % 144      = 89
(15)  144 % 89       = 55
(16)  89 % 55        = 34
(17)  55 % 34        = 21
(18)  34 % 21        = 13
(19)  21 % 13        = 8
(20)  13 % 8         = 5
(21)  8 % 5          = 3
(22)  5 % 3          = 2
(23)  3 % 2          = 1
(24)  ggt = 1
```

Trace des Euklidischen Algorithmus nach Aufruf mit den benachbarten Fibonacci-Zahlen 121393 und 75025.

Aufeinanderfolgende Fibonacci-Zahlen (oder ein gemeinsames Vielfaches aufeinanderfolgender Fibonacci-Zahlen) für a, b benötigen (relativ zu Zahlen ähnlicher Grösse) besonders lang bei der Divisionsversion.

Eine solche Eingabe produziert auch (gleichermassen für die Divisons- wie für die Subtraktionsmethode) hinsichtlich der Zwischenresultate einen Präfix der Fibonacci-Folge.

Man beachte bei diesem Beispiel, dass 75025 fünf Stellen hat; nach dem Theorem von Lamé (→ spätere Slides) gibt es daher nicht mehr als fünf Mal so viele, also 25, Iterationen.

Euklidischer Algorithmus und Fibonacci-Zahlen

Schon der deutsche Rechenmeister Simon Jacob (1510 – 1564) erkannte, dass die Fibonacci-Zahlen (die damals allerdings noch nicht so genannt wurden) besonders viele Rechenschritte bei der ggt-Berechnung induzieren; er beschreibt dies in seinem Rechenbuch von 1560, das entsprechend der Sitte der damaligen Zeit einen langen erklärenden und werbenden Titel trug:

„Ein new und wolgegründt Rechenbuch, auff den Linien und Ziffern, sampt der Welschen Practica und allerley Vortheilen, neben der Ex= traction Radicum, und von den Proportionen, mit vielen lustigen Fragen und Aufgaben. Dessgleichen ein vollkommener Bericht der Regel Falsi, mit neuwen Inventionibus, De= monstrationibus, und Vortheilen, so biß anher für unmüglich geschetzt, gebessert, derglei= chen noch nie an Tag kommen. Und dann von der Geometrie, wie man mancherley Felder und Ebenen, auch allerley Corpora, Regularia und Irregularia, messen, Aream finden und rech= nen soll."


Ein new vnd wol
gegründt Rechenbuch/auff den Linien vñ Ziffern/
sampt der Welschen Practic vnd allerley vortheilen/neben der
extraction Radicum, vñ von den Proportionen/mit vilen lustigen Fragen vñ
Auffgaben/ꝛc. Deßgleichen ein vollkomner Bericht der Regel Falsi/mit neuwen Inuentio-
nibus/Demonstrationibus/ vnd vortheilen/so biß anher für vnmüglich geschetzt/gebessert/der-
gleichen noch nie an tag kommen. Vnd dann von der Geometria/wie man mancherley Fel-
der vnd ebne/auch allerley Corpora/Regularia vnd Irregularia/messen/Aream finden vñ rech-
nen sol. Alles durch Simon Jacob von Coburg/Bürger vnd Rechenmeister zu
Franckfurt am Main/mit fleiß zusammen getragen/vnd jetzt
erstmals getruckt.
Mit Röm. Keyserl. Mt. Gnad vnd Freiheit nicht nachzuwercken.
Getruckt zu Franckfurt am Mayn/1565.

Im Buch ist bezüglich des euklidischen Algorithmus folgendes interessant:

Wie findet mann ein zal und die größt/
die disen bruch $\frac{77002051219739}{124591936076998}$ [...]
zum kleinesten macht?

Die allgemeyne Regel ist/ Theil des Bruchs nenner durch seinen zeler/ nim nach dem den zeler vnn teil durch die zal/ di vberblieben/ ferner theil weiter den theiler diser andern Division durch den rest so bliben ist/ vnd solche theilung widerhole so offt/ biß endlich ein mal nichts vberbleibt/ welche zal dann in solcher arbeit der letste theyler ist/ die macht den bruch kleiner an seinen zaln/ vnd ist in dem fürgebrachten Bruch 19/ Hat solche Regel ihr Beweisung auß der 2 Proposition des 7 Buchs Euclidis/ darauß dann auch vernommen wird/ wann der letste theyler eins/ daß der Bruch kleiner zu= machen vnmöglich were/ vnd durch solche regel findt mann alle mal ein solche zal/ die die zalen des bruchs so erkleinert/ daß sie kleinerzumachen vnmöglich sein. Vnd hat obgesatzter Bruch ein wunderbarlich art in ihm/ Nemlich daß er sich 54 mal diuidirn leßt/ ehe mann das gemein maß oder die größt zal damit er auffgehaben wirdt/ findet/ mag derhalb wol ein Arith= metisch labyrint genant werden/ wird gemacht auß der vorgesatzten ord= nung der zaln/ da je die zwo so nechst vff einander folgen/ souil thun als die dritt folgend/ bringen je die erst vnd dritt inn einander multiplicirt eins weniger dann das quadrat der mittleren/ darumb je weiter mann solche ordnung erstreckt/ je näher man zu der Proportz kommt/ Dauon Euclide die 11 Proposi des 2/ vnnd 30 des 6 Buch handeln/ vnnd wiewol mann jmmer jhe näher kompt/ mag doch nimmermehr dieselb erreicht/ auch vbertretten werden.

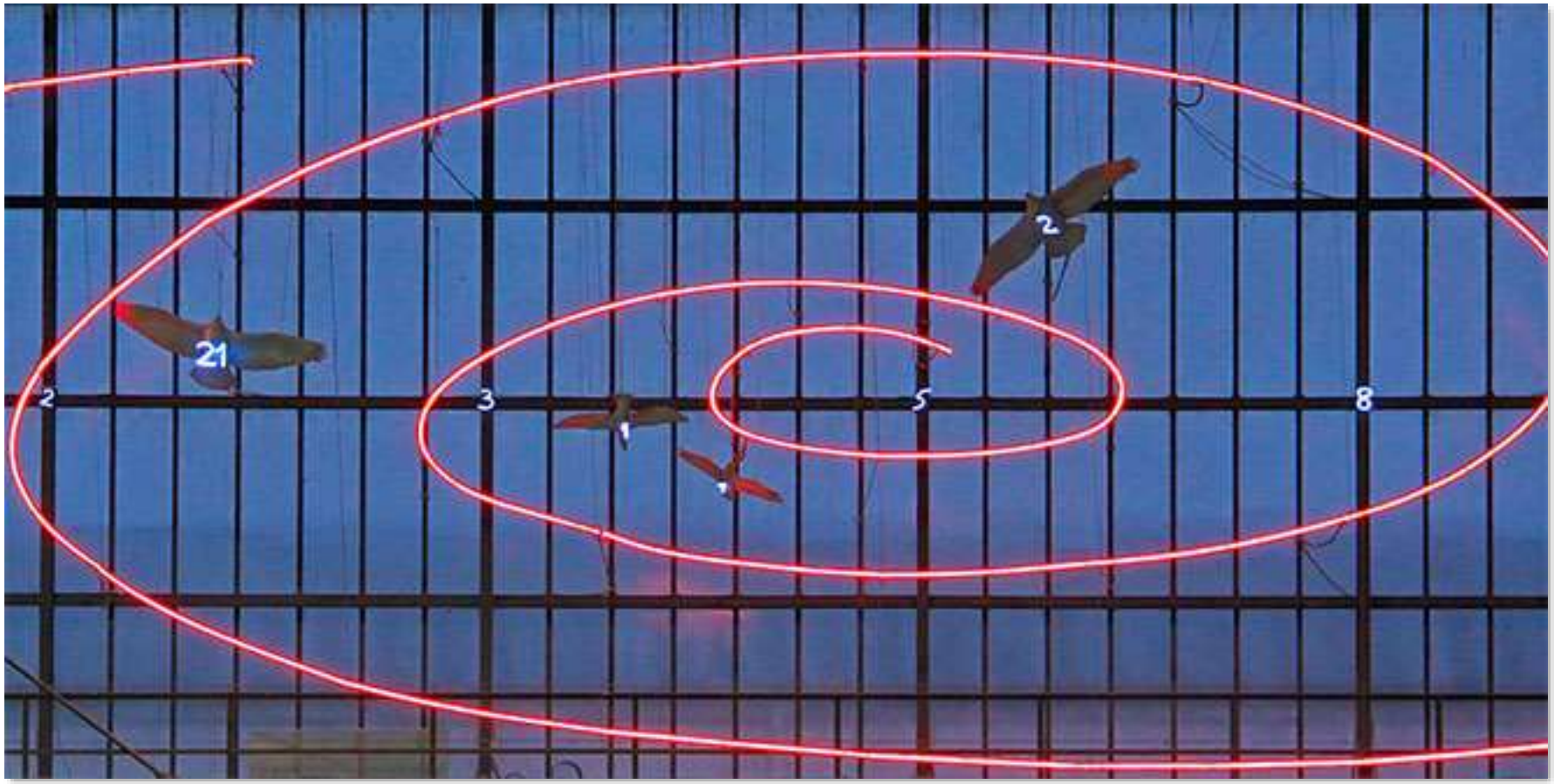
Das Frühneuhochdeutsch des Textes ist zwar gut verständlich, wirkt aber mit seiner inkonsistenten Orthographie und teilweise lautmalerischen Vokabelschreibweise aus heutiger Sicht etwas skurril. (Statt „54 mal diuidirn leßt" muss es übrigens korrekt „64 mal..." heissen, offenbar handelt es sich um einen Druckfehler.)

