Praktikum Software-Engineering

Systementwurf: Stilanalyse von Texten

Jan Tammen (jan.tammen@fh-konstanz.de)
Christoph Eck (christoph.eck@fh-konstanz.de)

23. November 2005

Großgruppe:
Eck/Tammen
Apell/Jehle
Lehmann/Pfeifer

Inhaltsverzeichnis

1 Festlegung der Einflussfaktoren auf Softwarearchitektur

1.1 Einsatzbedingungen

Definiert im Pflichtenheft.

1.2 Umgebungs- und Randbedingungen

Definiert im Pflichtenheft.

1.3 Nichtfunktionale und Qualitätsanforderungen

1.3.1 Nichtfunktionale Anforderungen

Es werden keine besonderen Anforderungen bezüglich Internationalisierung, Skalierbarkeit und Wiederverwendbarkeit gestellt.

1.3.2 Qualitätsanforderungen

Es werden keine besonderen Anforderungen bezüglich Zuverlässigkeit, Änderbarkeit und Effizienz gestellt.

2 Entwurf der Benutzungsschnittstelle

Detaillierte Entwürfe der grafischen Benutzungsschnittstelle sind im Pflichtenheft enthalten. Das folgende Diagramm zeigt den schematischen Aufbau der Oberfläche.

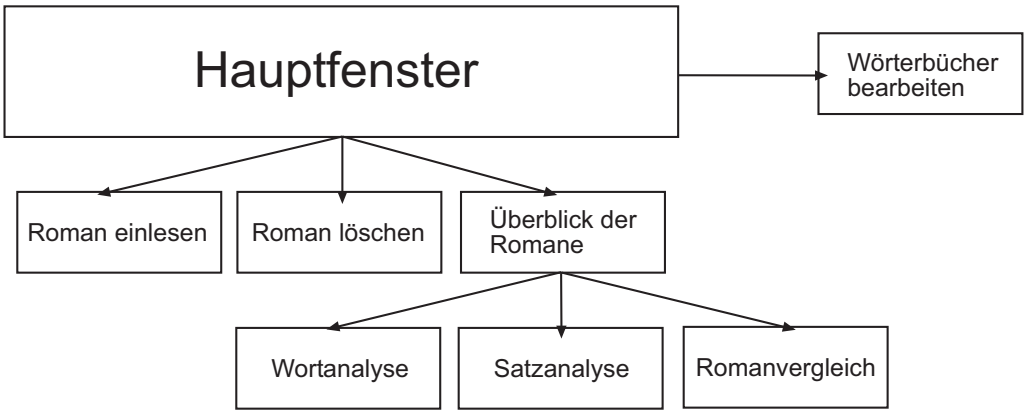

Abbildung 1: Benutzungsschnittstelle

3 Grobarchitektur

Das folgende Diagramm stellt die im System enthaltenen Komponenten und deren Abhängigkeiten untereinander dar.

