

“Programmieren Best Practices“ mit Lerngruppenbörse

Dienstag, 21.11.2023, um 17:45 Uhr



<https://www.informatik.kit.edu/imstudium.php>

Agenda

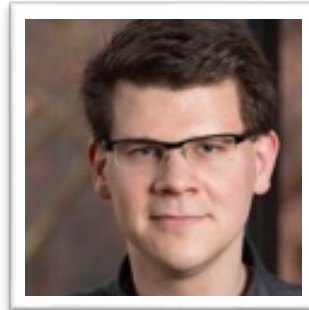
- Einführung
- „Programmieren Best Practices“ mit Timur Sağlam (SDQ)
- Q&A
- Lerngruppenbörse

Best Practices für „Programmieren“

Info² | Info-Bites | Timur Sağlam



Das Programmieren-Team



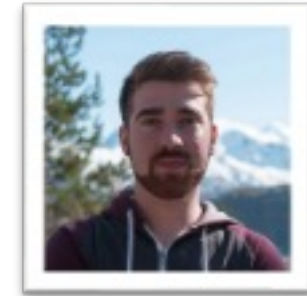
PD Dr. rer. nat. Robert Heinrich



Yves Kirschner



Dominik Fuchß



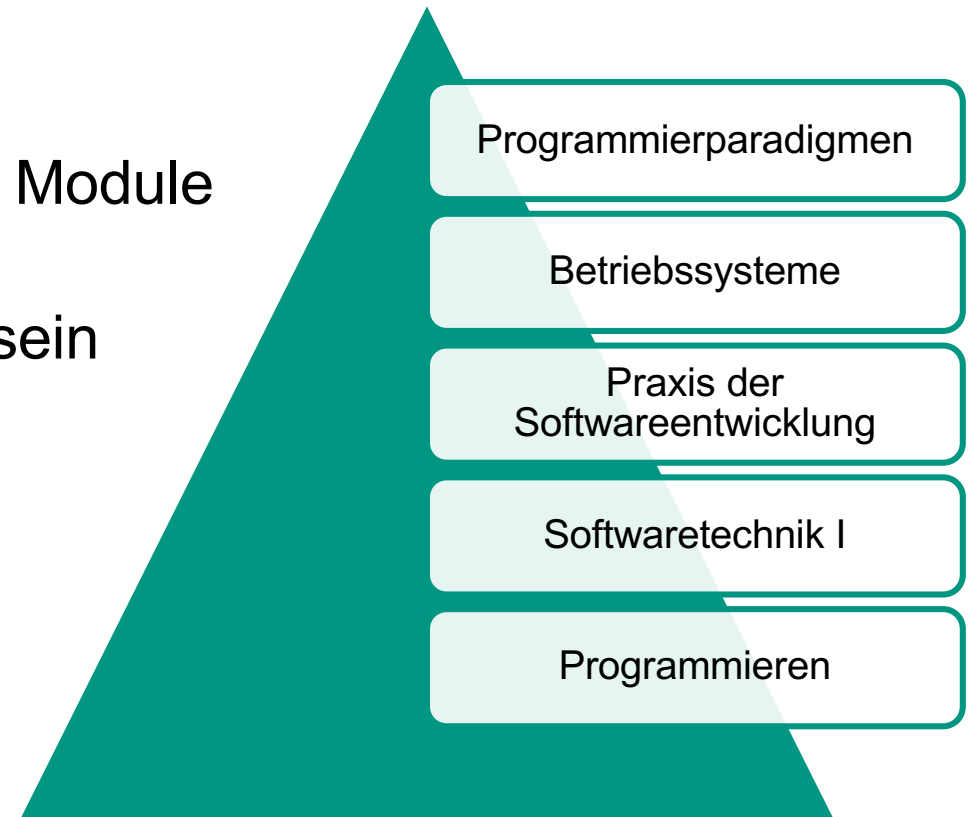
Timur Sağlam

Best Practices für „Programmieren“

- Heutige Ziele
 - Teilen von Erfahrungen, woran es scheitern könnte
 - Tipps um Frust sparen und beste Chancen zu haben
- Nicht alle Praktiken werden bei den ersten Blättern anwendbar sein
 - Es muss nicht unbedingt alles sofort verstanden werden
 - Aber spätestens bei den Abschlussaufgaben sollte darauf zurückgegriffen werden

