Objektorientierung – ein kurzer Ausblick

Einführung in die Programmierung 1

Wintersemester 21/22

Überblick

- Objektorientierung
- Datenstrukturen und Objektorientierung

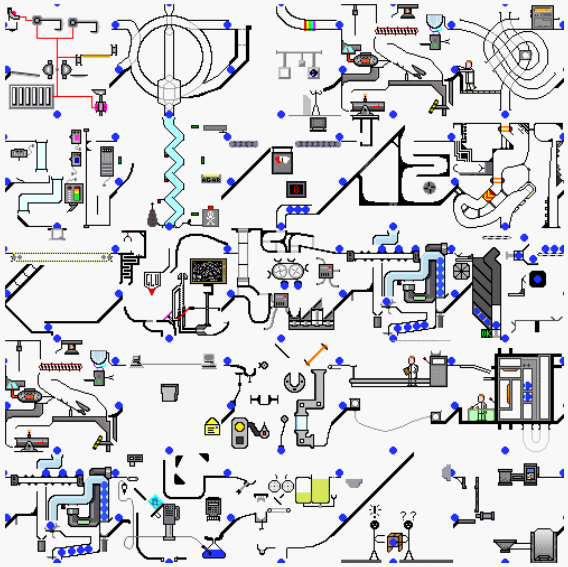
Objektorientierung

Motivation

- Was wurde bisher besprochen?
  - Variablen, Anweisungen, Arrays, Methoden
- Was kann damit gemacht werden?
  - Algorithmen implementieren
  - Kleinere Programme schreiben
- Große Programme?
  - Ja, aber nicht sinnvoll

Komplexität

- Komplizierte Algorithmen versus komplexe Software
- Probleme bei der Softwareentwicklung
  - Viele Funktionen müssen organisiert werden
  - Anforderungen an die Software können sich ändern



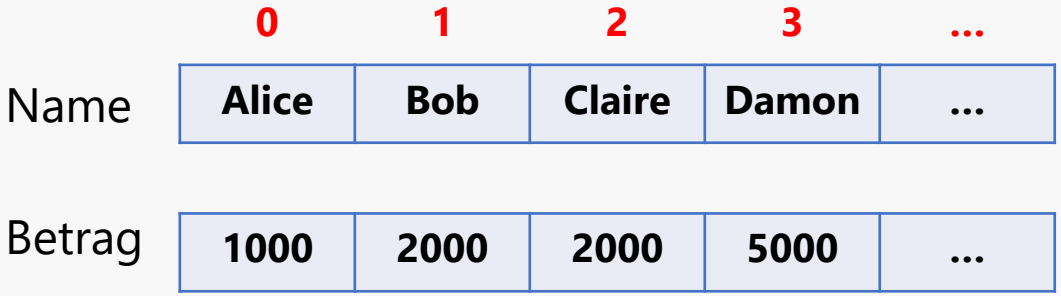
Lösung?
Objektorientierung!

Objektorientierung – Beispiel

- Einfaches Bankkonto
  - Name und Betrag sollen pro Person gespeichert werden
  - Bestimmter Name und bestimmter Betrag gehören zusammen
- Beispiel

