# 1. Einführung

Quelle: ep2-01_Einführung.pdf
Beinhaltet: Einführung

---

# Klassifizierung und Modularisierung

## 1. Beispielcode

```java
public class TestFactorial {
    public static void main(String[] args) {
        System.out.println(Factorial.fact(3));
    }
}

public class Factorial {
    public static long fact(int n) {
        return n < 2 ? (long)n : n * fact(n - 1);
    }
}
```

## 2. Erklärung des Codes

### TestFactorial.java

- Zweck: Startpunkt der Programmausführung.
- In der `main`-Methode wird die Methode `fact(3)` aus der Klasse `Factorial` aufgerufen.

### Factorial.java

- Methode: `public static long fact(int n)`
  - Berechnet rekursiv die Fakultät von `n`.
  - Bedingung:
    - Wenn $n < 2$, wird $n$ direkt zurückgegeben.
    - Sonst wird $n \cdot fact(n-1)$ rekursiv berechnet.

## 3. Verweise auf Klassen und Methoden

### Verweise auf andere Klassen

- Der Aufruf `Factorial.fact(3)` in `TestFactorial` ist ein Verweis auf eine Methode in einer anderen Klasse.
- Aber Auch `System` und `String[]` sind Verweise auf Methoden von anderen Klassen
  - Klassenname als Präfix: Da `fact` statisch ist, wird sie über den Klassennamen aufgerufen: `Factorial.fact(...)`.

### Verweis auf Methode innerhalb derselben Klasse

- Innerhalb der Klasse `Factorial` ruft `fact` sich selbst auf mit `fact(n - 1)`.
  - Dies ist ein impliziter Verweis auf eine Methode innerhalb derselben Klasse.
  - Da es sich um dieselbe Klasse handelt, ist kein Präfix notwendig.

## 4. Fazit

- Interner Methodenaufruf: `fact(n - 1)` → innerhalb von `Factorial`, daher kein Klassennamen nötig.
- Externer Methodenaufruf: `Factorial.fact(3)` → von `TestFactorial` aus, daher mit Klassennamen.

### Klassifizierung und Modularisierung

```
public class TestFactorial {
    public static void main(String[] args) {
        System.out.println(Factorial.fact(3));
    }
}
```

Verweise auf andere Klassen

```
public class Factorial {
    public static long fact(int n) {
        return n < 2 ? (long)n : n * fact(n - 1);
    }
}
```

Verweis auf Methode innerhalb der Klasse entspricht `Factorial.fact(n - 1)`

# Verwendungsbeispiele abstrakter Datentypen

```
private static void printFifthChar(String s) {
    if (s != null && s.length() > 4) {
        System.out.print(s.charAt(4));
    }
}

private static void echo() {
    Scanner scanner = new Scanner(System.in);
    while (scanner.hasNextLine()) {
        System.out.println(scanner.nextLine());
    }
}
```