Grundlagenstudium praktische Informatik

- Programmieren...
 - ...bildet die Grundlage für weitere Module
 - ...ist Orientierungsprüfung
 - ...sollte vor SWT abgeschlossen sein
- Was kann man mit guten Programmierkenntnissen tun?
 - Studentische Hilfskraft
 - Werkstudent
 - Open-Source-Projekte



Was Sie aus diesem Modul mitnehmen sollten

- Die Grundlagen der objektorientierten Programmierung in Java
- **Abbildung:** Problemmodellierung
 - Wie man alltägliche Probleme in Programmiersprache modelliert
- **Automatisierung:** Lösungsformulierung
 - Entwicklung von Verfahren (Algorithmen) zur Lösung einfacher Probleme
- **Abstraktion:** Problemformulierung
 - Wie man nicht nur eine Probleminstance löst, sondern eine allgemeine Lösungsmethode findet, die für viele Probleminstance funktioniert
- **Sauber programmieren!**
 - Wichtig für Wartbarkeit, Wiederverwendbarkeit und Zusammenarbeit



Allgemeine Tipps zur Studienorganisation

- Termine und Fristen verfolgen
 - Bereits zu Beginn des Semesters veröffentlicht: s.kit.edu/programmieren
 - In den eigenen Kalender mit Erinnerung übernehmen
 - Keine nachträglichen Anmeldungen für Prüfungsleistungen

- Studien- und Prüfungsordnung beachten
 - Arbeit muss selbstständig angefertigt werden
 - Der Übungsschein kann mehrfach wiederholt werden
 - Abschlussaufgaben können nach dem Übungsschein erworben werden
 - Die Abschlussaufgaben können nur einmal wiederholt werden
 - Muss bis zum Ende des dritten Semesters bestanden werden
 - Keine mündliche Nachprüfung

Vorlesung und Übung nachbereiten

- Folien der Vorlesung während des Semesters aktiv nacharbeiten
 - Im Semester nacharbeiten und nicht kurz vor den Abschlussaufgaben
 - Aufzeichnungen der Vorlesung sind auf YouTube verfügbar
 - Vorlesungsfolien sind im ILIAS vorhanden: s.kit.edu/ilias
 - Übungsblätter orientieren sich an der Struktur der Vorlesung
- Beispiellösungen der Übungsblätter nachvollziehen
 - Geben Hinweise zur Bearbeitung der Abschlussaufgaben
 - Nur Beispiele dafür, wie die Aufgabe gelöst werden könnte
 - Verstehen der verwendeten Konzepte in den Beispiellösungen
 - Entwurfsentscheidungen versuchen zu verstehen
 - Das Vorgehen nachvollziehen und verstehen

Unterstützung während des Semesters

Tutorien

- Möglichkeit zum Nachfragen
- Übungsblatt vor- und nacharbeiten
- Vorbereitung auf Präsenzübung
- s.kit.edu/programmieren

Unterstützung im Studium

- Info² und Lerngruppenbörse
- Mentoring und Kennenlernen Anderer
- Fachliche und persönliche Unterstützung
- Studienplanung Hilfsmittle
- [s.kit.edu/Unterstützung im Studium](https://s.kit.edu/Unterstützung%20im%20Studium)

MINT

- Wiederholung des Vorlesungsinhalt
- Begrenzte Teilnehmerzahl
- Aufzeichnungen im ILIAS
- s.kit.edu/mint