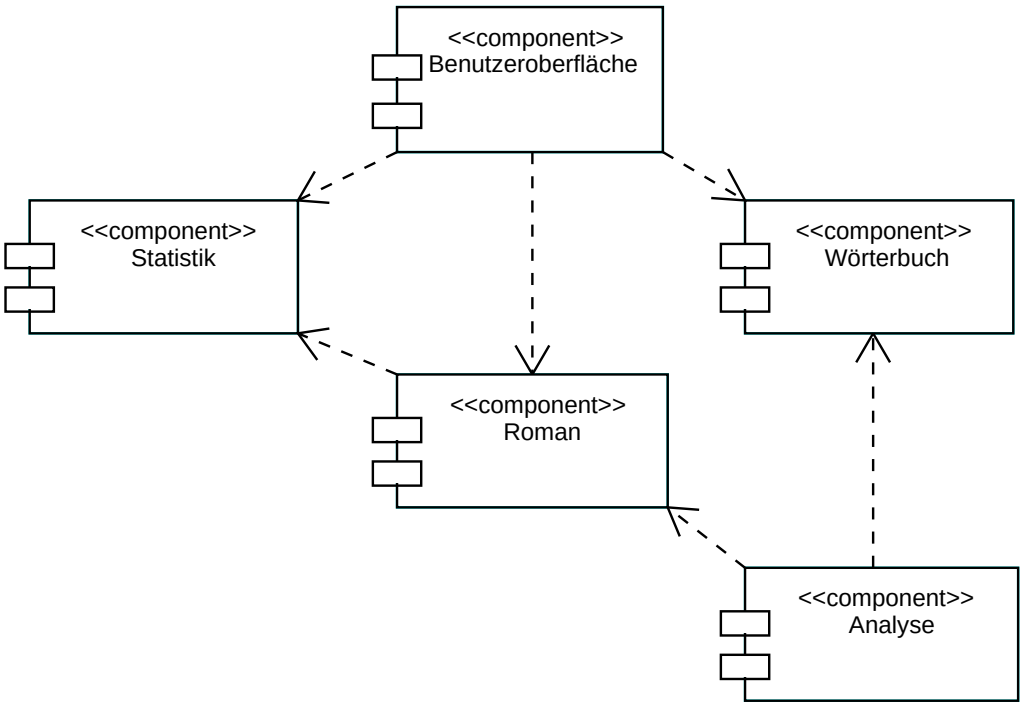

Abbildung 2: Grobarchitektur: Komponenten

3.1 Beschreibung der Komponenten

3.1.1 Komponente Benutzeroberfläche

Die Komponente Benutzeroberfläche ist verantwortlich für die Aufbereitung und Anzeige der Analyse-Ergebnisse sowie für das Entgegennehmen der Benutzereingaben. Sie benutzt die Komponenten Statistik, Roman sowie Wörterbuch.

3.1.2 Komponente Roman

Die Komponente Roman speichert den Text eines Romans ab und verwaltet diesen. Die Komponente beinhaltet folgende Module:

- RomanVerwaltung
- Roman
- Text
- Textdatei
- Satz
- Wort

Die Komponente Roman benutzt die Komponenten Statistik und Wörterbuch.

3.1.3 Komponente Analyse

Die Komponente Analyse analysiert den eingelesenen Roman, teilt die Analyse in Wort- und Satzanalyse auf und kann verschiedene Romane miteinander vergleichen. Die Komponente beinhaltet folgende Module:

- Wortanalyse
- Satzanalyse
- Vergleichsanalyse
- Analyse

Die Komponente Analyse benutzt die Komponenten Roman und Wörterbuch.

3.1.4 Komponente Wörterbuch

Die Komponente Wörterbuch verwaltet die Wörterbücher. Die Komponente beinhaltet folgende Module:

- WörterbuchVerwaltung
- Wörterbuch
- Wortart

Die Komponente Wörterbuch benutzt keine Komponente.

3.1.5 Komponente Statistik

Die Komponente Statistik ist für die Verwaltung der bei der Textanalyse anfallenden statistischen Daten zuständig. Die Komponente beinhaltet folgende Module:

- Wortstatistik
- Satzstatistik

Die Komponente Statistik benutzt keine Komponente.

4 Feinarchitektur

Die folgenden Abschnitte beinhalten die detaillierten Diagramme der einzelnen Komponenten sowie die Beschreibungen der in ihnen enthaltenen Module (Klassen).

