Karlsruher Institut für Technologie

Programmieren WS 2012/2013

Forschungsgruppe: Verifikation trifft Algorithmik
http://verialg.iti.uka.de/prog_ws12.php
Dr. Carsten Sinz, Florian Merz, Markus Iser

Übungsblatt 4
(v1.0)

Ausgabe:	03.12.2012
Abgabe:	17.12.2012, 13Uhr
Besprechung:	07.01.2013 - 11.01.2013

Hinweise

Beachten Sie bei Ihrer Abgabe inbesondere auf das Einhalten ...

- der maximale Zeilenlänge von 120 Zeichen,
- der Kompilierbarkeit,
- der Whitespace-Regeln [1],
- der JavaDoc-Regeln [2] und
- der vorgegeben Paketnamen und Paketzugehörigkeiten der Klassen.

Beachten Sie auch, dass die Einhaltung der Regeln vom Praktomaten automatisiert geprüft wird. Besteht eine Abgabe eine dieser Prüfungen nicht, so wird sie hiedurch automatisch ungültig und nicht in eine Bewertung einbezogen. Fangen Sie daher frühzeitig mit dem Lösen Ihrer Aufgaben an und testen Sie die Abgabe im Praktomaten.

Neben den spezifisch für die Aufgaben gestellten Anforderungen gelten für dieses Blatt die folgenden Erwartungen, deren Nichtbeachtung zu Punktabzügen führt.

- Klassen, Methoden und Attribute erhalten sinnvoll gewählte Sichtbarkeiten.
- Es werden keine externen Pakete verwendet. Das Paket `java.lang` ist hiervon ausgenommen.
- Im normalen Programmlauf soll Ihre Abgabe keine Ausgaben, beispielswiese zum Debugging, mehr produzieren.

Denken Sie daran Ihre Lösung selbständig zu testen, um unnötige Punktabzüge für funktionale Fehler zu vermeiden.

Klären Sie offene Fragen zur Aufgabenstellung im Forum des Praktomaten.

[1] `http://baldur.iti.uka.de/programmieren/whitespace-checks.xml`
[2] `http://baldur.iti.uka.de/programmieren/javadoc-checks.xml`

Einleitung

Auf den bisherigen Übungsblättern haben Sie lediglich kleinere Programme und Programmteile implementieren müssen. In diesem Übungsblatt werden Sie nun zum ersten Mal mit einem größeren Softwareprojekt konfrontiert sein, dessen Funktionalität über mehrere Klassen hinweg verteilt realisiert ist. Sie sollen üben sich in ein solches Projekt einzuarbeiten und in ihm zurechtzufinden, ohne dabei die Übersicht zu verlieren.

Am Ende dieses Übungsblatts werden Sie den Kern eines Textverarbeitungsprogramms implementiert haben. Alle Klassen, die Sie hierfür benötigen, stehen auf der Webseite der Vorlesung[3] zur Verfügung. Ihre Aufgabe ist es dabei nicht, die Klassen zu entwerfen, sondern lediglich die vorgegebenen Methoden anhand ihrer Javadoc-Beschreibung zu implementieren. Sie werden lernen, wie Sie ein großes Projekt erfolgreich abschließen können, indem Sie es in kleinere Aufgaben aufteilen, die sie unabhängig voneinander lösen können.

Da Sie in der Vorlesung nicht gelernt haben graphische Benutzeroberflächen zu programmieren wird die Steuerung des Textverarbeitungsprogramms direkt über die Klasse `WordProcessor` erfolgen. Dennoch ist diese Klasse so gestaltet, dass sie der Steuerung eines Textverarbeitungsprogramms über eine graphische Oberfläche ähnelt. Sie müssen diese Klasse nicht selbst implementieren, Sie finden Sie vollständig implementiert auf unserer Webseite.

Zunächst geben wir Ihnen eine kleine Übersicht über das Endprodukt.

Übersicht

Unser Textverarbeitungsprogramm kann genau ein *Dokument* öffnen. Ein solches Textdokument enthält formatierten Text, wobei als Formatierungsmöglichkeiten **bold**, also fett, und *italic*, also kursiv, zur Verfügung stehen. Da unterschiedliche Teile des Textes mit unterschiedlichen Formatierungen versehen sein können, ist der Text des Dokuments in sogenannte *Textfragmente* aufgeteilt. Jedes Fragment ist dabei ein zusammenhängendes Stück Text mit einheitlicher Formatierung. Die Fragmente eines Dokuments werden mit Hilfe einer doppelt-verketteten Liste verwaltet.