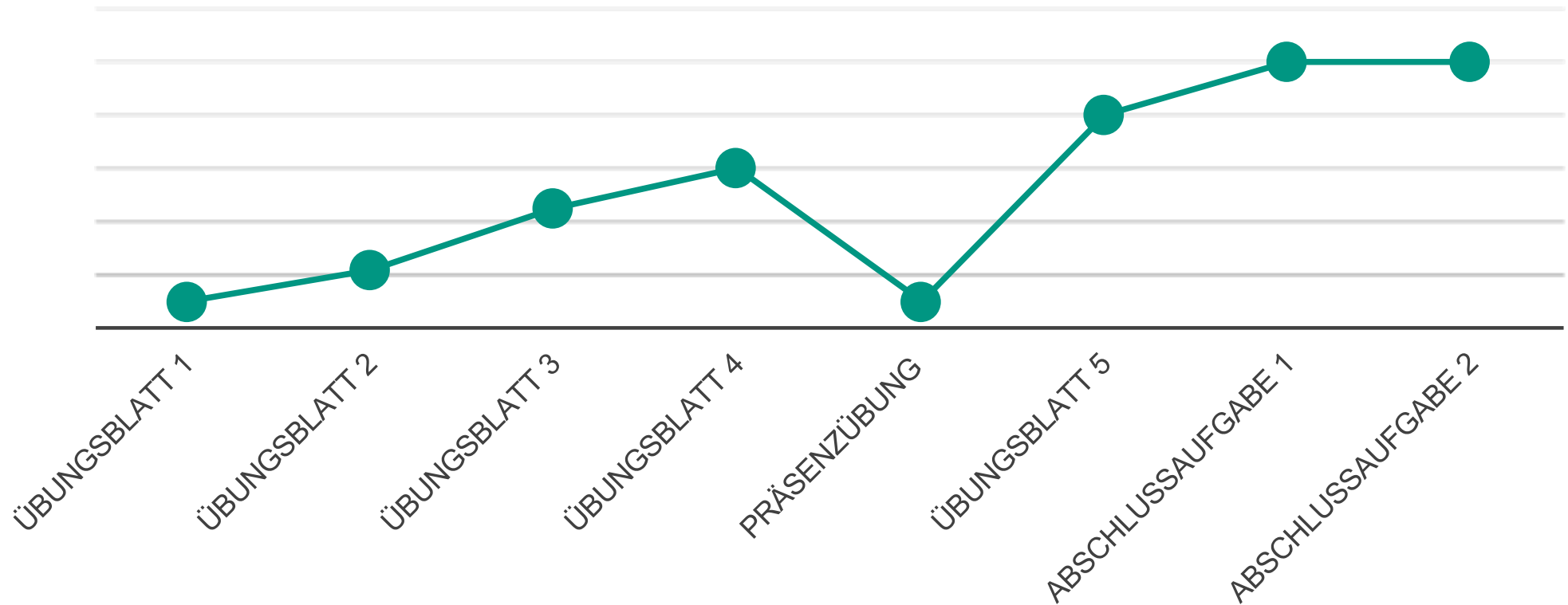
Forum-Wi / (Aktive) Fachschaft

- Vielfältigen Angebot auf dem Campus
- Klausuren und Prüfungsprotokollen
- Fachliche und persönliche Unterstützung
- s.kit.edu/fsmi ■ s.kit.edu/forum

Programmieren Präsenzübung

- Prüft Wissen, welches bei der Bearbeitung der Übungsblätter erworben worden sein sollte
 - Kleine Quelltextausschnitte schreiben oder ergänzen
 - Verhalten von kleinen Quelltextausschnitten analysieren
 - Verbeißen Sie sich nicht zu lange in eine einzige Aufgabe
- Sie sind bereits gut vorbereitet, wenn Sie die Übungsaufgaben gewissenhaft und erfolgreich gelöst haben!
 - Kein besonderes Hintergrundwissen nötig
 - Keine Vorgaben zu Checkstyle-Regeln
- Zur Vorbereitung auf das „Programmieren auf Papier“
 - Frischen Sie die Java-Syntax auf
 - Nicht die Folien auswendig lernen

Subjektive Schwierigkeit



Übungsblätter und Abschlussaufgaben

Übungsschein nicht erworben?