4.1 Komponente Roman

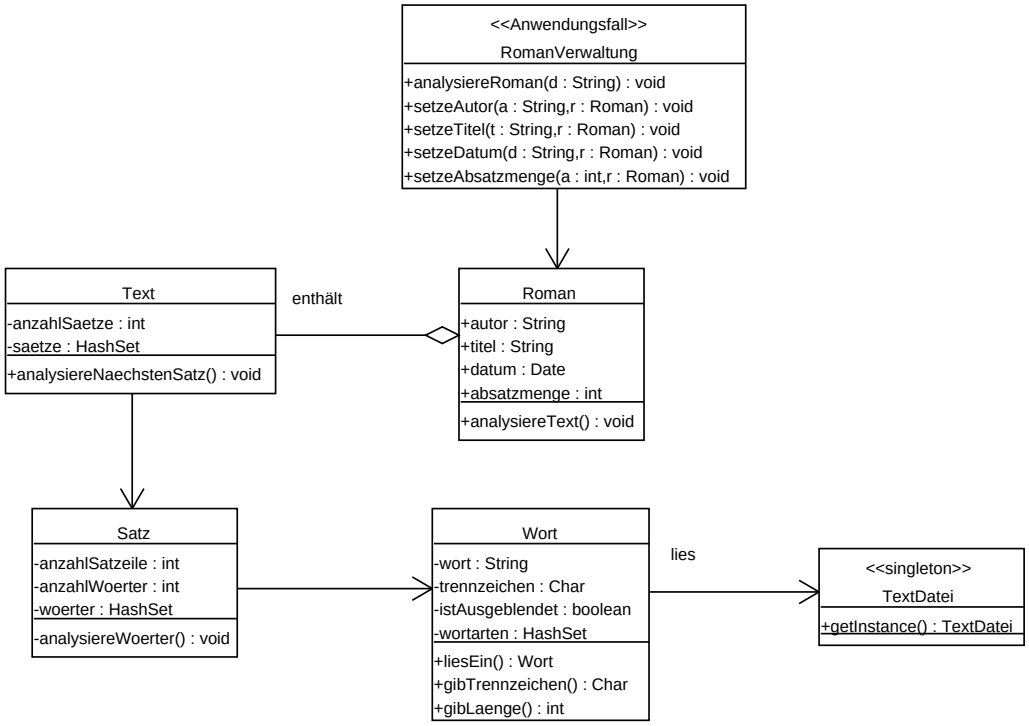

Abbildung 3: Komponente Roman

4.1.1 Klasse RomanVerwaltung

Diese Klasse dient als Schnittstelle zwischen der Benutzeroberfläche und der Klasse `Roman`. Über sie werden die Angaben zu einem Roman wie Autorname, Erscheinungsdatum usw. manipuliert. Außerdem startet der Benutzer über diese Klasse die Textanalyse eines Romans. Sie enthält die folgenden Operationen:

Operation	Aufgabe	Parameter
`analysiereRoman`	Stößt die Analyse des Romans an	*in*: Quell-Dateiname (Typ: `String`)
`setzeAutor`	Setzt den Autornamen des Romans	*in*: Roman (Typ: `Roman`), Autor-Namen (Typ: `String`)
`setzeTitel`	Setzt den Titel des Romans	*in*: Roman (Typ: `Roman`), Titel (Typ: `String`)
`setzeDatum`	Setzt das Erscheinungsdatum des Romans	*in*: Roman (Typ: `Roman`), Erscheinungsdatum (Typ: `Date`)
`setzeAbsatzmenge`	Setzt die Absatzmenge des Romans	*in*: Roman (Typ: `Roman`), Absatzmenge (Typ: `Integer`)

Tabelle 1: Operationen der Klasse RomanVerwaltung

4.1.2 Klasse Roman

Diese Klasse repräsentiert einen Roman (Anwendungsweltobjekt) und enthält Informationen über diesen. Die Klasse wird benutzt von der Klasse `Analyse` und benutzt selbst die Klasse `Text` („hat-ein“-Beziehung).

Operation	Aufgabe	Parameter
`analysiereText`	Löst die Analyse des Textes aus	keine
`setzeAutor`	Setzt den Autornamen des Romans	*in*: Autor-Namen (Typ: `String`)
`setzeTitel`	Setzt den Titel des Romans	*in*: Titel (Typ: `String`)
`setzeDatum`	Setzt das Erscheinungsdatum des Romans	*in*: Erscheinungsdatum (Typ: `Date`)
`setzeAbsatzmenge`	Setzt die Absatzmenge des Romans	*in*: Absatzmenge (Typ: `Integer`)

Tabelle 2: Operationen der Klasse Roman

4.1.3 Klasse Text

Diese Klasse nimmt den eigentlichen Text eines Romans auf. Ein Text wird dabei als Menge von Sätzen angesehen; deshalb enthält die Klasse `Text` einen Verweis auf einen speziellen mengenbasierten Datencontainer (HashSet), in welchem die einzelnen Sätze effizient abgelegt werden können. Die Klasse wird benutzt von der Klasse `Roman` und benutzt selbst die Klasse `Satz`.

Operation	Aufgabe	Parameter
`analysiereNaechstenSatz`	Wird innerhalb der Klasse `Roman` aufgerufen und untersucht den nächsten Satz des Quelltextes	keine

Tabelle 3: Operationen der Klasse Text

4.1.4 Klasse TextDatei

Diese Klasse bietet einen Zugriff auf die Quelldatei des zu analysierenden Romans. Damit es in der Anwendung stets nur einen gleichzeitigen Zugriff auf eine Datei gibt, wird diese Klasse mithilfe des Singleton-Musters implementiert. Die Klasse wird benutzt von den Klassen `Wort` und `Analyse`.