Um Text und Formatierungen verändern zu können stehen sogenannte *Cursor* zur Verfügung. Diese sind konzeptionell eng an Textcursor aus realen Textverarbeitungsprogrammen angelehnt und haben auch eine starke Ähnlichkeit zum Iterator-Konzept, das Sie in der Vorlesung kennenlernen werden.

Schließlich kann der Text eines Dokument auch noch als unformatierter Text sowie als `HTML`-Code ausgegeben werden.

Vorgehensweise

In Teil A dieses Übungsblattes sollen Sie alle notwendigen Klassen implementieren um die grundlegenden Funktionen von Textfragmenten umzusetzen. Beginnen Sie mit der Implementierung der Klasse `Style`, wie in Aufgabe A.1 beschrieben. Fahren Sie in Aufgabe A.2 mit der Implementierung der Klasse `Fragment` selbst fort. Schließen Sie dann Teil A dieses Übungsblattes mit der in Aufgabe A.3 beschriebenen Implementierung der Klasse `FragmentCursor` ab. Hierbei handelt es sich um einen Cursor welcher nur innerhalb eines einzigen Textfragments bewegt werden kann.

In Teil B sollen Sie dann zunächst eine doppelt-verkettete Liste für Textfragmente erstellen. Hier-

[3] `http://baldur.iti.kit.edu/programmieren`

für müssen Sie die Klassen `FragmentList`, die eine Liste von Fragmenten darstellt, die Klasse `FragmentList.Element`, die Elemente in einer solchen Liste darstellt, sowie die Klasse `FragmentList.Cursor`, die einen Cursor auf eine solche Liste umsetzt, implementieren. Beachten Sie, das Sie für die Implementierung der doppelt-verketteten Liste die Fragmente selbst noch nicht implementiert haben müssen, da diese Liste lediglich Referenzen auf `Fragmente` verwaltet, diese jedoch selbst nicht benutzt.

Eine echte Herausforderung folgt in Teilaufgabe B.2, in der Sie die Klasse `DocumentCursor` implementieren sollen, die einen Cursor zur Verfügung stellt, der sich in einem Dokument über Fragmentgrenzen hinweg bewegen kann. Intern soll diese Klasse einen `FragmentList.Cursor` benutzen um sich zwischen Fragmenten zu bewegen, sowie einen `FragmentCursor` für die Bewegung innerhalb eines Fragments. In Teilaufgabe B.3 schreiben Sie abschließend die Klasse `Document`, in der Sie die Einzelteile zusammenfügen.

Abbildung 1 zeigt eine grafische Übersicht über das gesamte Projekt, aus der noch einmal ersichtlich wird, dass das Projekt aus mehreren, relativ unabhängigen Teilen besteht. Achten Sie auch bei Ihrer Implementierung auf den Aspekt, dass diese Teile möglichst unabhängig voneinander sind. So müssen zum Beispiel Fragmentlisten nicht wissen, wie Fragmente aufgebaut sind und Fragmente wiederum müssen nicht wissen, dass sie in Listen verwaltet werden. Ferner bietet die Klasse `DocumentCursor` nach außen hin Methoden an die auf Zeichen basieren und versteckt damit die zugrunde liegende Fragment-basierte Implementierung.

Ausnahmebehandlung

In den zu implementierenden Klassen gibt es einige Methoden, deren korrektes Funktionieren nur gegeben ist, wenn bestimmte Vorbedingungen zum Zeitpunkt des Methodenaufrufs gelten. Um subtile Fehler im Programmablauf zu vermeiden, ist es sinnvoll diese am Anfang der Methode zu überprüfen und das Programm dann sofort zu beenden. Da die eigentlich hierfür gedachten Java-Sprachkonstrukte zur Ausnahmebehandlung (Exceptions) noch nicht Stoff der Vorlesung waren, stellen wir Ihnen hierfür die statische Methode `abortIf(boolean condition, String message)` aus der Klasse `Helper` zur Verfügung.