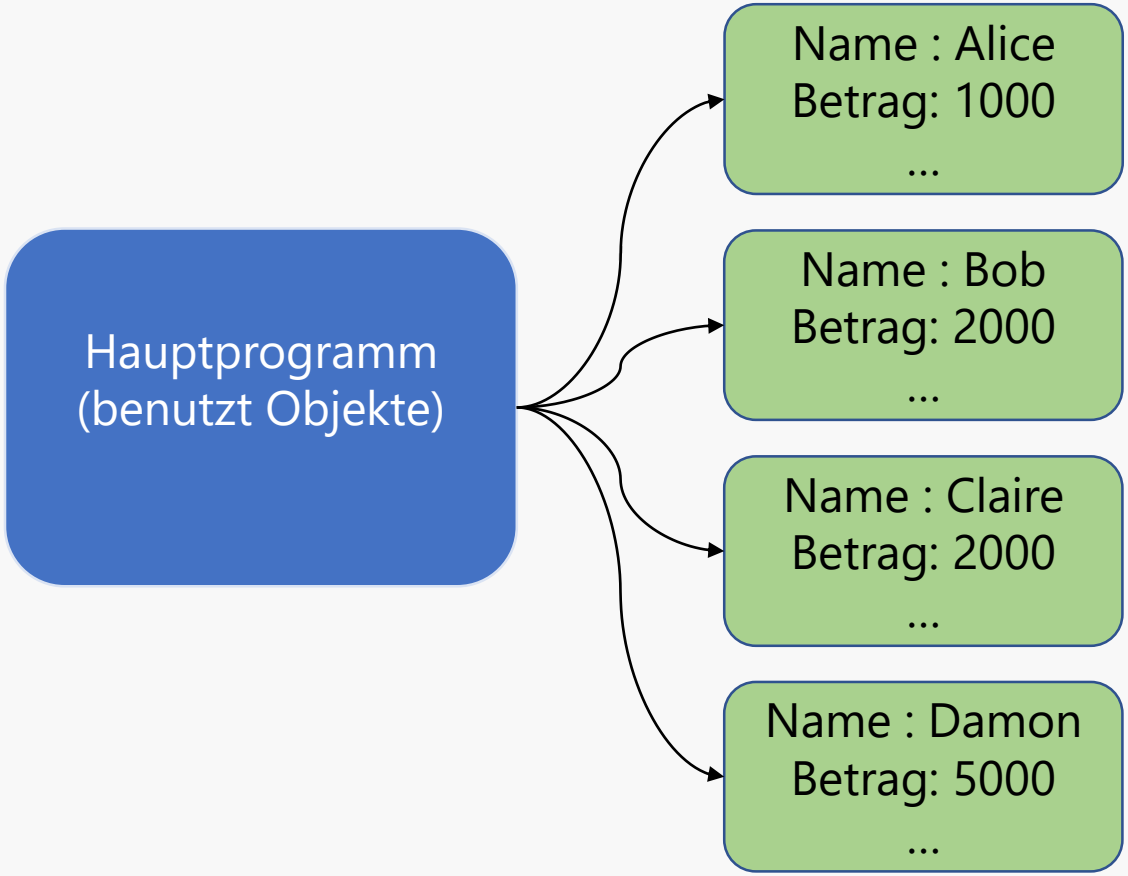
Sehr schlechte Lösung mit Arrays

- 2 Arrays für die Daten
  - Mehrere Methoden operieren darauf



- Daten von Alice? Index 0 in beiden Arrays
- Einfügen bzw. Löschen?
  - 2 Arrays an denselben Indexpositionen verändern

Andere Sichtweise – Objekte



Objekt

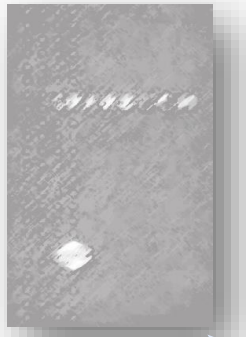
- Objekt ist eine Datenkapsel
  - Objekt bündelt Daten (Zustände)
- Objekt bietet Operationen an
  - Spezifizieren, welche Funktionalität ein Objekt bereitstellt
  - Schnittstelle = Menge der Operationen eines Objekts
  - Realisierung von Operationen in Java durch Methoden

Klasse

- Eine Klasse beschreibt die gemeinsamen Eigenschaften und Operationen einer Menge von gleichartigen Objekten
  - Struktur und Verhalten der Objekte einer Klasse sind gleich (unterscheiden sich nur in den Zuständen)
  - Klassen stellen daher einen **Konstruktionsplan** für Objekte dar

