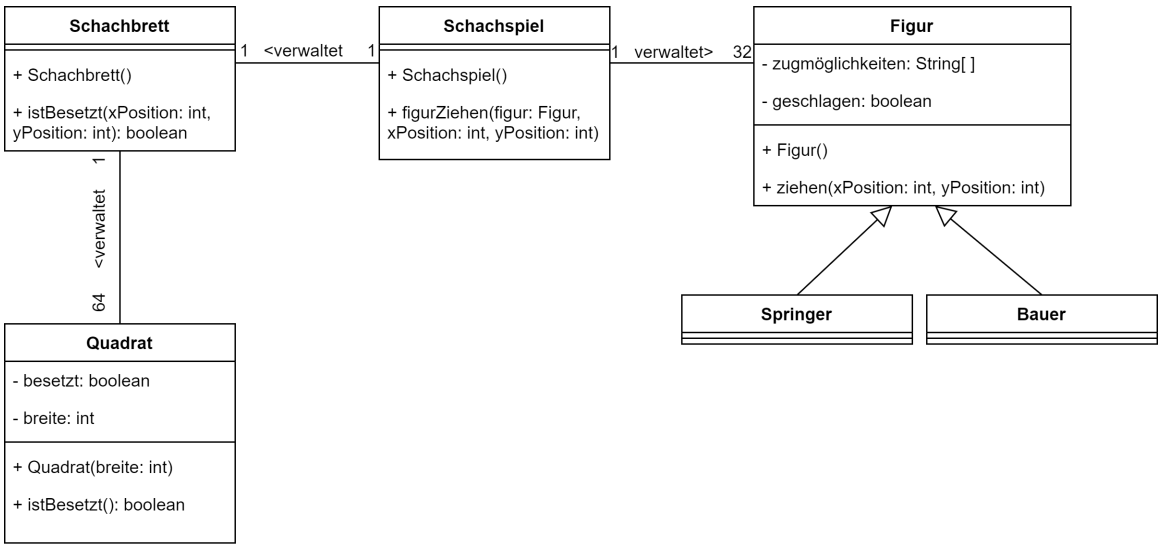
Assoziationen und Klassendiagramme

Bisher haben wir hauptsächlich **Aggregationen** zur Beschreibung verwendet, nicht immer ganz zutreffend. Zur Erinnerung: eine Aggregation entspricht einer „enthält“ - Beziehung zwischen zwei Objekten. Bei den Holfzfällern und den Holzstapeln ist das eigentlich nicht ganz zutreffend. Die Objekte kommunizieren zwar miteinander, aber ein Holzfäller „besteht“ nicht aus einem Holzstapel oder umgekehrt. Die bessere Variante eine solche Beziehung zu beschreiben ist eine einfache **Assoziation**, repräsentiert durch eine einfache Linie zwischen den beiden Klassen. Man kann auch hier durch **Multiplizitäten** angeben, auf wie viele Objekte der jeweiligen Klasse zugegriffen wird.
Beispiel:



Wir betrachten eine - sehr einfache - Variante eines Schachspiels (das Spiel wäre so natürlich noch nicht vollständig funktional, aber für uns ist es ausreichend, für die Profi-Programmierer: das Design ist so auch nicht wirklich geschickt, aber anschaulich :)).

Die zentrale Klasse ist das **Schachspiel**, ein Nutzer soll in dieser Implementierung nur mit dem Schachspiel interagieren (hier kann der Nutzer nur Figuren ziehen und schlagen).

Gehen wir davon aus, dass wir unser Schachbrett wieder grafisch mit einem Canvas ausgeben wollen (hier weggelassen), dann muss ein Objekt der Klasse Schachspiel natürlich ein Schachbrett „kennen“, hier repräsentiert durch eine **1:1-Assoziation**.

Für die Implementierung ist auch die Bezeichnung und besonders die Pfeilspitze mit der Leserichtung auf der Assoziationslinie wichtig. Hier soll dargestellt werden, dass das Schachspiel das Schachbrett verwaltet, d.h. es gibt nur eine Referenz im Objekt der Klasse Schachspiel auf ein Objekt der Klasse Schachbrett und nicht umgekehrt!

Genauso muss das Schachspiel aber z.B. auch die Figuren kennen (hier als Oberklasse Figuren implementiert, neben dem Springer und dem Bauer muss es natürlich noch weitere Unterklassen geben). Da es mehrere Figuren gibt und nicht nur eine, handelt es sich hier um eine **1:n - Assoziation** (analog zu Datenbanken!), die man in unserem Fall noch präziser angeben kann, da klar ist, dass es nur 32 Figuren im Standard-Schach gibt. Auch hier gilt wieder: das Schachspiel muss auf die Figuren zeigen, um sie „ziehen“ oder „schlagen“ zu können. Umgekehrt ist dies aber nicht der Fall! Eine mögliche Implementierung wäre also, ein Feld der Länge 32 im Schachspiel anzulegen.

Theoretisch sind auch **n:m-Assoziationen** möglich, diese sind (gerade bei unseren „einfachen“ Programmen) aber deutlich seltener und sollen deswegen nicht vertieft werden.
Hinweise:

- Auch wenn es im tatsächlichen Code ein Referenzattribut gibt, z.B. vom Schachspiel zum Schachbrett (also *Schachbrett brett;*), so wird dies nicht extra im Diagramm als Attribut angegeben, da die Kante diese Information schon trägt.
- Im Wesentlichen sagen die Kanten (und die Pfeilspitzen) aus, welches Objekt auf anderen Objekten der entsprechenden Klassen Methoden aufrufen oder Attribute verändern kann.

Aufgabe: (Erstellung von Klassendiagrammen und Implementierung)
a) Erstelle ein Klassendiagramm, das folgende Vorgaben beachtet:

- Es soll vier Klassen geben: Bibliothek, Besucher, Buch, Regal.
- Die Bibliothek besteht aus 64 Regalen mit Büchern und hat einen Namen.
- Ein Buch wird eindeutig durch seine ISBN-Nummer bestimmt (vereinfacht hier als int dargestellt) und ein Attribut, in dem die Mitgliedsnummer des Besuchers gespeichert wird, der es gerade ausgeliehen hat. Von einem Buch gibt es immer nur ein Exemplar. (Der Inhalt des Buches wird der Einfachheit halber nicht dargestellt!)
- Ein Regal enthält 32 Bücher und eine eindeutige Regalnummer.
- Ein Besucher hat einen Namen und eine Mitgliedsnummer in der Bibliothek. Außerdem kann er bis zu sechs Bücher gleichzeitig ausleihen. Er „weiß“ natürlich welche!
- Ein Besucher kann über die Bibliothek ein Buch ausleihen, dazu muss die Bibliothek in den verschiedenen Regalen nach dem Buch „suchen“. Dann wird das Buch „ausgeliehen“ - durch Setzen der entsprechenden Werte im Buch und beim Besucher!

b) Implementiere dein Klassendiagramm!