Mit „vorgesatzte ordnung" ist die Folge der Fibonacci-Zahlen von 1 bis 317811 gemeint, die als Randnotiz erscheint. Mit „bringen je die erst vnd dritt inn einander multiplicirt eins weniger dann das quadrat der mittleren" meint Simon Jacob, dass fib(i) * fib(i+2) bis auf eine Differenz von 1 dem Quadrat von fib(i+1) entspricht; dies ist bemerkenswert, da es sich um die sogenannte Cassini-Identität handelt, die eigentlich erst 1680 von Jean-Dominique Cassini, dem seinerzeitigen Direktor der Pariser Sternwarte, entdeckt wurde! (Ein informelles Argument dazu, unter Herausfaktorisierung von √5, wäre: $\Phi^n * \Phi^{n+2} = \Phi^{2n+2} = \Phi^{n+1} * \Phi^{n+1}$.)

Beim angesprochenen 11. Satz des 2. Buches von Euklid handelt es sich um den Goldenen Schnitt bzw. dessen Verhältniszahl φ = (1+√5)/2 = 1.618...; Simon Jacob behauptet also $\lim_{(n\to\infty)} fib(i)/fib(i-1) = \varphi$; ebenfalls eine bemerkenswerte Erkenntnis!

Mit „arithmetisch labyrint" bezeichnet Jacob einen Bruch aus (gemeinsamen Vielfachen) aufeinanderfolgender Fibonacci-Zahlen; die „wunderbarlich art" ist die Eigenschaft, dass dabei der euklidische Algorithmus besonders viele Schritte benötigt. Im konkreten Fall des von Jacob „fürgebrachten" Bruchs ist 77002051219739 = 19 * 4052739537881 = 19 * fib(62) und 124591936076998 = 19 * 6557470319842 = 19 * fib(63). Man darf wohl annehmen, dass Jacob davon ausging (bzw. vermutlich sogar überzeugt war), dass benachbarte Fibonacci-Zahlen den „worst case" darstellen – ob dahinter mehr steckt als begründete Intuition, die aus der Praxis gewonnen wurde, bleibt allerdings offen.

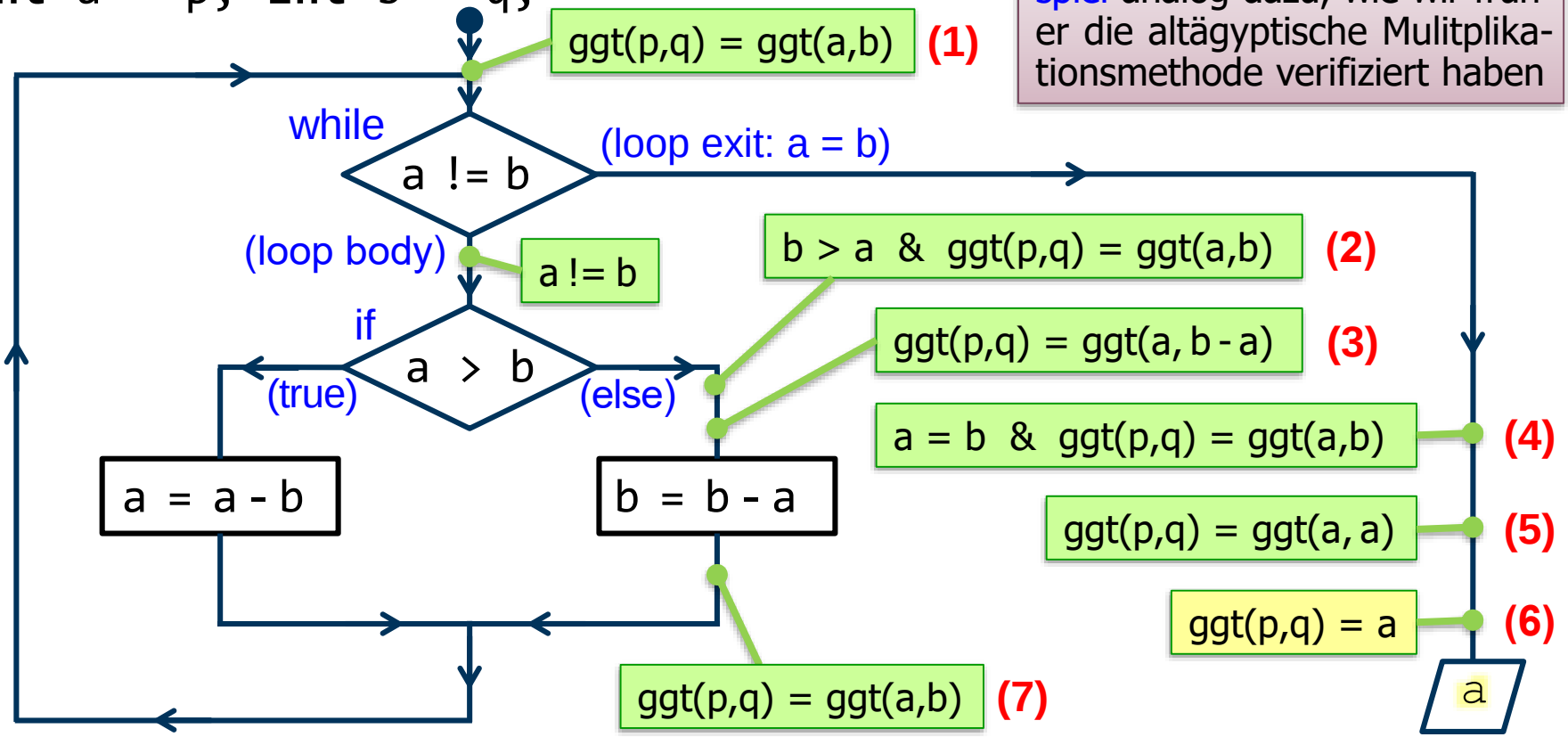
Fibonacci in Zürich



Wo in Zürich befindet sich diese Lichtskulptur zu den Fibonacci-Zahlen?

Euklidischer Algorithmus für ggT – Verifikation

```
public int ggT(int p, int q)
{int a = p; int b = q;
```



ggT-Verifikation als Übungsbeispiel analog dazu, wie wir früher die altägyptische Mulitplikationsmethode verifiziert haben

Die formalen Parameter p und q werden in der Methode nicht verändert. (3) folgt aus (2), da (für b>a) jede Zahl, die a und b teilt, auch b-a teilt (der Teiler kann ausgeklammert werden). (7) folgt aus (3) aufgrund des Zuweisungsaxioms (vgl. frühere Diskussion bei der Hoare-Logik). (1) bzw. (7) ist die Schleifeninvariante. (6) folgt aus (5), da offenbar stets ggt(x,x) = x gilt. Der nicht detaillierter ausgeführte linke if-Ast entspricht dem rechten in analoger Weise.