Die Methode prüft ob der boolesche Ausdruck `condition` wahr ist und gibt in diesem Fall die Nachricht `message` auf der Konsole aus und beendet das Programm. Ist der Boolesche Ausdruck falsch so ist die Methode ohne Effekt. Während Sie einerseits diese Methodenaufrufe an den richtigen Stellen in den Code einfügen sollen, sollen sie andererseits dafür sorgen, dass bei Aufruf dieser Methoden die Vorbedingungen immer erfüllt sind und daher `Helper.abortIf()` in Ihrer Lösung niemals das Programm wirklich beendet.

Die explizite Behandlung von Fehlern mithilfe selbst verfasster Fehlermeldungen wird Ihnen bei der Entwicklung helfen. Benutzen Sie daher `Helper.abortIf()` mindestens an den Stellen, an denen wir in der JavaDoc-Beschreibung explizit darauf hingewiesen haben, dass die Programmausführung unter bestimmten Bedingungen abbrechen soll.

Unit-Tests

Um Ihnen die Arbeit ein wenig zu erleichtern, stellen wir Ihnen sogenannte Unit-Tests zur Verfügung. Benutzen Sie diese, um die bereits erstellten Komponenten auf eine korrekte Arbeitsweise zu prüfen. Um die Unit-Tests auszuführen, laden Sie sich bitte die JUnit-Bibliothek unter `https://github.com/KentBeck/junit/downloads` herunter. Beachten Sie, dass diese sowohl beim

kompilieren der Tests mit `javac`, als auch beim Ausführen der Tests mit `java` im Classpath vorhanden sein muss. Sie können sowohl bei `javac` als auch bei `java` den Classpath explizit setzen: `javac -cp path/to/junit-4.11.jar wordprocessor.SomeTest`. Falls Sie mehrere Pfade in den Classpath aufnehmen müssen, trennen Sie diese unter Linux mit ":" und unter Windows mit ";". Also so zum Beispiel: `javac -cp path/to/junit-4.11.jar:../src:. wordprocessor.SomeTest`. In diesem Beispiel liegt sowohl die JUnit-Bibliothek (mit `path/to/junit-4.11.jar`), der Quelltext Ihrer Lösung (mit `../src` als auch das aktuelle Verzeichnis (mit `.`) auf dem Classpath.

Eigene Tests

Wir können nicht alle Anwendungsfälle mit unseren Unit-Tests abdecken. Daher sind Sie immer angehalten eigene Tests zu schreiben. Das können Sie beispielsweise tun, indem Sie einer Ihrer Klassen eine `main()`-Methode hinzufügen, verschiedene Anwendungsszenarien simulieren und bestimmte Testausgaben kontrollieren. Des weiteren stellen wir Ihnen eine Klasse `WordProcessor` zur Verfügung, die ein Toplevel-Interface zur Verfügung stellt, mit dem Sie Ihre Klassen abschließend testen können.

HTML - Hypertext Markup Language

Da das Textverarbeitungsprogramm, das Sie hier schreiben, markierten Text als **fett** oder *kursiv* auszeichnen kann, haben wir uns dafür entschieden, diese Information in der Ausgabe mit den dafür üblichen HTML-Tags zu codieren. Sie sollen hier keine HTML-Dokumente erzeugen, aber fetter Text sollte von den Tags `<b>` und `</b>` geklammert sein und kursiver Text von `<i>` und `</i>`. Zum Beispiel ist in folgendem Text das Wort "Forschungsergebnisse" als fett darzustellen markiert, "Tim Berners-Lee" wäre kursiv darzustellen und "World Wide Web" fett und kursiv.

> Die Hypertext Markup Language (HTML) wurde Anfang der 90er Jahre des letzten Jahrhunderts entwickelt um Forschungsergebnisse digital auszutauschen. Sie entstand am europäischen Kernforschungszentrum CERN als Nebenprojekt von Tim Berners-Lee, der damit zusammen mit dem Netzwerkprotokoll HTTP und anderen Technologien, die Grundlage für das World Wide Web (WWW) legte.

Abgabemodalitäten

Sie erhalten zwei `zip`-Dateien, eine mit den zu bearbeitenden Quellen und eine mit den Tests. Geben Sie den Inhalt der ersteren nach dem Bearbeiten im Praktomaten ab. Geben Sie nicht die Tests ab.