Beispiel – Klasse Bankkonto

Klasse

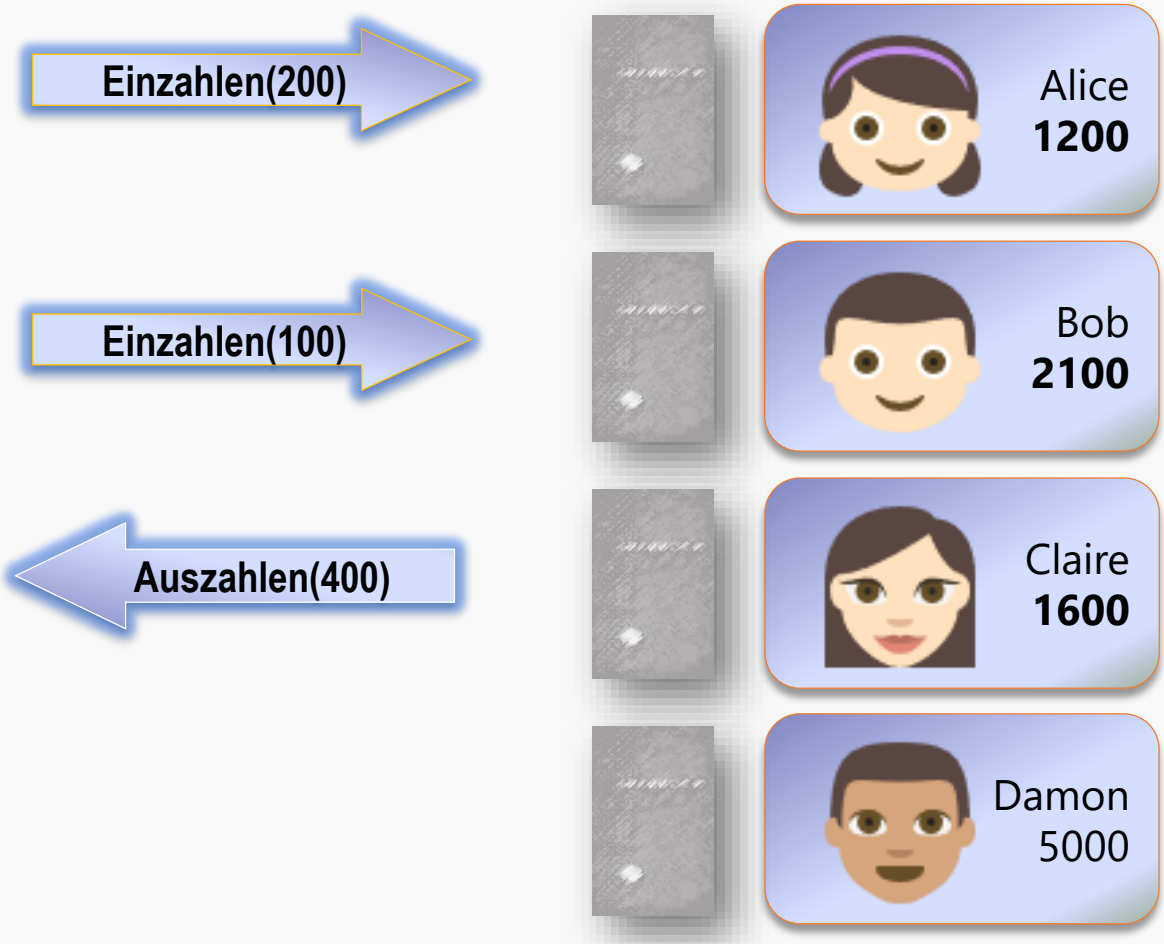


Eigenschaften:
Name
Betrag

Operationen:
Einzahlen
Auszahlen
Ausgabe des Inhalts

Objekte

Beispiel (Operationen)



Beispiel Bankkonto

- Einfacher Ansatz für ein Bankkonto

```
public class Account {
    private String name;
    private int amount;
    //...
}
```

Variablen werden als **Attribute**, **Membervariablen**, **Instanzvariablen** oder **Objektvariablen** bezeichnet. Sollten immer `private` sein!

