ZUSAMMENFASSUNG

1. Einleitung

- Konzept des Kurses. Fokus: Problemlösung.
- Computer und Programmierung
- Programmiersprache Java. Pascal →Java
  - Klassen vs. Records
  - Methoden vs. Prozeduren
  - Referenzsemantik und fehlende Referenzparameter in Java
- Erstes Java-Programm

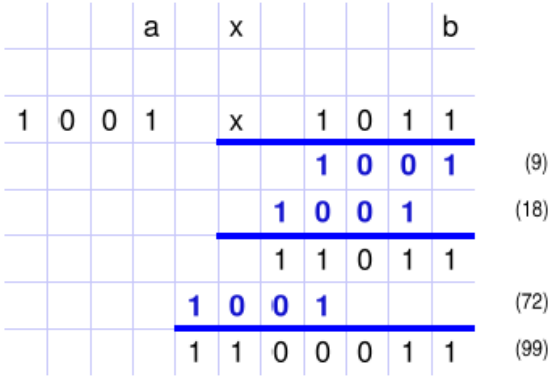
1. Fallstudie: Altägyptische Multiplikation

- Alorithmus basiert auf Addition, Verdoppeln und Halbieren
- Verhalten eines Programmes (Korrekt, inkorrekt, nicht terminierend)
- Rekursive Formulierung des Algorithmus, Induktiver Beweis
- Implementation in Java. Gültigkeit des Beweises für das Programm?
- Exception Handling

```
try { … }
catch (Fehlertyp1) { … }
```

- Elementare Codeumformungen → Endrekursive und iterative Formulierung des Algorithms
- Invarianten

```
static int fi(int a, int b) {
    int res = 0;
    while (b > 0) {
        res += a * (b%2);
        a *= 2;
        b /= 2;
    }
    return res;
}
```



2. Java

- Klassen, Packages, Namespaces, Qualifizierte Bezeichner
- Statische Methoden in Java – Prozeduren in Pascal
- Typisierung von Java – Typkonversionen: explizit, implizit. Prüfung durch Compiler oder Laufzeit
- Arrays: dynamische Objekte. Referenzsemantik!
- Programmargumente `args[]`
- System.in, System.out, System.error und Weiterleitung mit `>`, `<` und `|`

```
int i = 100;

float f = 2.712f;

i = f;              // type mismatch.
i = (int) f;        // explizit konvertiert

f = i;              // implizit konvertiert

Tier tier;

...

Hund hund = (Hund) tier; // dynamisch
```

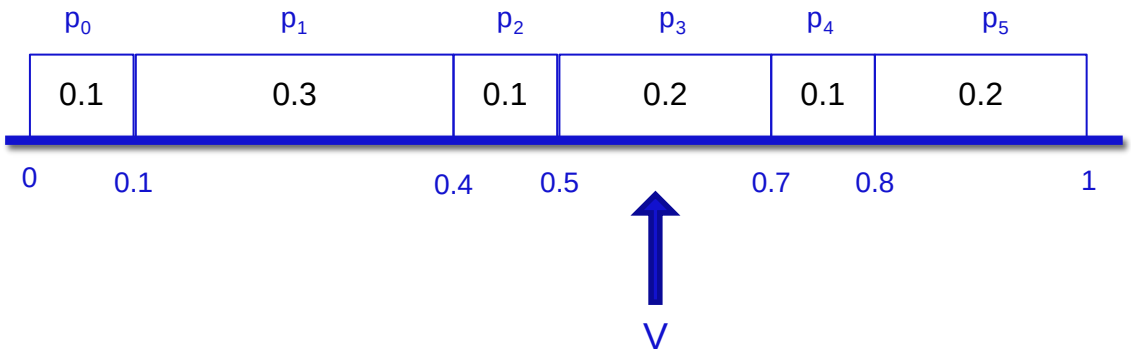
```
int [] x = new int[10];

int [] y;

y = x; // was passiert hier?

y[3] = 10; // und hier?
```

2. Fallstudie: Zufallssurfer

- Modellierung [als Markovkette] mit Matrix P der Übergangswahrscheinlichkeiten
- Simulation einer Zufallsvariablen

W't von der Seite 1 startend auf die Seite 3 zu wechseln

$$\begin{bmatrix} .02 & .92 & .02 & .02 & .02 \\ .02 & .02 & .38 & .38 & .20 \\ .02 & .02 & .02 & .92 & .02 \\ .92 & .02 & .02 & .02 & .02 \\ .47 & .02 & .47 & .02 & .02 \end{bmatrix}$$



- Anwendung der Simulation auf die Zeilen der Übergangsmatrix P.

