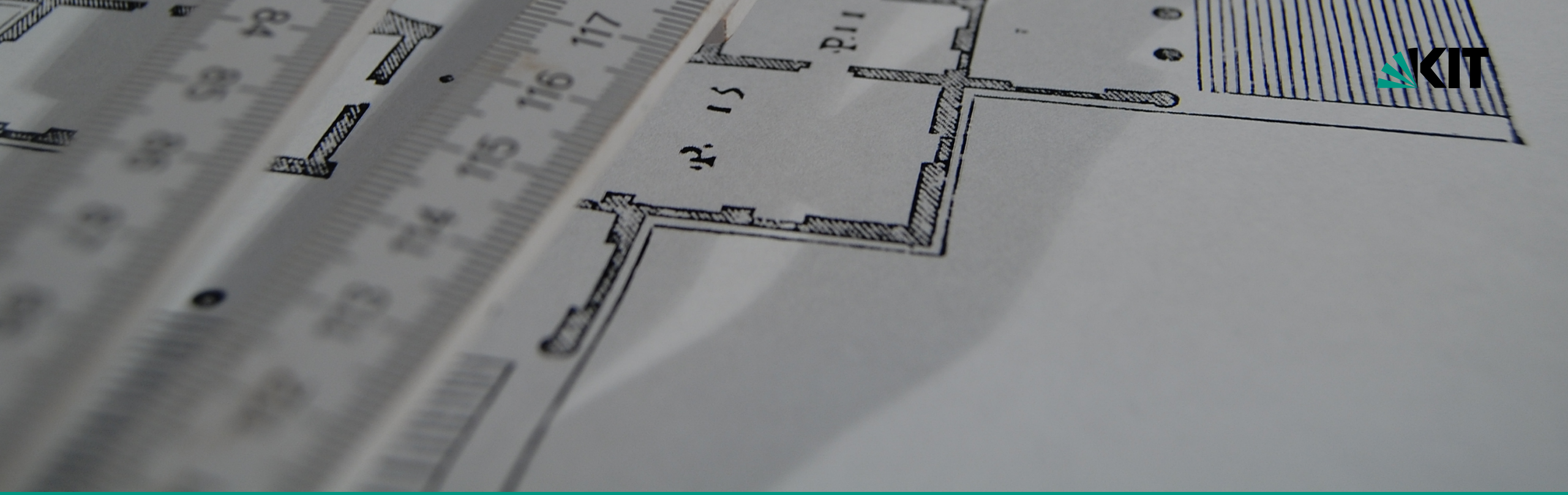

Assertions und Lambdas

Programmieren Tutorium Nr.17

Aleksandr Zakharov | 04. Februar 2026

Assertions

Allgemein

Zusicherungen (engl. Assertions) sind ein weiteres wichtiges Hilfsmittel für die Entwicklung von Software.
Sie werden verwendet, um

- Restriktionen für Parameter und globale Variablen von Methoden anzugeben (Vorbedingung)
- den Effekt von Methoden formal zu beschreiben (Nachbedingung)
- an zentralen Programmpunkten wichtige Eigenschaften (z.B. mögliche Werte von Variablen) explizit festzuhalten

Mit Hilfe von Zusicherungen ist es möglich Korrektheitsaussagen bzgl. eines gegeben Programmes mathematisch zu beweisen.

Keine Verwendung in diesem Modul!

Assertions

Java

- Java bietet die Möglichkeit Assertions direkt in den Programmtext zu schreiben (`assert`)
- Automatisierte Überprüfung, aber nur während der Laufzeit
- Eingeschränkt durch Java-Syntax im Vergleich zu Kommentaren
- Assertions werden standardmäßig nicht ausgewertet. Sie müssen beim Programmstart mit `java -ea MyClass` aktiviert werden (**e**nable **a**ssertions)

Syntax

```
assert Boolesche Bedingung;
assert Boolesche Bedingung: "Detaillierte Fehlermeldung";
```

Assertions

Beispiel

```
double multiplyScalar(double[] u, double[] v) {
    assert u.length == v.length; // Precondition
    double s= 0;
    for (int i = 0; i < u.length; i++) {
        assert s == sum(i, u, v); // Invariante
        s += u[i] * v[i];
    }
    assert s == sum(u.length - 1, u, v); // Postcondition
    return s;
}
```

Assertions

Eigenschaften

Eine Zusicherung gibt eine Eigenschaft an, die bei der Ausführung des Programms an der entsprechenden Stelle erfüllt sein muss.

- Eine **Precondition** muss vor der Abarbeitung des entsprechenden Methodenrumpfs erfüllt sein
- Eine **Postcondition** muss erfüllt sein, nachdem die Methode abgearbeitet wurde
- Eine **Instanz-Invariante** ist eine Zusicherung, die sowohl vor als auch nach jedem Methodenaufruf (Ausnahme: private Hilfsmethoden) des zugehörigen Objekts gültig sein muss. Beispiel: `int length; assert length >= 0;`
- Eine **Schleifen-Invariante** ist eine Zusicherung, die am Anfang und Ende eines jeden Durchlaufes der zugehörigen Schleife erfüllt sein muss

Assertions

assert vs if

```
void setBalance(double b) {
    assert (b >= 0);
    this.balance = b;
}

void setBalance(double b) throws IllegalArgumentException {
    if (b < 0) {
        throw new IllegalArgumentException();
    }
    this.balance = b;
}
```

Assertions

assert

- Dokumentation: Nur Überprüfungszweck
- Zur eigentlichen Laufzeit abschaltbar

if

- wird immer ausgeführt
- ggf. teuer zur Laufzeit

Lambda-Ausdrücke

Anonyme Klassen

- Manchmal möchte man ein einmaliges Objekt erschaffen
  - “Anonyme Klasse”
  - Erweitern existierende Klassen

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button geklickt!");
    }
});
```

Lambda Ausdrücke

- Kompakte Schreibweise einer anonymen Klasse
- Syntax:

```
(Parameterliste)    -> Ausdruck
()                  -> Ausdruck
(Parameterliste)    -> {
                        Ausdruck1;
                        ...
                        return Wert; // Optional, nur wenn es einen
                          R ckgabetyp geben soll
                       }
```

Beispiel (1/2)

```
List<Integer> list = new ArrayList:>();
// Add some elements//.

// ------ 1. Variante: Anonyme Klasse ------
list.sort(new Comparator<Integer>() {
    @Override
    public int compare(Integer x, Integer y) {
        return y - x;
    }
});

// ------ 2. Variante: Lambda Ausdruck ------
list.sort( (x, y) -> y - x );
```

Beispiel (2/2)

```java
// ------ 3. Variante: Lambda Ausdruck als Variable ------
Comparator<Integer> reverseIntegerComparator = (x, y) -> (y - x);
list.sort(reverseIntegerComparator);

// ------ 4. Variante: Wiederverwenden von anderen Methoden ------
list.sort(MyClass::compareIntegersReversed);
```

Parameter und Rückgabetyp müssen stimmen! (und static!)

```java
public class MyClass {

    public static int compareIntegersReversed(int x, int y) {
        return y - x;
    }

}
```