- Vorlesung beginnt bei null
- Übungsblätter orientieren sich an der Vorlesung
- Sehr steile Lernkurve
 - Sehr anspruchsvoll, wenn keine Vorkenntnisse vorhanden sind
 - Fortgeschrittene Studenten sollten für die Methodik am Ball bleiben

Übungsschein bereits erworben?

- Übungsblätter als erneute Vorbereitung nutzen
- Das Abgabesystem anschauen
- Programme selbstständig implementieren
 - Aufgaben aus dem Internet
 - Algorithmen aus den Mathematikvorlesungen

Organisatorische Tipps für die Übungsblätter

- Übungsblätter als Vorbereitung nutzen
 - Durch das eigenständige Bearbeiten lernt man am meisten
 - Praktisch fast kein zusätzlicher Lernaufwand für die Präsenzübung
 - Das letzte Übungsblatt als Vorbereitung auf die Abschlussaufgaben nutzen
 - Von Anfang an „saubere“ Programmierung denken und sich daran gewöhnen
- Früh genug anfangen, die Übungsblätter zu bearbeiten
 - Unklarheiten können so frühzeitig beseitigt werden
 - Die Übungsblätter werden schnell deutlich anspruchsvoller
 - Auf den ersten Blättern so viele Punkte wie möglich sammeln
- Früh genug anfangen, die Übungsblätter abzugeben
 - Frühzeitiges erproben von Git und Artemis
 - Lösungen können beliebig häufig hochgeladen werden
 - Tutoren sieht nur die letzte gültige Abgabe