Euklidischer Algorithmus (Euklid: Elemente)

Historische Notiz

Darstellung von Euklid durch Justus von Gent, niederländischer Maler und Zeichner (ca. 1430)

Der 2. Satz.

Es sind zwey Zahlen gegeben, die nicht Primzahlen zu einander sind: man soll ihr gröſtes gemeinſchaftliches Maaß finden.

Erſter Fall.

Es seyen, A, B, die beyden gegebnen Zahlen, so daß die gröſſere, A, von der kleinern, B, genau gemeſſen werde. Nun mißt B auch sich selbſt, folglich ist B ein gemeinſchaftliches Maaß von A und B; aber auch das gröſte, weil offenbar keine gröſſere Zahl, als B, die B meſſen kann.

A 8 B 4

Zweyter Fall.

Es seyen AB, CD, die beyden gegebnen Zahlen, so daß die gröſſere, AB, von der kleinern, CD, nicht genau gemeſſen werde. Nimm immer die kleinere von der gröſſern weg, bis ein Rest kommt, welcher die nächſtvorgehende Zahl genau mißt. Dieſer Rest ist das gröſte gemeinſchaftliche Maaß der beyden gegebnen Zahlen.

A 4 E 6 B
C 2 F 4 D
G —

1.) Daß dieſer Rest gröſſer als die Einheit, folglich eine Zahl sey, siehet man daher, weil sonſt (7, 1. S.) AB, CD, Primzahlen zu einander wären, gegen das Angenommene.

Euklidischer Algorithmus (Euklid: Elemente)

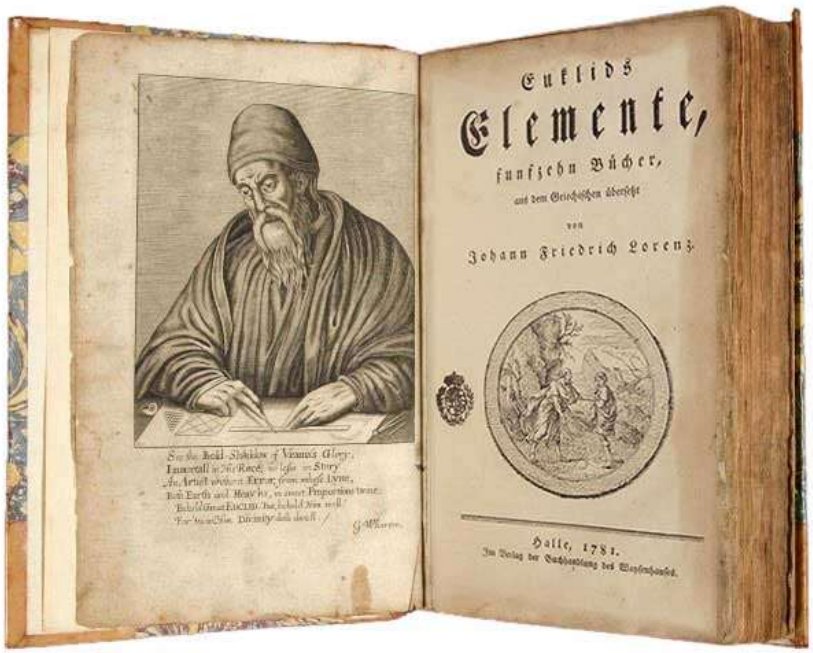
Historische Notiz

2.) Daß dieſer Reſt das gemeinſchaftliche Maaß von AB, CD, ſey, wird ſo gezeigt: Es laſſe AB, von CD gemeſſen, den Reſt AE < CD, und CD, von AE gemeſſen, den Reſt CF, welcher die nächſtvorgehende Zahl AE genau meſſe. CF mißt AE, und AE die DF, folglich mißt CF die DF; aber auch ſich ſelbſt, folglich die ganze CD. Nun mißt CD die BE. Folglich mißt auch CF die BE; aber auch die EA, folglich die ganze AB. Demnach iſt der Reſt CF ein gemeinſchaftliches Maaß von AB, CD.

3.) Daß dieſer Reſt CF aber auch das gröſte gemeinſchaftliche Maaß ſey, beweißt man alſo: Es ſey ein andres, G > CF. G mißt die CD, und CD die BE, folglich mißt G die BE; aber auch die ganze AB, folglich auch den Reſt AE. Nun mißt AE die DF. Folglich mißt G die DF; aber auch die ganze CD, folglich den Reſt CF, welches unmöglich, weil G > CF.

Zuſatz.

Hieraus erhellet: wenn zwey Zahlen von einer dritten gemeſſen werden, daß auch das gröſte gemeinſchaftliche Maaß der beyden Zahlen von dieſer dritten Zahl müſſe gemeſſen werden.



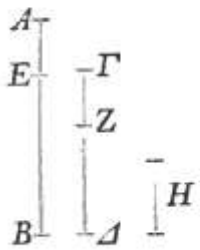
→ Euklids Elemente, funfzehn Bücher, aus dem Griechischen übersetzt von Johann Friedrich Lorenz, Halle 1781 (Buch 7, 2. Satz).

Die älteste erhaltene griech. Handschrift der „Elemente“ stammt von 888; sie entspricht der Ausgabe in der Bearbeitung von Theon von Alexandria. Relevant ist vor allem eine Handschrift aus dem 10. Jhd. (Vatikanbibliothek), die einen Text vor der Bearbeitung durch Theon enthält; auf ihr beruhen alle neueren Editionen. Die erste gedruckte Ausgabe erschien (auf griechisch) 1533 in Basel; bis Mitte des 19. Jahrhunderts war das Werk neben der Bibel das auflagenstärkste Buch der Menschheitsgeschichte.

Euklidischer Algorithmus (Euklid: Elemente)

Historische Notiz

at si $\Gamma\Delta$ non metitur AB, minore numerorum AB, $\Gamma\Delta$ semper uicissim a maiore subtracto relinquetur numerus aliquis, qui proxime antecedentem metietur. unitas enim non relinquetur; sin minus, AB, $\Gamma\Delta$ inter se primi erunt [prop. I]; quod contra hypothesim est. ergo numerus aliquis relinquetur, qui proxime antecedentem metietur. et $\Gamma\Delta$ metiens BE relinquat se



At si $\Gamma\Delta$ non metitur AB, minore numerorum AB, $\Gamma\Delta$ semper vicissim a maiore subtracto relinquetur numerus aliquis, qui proxime antecedentem metietur.