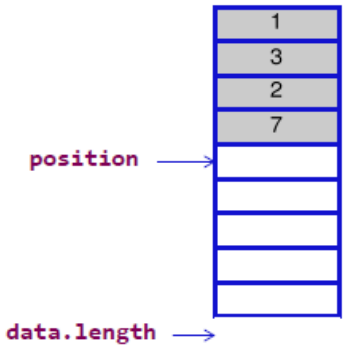
3. Klassen

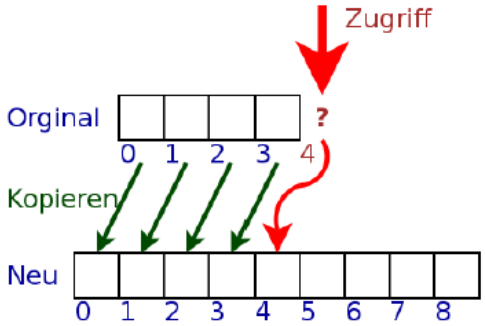
- Klassen beinhalten Daten und Code.
- Referenzsemantik → Allokation mit new
- Konstruktoren, Überladen von Methoden
- Datenkapselung: wichtiges Konzept der Programmierung:
  - Invarianten
  - Abstraktion, Verstecken der Implementierungsdeatails
- Zugriff zu den Klassenvariablen via (implizitem) `this`-Parameter
- Keine Referenzparameter in Java. Wegen Referenzsemantik von Variablen werden Variablen vom Klassentyp trotzdem als Referenzen übergeben.

```
class Rational {
    int numerator, denominator;

    public Rational(int num, int
    denom){ ... }
    public Rational(){ ... }

    public int Numerator(){
        return numerator;
    }

    public int Denominator(){
    return denominator;
    }

    ...
}

int n = r.Numerator();
int d = r.Denominator();
```

3. Fallstudie: Online-Statistik

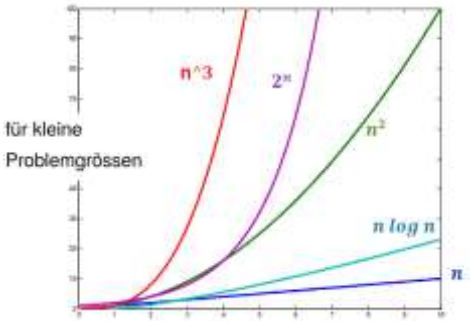
- Objekt zur schnellen Berechnung Mittelwert / Varianz / Median von Daten
- Konzept für begrenzte Datenmenge: zirkulärer Puffer, Begrenzung aufheben: dynamisches (wachsendes) Array
- Algorithmus auf dem Puffer: O(n) bei jeder Berechnung.
- Besserer Algorithmus: Provisional Means. Update-Schritt: O(1).
- Median: Berechnung komplizierter. Auswahlproblem.

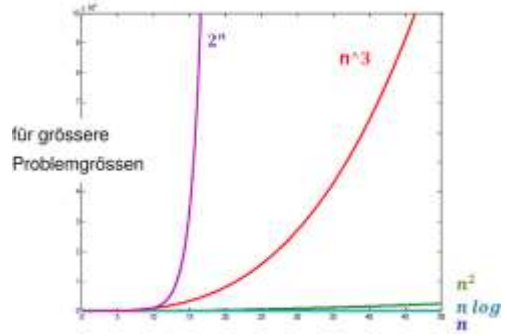




4. Komplexität

- Algorithmen verbrauchen Resourcen: Prozessorzeit, Speicher, Energie
- Betrachten Problemgrössen n und wollen von unwesentlichen Details abstrahieren.
- Betrachten Fälle wir "bester", "schlechtester", "im Mittel" und betrachten Aufwand für wachsende n
- O(g) Notation
- Kategorien: konstant, logarithmisch, linear, loglinear, quadratisch, polynomial, exponentiell
- Wenn eine Maschine 10 mal schneller wird, was passiert mit der berechenbaren Grössenordnung bei gegbenem Zeitlimit?