Klasse als Datentyp

- Klassen sind selbst definierte Datentypen
- Erlauben mehrere Elemente zu einem neuen Element zusammenzufassen und unter einem gemeinsamen Namen anzusprechen
- Können analog zu den eingebauten Datentypen verwendet werden

Klasse als Datentyp – Beispiel

- Klasse Account

```
Account x, y;
```

  - Damit sind x und y vom Typ `Account`
  - x und y sind Referenzen

Objekterzeugung

- Erzeugung mit new

```
x = new Account();
y = new Account();
```

  - x und y **zeigen nun** auf **Account – Objekte**
  - Beide Objekte haben die **gleiche Struktur**, aber eine **unterschiedliche Identität** (wird intern verwaltet)

Identität

- Objekte haben immer eine eigene Identität
- Zwei Objekte können dadurch immer unterschieden werden, auch wenn sie den gleichen Zustand haben
- Allgemeines Beispiel
  - Datei A und Datei B haben gleichen Inhalt (Zustand) sind aber trotzdem unterschiedlich

Automatische Initialisierung

- Objektvariablen werden **automatisch** initialisiert

`int`	`0`
`double`	`0.0`
`boolean`	`false`
`char`	`\u0000`
`Referenztypen`	`null`

- Objektvariablen können aber auch wie lokale Variablen initialisiert werden

Konstruktoren

- Initialisierungsmethoden, die bei der Erzeugung des Objekts aufgerufen werden
  - Eine Klasse kann mehrere Konstruktoren haben
  - Objektvariablen können damit mit sinnvollen Werten initialisiert werden
- Konstruktor
  - Hat den gleichen Namen wie die Klasse
  - Ist eine Methode ohne Rückgabetyp
  - Kann aber Parameter haben

Konstruktor – Parameter

- Konstruktoren mit unterschiedlichen Parametern (überladen!)
  - Übergabe von Werten (für die Initialisierung)
  - Beispiele für die Klasse Account

```
public Account(String name, int amount){…}
public Account(String name){…}
public Account(int amount){…}
```

this-Referenz

- `this`-Referenz
  - Jedes Objekt hat eine `this`-Referenz
    - Referenz auf sich selbst
  - Ist in jeder Methode automatisch vorhanden
  - Damit können Objektvariablen von lokalen Variablen unterschieden werden
    - Objektvariablen und lokale Variablen können den gleichen Bezeichner haben
- Siehe nachfolgendes Beispiel!
  - Weitere Operationen (Einzahlen, Auszahlen, Ausgabe) durch einfache Methoden realisiert

Beispiel (Account)

```java
public class Account {
    private String name;
    private int amount;

    public Account(String name, int amount) {
        this.name = (name != null) ? name : "";
        this.amount = Math.max(amount, 0);
    }

    public void payInto(int amount) {
        if (amount > 0) {
            this.amount += amount;
        }
    }

    public void drawOut(int amount) {
        if (amount > 0 && amount <= this.amount) {
            this.amount -= amount;
        }
    }

    public String print() {
        return "Account{" + "name='" + name + "'" + ", amount=" + amount + '}';
    }
}
```

Klasse Account

- Verwaltet ein Konto
- Operationen (Methoden)
  - Account erzeugen
  - Einzahlen
  - Auszahlen
  - Ausgabe des Inhalts

Beispiel (AccountClient)

```
public class AccountClient {
    public static void main(String[] args) {
        Account account1, account2, account3, account4;
        account1 = new Account("Alice", 1000);
        account2 = new Account("Bob", 2000);
        account3 = new Account("Claire", 2000);
        account4 = new Account("Damon", 5000);
        System.out.println(account1.print());
        System.out.println(account2.print());
        System.out.println(account3.print());
        System.out.println(account4.print());
        account1.payInto(1000);
        System.out.println(account1.print());
        account1.drawOut(100);
        System.out.println(account1.print());
        account2.drawOut(2000);
        System.out.println(account2.print());
        account2.drawOut(2000);
        System.out.println(account2.print());
        System.out.println(account3.print());
        System.out.println(account4.print());
    }
}
```