Operation	Aufgabe	Parameter
`getInstance`	Gibt aktuelle Instanz zurück (Typ: `TextDatei`). Wird innerhalb der Klasse `Wort` aufgerufen, um anschließend von der Datei lesen zu können	keine

Tabelle 4: Operationen der Klasse TextDatei

4.1.5 Klasse Satz

Diese Klasse ist für die Speicherung von Sätzen zuständig. Ein Satz wird dabei als Menge von Wörtern angesehen; deshalb enthält die Klasse `Satz` einen Verweis auf einen speziellen mengenbasierten Datencontainer (HashSet), in welchem die einzelnen Wörter effizient abgelegt werden können. Weiterhin enthält sie Informationen über die Anzahl der Wörter sowie Anzahl der Satzteile innerhalb des Satzes. Die Klasse wird benutzt von der Klasse `Text` und benutzt selbst die Klassen `Wort` und `Satzstatistik`.

Operation	Aufgabe	Parameter
`analysiereWoerter`	Wird innerhalb der Klasse `Text` aufgerufen und extrahiert solange Wörter aus dem Quelltext, bis das Satzende erreicht ist	keine

Tabelle 5: Operationen der Klasse TextSatz

4.1.6 Klasse Wort

Diese Klasse bildet die kleinste im System bekannte Einheit eines Textes ab – das Wort. Ein Wort besteht aus einer Menge von Zeichen, welche in der Klasse in einem `String`-Datencontainer

gespeichert werden. Ebenso wird dasjenige Zeichen (Typ: `Char`) gespeichert, mit welchem das Wort beendet wurde, also z.B. ein Leerzeichen oder Komma. Um das Wort bei der Berechnung der Statistik auf Wunsch des Nutzers ignorieren zu können, gibt es ein enstprechendes Flag (`istAusgeblendet`).

Beim Einlesen eines Wortes aus dem Quelltext muss zunächst überprüft werden, ob das Wort bereits im Wörterbuch vorhanden ist und wenn dies der Fall ist, welchen Wortarten (s. Kapitel 4.3.3) es zugeordnet ist. Dabei ist es vorteilhaft, stets die Verknüpfungen zwischen Wort und Wortarten präsent zu haben – daher enthält ein Wort in einem entsprechenden Datencontainer (`wortarten`, HashSet) Referenzen auf die zugehörigen Wortarten.

Die Klasse wird benutzt von der Klasse `Satz` und benutzt selbst die Klassen `TextDatei`, `Wörterbuch` und `Wortstatistik`.

Operation	**Aufgabe**	**Parameter**
`liesEin`	Solange zeichen- bzw. wortweise aus der Quelldatei lesen, bis definiertes Trennzeichen erreicht ist, anschließend Selbstreferenz zurückgeben (Typ: `Wort`)	keine
`gibLaenge`	Länge des Wortes/Strings zurückgeben (Typ: `Integer`)	keine
`gibTrennzeichen`	Gibt das Trennzeichen zurück (Typ: `Char`), anhand dessen die Klasse `Satz` erkennen kann, ob es sich um ein Satzende oder z.B. den Anfang eines Satzteils handelt	keine

Tabelle 6: Operationen der Klasse Wort

4.2 Komponente Analyse

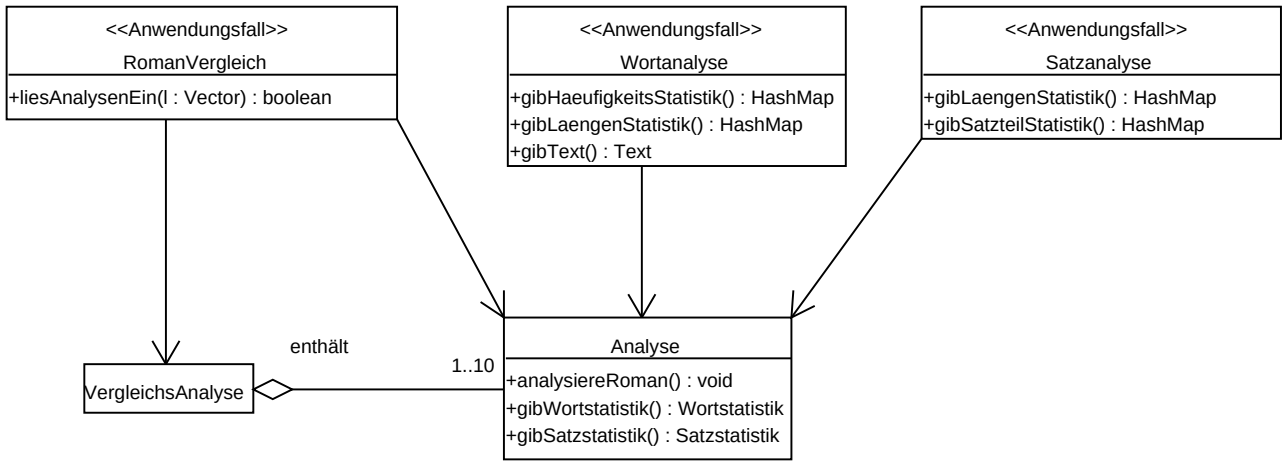

Abbildung 4: Komponente Analyse

4.2.1 Klasse RomanVergleich

Diese Klasse dient als Schnittstelle zwischen der Benutzeroberfläche und der Klasse `Vergleichsanalyse`. Sie dient dazu, mehrere ausgewählte Roman-Analysen vergleichen zu können. Die Klasse benutzt die Klassen `VergleichsAnalyse` und `Analyse`.