Vicissim (Adv.) wird hier in der Bedeutung *untereinander*, *gegenseitig*, *abwechselnd* gebraucht; bezeichnet eine abwechselnde Handlung eines oder zweier Akteure

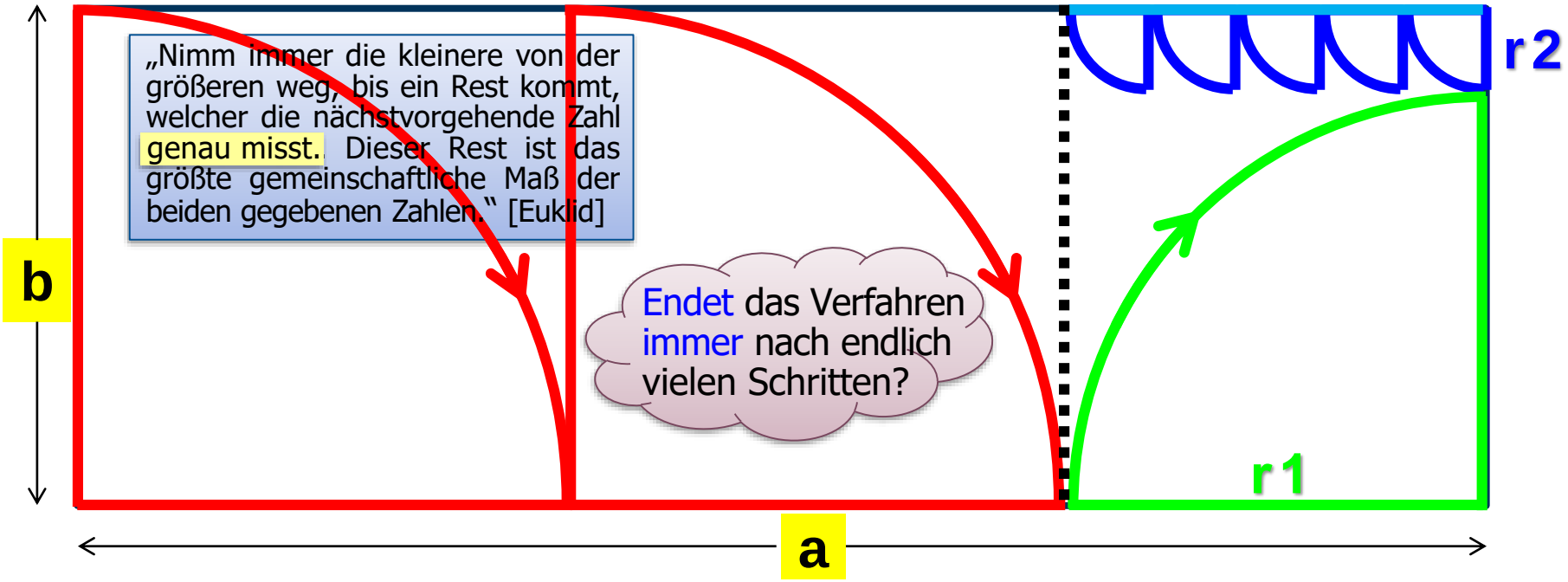
Εἰ δὲ οὐ μετρεῖ ὁ $\Gamma\Delta$ τὸν AB, τῶν AB, $\Gamma\Delta$ ἀνθυφαιρουμένου ἀεὶ τοῦ ἐλάσσονος ἀπὸ τοῦ μείζονος λειφθήσεταί τις ἀριθμός, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. μονὰς μὲν γὰρ οὐ λειφθήσεται· εἰ δὲ μή, ἔσονται οἱ AB, $\Gamma\Delta$ πρῶτοι πρὸς ἀλλήλους· ὅπερ οὐχ ὑπόκειται. λειφθήσεταί τις ἄρα ἀριθμός, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. καὶ ὁ μὲν $\Gamma\Delta$ τὸν BE μετρῶν

Partizipform ἀνθυφαιρουμένου zum Prozess des wechselseitigen Wegnehmens (→ „Wechselwegnahme“ = *Antiphairesis* bzw. *Antepheiresis, Anthyphairesis* etc., vgl. aphairesis: Wegnahme, Entziehen)

Euclidis Opera Omnia. 9 Bände, Teubner, Leipzig 1888–1916, siehe auch https://el.wikisource.org/wiki/Στοιχεία/ζ (2.2)

Wechselwegnahme am Rechteck

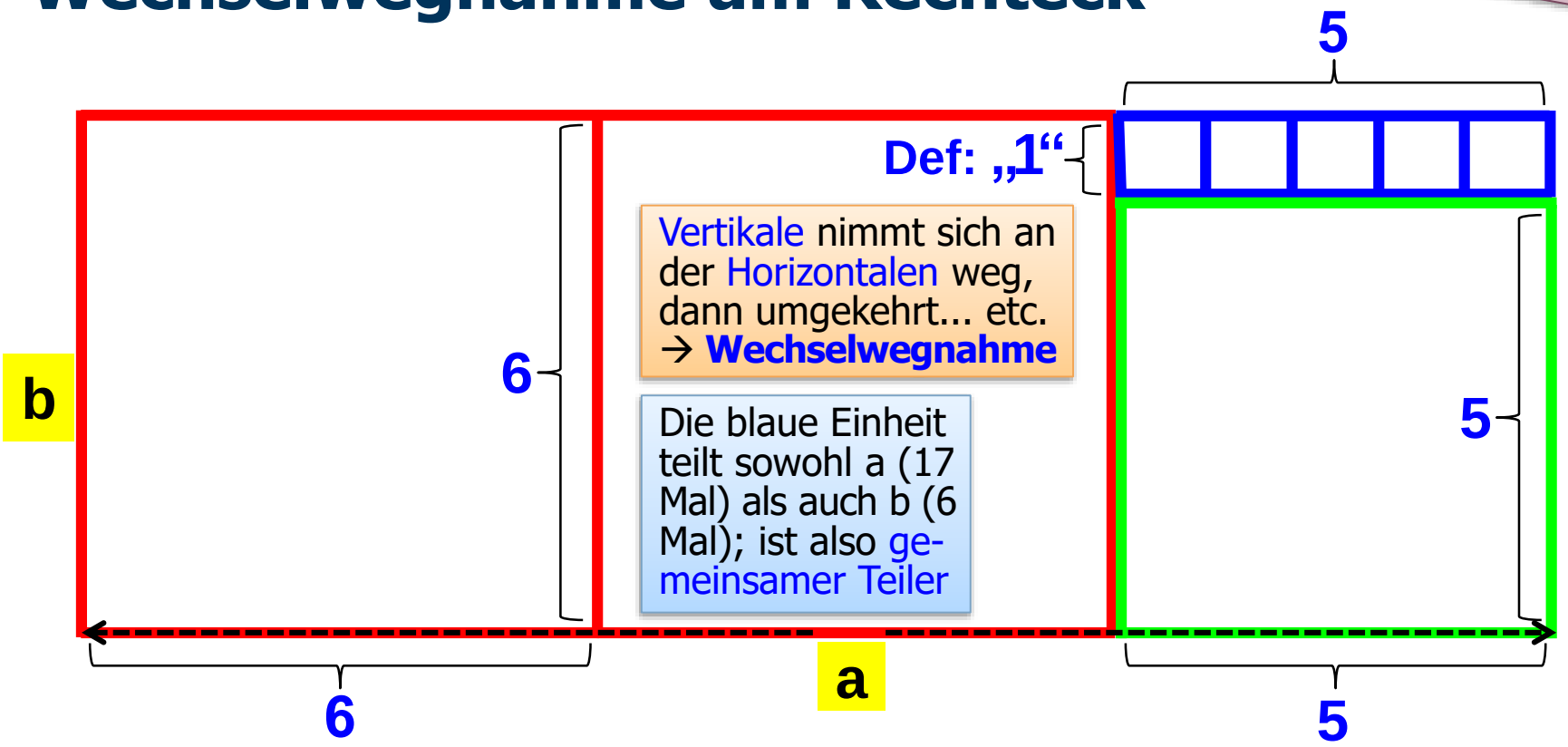
Historische Notiz



Die kleinere Seite b wird (z.B. mit dem Zirkel) so oft an der grösseren, vertikalen Seite a abgetragen (rot), bis ein Rest r1 bleibt. Danach wird dieser an der (nunmehr längeren, horizontalen) Seite b abgetragen (grün) bis zu einem dortigen Rest r2. Mit diesem wird dann wiederum die vertikale Seite abgemessen (blau); in diesem Beispiel geht dies nach fünf Mal exakt auf – das gemeinsame Mass entspricht dem blauen Quadrat. Die horizontale und vertikale Seite wechseln im Prozess mehrfach ihre Rolle.