Klasse AccountClient

- Einfaches Testprogramm
- Legt Kontos an
- Führt Operationen aus

```
Account{name='Alice', amount=1000}
Account{name='Bob', amount=2000}
Account{name='Claire', amount=2000}
Account{name='Damon', amount=5000}
Account{name='Alice', amount=2000}
Account{name='Alice', amount=1900}
Account{name='Bob', amount=0}
Account{name='Bob', amount=0}
Account{name='Claire', amount=2000}
Account{name='Damon', amount=5000}
```

Hinweis zur Ausgabe (1)

- Ausgabe bei einer Klasse kann anders implementiert werden – Aufrufe werden dann kürzer
- Implementierung der Methode toString

```
public class Account {
    private String name;
    private int amount;

    …
    public String toString() {
        return "Account{" + "name='" + name + "'" + ", amount=" + amount + '}';
    }
}
```

Gleiche Implementierung wie `print`, aber anderer Name!

```
public class AccountClient {
    public static void main(String[] args) {
        Account account1, account2, account3, account4;
        account1 = new Account("Alice", 1000);
        System.out.println(account1);
    }
}
```

`toString` muss nicht explizit aufgerufen werden – `println` ruft diese Methode „automatisch“ auf.

Hinweis zur Ausgabe (2)

- Jede Klasse in Java erbt von der Klasse `Object`
  - Neue Klasse übernimmt Implementierung (Variablen, Methoden)
    - Einschränkungen dazu werden in EP2 noch genauer besprochen
  - Neue Klasse kann zusätzliche Variablen und Methoden festlegen
- In `Object` ist `toString` sehr einfach implementiert
  - Kann in jeder Klasse, die von `Object` erbt, benutzt werden
  - Kann in einer neuen Klasse (wie Account) neu implementiert werden (die Methode wird „überschrieben“) und dann wird bei Objekten dieser Klasse die neue Implementierung benutzt

Default-Konstruktor

- Parameterloser Konstruktor

```
public Account(){…}
```

  - Wenn bei einer Klasse kein Konstruktor angegeben wird, dann fügt Java **automatisch** einen parameterlosen leeren Konstruktor (Default-Konstruktor) ein
- Sobald ein Konstruktor angegeben wird, wird kein Default – Konstruktor automatisch erzeugt
  - Wenn ein Default-Konstruktor benötigt wird, muss dieser explizit ausprogrammiert werden
  - Beispiel

```
public Account(){ }
```

Beispiel (Rational)

- Klasse für positive Bruchzahlen
  - Bruch hat Zähler und Nenner
    - Einzeln speichern
  - Es werden mehrere Operationen (+, -, *, /) auf Brüche unterstützt
    - Mit Hilfe von Methoden implementiert
- Code in TUWEL
  - `Rational.java`
  - `RationalTester.java`

Statische Komponenten

- Wiederholung
  - Objektvariablen und Methoden können auch mit dem Schlüsselwort `static` gekennzeichnet werden - Unterschied?
- Klasse ist auch ein Objekt
  - Klassenobjekt ist eine Schablone, die das Aussehen der Klasse festlegt
  - Sowohl das Klassenobjekt als auch die Objekte können Variablen und Methoden haben
    - Mit `static` – gehören zum Klassenobjekt (statische Komponenten)
    - Ohne `static` – gehören zu den Objekten (objektbezogene Komponenten)