Operation	Aufgabe	Parameter
`liesAnalysenEin`	Liest für jeden der gewählten Romane die Analyse ein	*in*: Romane (Typ: `Vector`)

Tabelle 7: Operationen der Klasse RomanVergleich

4.2.2 Klasse Wortanalyse

Diese Klasse dient als Schnittstelle zwischen der Benutzeroberfläche und der Klasse `Analyse`. Sie dient dazu, die Wortanalyse eines Romans geeignet aufzubereiten bzw. an die Oberfläche weiterzuleiten. Die Klasse benutzt die Klasse `Analyse`.

Operation	Aufgabe	Parameter
`gibHaeufigkeitsStatistik`	Häufigkeits-Statistik zurückgeben (Typ: `HashMap`)	keine
`gibLaengenStatistik`	Längen-Statistik zurückgeben (Typ: `HashMap`)	keine
`gibText`	Text des Romans zurückgeben (Typ: `Text`), für die Anzeige in der Wortanalyse	keine

Tabelle 8: Operationen der Klasse Wortanalyse

4.2.3 Klasse Satzanalyse

Diese Klasse dient als Schnittstelle zwischen der Benutzeroberfläche und der Klasse `Analyse`. Sie dient dazu, die Satzanalyse eines Romans geeignet aufzubereiten bzw. an die Oberfläche weiterzuleiten. Die Klasse benutzt die Klasse `Analyse`.

Operation	Aufgabe	Parameter
`gibHaeufigkeitsStatistik`	Häufigkeits-Statistik zurückgeben (Typ: `HashMap`)	keine
`gibLaengenStatistik`	Längen-Statistik zurückgeben (Typ: `HashMap`)	keine

Tabelle 9: Operationen der Klasse Satzanalyse

4.2.4 Klasse Vergleichsanalyse

Diese Klasse dient dazu, mehrere Analysen verschiedener Romane zu vergleichen. Die Klasse benutzt daher die Klasse `Analyse`.

4.2.5 Klasse Analyse

Diese Klasse verwaltet die für die Textanalyse benötigten Elemente: den Roman sowie die Wort- und Satzstatistik. Die Klasse benutzt daher die Klassen `Roman`, `Wortstatistik` und `Satzstatistik` und wird benutzt von den Klassen `VergleichsAnalyse`, `RomanVergleich`, `Wortanalyse`, `Satzanalye` und `RomanVerwaltung`.

Operation	Aufgabe	Parameter
`analysiereRoman`	Stößt die Erstellung der Text-Analyse des Romans an	keine
`gibWortstatistik`	Gibt die Wortstatistik zurück (Typ: `Wortstatistik`)	keine
`gibSatzstatistik`	Gibt die Satzstatistik zurück (Typ: `Satzstatistik`)	keine

