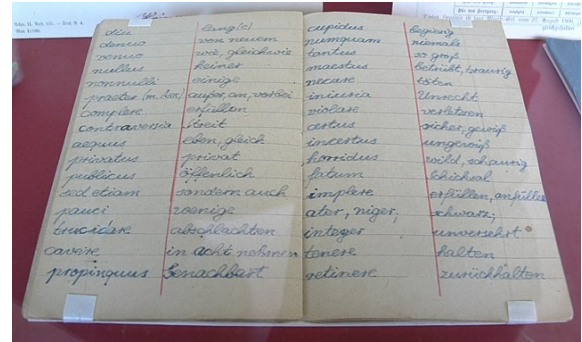


Betrachten wir als Beispiel einen **Vokabeltrainer**:
Wir möchten in einer Software Vokabeln speichern,
um damit unser Gedächtnis zu trainieren.

Zunächst kann man sich überlegen, dass die
Vokabeln in einer dynamischen Liste gespeichert
werden sollten, damit laufend neue Vokabeln
hinzugefügt und die Liste evtl. nach bestimmten
Kriterien geordnet werden kann.

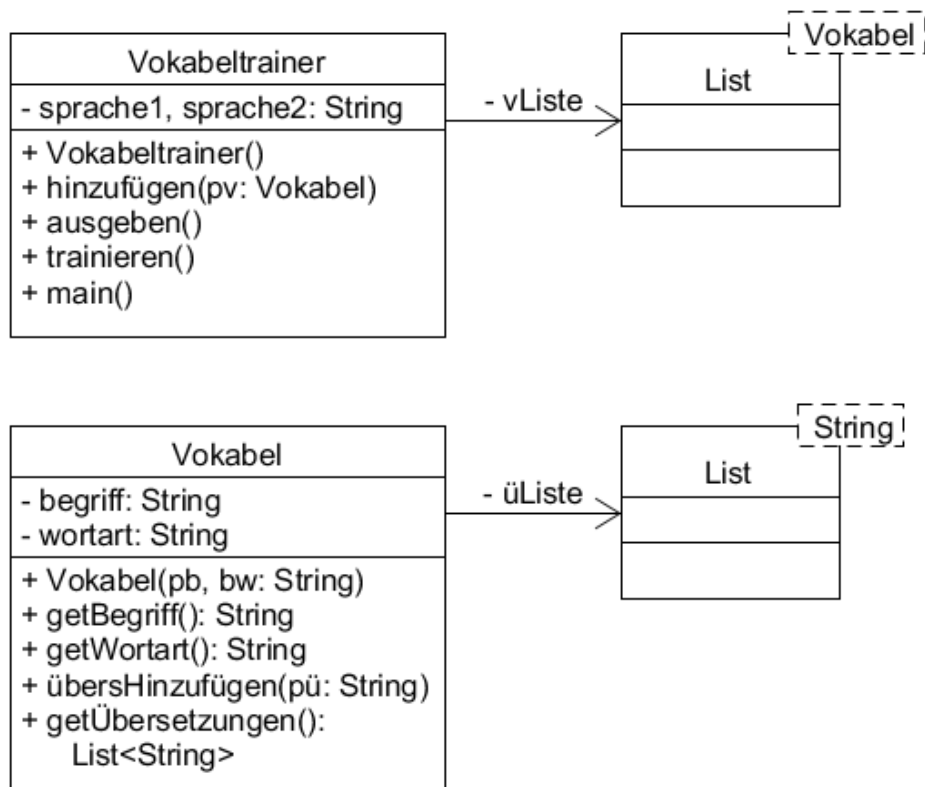


Genügt dann für jede einzelne Vokabel **ein Objekt**?

Die meisten Vokabeln haben nicht nur eine, sondern mehrere mögliche Übersetzungen.
Das Wort „laufen“ kann auf Englisch z.B. mit „to run“ oder „to jog“ übersetzt werden.
Die Anzahl der Übersetzungen ist dabei nicht festgelegt.
Es wäre also sinnvoll, für jede Vokabel eine Liste von Übersetzungen zu speichern.