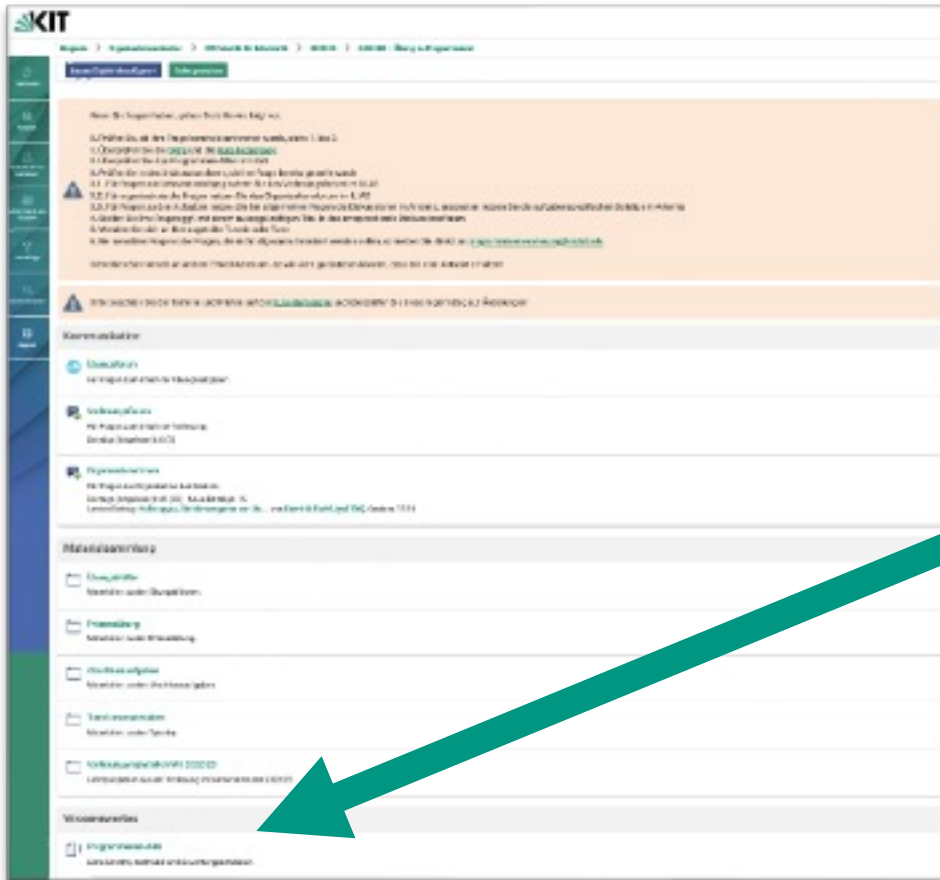
Praktische Tipps für die Aufgaben

- Quelltextklone vermeiden
 - Sich nicht Wiederholen und alles nur einmal im Quelltext implementieren
 - Aus doppelten Quelltextstellen versuchen, Methoden zu extrahieren
- Lange Methoden vermeiden
 - Methoden zerteilen und auf mehrere kleine (private) Hilfsmethoden verteilen
 - Dokumentation der Hilfsmethoden durch Javadoc-Kommentare
- Over-Engineering vermeiden
 - Quelltext so einfach wie möglich halten
 - Keine nicht benötigte Funktionalität
- Trennung der Anliegen
 - Datenstrukturen nicht nach außen geben
 - Trennung von Anwendungslogik und Benutzungsschnittstelle
- Geeignete Datentypen verwenden
 - Enums bei abgeschlossenen Mengen
 - Primitive Datentypen verwenden
 - Final, wenn nur einmal zugewiesen
 - Nur Klassenvariablen sollten statisch sein
- Herangehensweise
 - Aufgabestellung genau lesen
 - Exakt die erwartete Ausgabe liefern
 - Bei Unklarheiten sofort nachfragen


Methodik Abzüge

- Neben Funktionalität wird auch Methodik geprüft
- Großer Teil der Bewertung von Übungsblättern und Abschlussaufgaben
- Alle Bewertungskriterien sind im Ilias-Wiki: s.kit.edu/wiki

Ilias-Wiki



Wissenswertes

 **Programmieren-Wiki**
Erste Schritte, Methodik und Bewertungsrichtlinien.

Ilias-Wiki: Übersicht

Hauptseite

Seitenübersicht [\[Ausblenden\]](#)

- 1 Erste Schritte
- 2 Methodik
- 3 Bewertungsrichtlinien
 - 3.1 Allgemeine Richtlinien
 - 3.2 Blatt 1
 - 3.3 Blatt 2
 - 3.4 Blatt 3
 - 3.5 Blatt 4
 - 3.6 Blatt 5

Das Programmieren Wiki wurde zum Start des Wintersemesters 2013/14 im Rahmen der Vorlesung Programmieren (1. Semester, Informatik / Informationswirtschaft) am KIT initiiert. Hier sollen immer wiederkehrende Anfängerfragen und Probleme möglichst umfassend und mit Beispielen erläutert werden. Das Wiki wird im Laufe der Semester stetig von den Lehrenden erweitert. Aber auch sinnvolle Beiträge von Kursteilnehmern werden die Qualität und Quantität des Wikis kontinuierlich steigern.

Ilias-Wiki: Bewertungsrichtlinien

3 Bewertungsrichtlinien

Die nachfolgenden Artikel beschreiben Teile unserer Bewertungsrichtlinien um die Modellierung z.B. in den Abschlussaufgaben zu bewerten. Die Liste enthält nicht alle Richtlinien, soll Ihnen aber einen guten Überblick geben, auf was Sie bei der Erstellung Ihrer Lösungen achten sollen. In der Regel kann der in der Vorlesung behandelte Stoff bei der Bearbeitung der Aufgaben verwendet werden, es sei denn, dies ist in der Aufgabe ausdrücklich untersagt.

3.1 Allgemeine Richtlinien

- Kein OOP
- Seiteneffekte
- Schlechte Modellierung
- Verwendung der Wrapper Klassen

3.2 Blatt 1

- Schwieriger Code
- Einheitliche Sprache
- Verschachtelungstiefe

3.3 Blatt 2

- Alles aus Blatt 1 und ...
- Schlechte Bezeichner
- Fehlende Trennung von Programmlogik und UI
- Final
- Ungeeigneter Schleifentyp
- Unnötige Komplexität

1 von 10 Blättern / 1 von 10 Knoten

Ilias-Wiki: Einheitliche Sprache

Einheitliche Sprache

Seitenübersicht [[Ausblenden](#)]

1 [Negativbeispiel](#)

2 [Positivbeispiel](#)

Kommentare und Ausgaben sollen in einheitlicher Sprache verfasst sein. Diese Bewertungsrichtlinie bezieht sich sowohl auf Javadoc als auch auf sogenannte Entwicklerkommentare, d.h. Zeilen- und Blockkommentare.