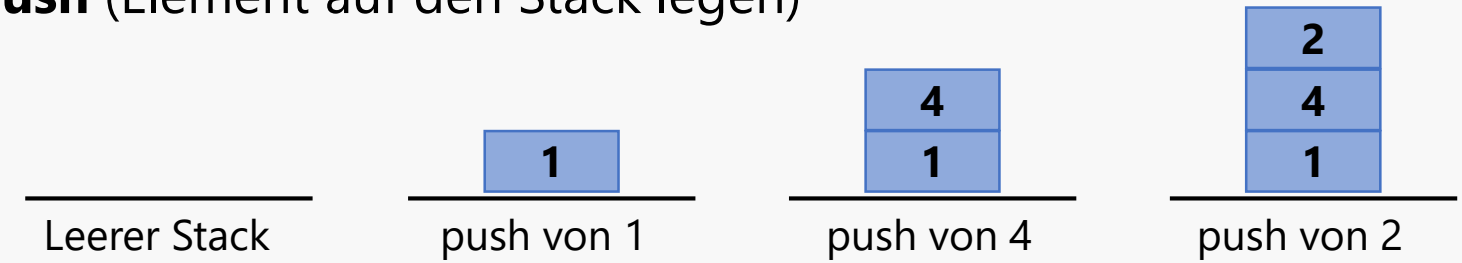
Weitere Beispiele

- Unterschiedliche Ebenen
  - Klassen benutzen
  - Einfache Klassen selbst implementieren
  - Design von Klassen
- Weitere Beispiele dazu auf https://introcs.cs.princeton.edu/java/30oop/

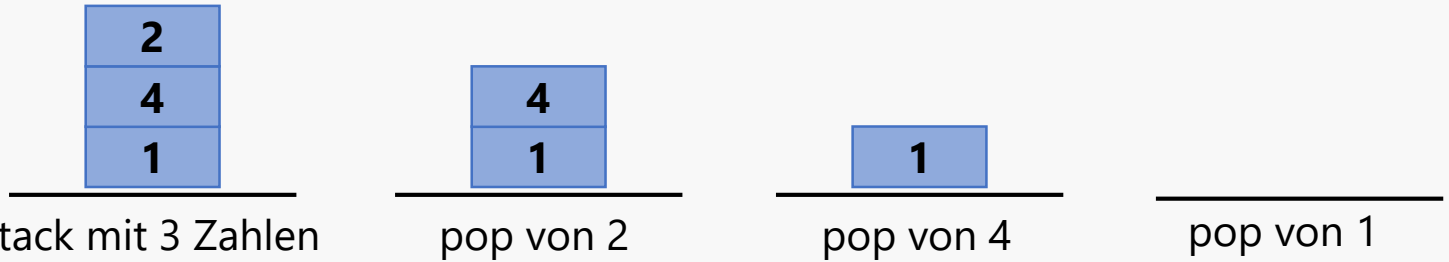
Datenstrukturen und Objektorientierung

Stack (Wiederholung)

- Einfache Datenstruktur
- Operationen (z. B. hier für einen Stack von Zahlen jeweils nach der Operation)
  - **push** (Element auf den Stack legen)



  - **pop** (oberstes Element vom Stack nehmen)



Stack als Klasse (Stack für Strings)

```
public class ArrayStackOfStrings {
    private String[] items;
    private int n;

    public ArrayStackOfStrings(int capacity) {
        items = new String[capacity];
    }

    public boolean isEmpty() {
        return n == 0;
    }

    public boolean isFull() {
        return n == items.length;
    }

    public void push(String item) {
        items[n++] = item;
    }

    public String pop() {
        return items[--n];
    }
}
```

Daten in einem Array speichern

Einfache Implementierung ohne Fehlerbehandlung!