Tabelle 10: Operationen der Klasse Analyse

4.3 Komponente Wörterbuch

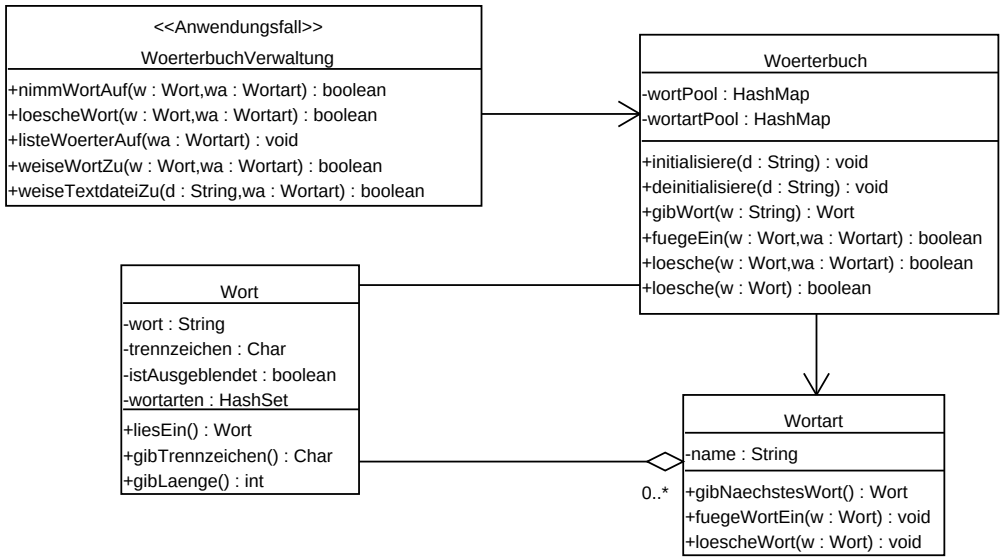

Abbildung 5: Komponente Wörterbuch

4.3.1 Klasse WoerterbuchVerwaltung

Diese Klasse verwaltet alle vorhandenen Wortarten. Alle bei der Analyse nicht zugeordneten Wörter gehören der Wortart „Sonstige Wörter“ an. Mit einer Textdatei mit Wörtern gleicher Wortart können die neuen Wörter in eine bestehende oder neue Wortart eingelesen werden.

Operation	**Aufgabe**	**Parameter**
`nimmWortAuf`	Wörter, die noch nicht spezifiziert sind, werden in die Wortart „Sonstige Wörter“ gespeichert	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`loescheWort`	Wort löschen, in jeder Wortart	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`listeWoerterAuf`	listet alle Wörter der Wortart auf	*in*: Wortart (Typ: `Wortart`)
`weiseWortZu`	Das Wort wird der Wortart zugeordnet	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`weiseTextdateiZu`	Durch Öffnen der gewünschten Textdatei werden die Wörter einer Wortart zugewiesen. Beim Einlesen müssen doppelte Wörter erkannt werden.	*in*: Dateiname (Typ: `String`), Wortart (Typ: `Wortart`)

Tabelle 11: Operationen der Klasse WoerterbuchVerwaltung

4.3.2 Klasse Woerterbuch

Diese Klasse implementiert das Wörterbuch und stellt die entsprechenden Methoden und Datenstrukturen für das Verwalten der Wörter zur Verfügung. Die Klasse wird benutzt von den Klassen `WoerterbuchVerwaltung`, `Analyse` und `Wort`; sie benutzt selbst die Klasse `Wortart`.

Operation	Aufgabe	Parameter
`initialisiere`	baut die Datenstruktur auf	*in*: Dateiname (Typ: `String`)
`deinitialisiere`	löscht die Datenstruktur im Speicher und speichert sie (serialisiert) in der Datei ab	*in*: Dateiname (Typ: `String`)
`gibWort`	Gibt das vollständige Wort – sofern im Datenbestand vorhanden – zurück	*in*: Wort (Typ: `Wort`)
`fuegeEin`	Wort der entsprechenden Wortart zuweisen	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`loesche`	lösche ein Wort (nur in der spezifizierten Wortart)	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`loesche`	lösche ein Wort (in allen Wortarten)	*in*: Wort (Typ: `Wort`)

Tabelle 12: Operationen der Klasse Woerterbuch

4.3.3 Klasse Wortart

Mithilfe dieser Klasse wird die im Pflichtenheft festgelegte Klassifizierbarkeit von Wörtern des Wörterbuchs ermöglicht. Ein Wort kann beliebig vielen Wortarten (z.B. „Substantiv“, „Science-Fiction-Helden“) zugewiesen werden und einer Wortart können ebenso beliebig viele Wörter zugewiesen werden. Details zur Realisierung der Datenstruktur finden sich in Kapitel 4.3.4.

Die Klasse wird benutzt von der Klasse `Woerterbuch` und benutzt selbst die Klasse `Wort`.

Operation	Aufgabe	Parameter
`gibNaechstesWort`	Gibt das nächste zu dieser Wortart gehörige Wort zurück (Typ: `Wort`)	keine
`fuegeWortEin`	Stellt eine Zuordnung zwischen dieser Wortart und dem übergebenen Wort her	*in*: Wort (Typ: `Wort`)
`loescheWort`	Entfernt die Zuordnung zwischen Wortart und übergebenem Wort	*in*: Wort (Typ: `Wort`)

Tabelle 13: Operationen der Klasse Wortart

4.3.4 Datenstruktur der Wörterbuch-Komponente

In diesem Abschnitt soll die für die Realisierung des Wörterbuchs verwendete Datenstruktur erläutert werden. Folgende Anforderungen werden an diese Datenstruktur gestellt:

- Ein Wort kann beliebig vielen Wortarten zugewiesen werden

- Einer Wortart können beliebig viele Wörter zugewiesen werden
- Suche nach einem Wort soll möglichst effizient sein
- Einfügen eines neuen Wortes soll möglichst effizient sein

Das Diagramm in Abbildung 6 soll die bereits im Klassendiagramm angedeutete Struktur verdeutlichen.