Während Bezeichner zwingend englische Namen haben müssen (siehe auch [Schlechte Bezeichner](#)), darf für Kommentare und Javadoc zwischen Deutsch und Englisch gewählt werden. Diese Wahl muss allerdings über das gesamte Programm einheitlich sein: Entweder werden alle Kommentare/Javadoc auf Deutsch oder auf Englisch verfasst. Außerdem muss die Ausgabe des Programms eine einheitliche Sprache haben. Da die Aufgabenstellung meist einige englische Ausgaben vorgibt, müssen also auch alle selbstgeschriebenen Fehlermeldungen auf Englisch sein.

Ilias-Wiki: Einheitliche Sprache

1 Negativbeispiel

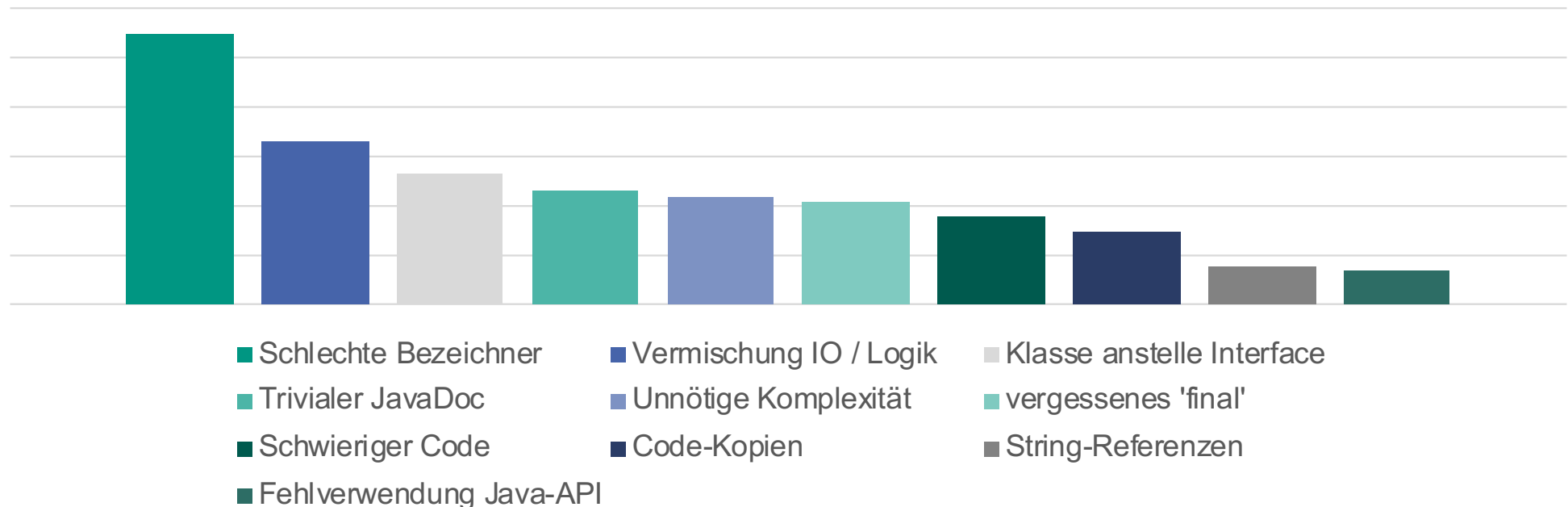
```
1  /**
2  * Prints the status of the player to the console.
3  * @param player The id of the player
4  */
5  public void printPlayerStatus(int playerId) {
6      // Falls der Spieler unbekannt ist wird null zurückgegeben
7      Player player = findPlayerById(playerId);
8
9      if (player == null) {
10         System.err.println("Error, unbekannter Spieler");
11     } else {
12         System.out.println("Player " + playerId + ": " + player.getHP() + "HP");
13     }
14 }
```

