# Heute

- Organisatorisches:
  Umstellung Studienplan
- Evaluierung
- Nachbesprechung Abschlusstest
- Einführung Rekursion

# Organisatorisches

- Neuer Studienplan ab WS 20/21:

  - VO+UE: IT I und VO+UE: IT II werden ersetzt durch

    VO: OOP (2st, 3.Sem.)

    UE: OOP A (2st, 3.Sem.)

    UE: OOP B (2st, 5.Sem.)

    PS: OOP (1st, 5.Sem.)

- Umstieg nach IT I:

  PS: OOP und UE: OOP B im 5.Sem.

# Organisatorisches

Zur Erinnerung:

VO: IT I wird evaluiert

# Nachbesprechung Abschlusstest

**Bankomatkarte:**

Aufruf Konstruktor der Oberklasse über

```
super(alteKarte.getKontonr(),
      alteKarte.getInhaber() );
```

# Nachbesprechung Abschlusstest

**Bankomatkarte:**

- Um alte Bankomatkarte ungültig machen zu können, braucht man Attribut, z.B.

*private Bankomatkarte alte;*

- Aufruf *macheUngueltig()* für Objekt *alte*
  (Aufruf *super.macheUngueltig()* würde neue Karte ungültig machen!)

# Nachbesprechung Abschlusstest

- Bankomatkarte:

  - Um alte Bankomatkarte ungültig machen zu können, braucht man Attribut, z.B.

    *private Bankomatkarte alte;*

  - Aufruf *macheUngueltig()* für Objekt *alte* (Aufruf *super.macheUngueltig()* würde neue Karte ungültig machen!)

# Rekursion

Mit Hilfe von Rekursion können einige Problemstellungen sehr einfach programmiert werden:

- Schnelle Sortieralgorithmen beruhen auf Rekursion.
- Auch kombinatorische Optimierungsprobleme werden oft rekursiv gelöst.

# Rekursive Methoden

- **Direkte Rekursion:**
  - Eine Methode ruft sich selbst auf.
- **Indirekte Rekursion:**
  - Eine Methode ruft eine zweite Methode auf, diese ruft wieder die erste Methode auf.
  - (Die zweite Methode könnte auch eine dritte Methode aufrufen, und diese wieder die erste. …)
- Damit es nicht zu einer endlosen Folge von Aufrufen kommt, muss es einen Basisfall geben, der ohne rekursiven Aufruf gelöst wird.
- Dieser Basisfall muss immer erreicht werden:
  - Es gibt bei jedem rekursiven Aufrufe einen Fortschritt in Richtung Basisfall.

# Teilsummenproblem

- Gegeben sei Menge oder Liste von ganzen Zahlen $A=\{a_1,a_2,\ldots,a_n\}$ und eine ganze Zahl $S$.
- Gibt es eine Teilmenge von $A$, die zu exakt $S$ aufsummiert?

- Z.B.: Kann Summe $S$ mit vorgegebenen Münzen genau bezahlt werden?

# Teilsummenproblem

- Gegeben sei Menge oder Liste von ganzen Zahlen $A=\{a_1,a_2,\ldots,a_n\}$ und eine ganze Zahl $S$.
- Suche Teilmenge von $A$, sodass Summe der Elemente maximal und höchstens $S$.

- Z.B.: Möchte Lkw mit gegebener Kapazität $S$ maximal beladen.

# Rucksack Problem

**Verallgemeinerung des Teilsummenproblem:**

- Mehrere Gegenstände stehen zum Transport zur Verfügung.
- Jeder Gegenstand hat einen Wert und ein Gewicht.
- Nur ein fixes Gesamtgewicht kann transportiert werden.
- Welche Gegenstände sollen transportiert werden, damit der Gesamtwert maximal ist?

# Kombinatorische Optimierungsprobleme

- Das Teilsummen- und das Rucksack-Problem gehören zu einer Klasse kombinatorischer Optimierungsprobleme (**NP-vollständige Probleme**), die im allgemeinen nur mit großem (Rechen-)Aufwand optimal gelöst werden können.
- Dabei sind diese Probleme relativ gutartig, da sie für typische Inputs recht schnell optimal oder fast optimal gelöst werden kann.

# Exhaustive Search

- Ausprobieren aller Möglichkeiten und berechnen der besten.
- Für Teilsummen- und Rucksack-Problem kann das mit einer rekursiven Methode sehr einfach programmiert werden.
- *Exhaustive Search* ist dabei für eine größere Anzahl von Gegenständen aber sehr langsam.

# Dynamisches Programmieren

- Löse "kleinere" Probleme und benutze Lösungen, um Lösung eines größeren Problems zu finden.
- Typischerweise wird das rekursiv gemacht.

- ***Beispiel:*** *Kürzester Weg von A nach Z (über mehrere Orte):*

  *- Gehe zunächst von A nach B.*

  *- Nimm kürzesten Weg von B nach Z (Rekursion!).*

  *- Wähle für B jenen Ort, sodass Distanz AB + BZ minimal.*

# Rekursion oder Schleife?

- Im Prinzip kann jede rekursive Methode in eine iterative Methode (d.h. eine Methode mit einer Schleife) umgeschrieben werden (und umgekehrt).
- Das kann sehr einfach und naheliegend sein:
  - Z.B. Binäres Suchen
  - (Head- oder Tail-Rekursion)
- Oder das kann recht mühsam sein, obwohl die rekursive Methode recht einfach ist:
  - Z.B. Exhaustive Search beim Rucksack-Problem

# Rekursionsberechnungen beschleunigen

- Berechnungen, die kein besseres als das bisherige Ergebnis liefern können, werden abgebrochen.

  - Ist z.B. bei der Berechnung des kürzesten Weges von A nach Z der Weg von A nach B länger als der bisher gefundene kürzeste Weg, muss B nicht berücksichtigt werden.

- Mehrfachberechnungen können durch Speichern der Zwischenergebnisse vermieden werden.
  - z.B. kann man bei der Berechnung des kürzesten Weges von A nach Z bereits gefundene kürzeste Wege von anderen Punkten P nach Z zwischenspeichern.