LAMBDA AUSDRÜCKE

Funktionale Programmierung

Lambda Ausdrücke

- Lambda Ausdrücke setzen ein funktionales Interfaces voraus.
- Ein funktionales Interfaces kann mit der Annotation @FunctionalInterface eingeleitet werden. Die Annotation erzeugt einen Compilerfehler, wenn das Interfaces mehr als eine Abstrakte Methode beinhaltet.
  - *Ein funktionales Interfaces darf nur eine einzige abstrakte Methode enthalten*
  - *Weitere default und auch static Methoden sind erlaubt*
- Lambda können auf finale variablen aus dem Selben Codeblock zugreifen
  - *final muss nicht deklariert sein, wird jedoch empfohlen*
    - effektives final sollte vermieden werden - ???

Lambda Ausdrücke

- Selten oder gar nicht wiederverwendete Klassen werden in Java gerne als anonyme Klassen geschrieben. Das beste Beispiel hierfür dürfte die Verwendung eines einfachen ActionListeners sein, der z.B. die Funktionalität eines Buttons steuert:

```
JButton btn_click = new JButton("Klick");
btn_click.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button angeklickt");
    }
});
```

- In einer anonymen Klasse wird das Interface ActionListener implementiert, diese beinhaltet nur eine einzige Methode. Seit Java 8.0 werden Interfaces die mit genau einer abstrakten Methode ausgestattet sind als funktionale Interfaces bezeichnet.

Lambda Ausdrücke

- Die in Java 8 eingeführten Lambda Ausdrücke ermöglichen es uns die Methoden von Funktionalen Interfaces ohne Anonyme Innere Klassen zu verwenden.
- Das Beispiel kann auf die folgende Weise umformuliert werden :

```
JButton btn_click = new JButton("Klick");
btn_click.addActionListener(e -> {
System.out.println("Button angeklickt");
});
```

- Es wird keine Vollständige Objektbildung mehr durchgeführt, welches die deutlichste Änderung darstellt. Stattdessen werden ein beliebiger Bezeichner des Methodenarguments und die Anweisungen innerhalb der Parameterklammern durch den Pfeiloperator (->) getrennt notiert. Es wird also quasi eine Kurzform der vollständigen Methodenimplementierung als Parameter der aufgerufenen Methode actionPerformed() angegeben.