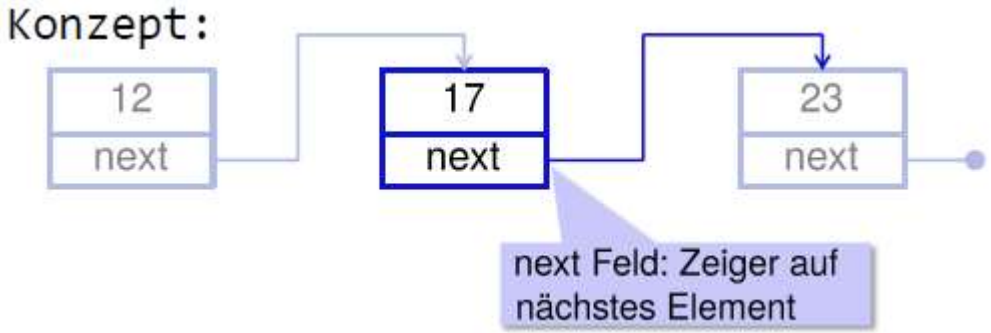




f(n)	alter und neuer Problemumfang
$\log n$	$N \rightarrow N^{10}$
n	$N \rightarrow 10\,N$
$n \log n$	$N \rightarrow$ fast $10N$ $(\rightarrow 10N$ für $N \rightarrow \infty)$
n^2	$N \rightarrow \sqrt{10}\,N \sim 3.16\,N$
n^3	$N \rightarrow 10^{\frac{1}{3}}N \sim 2.15\,N$
2^n	$N \rightarrow N + \log_2 10 \sim N + 3.3$

5. Dynamische Datenstrukturen

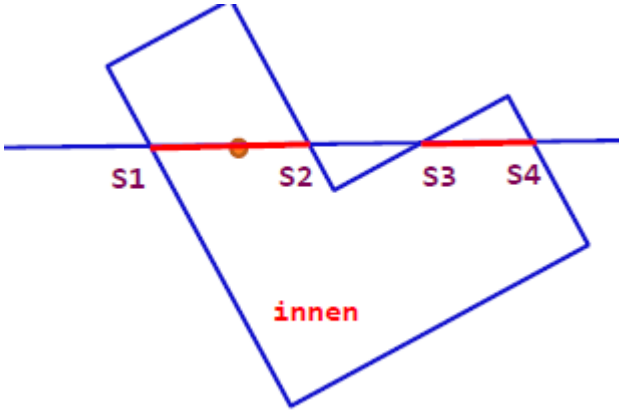
- Motivation: Aufwand beim Einfügen oder Löschen von Einträgen in Arrays
- Verkettete Listen: Speichere Nutzdaten zusammen mit eine Zeiger auf das jeweils nächste Element.

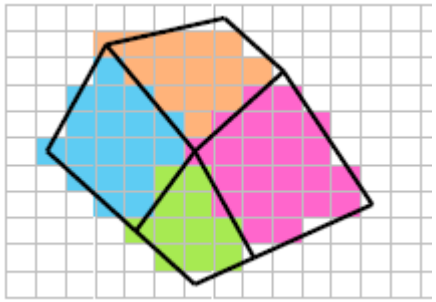


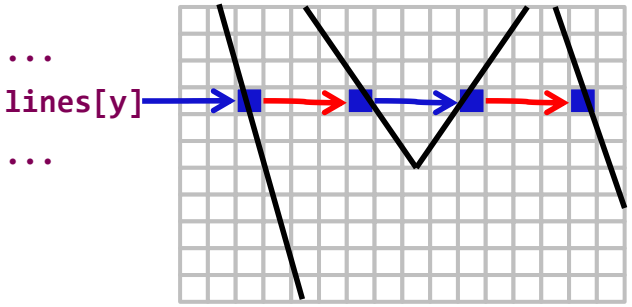
- Implementation von Stack und Queue als Klasse mit next Feld
- Beim Einfügen / Löschen müssen Spezialfälle berücksichtigt werden, wie z.B. leere Liste
- Traversieren durch benutzen eines "running pointers". Beim sortierten Einfügen: halte Referenz zum vorigen Element beim Suchen nach Einfügeposition