**String**

Referenz auf Objekt

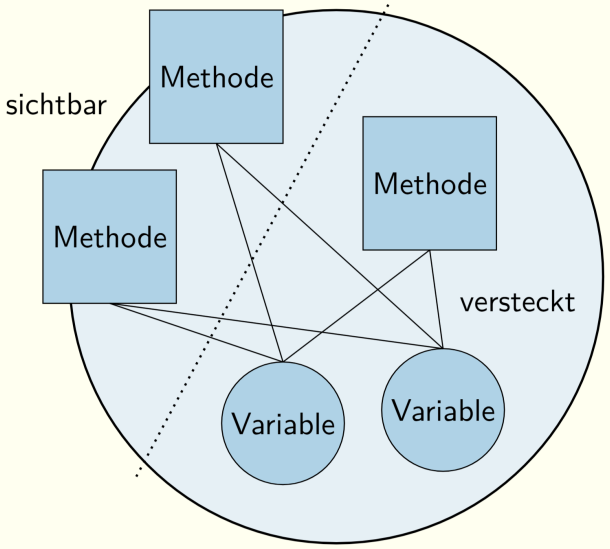
**Scanner**

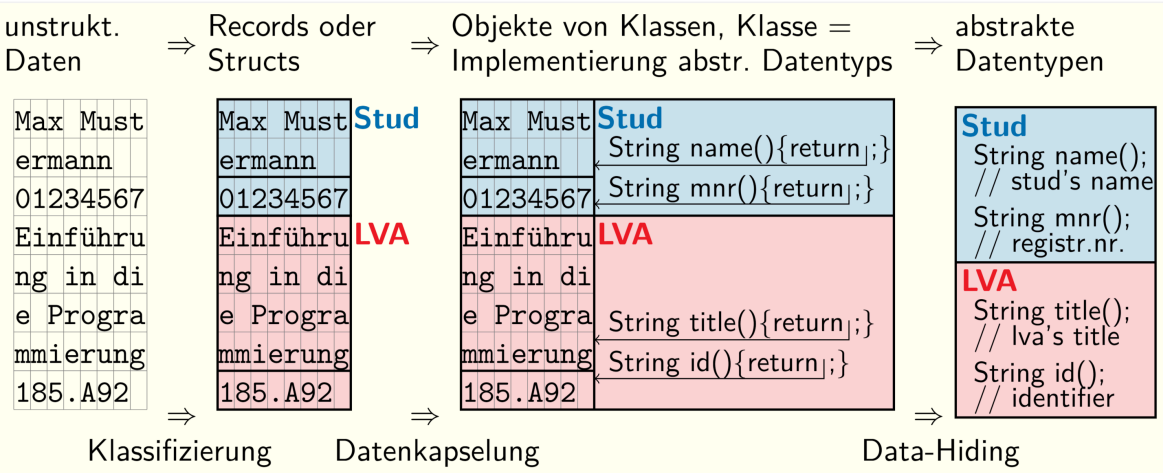
# Datenabstraktion

Datenabstraktion = Datenkapselung + Data-Hiding

Datenkapselung = Zusammenfassen
von Methoden und Variablen zu Einheit

Data-Hiding = Verstecken
vor Zugriffen von außen



# Abstrakte Datentypen

## Entwerfen eines abstrakten Datentypens:

```
/*******************************************************
class BoxedText: Rectangular text within border lines.
public methods:
    void newDimensions(int width, int height);
    void setLine(int index, String txt);
    void print();
    String toString();
*******************************************************/

private static void testBoxedText() {
    BoxedText t = new BoxedText();
    t.newDimensions(10, 3);
    t.setLine(1, "Das ist ein Text");
    t.print();
}
```

BoxedText als abstrakter Inhalt einer Objektreferenz

### Implementierung des abstrakten Datentypens

```
public class BoxedText {
    // Deklarationen von Objektvariablen
    private int textWidth = 0;
    private int textHeight = 0;
    private char[][] text = new char[0][];

    // Objektmethode
    public void newDimensions(int width, int height) {
        // Zugriffe auf Objektvariablen
        textWidth = width;
        textHeight = height;
        text = new char[height][width];

        for (char[] line : text) {
            fill(line, 0);
        }
    }

    // Private Objektmethode
    private void fill(char[] line, int i) {
        for (; i < textWidth; i++) {
        line[i] = ' ';
```

```
    }
}
```

Um so eine Klasse zu erstellen siehe hier 2. Data Hiding und co > Klassen erstellen

---

# Arten von Methoden

## 1. Objektmethode

- Definition: Ohne den Modifier `static`
- Zugehörigkeit: An ein bestimmtes Objekt gebunden
- Zugriff auf Objektvariablen: Direkter Zugriff möglich
- Nutzung:
  - Häufig in abstrakten Datentypen
  - Ermöglicht enge Zusammenarbeit zwischen Methoden und Objektzustand
- Beispiele:
  - Aufruf auf Objekt: `x.newDimensions(10, 3)` *(wenn `x` vom Typ `BoxedText`)*
  - Aufruf im selben Objektkontext: `fill(line, 0)` *(vereinfachter Aufruf ohne Objektpräfix)*

## 2. Klassenmethode

- Definition: Mit dem Modifier `static`
- Zugehörigkeit: An die Klasse, nicht an ein Objekt
- Zugriff auf Objektvariablen: Kein direkter Zugriff möglich
- Nutzung:
  - Nicht für abstrakte Datentypen im engen Sinn geeignet
  - Sollte eher sparsam eingesetzt werden, vor allem wenn Objektbezug notwendig ist
- Beispiele:
  - Aufruf über Klasse: `Factorial.fact(n - 1)`
  - Aufruf innerhalb derselben Klasse: `fact(n - 1)` *(vereinfachter Aufruf ohne Klassennamen)*

---

# Arten von Variablen

## 1. Parameter

- Deklaration: In der Parameterliste einer Methode
- Sichtbarkeit: Nur innerhalb des Methodenrumpfs
- Lebensdauer: Nur während der Ausführung der Methode
- Beispiel: `void methode(int x) { ... }`

## 2. Lokale Variablen

- Deklaration: Innerhalb des Methodenrumpfs
- Sichtbarkeit: Nur in dem Block, in dem sie deklariert wurden
- Lebensdauer: Nur während der Ausführung des Blocks
- Initialisierung: Muss manuell erfolgen – keine automatische Initialisierung

## 3. Objektvariablen (Instanzvariablen)

- Deklaration: In der Klasse, ohne `static`
- Zugehörigkeit: Zu einem bestimmten Objekt
- Existenz: Einmal pro erzeugtem Objekt
- Zugriff: In Objektmethoden derselben Klasse direkt möglich
- Initialisierung: Automatisch mit Standardwerten (`0`, `0.0`, `null`)

## 4. Klassenvariablen (statische Variablen)

- Deklaration: In der Klasse, mit `static`
- Zugehörigkeit: Zur Klasse, nicht zu einem Objekt
- Existenz: Nur einmal im gesamten Programm
- Zugriff: In Objekt- und Klassenmethoden zugreifbar
- Initialisierung: Automatisch mit Standardwerten (`0`, `0.0`, `null`)
- Hinweis: Verwendung möglichst vermeiden (insbesondere bei Objektbezug)