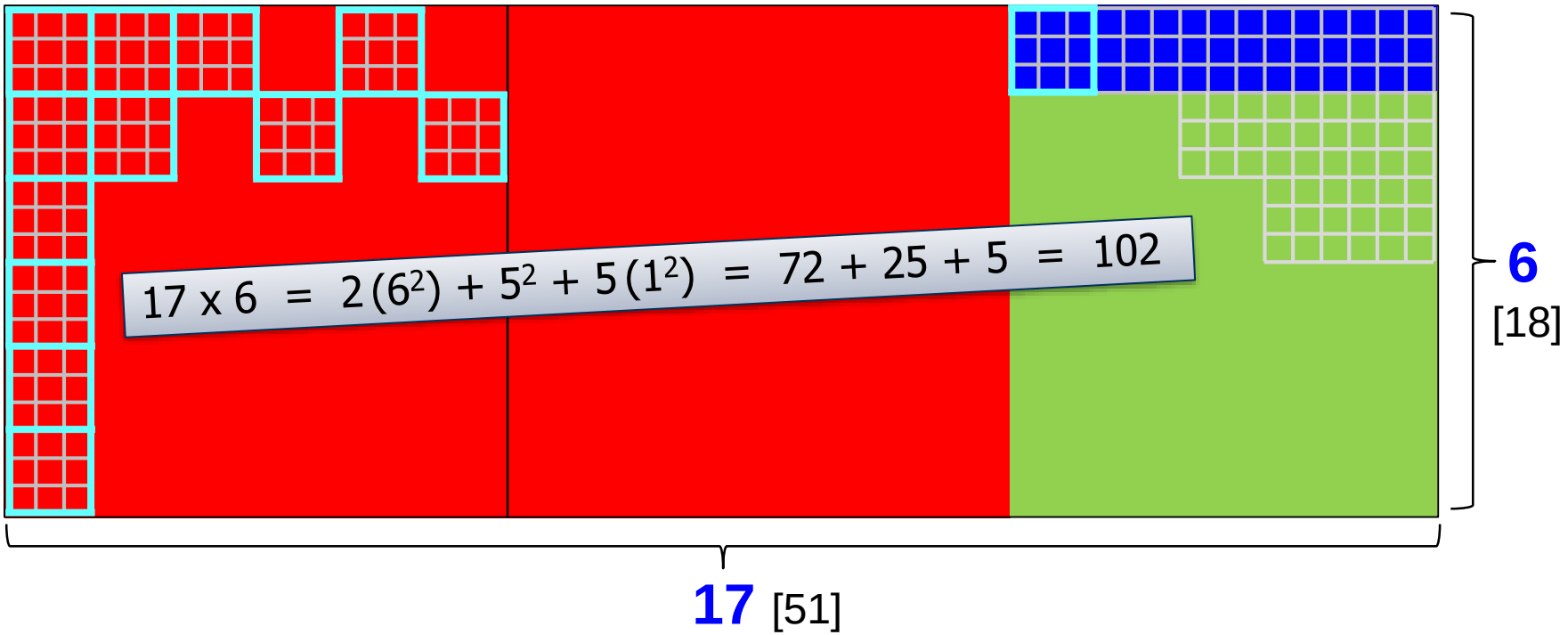
Wechselwegnahme am Rechteck

Historische Notiz



Das grüne Quadrat ist 5 blaue Einheiten lang und breit, das rote 6. Damit ist im ursprünglichen Rechteck a = 17 (es ist also 17 „blaue“ lang), die Höhenkante b misst 6 „blaue“. „Ein blau“ stellt die gemeinsame Masseinheit dar. Wäre das blaue Quadrat z.B. 12 mm „im Quadrat“, dann wäre a = 204 mm und b = 72 mm. Da die blaue Einheit in a und b ganzzahlig enthalten ist, ist 12 ist also (grösster!?) gemeinsamer Teiler von 204 und 72. Das Prinzip der Wechselwegnahme war Handwerkern und Baumeistern (die mit „Mass-Stäben“ und Messschnüren hantiert haben, um das gemeinsame Mass zweier Strecke zu finden) sicherlich schon lange bekannt, bevor die Harmonie von Zahlenverhältnissen von den Pythagoreern zum mathematischen und metaphysischen Prinzip erhoben wurde und Euklid den Algorithmus aufschrieb.

Wechselwegnahme am Rechteck

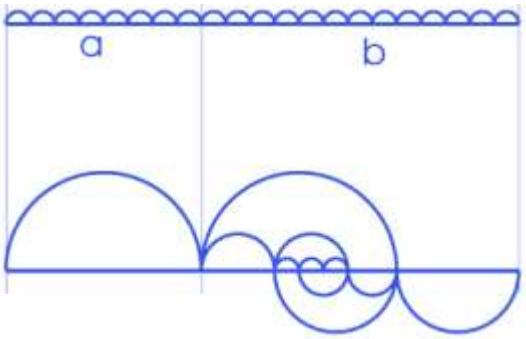


- Für 17 und 6 ist der ggT = 1; statt mit 102 Quadraten der Grösse 1 x 1 könnte man (in einem anderen „Massstab" und entsprechend feinerem Koordindaten-gitter) das Rechteck aber z.B. auch mit 918 Quadraten der Grösse 1/3 x 1/3 (relativ zum Originalmassstab) pflastern – man sucht aber ja gerade das *grösste* gemeinsame Mass
- Bei 51 und 18 (→ 918) wäre der ggT dann nicht 1, sondern 3 mal grösser, also 3

Wechselwegnahme

Auf dem Tisch liegen 2 Reihen von Streichhölzern. Von der jeweils längeren Reihe sollen so viele weggenommen werden, wie in der kürzeren vorkommen. Aufgehört wird, wenn beide Reihen gleich lang sind.

Historische Notiz



Das Prinzip der Wechselwegnahme wurde in der griechischen Mathematik der Pythagoreer vor allem auf die Grössen von Strecken geometrischer Figuren angewendet. Man suchte nach dem gemeinsames Mass e zweier Strecken a, b, das sich dann ergibt, wenn beim euklidischen Algorithmus schliesslich ein Rest von 0 herauskommt. Diese Strecken haben a priori zwar keine ganzzahlige Länge – aber dafür muss man die (gemeinsame) Längeneinheit doch nur klein genug wählen, oder?

Restzweifel bleiben: Gibt es vielleicht zusammengehörige Strecken, die kein gemeinsames Mass besitzen, also inkommensurabel sind? Wo die fortgesetzte wechselseitige Wegnahme dann nie terminieren würde? Und was würde dies dann bedeuten?

Und wirklich stellte sich heraus – wohl zum Entsetzen der Pythagoreer, die ja daran glaubten, dass der Kosmos eine nach bestimmten Zahlenverhältnissen aufgebaute harmonische Einheit bildet – dass z.B. Diagonale und Seite des Quadrats, und vor allem auch des regelmässigen Fünfecks, inkommensurabel sind; moderner ausgedrückt, dass das Grössenverhältnis keine rationale Zahl darstellt. Tatsächlich führt eine entsprechend geeignete geometrische Konstruktion für die Wechselwegnahme zu einem endlosen Regress immer kleinerer selbstähnlicher Figuren; algebraisch interpretiert, zu einem nie abbrechenden Kettenbruch! Das Streben nach Harmonie erlitt so gesehen einen herben Rückschlag. Sind Figuren mit inkommensurablen Strecken vielleicht nur Ideale, also Hirngespinste, aber nicht real existent? Aber Denkschmerz beiseite: Mit der Entdeckung der Inkommensurabilität sowie irrationaler Verhältnisse und Zahlen hält, zögerlich zunächst, die Unendlichkeit Einzug in die Mathematik. Eine gewaltige Bereicherung, wie die folgenden Jahrhunderte – um nicht zu sagen Jahrtausende – zeigen sollten!

Als Übung

Simultane Berechnung des ggT und kgV

Von E.W. Dijkstra (**EWD 316**, 1971) stammt folgender nette Algorithmus, der simultan ggT und kgV (von A, B > 0) berechnet:

Exercise. Prove that the following program will print in addition to the greatest common divisor of A and B the smallest common multiple of A and B. Hint: Follow the expression a x c + b x d.