5. Fallstudie: Point-In-Polygon

- Konsequenz des Jordan-Theorems: Identifiziere ob Punkt p innen liegt durch Schnitt einer Halbgeraden von p mit Polygon
- Implementation etwas "tricky" wegen der Spezialfälle. Diskretisierungseffekte und Eckpunkte.
- Erste Implementation mit Fliesskommaarithmetik
- Bresenham Algorithmus zum Linien Zeichnen
- Repräsentation des Polygons als Array von verketteten Listen von Schnittpunkten. Füllen der Datenstruktur mit Bresenham.

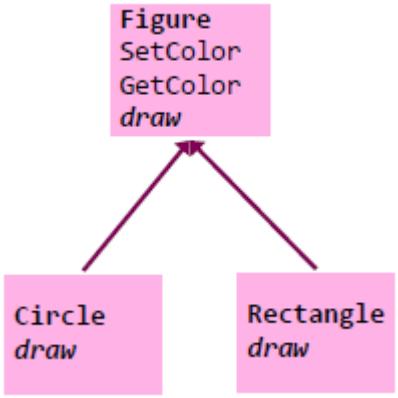






6. Objektorientierte Programmierung

- Motivation: Graphikbibliothek bauen
  - 1. Versuch: Konventionelle Programmierung ohne Vererbung / Polymorphie
  - Probleme: Abundanz, Adminstrativer Overhead, keine Erweiterbarkeit
- Teil 1 der Lösung: Vererbung
  - Generalisierung: Gemeinsame Eigenschaften in der Basisklasse. Spezialisierung: abweichende Eigenschaften in der abgeleiteten Klasse
  - Kompatibilitätsregel: erweitertes Objekt da möglich, wo Basisobjekt gefordert. Umgekehrt nicht.
  - Typcheck und Typtest
- Teil 2 der Lösung: Polymorphie
  - der dynamische Typ des Objektes entscheidet über die auszuführende Methode (zur Laufzeit)




```
public class TestDynamicDispatch {

    public static void main(String[] args) {
        int width = 400;
        int height = 300;
        BufferedImage img = new BufferedImage(width, height, BufferedImage.
        ImageViewer panel = new ImageViewer(img);

        Figure f1 = new Rectangle(100,100,200,100);
        Figure f2 = new Circle(100,100,50);
        f1.draw(img); // wählt draw von Rectangle
        f2.draw(img); // wählt draw von Circle

        panel.repaint();
    }
}
```

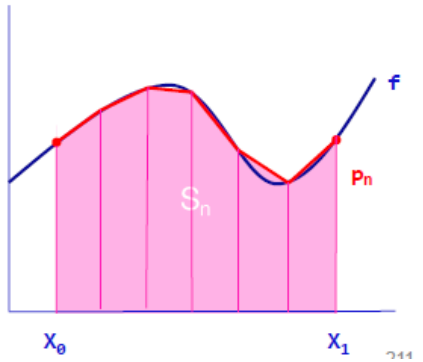
6. Fallstudie: Numerische Integration

- Ziel: generisches Framework zur numerischen Integration beliebiger reellwertiger Funktionen
- Abstrakte Klassen `Function` und `Integrator`
- Spezialisierung mit verschiedenen Funktionen und Integratoren
  - Parabel, Dichte Normalverteilung
  - Rechteck-, Trapezoidal-, Simpson- und Monte-Carlo Integrator
- Konnten die theoretischen Fehlerterme $O(\Delta^3)$ und $O(\Delta^5)$ bestätigen

```
public abstract class Function {

     public abstract double Evaluate(double x);

}
```

