Warum betreiben wir modularen Klassenentwurf?

An dieser Stelle kann man sich mit zwei Fragestellungen befassen:

1. **Warum macht man das überhaupt?** Könnte man nicht einfach alles Funktionalität in einer Klasse unterbringen, anstatt das Programm auf viele einzelne Klassen(dateien) zu verteilen?
2. Wenn die OO-Modellierung eine Problems nicht eindeutig ist - **woran erkennt man dann, ob man es „gut“ gemacht hat?**

Warum verteilt man die Funktionalität und den Code auf mehrere Klassen?

Wenn man ein Problem sinnvoll modularisiert und modelliert, hat das viele Vorteile:

- **Lesbarkeit des Quellcodes** → Etwas stimmt mit dem Tor nicht? Also muss man in der „tor“-Klasse schauen und nicht 5000Zeilen Code durchscrollen, bis man zu dem Teil kommt, der das Tor erzeugt. Es **erleichtert Änderungen** an der Funktionalität, wenn stets klar ist, wo bestimmte Eigenschaften und Fähigkeiten festgelegt sind.
- Wenn man **Klassen** geschickt modelliert, kann man Sie in anderen Programmen **wiederverwenden** - nicht umsonst spricht man von „Klassenbibliotheken“.
- **Neue Objekte** können durch **neue Klassen** ein ein Modell eingefügt werden - du willst Hindernisse auf dem Spielfeld? Kein Problem mit der zusätzlichen „hindernis“-Klasse.

Wann ist ein Klassenentwurf "gut"?

Ein Klassenentwurf ist also „gut“, wenn er die oben genannten Vorteile maximal unterstützt - hierführ kann man zwei Eigenschaften des Entwurfs betrachten:

- **Kohäsion**: Die Kohäsion einer Klasse definiert ihren logischen Zusammenhalt als Einheit. Erennen kann man das daran, wie deutlich Sie sich von anderen Klassen des Modells abgrenzt, aber auch daran, dass ihre Methoden für klar umrissene Aufgaben zuständig sind. Beispiel: Wenn man bei der Implementation einer Liste eine Methode`isEmpty(): boolean` definiert, die nur schaut, ob der Startknoten null ist, erscheint das im ersten Augenblick übertrieben, erhöht jedoch die Kohäsion der Klasse. Antatt an vielen Stellen eine interpretationsbedürftige Abfrage `start==null` im Code zu haben, hat man eine semantisch klare Methode `isEmpty()`. **Man möchte also eine hohe Kohäsion der Klassen haben.**
- **Kopplung**: Die Eigenschaft „Kopplung“ beschreibt die Bindung zwischen den Klassen. **Die Kopplung soll möglichst gering sein**, das erreicht man dadurch, dass die Abhängigkeiten zu anderen Klassen möglichst klein gehalten werden sollten. Schnittstellen zwischen Klassen sollten klar definiert sein, mit bedeutungsvollen Parametern. Anstatt also eine Getter-Methode zu schreiben, die einen Parameter benötigt, der ihr mitteilt, welches Attribut zurückgegeben werden soll, bekommt jedes Attribut einen eigenen Getter mit sprechendem Namen - und ohne Parameter.