```
int a = A, b = B, c = B, d = 0, gcd, scm;
while (a != b)
{ while (a > b) {a = a - b; d = d + c};
  while (b > a) {b = b - a; c = c + d};
}
gcd = a; scm = c + d;
print(gcd); print(scm);
```

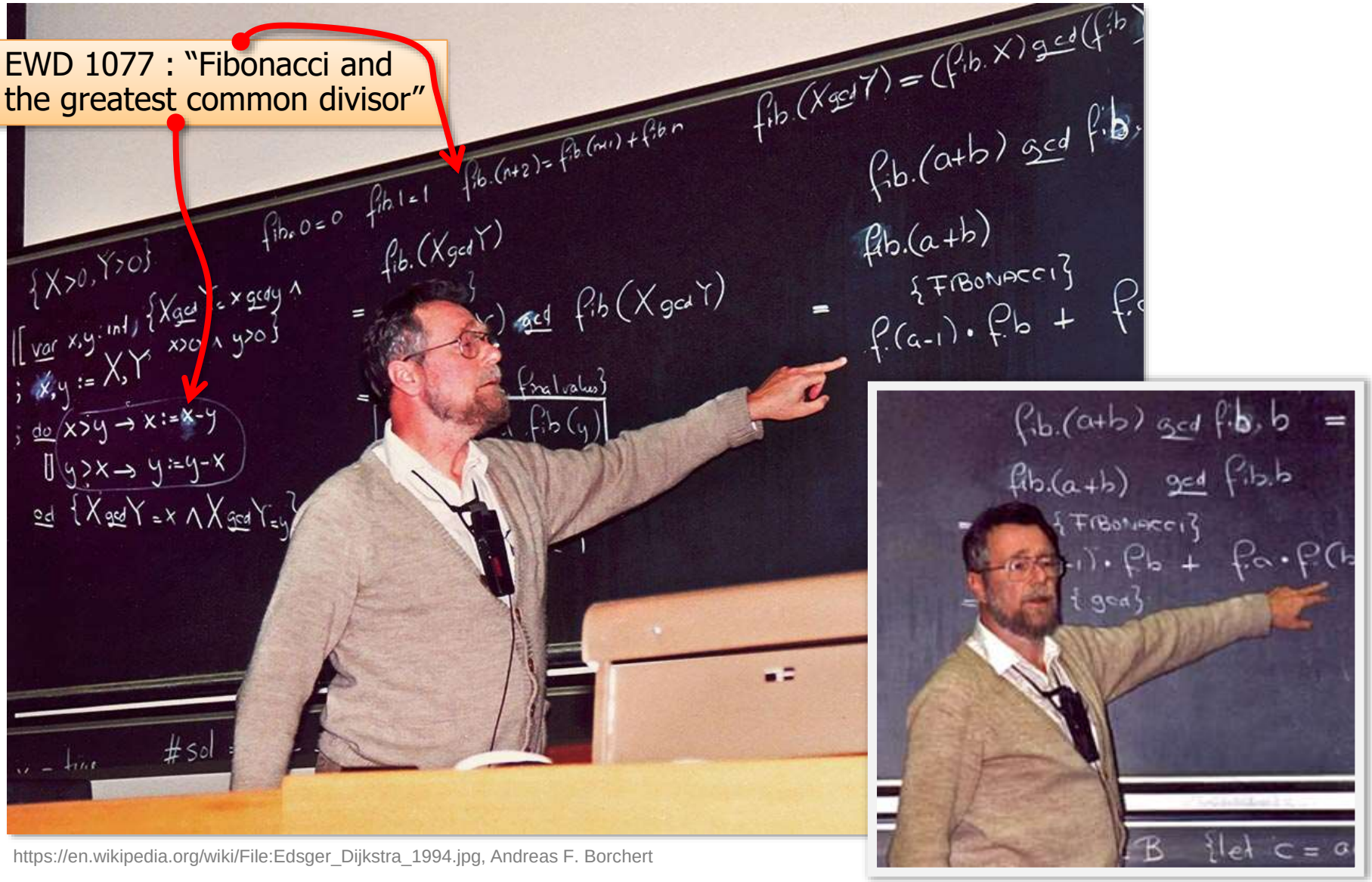
www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD316.4.html

The program and the correctness proof grow hand in hand. [E.W. Dijkstra in seiner Turing Award lecture 1972]

Dijkstra never wrote his articles using a computer. He preferred to rely on his typewriter and after 1972 on his Mont Blanc fountain pen. (His handwriting was so perfect and distinct that in the late 1980s Luca Cardelli designed a 'Dijkstra' font.) These articles were then distributed in an old-fashioned way: he sent copies to a few friends and associates who then served as the source nodes of the distribution centres. They are rarely longer than 15 pages and are consecutively numbered. The last one, EWD 1318, is from April 2002. Within computer science they are known as the EWDs. [Krzysztof Apt]

Historische Notiz

Dijkstra im März 1994 an der ETH Zürich



https://en.wikipedia.org/wiki/File:Edsger_Dijkstra_1994.jpg, Andreas F. Borchert

Historische Notiz

(0) $f.0 = 0 \;,\; f.1 = 1 \;,\; f.(n+2) = f.(n+1) + f.n$.

Then, f application distributes over gcd, i.e.

(1) $f.(X \text{ gcd } Y) = f.X \text{ gcd } f.Y$.

* * *

EWD 1077 : "Fibonacci and the greatest common divisor"

Our interest is not in the above theorem, nor in its proofs. We wish to explore how we could design a proof for it.

We know the gcd for positive operands as the outcome of Euclid's Algorithm:

(2)
```
x,y := X,Y
; do x>y → x:= x-y
  [] y>x → y:= y-x
  od {x = X gcd Y ∧ y = X gcd Y} ,
```

and this knowledge raises the question of whether we can prove (1) by a properly chosen invariant for program (2) .

Dijkstra war ein starker Verfechter des „zero-based numbering"; in EWD 831 "Why numbering should start at zero" (1982) begründet er dies. ("The moral of the story is that we better regard — after all those centuries! — zero as a most natural number.")

"His reports […] begin with page 0, and when he was writing about *n* processes, they were invariably numbered 0, …, *n*−1" [K. Apt]. Angeblich hatte der Springer-Verlag extra für die Dijkstra-Texte die Latex-Makros entsprechend verändert.

(Und wieso gibt es in Java eigentlich ein 0. Array-Element?)

Mehr zu Dijkstra später

Zero-based numbering?

Auf einer Reise geriet der Mathematiker Waclaw Sierpinski plötzlich in Panik, weil er ein Gepäckstück vergessen zu haben glaubte. „Aber Liebling" beruhigte ihn seine Frau, „alle sechs Koffer sind da." „Das kann nicht sein", entgegnete Sierpinski, „ich habe zweimal nachgezählt: null, eins, zwei, drei, vier, fünf."
[Nach: John H. Conway, Richard Guy: The Book of Numbers]

- Beim Abzählen zählen wir 1, 2, 3,...; fangen also bei 1 an zu zählen, wobei wir dabei oft auf das erste Objekt zeigen, dann auf das zweite etc.
  - Entsprechend ist z.B. Folgendes praktisch nur bei bzw. mit „1" (oder grösseren Zahlen) sinnvoll, nicht aber bei „0": Gleis 1, Paragraph 1, Phase 1, Version 1, Einbettzimmer, Einfamilienhaus, Einhorn, Einmalhandtuch, Einmaster, Einpersonenhaushalt, Eintagsfliege, Eintopfgericht, Einwegbehälter, eindeutig, eineiige Zwillinge, einfarbig, eingleisig, einseitig, einsilbig,...
- Das Erste spielt oft eine besondere Rolle, daher gibt es viele entsprechende Wortbildungen, wo „0" typischerweise sinnlos ist:
  - Erstauflage, Erstbesitz, Erstbezug, erstgeboren, erstgenannt, Erstimpfung, erstklassig, Erstschlag, Erstsemester, Erstveröffentlichung, erste Hilfe, König Ludwig I.,...
- Die Null kommt bei Wortzusammensetzungen meist nur dann ins Spiel, wenn man die Abwesenheit einer Eigenschaft hervorheben will (oder bei manchen Aufzählungen, wo es vor der 1 tatsächlich etwas gibt):
  - Nulldiät, Nullwachstum, Nullrunde, Nullstellung, Nullnummer, Nulltarif, Nullsummenspiel, Null-Bock-Generation, Windstärke 0, Nullgradgrenze, Nullmeridian,...
- Irritationen hinsichtlich 0 und / oder 1 gibt es allerdings manchmal doch:
  - Ist das Erdgeschoss der 0. oder 1. Stock?
  - Folgt auf das Jahr 1 v. Chr. direkt das Jahr 1 n. Chr.? (Wo bleibt dann das Jahr „0"?)

Who's on first? Zero or one?

Danny Cohen (1981): On holy wars and a plea for peace. *Computer*, *14*(10), 48-54

People start counting from the number one. The very word *first* is abbreviated as 1st, which indicates one. This, however, is a very modern notation. The older concepts do not necessarily support this relationship. In English and French the word *first* is not derived from the word *one*, but from an old word for *prince*, which means foremost. Similarly, the English word *second* is not derived from the number two but from an old word which means "to follow." Obviously, there is a close relation between third and three, fourth and four, and so on. [...]

