

## Fragen, Antworten, Kommentare zur aktuellen Vorlesung

Hinweis: Zu einzelnen Aufgaben werden Videos mit Beispiellösungen von mir hochgeladen. Sollten Sie die Aufgabe korrekt gelöst haben, besteht keine Notwendigkeit sich das Video zur Aufgabe anzusehen. Die Lösungen können nicht die Vorlesungsvideos ersetzen, da sie nichts Grundlegendes erklären. Die Diskussion einer Beispiellösung zur Aufgabe 6 zur Findung von beschreibenden Objekteigenschaften finden Sie unter <https://youtu.be/yK48Z1zq1l0> (11:20). Die Diskussion einer Beispiellösung zur Aufgabe 7 mit der Modellierung der Fahrkarte finden Sie unter <https://youtu.be/U829ns8LutU> (26:30). Sollte es Korrekturen zu Videos geben, stehen diese immer in den Kommentaren zum Video.

Info: Unsere Klausur ist (leider erst) für den 30.1.25 geplant, weitere Details und eine Musterklausur kommen später. Wieso „leider erst“? Der späte Termin suggeriert, dass durch fleißiges Lernen nach der Vorlesungszeit der Stoff nachgeholt werden kann. Meine Erfahrungen zeigen, dass das leider nie klappt, ich keine Person kenne, die während der Praktika enorme Schwierigkeiten hatte und dann die Klausur mitgeschrieben und bestanden hat. Falls es so jemanden geben sollte, wäre ich danach an einem Erfahrungsaustausch sehr interessiert.

Früher waren Klausuren einfach in den zwei Wochen nach der Vorlesungszeit. So war klar, dass ein Nachlernen kaum möglich ist. Da in der Programmierklausur auf Papier programmiert wird, ist auch klar, dass ein Auswendiglernen sinnlos ist. „Auf Papier programmieren“ klingt zunächst merkwürdig, hat aber einen didaktischen Vorteil. Vor der Lösung muss ein Verfahren im Kopf entstehen, mit dem die Aufgabe gelöst wird. Es kann nicht mit Try-And-Error versucht werden zum Ergebnis zu kommen. Try-and-Error ist genau der Ansatz den gute Leute in der Programmierung nicht nutzen, da so oft versteckte Fehler zurückbleiben. Es wird schrittweise ein komplexeres System erdacht und dann entwickelt und zwischenzeitlich getestet. Diese Tests allerdings fallen auf Papier weg. Dafür spielt beim Schreiben die exakte Syntax (vergessene Klammer, vergessenes Semikolon) keine Rolle und ist die Idee zielführend, die Umsetzung aber nicht, gibt es abhängig vom Fehler 50-80% der Punkte.

Frage: Ich habe das mit der ArrayList nicht genau verstanden, warum soll ich die nicht auch bei der Klasse Linie nutzen, die hat ja auch mehrere Punkte?

Antwort: Generell ist der Ansatz nicht völlig falsch, erhöht aber bei der späteren Nutzung und Erweiterung der Funktionalität der Klasse den Arbeitsaufwand deutlich. Bei einer Linie werden immer exakt zwei Punkte benötigt, die deshalb auch in zwei Objektvariablen festgehalten werden. Erst wenn die Anzahl der möglichen Objekte größer wird *und* es unklar ist, wieviele es werden können, ist eine ArrayList (später allgemeiner eine Art von Collection) die richtige Modellierung. Wollen Sie z. B. ein Bild modellieren, das daraus besteht, dass mehrere Punkte miteinander in einer Zeichnung verbunden werden (z. B. Haus des Nikolaus) ist eine Liste von Punkten (ArrayList<Punkt>) ein guter Ansatz.

Nebenbei wäre dort ein Ansatz ArrayList<Linie> nicht sinnvoll, da so erst sichergestellt werden muss, dass die einzelnen Linien untereinander verbunden sind. (Richtig: ein Dreieck wird als drei Punkte modelliert, falsch: ein Dreieck wird als drei Linien, also sechs Punkte, modelliert.)

Wenn die maximale Anzahl von Objekten bekannt ist, kann als Typ auch ein Array (geschrieben Punkt[]) genutzt werden. Den Typen lernen wir aber erst später kennen, da er ein Spezialfall ist.