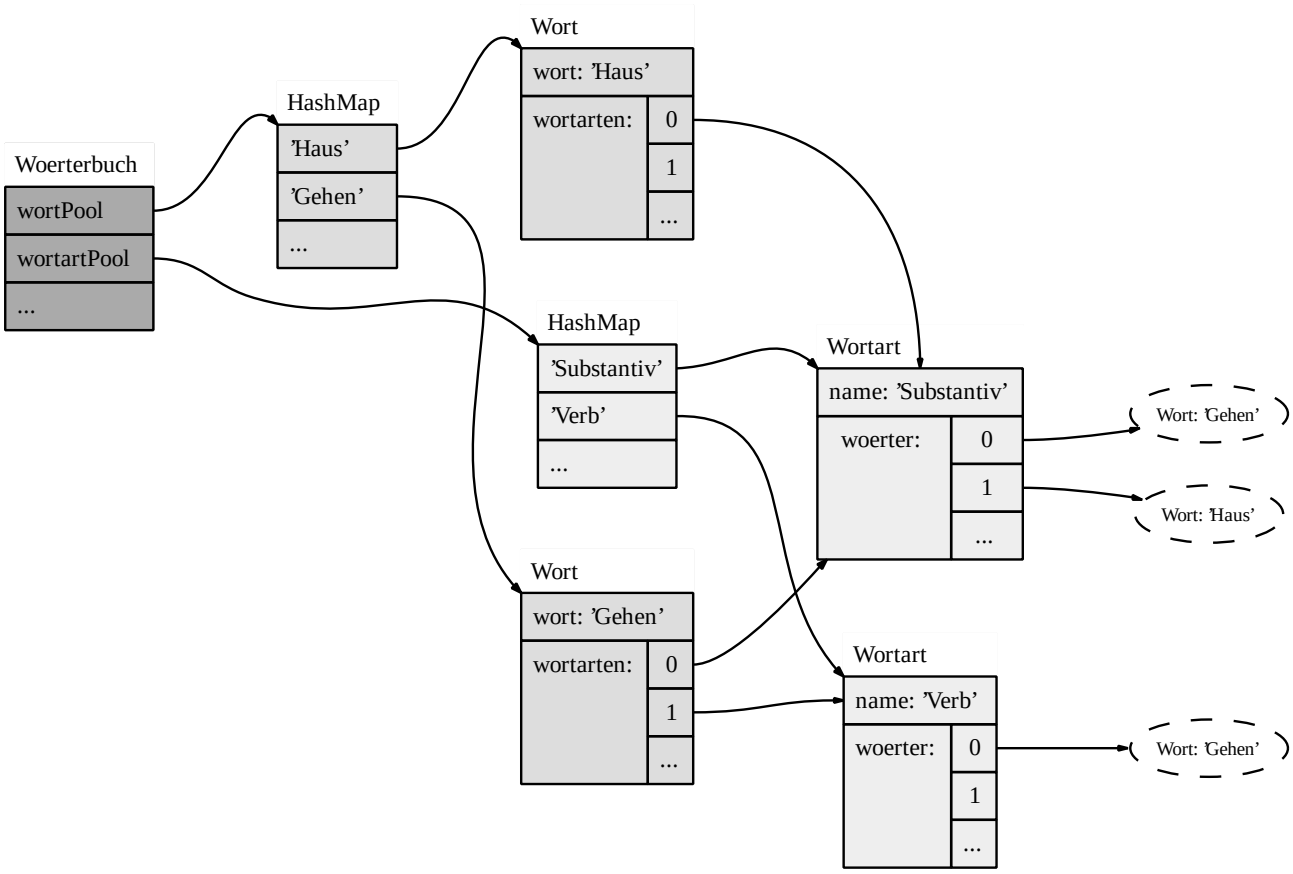

Abbildung 6: Datenstruktur Wörterbuch

Die Grafik stellt den „Speicher-Auszug“ einiger Objekte dar, nachdem das Wörterbuch entsprechend initialisiert wurde. Folgende Elemente sind hier aufgeführt:

1. `Woerterbuch`
   - enthält in der Variablen `wortPool` eine Referenz auf eine Datenstruktur (HashMap), welche alle Wortarten aufnimmt (bzw. Referenzen auf `Wort`-Objekte)
   - enthält in der Variablen `wortartPool` eine Referenz auf eine Datenstruktur (HashMap), welche alle Wortarten aufnimmt (bzw. Referenzen auf `Wortart`-Objekte)

2. `Wort`
    - enthält in der Variablen `wort` das tatsächliche Wort als String („Haus“ und „Gehen“)
    - enthält in der Variablen `wortarten` eine Referenz auf eine Datenstruktur[1] (HashSet), welche Referenzen auf diejenigen `Wortart`-Objekte enthält, zu denen dieses Wort zugehörig ist. Im Beispiel ist also das Wort „Haus“ der Wortart „Substantiv“ zugeordnet; das Wort „Gehen“ ist den Wortarten „Substantiv“ und „Verb“ zugeordnet.
3. `Wortart`
    - enthält in der Variablen `name` die Bezeichnung der Wortart als String („Substantiv“ und „Verb“)
    - enthält in der Variablen `woerter` eine Referenz auf eine Datenstruktur[2] (HashSet), welche Referenzen auf diejenigen `Wort`-Objekte enthält, welche dieser Wortart zugeordnet sind. Im Beispiel enthält also die Wortart „Substantiv“ Referenzen auf die Wörter „Haus“ und „Gehen“ – im Diagramm sind diese Referenzen aus Übersichtsgründen jedoch nur angedeutet (gestrichelte Ellipsen).

Mithilfe dieser Datenstruktur können die oben genannten Anforderungen erfüllt werden.

4.4 Komponente Statistik

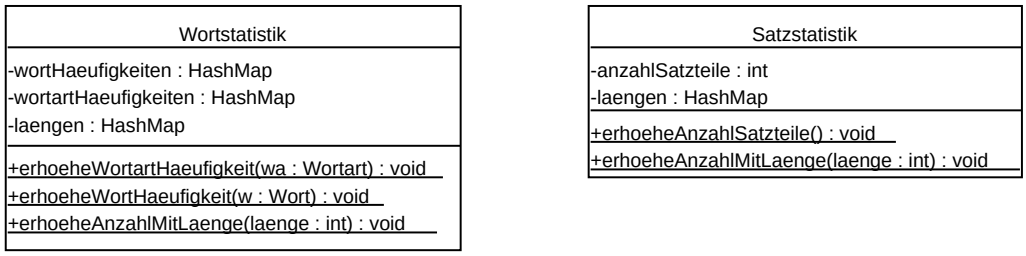

Abbildung 7: Komponente Statistik

4.4.1 Klasse Wortstatistik

Diese Klasse speichert statisch die Wort-Häufigkeiten (`wortHaeufigkeiten`), die Wortart-Häufigkeiten (`wortartHaeufigkeiten`) und die Längen (`laengen`) aller Wörter in einer HashMap.

[1] Hinweis: Aus Übersichtsgründen ist im Diagramm die Datenstruktur direkt in das `Wort`-Objekt integriert
[2] Hinweis: Aus Übersichtsgründen ist im Diagramm die Datenstruktur direkt in das `Wortart`-Objekt integriert

Operation	Aufgabe	Parameter
`erhoeheWortartHaeufigkeit`	erhöhe die Anzahl der Wörter dieser Wortart um eins	*in*: Wortart (Typ: `Wortart`)
`erhoeheWortHaeufigkeit`	erhöhe die Anzahl des Wortes um eins	*in*: Wort (Typ: `Wort`)
`erhoeheAnzahlMitLaenge`	erhöhe die Anzahl des Wortes gleicher Länge	*in*: laenge (Typ: `int`)

Tabelle 14: Operationen der Klasse Wortstatistik

4.4.2 Klasse Satzstatistik

Diese Klasse speichert statisch die Anzahl der Satzteile (`anzahlSatzteile`) und die Anzahl der jeweiligen Satzlängen (`laengen`) in einer HashMap.

Operation	Aufgabe	Parameter
`erhoeheAnzahlSatzteile`	erhöhe die Anzahl der Satzteile	*in*: keine
`erhoeheAnzahlMitLaenge`	erhöhe die Anzahl des Satzes mit gleicher Länge	*in*: laenge (Typ: `int`)

Tabelle 15: Operationen der Klasse Satzstatistik