Testklasse für Stack-Klasse

```
public class ArrayStackOfStringsTester {

    public static void main(String[] args) {
        ArrayStackOfStrings stack = new ArrayStackOfStrings(3);
        stack.push("Hello");
        stack.push("World");
        stack.push("EP1");
        System.out.println(stack.isEmpty());
        System.out.println(stack.isFull());
        while(!stack.isEmpty()) {
            System.out.println(stack.pop());
        }
    }

}
```

```
false
true
EP1
World
Hello
```

Adaptiver Stack

- Adaptiver Stack
  - Bei Überlauf automatisch Stack vergrößern
  - Bei wenig Elementen Stack verkleinern (freie Plätze reduzieren)
- Code in TUWEL
  - `ResizingArrayStackOfStrings.java`
  - `ResizingArrayStackOfStringsTester.java`

Beispiel für Benutzung der Stack-Klasse

- Auswertung von einfachen arithmetischen Ausdrücken
  - Klammern
  - Operatoren
  - Operanden
- Zwei Stacks
  - Einer für Operatoren
  - Einer für Operanden

Algorithmus

- Aktionen
  - Operand wird auf den Operanden-Stack gelegt
  - Operator wird auf den Operatoren-Stack gelegt
  - Linke Klammer wird ignoriert
  - Rechte Klammer
    - Nimm einen Operator und zwei Operanden von den jeweiligen Stacks
    - Lege das Ergebnis der Operation auf den Operanden-Stack
- Am Ende liegt das Ergebnis am Operanden-Stack
- Code in TUWEL
  - `Evaluate.java`

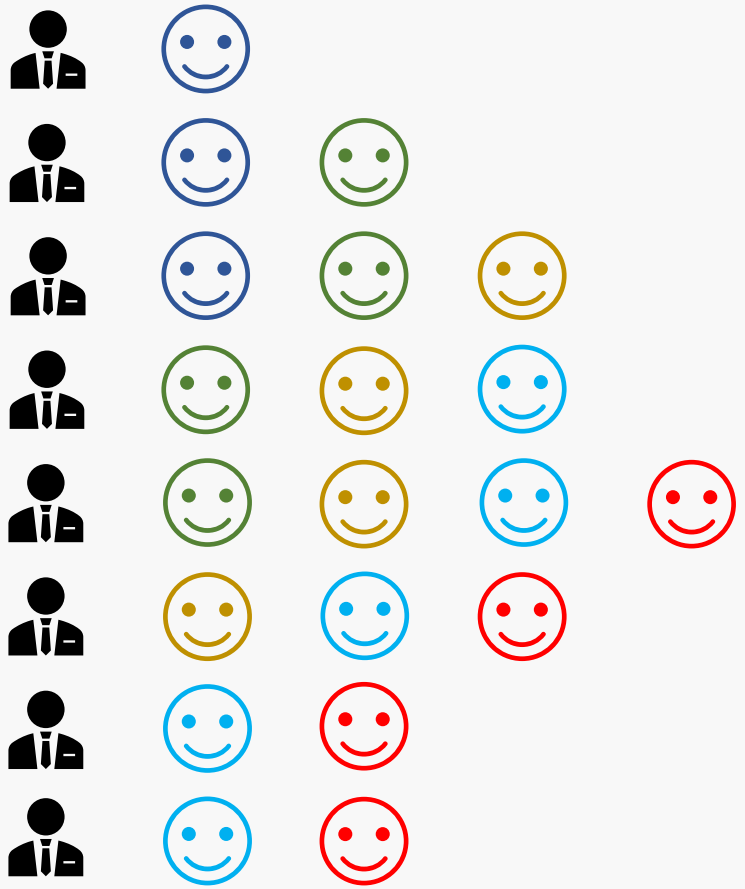
Beispiel

(1 + ((2 + 3) * (4 * 5)))
+ ((2 + 3) * (4 * 5)))
((2 + 3) * (4 * 5)))
+ 3) * (4 * 5)))
3) * (4 * 5)))
) * (4 * 5)))
* (4 * 5)))
(4 * 5)))
* 5)))
5)))
)))
))
)