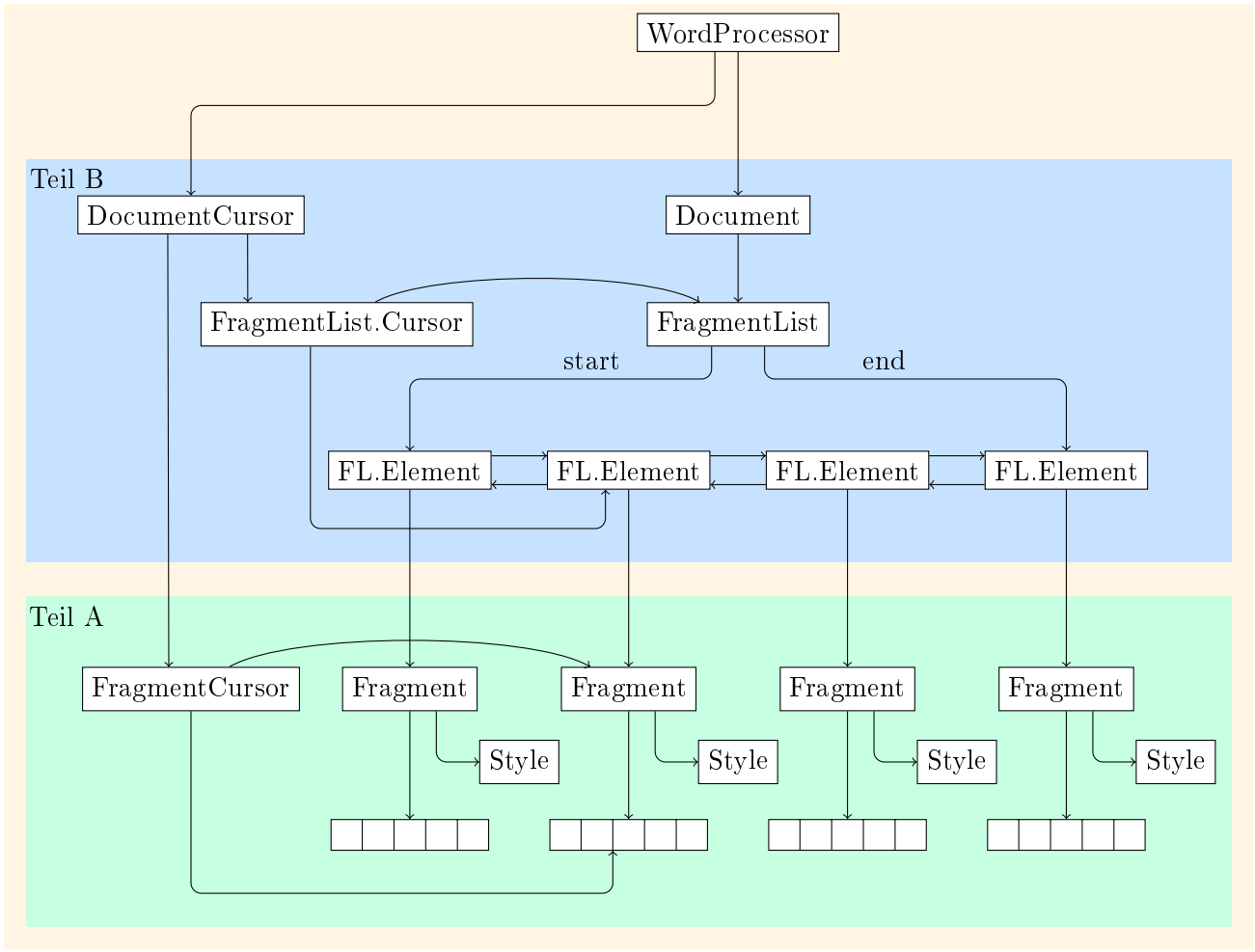

Abbildung 1: Übersicht über die Textverarbeitung

A Formatierte Textfragmente (10 Punkte)

In diesem Teil des Blatts implementieren Sie die Klassen `Style`, `Fragment` und `FragmentCursor`. Rahmenimplementierung, inklusive Klassenattribute, Methodensignaturen und JavaDocs, sind auf `http://baldur.iti.kit.edu/programmieren` verfügbar. Studieren Sie insbesondere die JavDoc Dokumentation sorgfältig, um sich einen Überblick über die Klassen zu verschaffen, da dieser beschreiben, wie sich die Methode verhalten sollen, die sie zu implementieren haben.