For a very long time, people have counted from one, not from zero. As a matter of fact, the inclusion of zero as a full-fledged member of the set of all numbers is a relatively modern concept, even though it is one of the most important numbers mathematically.

A nice mathematical theorem states that for any basis b the first b^N positive integers are represented by exactly N digits (leading zeros included). This is true if and only if the count starts with zero (hence, 0 through b^N - 1), not with one (for 1 through b^N).

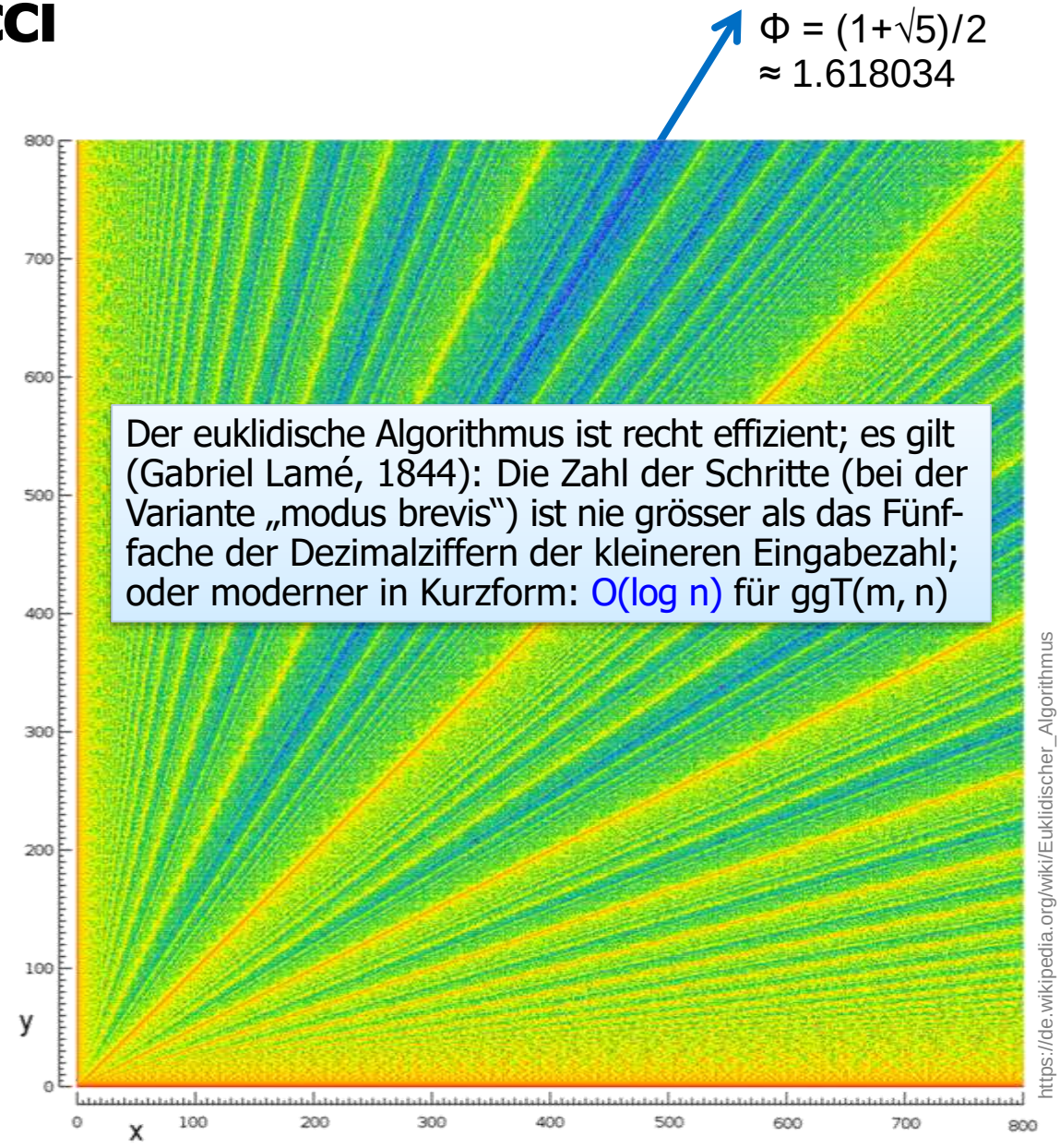
This theorem is the basis of computer memory addressing. Typically, 2^N cells are addressed by an N-bit addressing scheme. A count starting from one rather than zero would cause the loss of either one memory cell or an additional address line. Since either price is too expensive, computer engineers agree to use the mathematical notation that starts with 0. Good for them!

This is probably the reason why all memories start at address 0, even those of systems that count bits from bit 1 up. The designers of the IBM 1401 were probably ashamed to have address 0. They hid it from the users and pretended that the memory starts at address 1.

Communication engineers, like most people, start counting from one. They never have to suffer the loss of a memory cell, for example. Therefore, they happily count one-to-eight, not zero-to-seven, as computer people do.

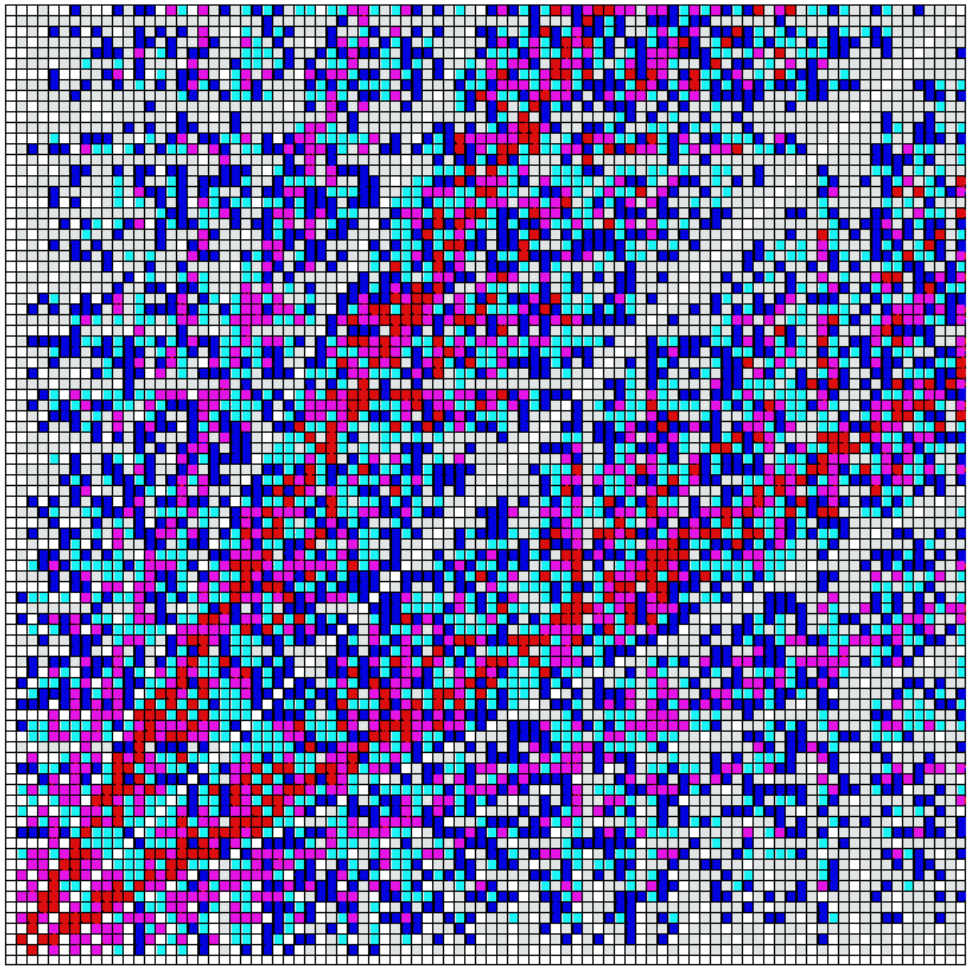
Euklid und Fibonacci

Der euklidische Algorithmus und die Fibonacci-Folge 1, 1, 2, 3, 5, 8, 13, 21,... [Fib(i) = Fib(i-1) + Fib (i-2)] hängen auf eine interessante Weise zusammen: Bei der Laufzeit stellt ein „schlimmer Eingabefall" zwei aufeinanderfolgende Fibonacci-Zahlen dar; dann ergibt sich als Rest stets die nächstkleinere Fibonacci-Zahl. Das Bild zeigt die Anzahl der Schritte bei Eingabe von x, y. Rote Punkte bedeuten wenige Schritte; gelbe, grüne und blaue Punkte relativ mehr Schritte. Die dunkelblaueste Linie hat die Steigung Φ, wobei $\Phi = (1+\sqrt{5})/2 \approx 1.618034$ die „Golden Zahl" des Goldenen Schnitts ist. Schon für kleine i gilt recht präzise $Fib(i+1) / Fib(i) \approx \Phi$. Ausserdem ist Φ^n „fast" eine natürliche Zahl, und $\Phi^n/\sqrt{5}$, gerundet auf ganze Zahlen, ist stets Fib(n).