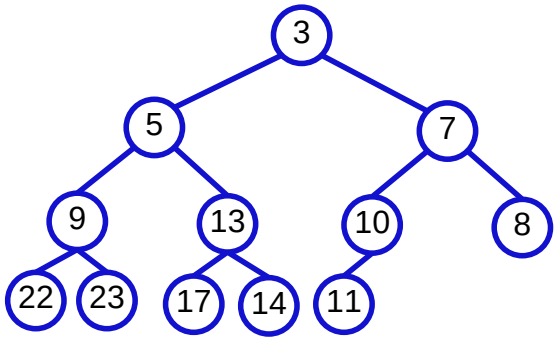
Abstrakter Integrierer:

```
public abstract class Integrator {

     int n;

     public void SetNumberPieces(int pieces) {

          n = pieces;

     }

     public abstract double

               Integrate(Function f, double x0, double x1);

}
```



7. Dynamische Datenstrukturen II: Bäume

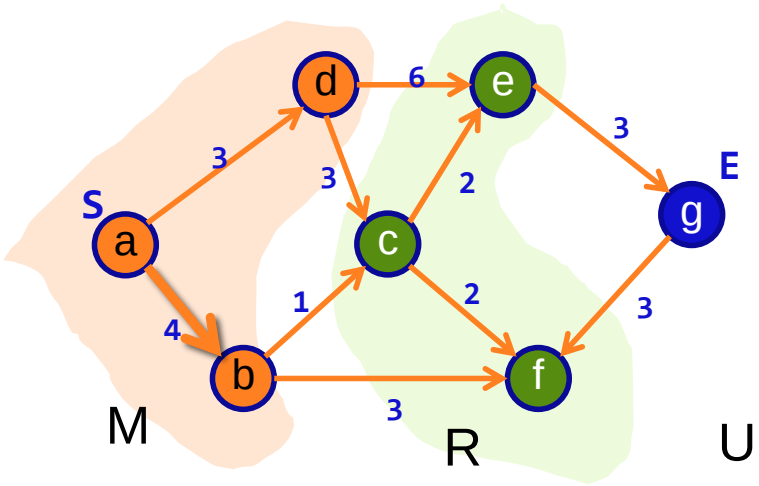
- Binärer Suchbaum: Binärer Baum mit Ordnung auf den Knoten $key_{x.left} < key_x < key_{x.right}$
  - Einfügen: Traversierung + Knoten anhängen
  - Löschen: Element durch symmetrischen Nachfolger ersetzen
- Suchbaum kann zu Liste degenerieren → Worst case Aufwand O(n) → Balancierte Bäume: AVL Trees mit Balancierung bei Update (→Rotationen)
- Min-Heap: Baum mit Heap-Eigenschaft
  - Kann als Array abgebildet werden
  - Einfügen: Element am Ende, Aufsteigen
  - Löschen der Wurzel: Ersetzen durch letztes Element, absinken



- Median eines Datensatzes mit Worst-case Update O(log n) durch Min- und Max-Heap um den Median herum

7. Fallstudie: Dijkstra's Kürzeste Wege

- Gegeben gerichteter Graph mit positiven Gewichten
- Frage: kürzester Weg von S nach E?
- Dijkstra: Betrachte im iterativen Algorithmus grundsätzlich nur kürzeste Wege → Garantie, dass jeder Pfad immer minimale Länge hat
- Mengen M, R und U
- Algorithmus: M={S}, dann iterativ
  - Nehme zu M Element aus R mit kürzestem Weg hinzu
  - Update von R gemäss M
  - Wiederhole bis E in M



- Implementation: R als MinHeap
- Vorläufiger Nachteil: "DecreaseKey" Operation in R noch nicht O(log n), da Element gefunden werden muss.