Jetzt können Sie beginnen, schrittweise den Kern Ihres eigenen Textverarbeitungsprogramms zu implementieren.

A.1 Formatierung

Die Klasse `Style`, die Textformatierungen modelliert, ist ein guter Anfangspunkt um für die folgenden Aufgaben warm zu werden. Implementieren Sie die Methoden der Klasse `Style` entsprechend der JavaDoc-Kommentare.

Beachten Sie bei der Implementierung der `equals`-Methode, dass diese alle Fälle abdeckt,

- in denen `this` und `other` auf das selbe Objekt zeigen,
- in denen `other` `null` ist,
- in denen sich die Typen von `this` und `other` unterscheiden und
- in denen `this` und `other` Instanzen der Klasse `Style` sind.

A.2 Textfragmente

Die Klasse `Fragment` benutzt ein Attribut des Typs `StringBuilder`, um den textlichen Inhalt des Fragments zu speichern. Bei der Implementierung der Klasse `Fragment` müssen Sie entsprechende Methoden von `StringBuilder` benutzen. Verschaffen Sie sich daher in der Java API Dokumentation einen Überblick über die Methoden dieser Klasse [4].

Implementieren Sie alle Methoden der Klasse so, dass ihr Verhalten dem der vorgegebenen JavaDoc-Kommentare entspricht.

A.3 Fragment-Cursor

Implementieren Sie alle Methoden der Klasse so, dass ihr Verhalten dem der vorgegebenen JavaDoc-Kommentare entspricht.

[4] `http://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html`

B Listen und Navigation (10 Punkte)

In diesem Teil des Blatts implementieren Sie die Klassen `FragmentList`, `DocumentCursor` und `Document`. Verschaffen Sie sich auch hier einen Überblick, indem Sie sich zunächst den Rumpf dieser Klassen von unserer Webseite herunterladen und den Code samt Dokumentation durchlesen. Danach können Sie darangehen die übrigen Datenstrukturen Ihres Textverarbeitungsprogramms zu implementieren.

B.1 Eine doppelt verkettete Liste: `FragmentList`

Die Klasse `FragmentList` mit ihren beiden inneren Klassen `FragmentList.Element` und `FragmentList.Cursor` modelliert eine doppelt verkettete Liste von Fragmenten. Implementieren Sie die Methoden dieser Klassen, und halten Sie sich dabei an die funktionale Beschreibung in den JavaDoc-Kommentaren.

B.2 Dokumenten-Cursor

In der Klasse `DocumentCursor` fließen jetzt die Entwicklungsstränge der vorherigen Aufgaben zusammen. `DocumentCursor` besitzt ein Attribut vom Typ `FragmentList.Cursor` zur Navigation in der `FragmentList`, während das Attribut vom Typ `FragmentCursor` für die Navigation innerhalb des aktuellen Fragments zuständig ist.

`DocumentCursor` bietet nach außen Zeichen-basierte Operationen an (z.B. einfügen, löschen, cursor links/rechts), wodurch Benutzern der Klasse der Umgang mit dem inneren Fragment-basierten Aufbau erspart bleiben soll. Nur Ihnen bleibt das nicht erspart. Sie müssen entscheiden wann Sie in das nächste Fragment springen und einen neuen Fragment-Cursor anlegen müssen und wann nicht. Auch das Erstellen neuer Fragmente wird Ihre Implementierung von `DocumentCursor` übernehmen müssen.

Implementieren Sie jetzt die Methoden der Klasse `DocumentCursor`. Halten Sie sich genau an unsere Vorgaben in der JavaDoc-Beschreibung. Sie gehören nicht nur zur Aufgabenstellung, sondern sind in diesem Fall vor allem auch eine Arbeitserleichterung.

B.3 Dokument

Implementieren Sie jetzt die Methoden der Klasse `Document` entsprechend der JavaDoc-Beschreibung.