Euklid und Fibonacci (2)

Hier das Anfangsstück nochmals in einer detaillierteren Auflösung. Karos von (x,y)-Paaren mit maximaler Schrittzahl sind hier rot gefärbt, solche mit der um 1, 2 oder 3 gegenüber der Höchstzahl verminderten Schrittzahl magenta, cyan bzw. blau. Alle Karos, die Paaren mit noch kleinerer Schrittzahl repräsentieren, erscheinen grau – bis auf diejenigen, die nur einen einzigen Schritt erfordern, sie wurden weiss gefärbt. Man erkennt die Symmetrie zur Diagonalen sowie die beiden roten „Worst-Case-Linien" mit der Steigung 1.61803398874989482... bzw. 0.61803398874989482...



http://horstth.de/wp-content/uploads/2013/10/EuklidGrafikNeuD.jpg

Euklid und Fibonacci

Gabriel Lamé: *Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers.* Comptes rendus hebdomadaires des séances de l'Académie des sciences, 19 (1844), 867-870

« Dans les traités d'Arithmétique, on se contente de dire que le nombre des divisions à effectuer, dans la recherche du plus grand commun diviseur entre deux entiers, *ne pourra pas surpasser la moitié du plus petit*. Cette limite, qui peut être dépassée si les nombres sont petits, s'éloigne outre mesure quand ils ont plusieurs chiffres. L'exagération est alors semblable à celle qui assignerait la moitié d'un nombre comme la limite de son logarithme; l'analogie devient évidente quand on connaît le théorème suivant :

» THÉORÈME. *Le nombre des divisions à effectuer, pour trouver le plus grand commun diviseur entre deux entiers* A, *et* B<A, *est toujours moindre que cinq fois le nombre des chiffres de* B.

par t'', t''', t^{IV}, t^{V},... les termes qui suivent t'', on aura les inéga-

$$2{,}5.10^k < t'' < 5.10^k,$$
$$4.10^k < t''' < 8.10^k,$$
$$6{,}5.10^k < t^{IV} < 13.10^k,$$
$$10{,}5.10^k < t^{V} < 21.10^k.$$

ne peut avoir plus de $(k+1)$ chiffres, et t'' moins de $(k+2)$ chif-

Erst später wurde diese Zahlenfolge (von Édouard Lucas) *Fibonacci* genannt.

COMPTE RENDU

DES SÉANCES

DE L'ACADÉMIE DES SCIENCES.

SÉANCE DU LUNDI 28 OCTOBRE 1844.

PRÉSIDENCE DE M. CHARLES DUPIN.

MÉMOIRES ET COMMUNICATIONS

DES MEMBRES ET DES CORRESPONDANTS DE L'ACADÉMIE.

ARITHMÉTIQUE. — *Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers ;* par M. LAMÉ.

Es handelt sich um eine frühe mathematische Abhandlung, in der die „Laufzeit" eines nicht-trivialen Algorithmus analysiert wird – die Veröffentlichung von Lamé stellt insofern einen Vorläufer aus dem Gebiet der Komplexitätstheorie dar, das erst 110 Jahre später begründet wurde (am Ende des Vorlesungskapitels „Komplexität von Algorithmen" kommen wir darauf zurück).

Komplexitätsanalyse von Euklids Algorithmus

Obwohl üblicherweise Lamé als Begründer der Komplexitätsanalyse des euklidischen Algorithmus angesehen wird, kommt diese Ehre statt dessen wohl Pierre Joseph Étienne Finck zu, der 1797 in Lauterburg, dem nordöstlichsten Punkt des Elsass, geboren wurde, 1829 in Strassburg promovierte und später dort auch Professor für Mathematik wurde. Er veröffentlichte 1841, drei Jahre vor Lamé, ein Buch *Traité élémentaire d'arithmétique à l'usage des candidats aux écoles spéciales*, in dem er den euklidischen Algorithmus analysiert:

Chaque reste est moindre que la moitié du dividende: car si le diviseur est égal à la moitié du dividende, le reste est 0; si le diviseur est plus grand que cette moitié, le quotient est 1, et le reste, devant faire avec le diviseur une somme égale au dividende, sera moindre que cette même moitié; enfin, si le diviseur est plus petit que la moitié du dividende, le reste le sera aussi comme étant moindre que le diviseur. De là il suit que si l'on nomme A et B les deux nombres dont on cherche le p. g. c. d., A étant > B, les restes successifs sont respectivement moindres que $A/2$, $B/2$, $A/4$, $B/4$, $A/8$, $B/8$, . . . , $A/2^n$, $B/2^n$; l'opération se terminera donc au plus tard lorsque $B/2^n \leq 2$, ou $B < 2^{n+1}$; car alors le reste moindre que $B/2^n$ sera au plus = 1, qui est le dernier diviseur; mais alors le nombre des divisions faites serait $2n + 1$. Ainsi, cherchez l'exposant de la plus petite puissance de 2, qui surpasse B, diminuez-le d'une unité, doublez le reste et ajoutez 1, ce sera une limite du nombre des opérations qu'il y aura à faire pour trouver le p. g. c. d. de A et B. Soit, pour exemple, les nombres 89 et 55: la plus petite puissance de 2, qui surpasse 55 est 64 ou 2^6; donc $n + 1 = 6$, $n = 5$ et $2n + 1 = 11$; et, en effet, dans ce cas, il faut neuf opérations.

Finck fand also heraus, dass bei ggT(m,n) die Zahl der Schritte $\leq 1 + 2 \log_2(n)$ beträgt; diese Schranke ist nur um einen Faktor von ca. 1.33 schlechter als die Schranke von Lamé („das Fünffache der Zahl der Ziffern von n"). Wieso Finck von Lamé nicht erwähnt wird, bleibt unklar – möglicherweise ist Lamé das frühere Resultat von Finck tatsächlich entgangen.

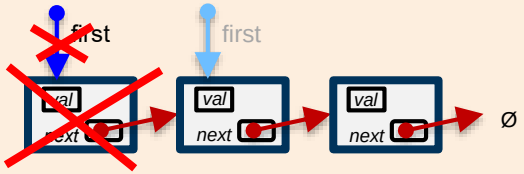
Übung: ggT-Algorithmus mit Schieberegistern

Digitalcomputer können typischerweise Zahlen in Registern sehr schnell verdoppeln und halbieren, indem die Schieberegister eine Stelle nach links oder rechts geschoben werden. Man mache sich die folgenden Identitäten klar und wende sie an, um damit einen effizienten Algorithmus zur Berechnung des ggT zu realisieren:

- $ggT(2a,2b) = 2\ ggT(a,b)$
- $ggT(2a,2b+1) = ggT(a, 2b+1)$
- $ggT(2a+1,2b+1) = ggT(2a-2b, 2b+1)$
- $ggT(a,b) = ggT(b,a)$

Resümee des Kapitels

- Pakete in Java
  - Beispiel verkettete Liste
  - Nutzung des Listenpakets
  - Beispiel-Nutzung: Stack mit Listen
  - Beispiel Bruchrechnung
  - Euklidischer Algorithmus (ggt)




```
import bruchPak; …
a = b.plus
    (new Bruch(1,8));
```