**Übung: Ritterturnier**

Schlossherr Owen Todtstelzer veranstaltet auch dieses Jahr wieder ein Ritterturnier auf Burg Hochgradnix.

  
Es kommen seine Verwandten, Zuschauer, Gaukler, Ritter und Knappen und ein paar Prominente sind auch immer mit dabei.   
Die Teilnehmer zum Ritterturnier hat der Burgherr in den vergangenen Jahren immer auf Papier vermerkt. Das wird ihm aber zu mühsam und deshalb wünscht er sich eine Softwarelösung dafür.

Überlege zuerst aufgrund der folgenden Beschreibung, welche Objekte es gibt, welcher Art die Beziehungen zwischen den Objekten sind und erstelle ein Klassendiagramm (und implementiere danach die entsprechenden Klassen).

Eine Teilnehmerliste soll die Teilnehmer an dem Ritterturnier verwalten. Teilnehmer sind erstmal die Ritter und in Zukunft vielleicht auch noch die Orks. Die Ritter haben für gewöhnlich Knappen, aber es gibt immer ein paar arme Kämpfer, die sich keine leisten können und deshalb allein anreisen. Knappen sind also **optional**. Die Ritter wählen am Anfang des Turniers außerdem eine Waffe, mit der sie kämpfen wollen. Zur Auswahl stehen neben Lanze, Schwert oder Keule auch noch andere. Das ist für Owen wichtig, weil er ja schlecht einen Ritter mit Lanze gegen einen mit Schwert kämpfen lassen kann. Er braucht also die Möglichkeit, die Kämpfer für einen bestimmten **Waffentyp zu filtern**.

Folgende Daten muss Owen Todtstelzer beim Check-In eingeben können:

**Ritter:** Name, Telefonnummer, Rufname (z.B.: „der Tapfere“). Wird der Ritter als Teilnehmer angelegt, so soll automatisch eine ID vergeben werden (von 1 aufsteigend), damit jedem Teilnehmer eine eindeutige Nummer zugeordnet ist.  
Damit Knappe bzw. Waffe hinzugefügt werden können, braucht es entsprechende Methoden dafür (*addKnappe* bzw. *addWaffe*).   
Außerdem muss man die Waffenart abfragen können, mit der der Ritter kämpft. Das ist vorerst einfach der Anfangsbuchstabe der **Waffe (also z.B. „L“ für Lanze, „S“ für Schwert und „K“ für Keule**).  
Es soll eine Methode *toString()* geben, die ID, Name und Rufname des Ritters zurückgibt und (wenn vorhanden) die Daten des Knappen und (wenn vorhanden) die der Waffe.

**Knappe:** Name, Telefonnummer, Ausbildungsgrad (das ist eine Nummer, die aussagt, welchen „Weg“ er schon zum Ritter zurückgelegt hat; beginnt mit 10). Den Ausbildungsgrad soll man später wieder abfragen können.  
Es soll eine Methode *toString()* geben, die den Namen und Ausbildungsgrad zurückgibt.

**Waffe:** Bezeichnung (z.B.: „Kampflanze“) und Art (Lanze, Schwert, Keule, …).  
Hinweis: Damit man für die Zukunft flexibel bleibt und nicht immer neue Waffenarten in der Klasse selbst kodieren muss, soll es hier die Erweiterungsmöglichkeit mittels Vererbung geben. D.h., es wird eine spezielle Waffe (z.B. Lanze) erzeugt und als Waffe dem Ritter hinzugefügt.  
Es soll eine Methode *toString()* geben, die die Beschreibung und die Art der Waffe (das ist der Anfangsbuchstabe „L“, „S“ oder „K“) zurückgibt.

Die **Teilnehmerliste** soll folgende Funktionalität bieten:

* Eine Methode (*addTeilnehmer*), um neue Teilnehmer hinzufügen zu können.
* Eine Methode (*listeAlleTeilnehmer*), um alle Teilnehmer ausgeben zu können.
  + Dazu soll jeder Teilnehmer eine Methode (*toString()*) haben, die aufgerufen werden kann.
    - Wenn ein Knappe vorhanden ist, dann sollen auch die Daten des Knappen mit ausgegeben werden.
    - Wenn schon eine Waffe zugeordnet ist, dann auch die Daten der Waffe.
    - **Wichtig**! 🡪 Auch wenn es keinen Knappen gibt bzw. noch keine Waffe, dann darf es trotzdem nicht zu einem Fehler bei der Ausgabe kommen!
* Eine Methode (*alleMitWaffenart(string waffenart)*), um alle Ritter mit einer bestimmten Waffenart zu erhalten (Die Waffenart ist wie oben erwähnt der Anfangsbuchstabe).
  + Auch hier soll die Methode *toString()* für die Ausgabe zum Einsatz kommen.
* Die „*addTeilnehmer*“-Methode soll eine **„*NameSchonVorhanden*“-Exception** werfen, wenn der Name schon vergeben ist.
  + Tritt das auf, soll die Exception gefangen und behandelt werden (das Programm darf dadurch nicht abstürzen).

Allgemeine Anforderungen an das Programm:

* Die Dateneingabe (d.h. das Dateneingabeformular beim Check-in) sowie die Ausgabe einer Teilnehmerliste (*listeAlleTeilnehmer, alleMitWaffenart*) soll **grafisch in WPF erfolgen**.
* Die Liste aller Teilnehmer soll vor Beenden des Programms in einer JSON-Datei gespeichert werden.
* Die Liste aller Teilnehmer soll nach dem Programmstart von einer JSON-Datei geladen werden können. Implementiere das Laden aus der Datei mittels asynchronen Aufrufen (async-await).
* Implementiere die entsprechenden Objekte und Methodenaufrufe so, dass das Verhalten sinnvoll **mittels Unit-Tests** getestet werden kann.

Exemplarische Ausgabe (hier noch auf Konsole):