8. Tabellen und Hashing

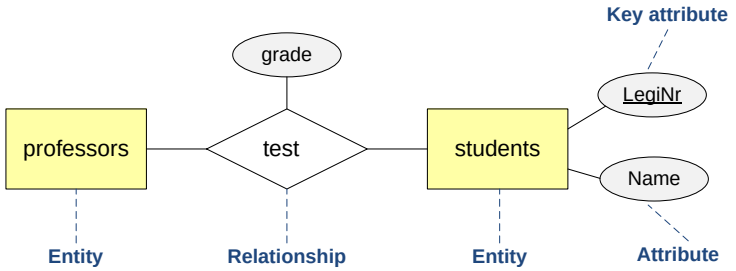
- Hash Table: Datensätze so speichern, dass man sie schnell nach Schlüssel finden kann.
- Idee: Speichere Referenz zu den Daten im Array. Berechne den Array Index zu einem Schlüssel k mit einer Hashfunktion.
- Einfache Hashfunktion $\mathrm{h}\colon \mathbb{N} \to \{0, \ldots, n-1\}$:

$$h(k) = k \bmod n$$

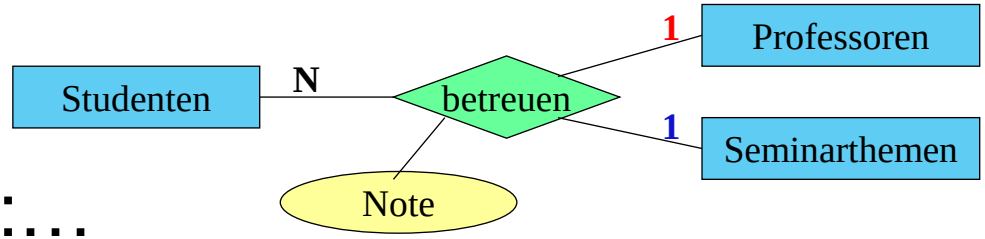
- Kollisionenbehandlung
  1. Array von verketteten Listen
  2. Offene Adressierung: nimm einen anderen freien Index (z.B. linear). Löschproblematik -> Stellvertreter für "gelöscht"
- Dynamisch wachsende Hashtabellen um den Belegungsgrad (und somit Kollisionswahrscheinlichkeit) zu begrenzen.

9. Datenbankysteme: ER Modell

- Konzeptuelle Modellierung der Miniwelt für eine Datenbank
- Entitäten, Beziehungen, Attribute, Schlüssel, Rollen



- Funktionalitäten



1:1, 1:N, N:M, 1:N:M:…
darstellbar als Funktion in Richtung der Eins

- Schwache Entitäten: können nicht ohne die starke Entität "leben". Immer 1:N oder 1:1

10. DB: relationales Modell

- Relationen $R \subseteq D_1$ x ... x D_n
- Schemata
  Telefonbuch: {[*Name: string, Strasse: string, Telefon#:integer*]}
- ER →Relationales Modell
  - Regel 1: Entitäten (Attribute →Attribute, Schlüssel → Schlüssel)
  - Regel 2: Beziehungen (alle Enittäen, Schlüssel = alle Schlüssel)
  - Regel 3: Zusammenfassen (nur) dort wo dieselben Schlüssel verwendet sind
- Relationale Algebra
  - σ Selektion
  - π Projektion
  - x kartesisches Produkt
  - ⋈ Join (Verbund)
  - ρ Umbenennung

11. DB: SQL

- Datendefinition, Datenmodifikation und Abfrage. Datentypen
- Einfache Queries

```
select s.Name, v.Titel
from Studenten s, hören h, Vorlesungen v
where s. Legi= h.Legi and h.VorlNr = v.VorlNr
```

- Aggregatsfunktionen sum, max etc. und Gruppierung
- Verschachtelte Abfragen, Korrelationen
  - exists, in, all

12. DB: Programmierschnittstelle

- JDBC: Datenbankzugriff aus Java
- Installation des Treibers, Öffnen der Datenbank
- SQL – Java
  - Queries als Strings, die der Datenbank übermittelt werden, nicht zur Compilezeit geprüft → Exception handling zum Abfangen vieler möglicher Laufzeitfehler
  - Resultat: ResultSet, cursor Ansatz
- Prepared Statements zum Parametrisieren von Anfragen. Kompaktheit und Effizienz.