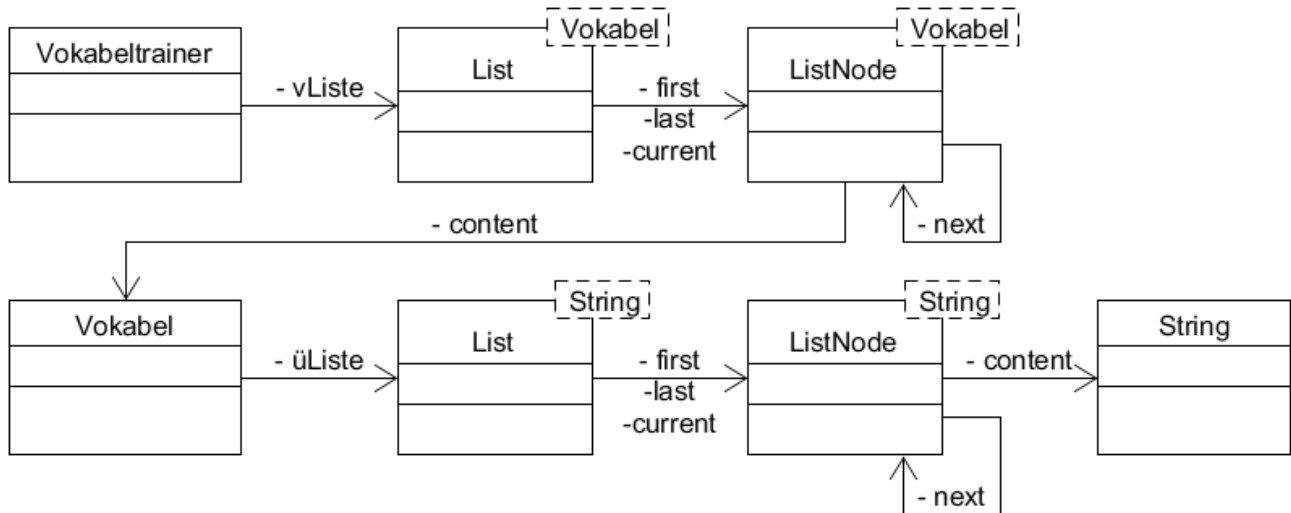
Das bedeutet: Der Vokabeltrainer hat eine Liste von Vokabeln, und jede Vokabel hat eine Liste von Übersetzungen. Solche Datenstrukturen nennt man „verschachtelt“.

Das folgende Klassendiagramm stellt einen einfachen Entwurf dar. `vListe` enthält die Liste der Vokabeln, `üListe` jeweils die Liste der Übersetzungen. Die „Verschachtelung“ drückt sich hier (nur) durch die Angabe des `ContentTypes` für die List-Objekte aus:



Aufgabe 1

Das folgende Diagramm zeigt zusätzlich auch die Klasse ListNode mit den zugehörigen Referenzen. Dadurch werden die Verbindungen aller Objekte des Projekts sichtbar:



Zeichne ein **Objektdiagramm** des Vokabeltrainers.

Er soll zwei Vokabeln mit jeweils zwei Übersetzungen enthalten, zum Beispiel:

Begriff „laufen“ → Übersetzungen „run“ und „jog“

Begriff „schauen“ → Übersetzungen „look“ und „watch“

Aufgabe 2

Das Ziel dieser Aufgabe ist nicht, einen funktionstüchtigen Vokabeltrainer zu programmieren.

Es geht hier darum, die Funktionsweise verschachtelter Datenstrukturen zu verstehen.

Der Programmieraufwand soll also überschaubar bleiben.

Nutze für die Aufgabe die Dokumentation auf der nächsten Seite.

- Entwickle einen **Algorithmus** für die Methode `ausgeben()` der Klasse **Vokabeltrainer**. Sie gibt alle in der Liste enthaltenen Vokabeln mit jeweils allen gespeicherten Übersetzungen auf der Konsole aus. Stelle den Algorithmus in Umgangssprache oder als Aktivitätsdiagramm dar.
- Implementiere** die in a) entwickelte Methode `ausgeben()` mithilfe der ausgeteilten Vorlage.

Dokumentation

Klasse Vokabel

`Vokabel(String pb)` (Konstruktor)

Erzeugt eine neue Vokabel und setzt dabei das Attribut `begriff` auf den Werte des Parameters `pb`.

Erzeugt außerdem die leere Liste `übersetzungen` für die Übersetzungen des Begriffs.

`String getBegriff()`

Gibt den zur Vokabel gehörenden Begriff (z. B. das Wort „laufen“) zurück.

`void übersHinzufügen(String pü)`

Fügt die im Parameter `pü` gegebene Übersetzung (z.B. das Wort „jog“) zur Liste der Übersetzungen hinzu.

`List<String> getÜbersetzungen()`

Gibt die Liste der Übersetzungen zurück.