Ilias-Wiki: Einheitliche Sprache

2 Positivbeispiel

```
1  /**
2  * Prints the status of the player to the console.
3  * @param player The id of the player
4  */
5  public void printPlayerStatus(int playerId) {
6      // Returns null if the player is unknown
7      Player player = findPlayerById(playerId);
8
9      if (player == null) {
10         System.err.println("Error, unknown player");
11     } else {
12         System.out.println("Player " + playerId + ": " + player.getHP() + "HP");
13     }
14 }
```

Die häufigsten methodischen Fehler der letzten Abschlussaufgaben




Weitere Bewertungsrichtlinien im Wiki: s.kit.edu/wiki


Schlechte Bezeichner

- Bezeichner sollten *sprechend* sein
- Keine Abkürzungen, konform zu Konventionen
- Ausnahmen: i, j, x, y, it

```
1 public static int s(final int a, final int b) {  
2     return a - b;  
3 }
```



```
1 public static int subtract(final int minuend, final int subtrahend) {  
2     return minuend - subtrahend;  
3 }
```



Final


- Attribute sollten falls möglich *final* sein
- Diese können dann nur einmal beschrieben werden

```
class Event {  
    private static int BASE_FEE = 1000;  
    private List<String> participants;  
  
    public Event() {  
        participants = new ArrayList<>();  
    }  
  
    public void addParticipant(String name) {  
        participants.add(name);  
    }  
}
```



Final

```
class Event {  
    private static final int BASE_FEE = 1000;           // constants should be final  
    private final List<String> participants;           // fields should be final if possible  
  
    public Event() {  
        participants = new ArrayList<>();  
    }  
  
    public void addParticipant(String name) {           // not required for parameters  
        participants.add(name);                       // (list is final, but not the content)  
    }  
  
    public int calculateProfit(int ticketPrice, int venueCost) { // not required for parameters  
        int fixedCost = BASE_FEE + venueCost;         // not required for local variables  
        return participants.size() * ticketPrice - fixedCost;  
    }  
}
```



Ungeeigneter Schleifentyp

- Boolescher-Ausdruck => While-Schleife, Do-While-Schleife
- Einfache Iteration => For-Each-Schleife
- Index benötigt => For-Schleife

```
1 public void printList(List<String> list){  
2     for(int i = 0; i<list.size(); i++){  
3         System.out.println(list[i]);  
4     }  
5 }
```



```
1 public void printList(List<String> list){  
2     for(String item : list){  
3         System.out.println(item);  
4     }  
5 }
```



Unnötige Komplexität

- Redundanz/Komplexität erhöht visuelles Rauschen
=> einfachste Lösung wählen

```
1  boolean isValid() {  
2      if (this.sold == false && !(this.price <= 0)) {  
3          return true;  
4      } else {  
5          return false;  
6      }  
7  }
```



```
1  boolean isValid() {  
2      return !this.sold && this.price > 0;  
3  }
```



Magic Numbers/Strings

- Alleinstehende Zahlen im Quelltext sind oft schwer zu verstehen
- Variablen erlauben Benennung
- Konstanten erlauben Wiederverwendung

```
1 final String password = "Test";  
2 if (password.length() < 8) {  
3     System.err.println("Password too short, please choose a longer password!");  
4 }
```



```
1 final String password = "Test";  
2 final int minimalPasswordLength = 8;  
3 if (password.length() < minimalPasswordLength) {  
4     System.err.println(PASSWORD_ERROR_MESSAGE);  
5 }
```



