Version 1 vom 4. Juni 2019 um 11:27 Uhr

URIs (15 Punkte)

In diesem Projekt werden Sie in Java Tests zu einer Spezifikation schreiben und diese anschließend implementieren.

1 Die Spezifikation

Die Spezifikation ist eine Abwandlung und Vereinfachung des Internetstandards RFC 3986.[1] Dieser legt eine Syntax für Bestandteile von URIs (Uniform Resource Identifier) fest, die alle verschiedenen URI-Schemata gemeinsam haben.
Beispiele für URIs sind zum einen Webadressen mit http/https-Schema, aber auch „mailto:a@b.com“, „tel:+49-681-302-0“ und viele weitere Schemata. Wir haben die Spezifikation der Syntax deutlich eingeschränkt und im wesentlichen bleiben nur URIs erlaubt, die ähnlich dem http-Schema aufgebaut sind.
Die Details der Spezifikation finden Sie in den JavaDoc-Kommentaren der Schnittstellen und Klassen im Paket `uri`: `Uri`, `IPv4Address`, `Host`, `UriParser` und `UriParserFactory`. Die akzeptierte Syntax der URIs ist insbesondere in `Uri` spezifiziert. Die Grammatik besteht aus Regeln, die jeweils ein Nicht-Terminal auf der linken Seite des Gleichheitszeichen definieren, dass die möglichen syntaktischen Formen der rechten Seite annehmen kann.
Eine ausführlichere Beschreibung dieser Form der Grammatikbeschreibung mit einzelnen Beispielen finden Sie in der Wikipedia.[2] Wir verwenden zusätzlich Wertebereiche mit Literalen in der Form `"a"-"f"` und `"0"-"9"`.
Wenn Ihnen Teile der Spezifikation unklar sind, stellen Sie bitte eine Frage im Forum.

2 Die Aufgaben

2.1 Testerstellung (7 Punkte) Abgabe: Di 11.06.2019 23:59 Uhr

Beachten Sie, dass zur Bewertung Ihrer Tests der eingebuchte Stand zum Abgabezeitpunkt (1 Woche vor Projektende) herangezogen wird!
In dieser Aufgabe müssen Sie JUnit-Tests implementieren. Diese sollen dazu dienen Implementierungen der Spezifikation zu überprüfen - insbesondere auch Ihre eigene. In diesem Sinne führen wir Ihre Tests mit sowohl fehlerhaften Implementierungen als auch einer fehlerfreien Implementierung der im Projekt vorhandenen Schnittstellen und Klassen aus. Für jede fehlerhafte Implementierung erwarten wir, dass einer Ihrer Tests auf dieser fehlschlägt, gleichzeitig aber auf der korrekten Implementierung erfolgreich verläuft.
Jede fehlerhafte Implementierung wurde von uns als JUnit-Test implementiert und über das Ergebnis dieser Tests können Sie sehen, ob Ihre Tests erfolgreich waren. Die fehlerhaften Implementierungen werden ihre Unzulänglichkeiten schon bei kurzen Eingabezeichenketten offenbaren und müssen auch nicht durch erschöpfende Suche aller möglicher Zeichen(ketten) getestet werden.
Denken Sie bei Ihren Tests aber an korrekte und ungültige Beispieleingaben an den Grenzen der Spezifikation und an mögliche Abweichungen und Fehlinterpretation der Grammatik in der Dokumentation der Klasse `Uri`. Beachten Sie, dass es zur Evaluation Ihres Projektes noch mehr fehlerhafte Implementierungen gibt. Schreiben Sie Ihre Tests deshalb gegen die Spezifikation und nicht nur gegen die daily Tests auf dem Server.
Bitte überprüfen Sie nicht, ob die Implementierungen riesigen Eingabegrößen standhalten.

Technische Hinweise
Alle Tests, die zur Bewertung dieser Aufgabe herangezogen werden sollen, müssen in dem Java-Paket `uri.tests` implementiert werden. Es ist unerheblich, ob Sie dazu mehrere Klassen benutzen, oder nicht. Sie können bereits kleine Beispieltests in der Klasse `uri.tests.SimpleTests` finden. Außerdem finden Sie dort zwei Hilfsfunktionen die Ihnen den Umgang mit `IPv4Address` Objekten erleichtern.