Operanden	Operatoren
1	
1	+
1 2	+
1 2	+ +
1 2 3	+ +
1 5	+
1 5	+ *
1 5 4	+ *
1 5 4	+ * *
1 5 4 5	+ * *
1 5 20	+ *
1 100	+
101	

Queue (Warteschlange)

- Datenstruktur wie Stack
- Unterschied zum Stack
  - Queue funktioniert nach dem FIFO-Prinzip
  - FIFO = First In First Out
- Beispiel
  - Bankschalter

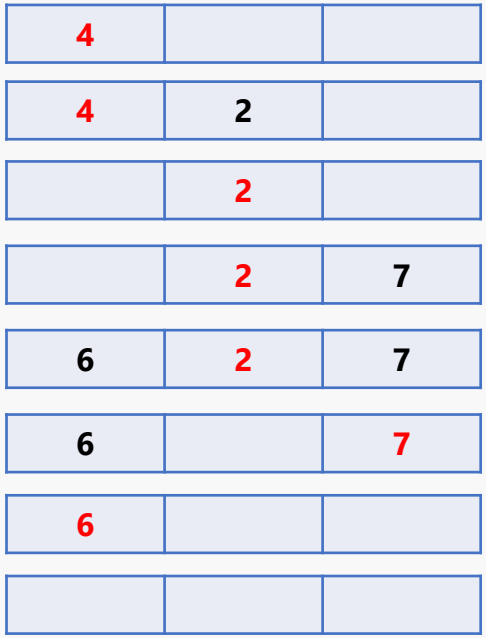


Queue (Implementierung)

- Typische Operationen
  - enqueue = Ein Element in die Queue einfügen
  - dequeue = Ein Element aus der Queue entfernen (zurückliefern)
- Implementierung in TUWEL
  - Queue basierend auf Array
  - Größe wird automatisch angepasst

Beispiel für eine Queue mit Zahlen

- Beispiel (mit 3 Plätzen)
  - enqueue von 4
  - enqueue von 2
  - dequeue
  - enqueue von 7
  - enqueue von 6
  - dequeue
  - dequeue
  - dequeue



Rot = erstes Element in der Queue, das beim nächsten Aufruf von deque entfernt (zurückgeliefert) wird

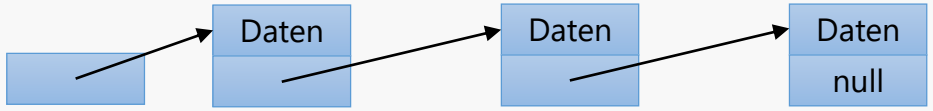
Ordnung des Inhalts: 2, 7, 6

Adaptive Queue

- Wie bei Stack automatisch vergrößern und verkleinern
- Code in TUWEL
  - `ResizingArrayQueueOfStrings.java`
  - `ResizingArrayQueueOfStringsTester.java`

Stack mit Listen (Ausblick auf EP2)

- Datenstruktur Liste
  - Abfolge von Knoten
  - Eigene Klassen für Knotenobjekte
    - Speichert Daten und Referenz auf nächsten Knoten
    - Keine speziellen Methoden
  - Knoten sind miteinander verbunden (über Referenzen)
- Mehr Informationen, Abbildungen, Code: https://introcs.cs.princeton.edu/java/43stack/



Beispiele für Listen in der Java-API

- ArrayList basierend auf Arrays
  - java.util.ArrayList
- LinkedList basierend auf verketteten Listen
  - java.util.LinkedList

Weitere Themen (Ausblick auf EP2)

- Vererbung
  - Mit Hilfe von Interfaces
  - Vererbung der Implementierung
- Ausblick mit Beispielen aus der Literatur

Weitere Themen (Ausblick auf EP2)

- Generische Programmierung
  - Beispiel: Eine Stack-Implementierung für beliebige Datentypen (Referenztypen)
- Ausnahmen
  - Informationen über bestimmte Programmzustände (Fehlerzustände) an andere Programmebenen zur Weiterbehandlung weiterreichen
- Behandlung von Dateien