Testen

- Testen hilft, Fehler zu finden!
 - Vergleich von tatsächlichem und erwünschtem Verhalten
 - Modularer Quelltext hilft beim Testen
- Ausführliches Testen vordem Hochladen
 - Nicht auf den Public-Test verlassen
 - Den Public-Test lokal ausführen
- Selber Tests schreiben
 - Für die Kommandozeileninteraktion
 - Über eigene Main-Methode
 - JUnit-Tests schreiben

Teststrategien

- Datenbasiert
 - Beispiele auf Aufgabenblättern
- Kontrollflussbasiert
 - Suche in Datenstrukturen
 - Alle Entscheidungen einmal treffen
- Grenzwertbasiert
 - Wertebereich bei Berechnungen
 - Off-by-one-Error

Rückfragen

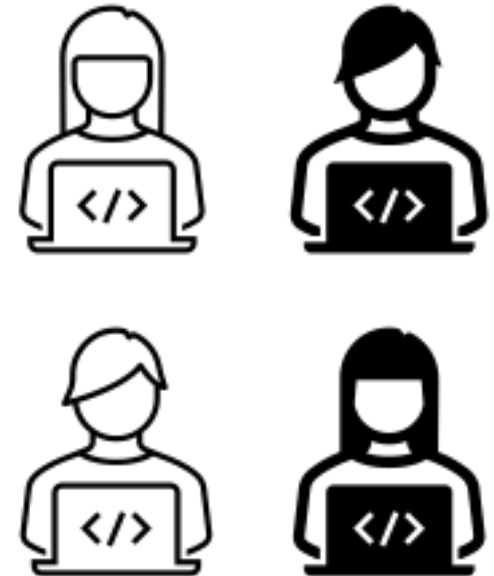
- Fragen und Unklarheiten können immer auftreten
 - Wenn ihr eine Frage habt, haben andere vielleicht die gleiche Frage
 - Wir bieten viele Möglichkeiten, Fragen zu stellen
- Bei Rückfragen werden Aufgaben gegebenenfalls aktualisiert
 - Lediglich minimale Aktualisierungen oder weitere Klarstellungen
 - Schaut immer mal wieder, ob die aktuelle Aufgabe aktualisiert wurde

Reihenfolge bei Fragen

1. Websuche
2. FAQ / Programmieren-Wiki prüfen
 - s.kit.edu/faq ■ s.kit.edu/wiki
3. Diskussionsforum prüfen
 - Suchfunktion verwenden
4. Im Diskussionsforum nachfragen
 - Aussagekräftigen Titel verwenden
5. Eure Tutoren fragen
6. Übungsleitung fragen
 - programmieren-vorlesung@cs.kit.edu

Zusammenfassung

- Nehmen Sie sich **ausreichend Zeit** für die Übungsblätter
- Fangen Sie **rechtzeitig** mit der Bearbeitung an
- Beachten Sie die **Bewertungskriterien** im Ilias-Wiki
- Verwenden Sie früh **Checkstyle**
- Besuchen Sie die **Tutorien** und nehmen Sie aktiv Teil



Danke für Ihre Aufmerksamkeit!

Links und Anlaufstellen:

- Programmieren Seite: https://sdq.kastel.kit.edu/wiki/Vorlesung_Programmieren_WS_2023/24
- Fachschaft Mathematik / Informatik: Beratung durch Studierende des Fachs Informatik / Mathematik: <https://www.fsmi.uni-karlsruhe.de/>
- Forum Wirtschaftsinformatik: Beratung durch Studierende der Fachrichtung Wirtschaftsinformatik: <https://www.forum-wi.de/>
- MINT-Kolleg Baden-Württemberg: Vorkurse, Semesterkurse (Einführung in die Programmierung mit Java), Aufbaukurse Informatik (in der vorlesungsfreien Zeit, usw): <https://www.mint-kolleg.kit.edu/index.php>
- Unterstützung im Studium (Fakultät für Informatik)
 - Webseite: <https://www.informatik.kit.edu/10509.php>
 - Youtube Playlist: <https://www.youtube.com/playlist?list=PL22ZNLSoHCREnOwW5FzHO1SGP-zK8oEGx>