[1] `https://www.ietf.org/rfc/rfc3986.txt`
[2] `https://de.wikipedia.org/wiki/Angereicherte_Backus-Naur-Form`

Wenn Sie eigene Hilfsmethoden/-klassen für Ihre Tests implementieren möchten, müssen diese ebenfalls in dem `uri.tests` Paket implementiert sein.
Ihren Tests stehen außerdem nur die vorgegebenen Schnittstellen `Uri`, `IPv4Address`, `Host` und `UriParser` und deren Methoden, aus dem Paket `uri` zur Verfügung. Insbesondere stehen Ihnen nicht Ihre eigenen Klassen aus dem Paket `uri.implementation` und anderen eigenen Paketen zur Verfügung.
Des weiteren wird eine von uns gestellte Klasse `UriParserFactory` vorhanden sein. Mit Hilfe dieser können Sie Instanzen der von uns gestellten Implementierungen der Schnittstellen erzeugen.
Wenn Sie in der zweiten Teilaufgabe Ihre eigene `UriParserFactory` implementieren, können Sie Ihre Tests ganz einfach in Eclipse ausführen, um darüber Ihre eigenen Implementierungen von `UriParser` zu testen.
Außerdem dürfen Sie, da die Testerstellung bewertet wird, hierfür keine Tests von Kommilitonen benutzen.
Als Hilfestellung gibt es den Test `prog2.tests.daily.TestsCorrect.all` der alle Ihre Tests einmal mit der korrekten Implementierung ausführt und die fehlschlagenden Tests ausgibt. Sie können also, wenn Sie sich an einzelnen Stellen der Spezifikation unsicher sind, einfach einen entsprechenden Test schreiben und überprüfen ob Ihr Verständnis der Spezifikation von der Referenzimplementierung bestätigt wird. Nach dem Abgabezeitpunkt werden Ihre Tests weiter durch den daily Test geprüft, aber nicht mehr bewertet.

2.2 Parserimplementierung (8 Punkte) Abgabe: Di 18.06.2019 23:59 Uhr

Beachten Sie, dass zur Bewertung Ihrer Implementierung der eingebuchte Stand zum Projektende herangezogen wird!
In dieser Aufgabe müssen Sie die Spezifikation selbst korrekt implementieren. Zur Hilfestellung haben wir bereits ein Paket `uri.implementation` angelegt, dass für die Schnittstellen `Uri`, `Host` und `IPv4Address` bereits Klassen enthält, die Sie nur noch ausfüllen und in Ihrem Parser benutzen müssen.
Sie müssen in jedem Fall noch eine Implementierung von `UriParser` hinzufügen. Die Datei dieser Implementierung müssen Sie mit `git add` zur Versionsverwaltung hinzufügen, ansonsten wird die Kompilierung Ihres Projekts auf unserem Server fehlschlagen. Erzeugen Sie eine Instanz dieser Klasse in `UriParserFactory` und geben Sie diese dort zurück.
Da Sie die Tests selbst schreiben, wird es zu Ihrer Implementierung erst zu Ende des Projekts Tests zur Bewertung geben. Es bietet sich also an mit der Testerstellung zu beginnen um frühzeitig Fehler in Ihrem Parser zu erkennen.
Sie dürfen keine Teile der Java-Standardbibliothek, weder direkt noch indirekt, verwenden, die URIs/URLs parsen.

3 Java Projekte – Eclipse

Dieses und die verbleibenden Projekte werden von Ihnen in Java implementiert. Um Ihnen dies zu erleichtern dürfen Sie die Entwicklungsumgebung *Eclipse*[3][4] benutzen. Zusätzlich benötigen Sie das Java Development Kit (JDK), entweder von Oracle[5] oder die open-source Variante OpenJDK[6]. Auf unseren Servern (und in der VM) ist OpenJDK Version 11 installiert, es sollte aber keinen Unterschied machen wenn Sie stattdessen mit dem Oracle JDK arbeiten. Anhand der daily Tests können Sie vergleichen, ob sich Ihre Implementierung auf unserem Server genauso verhält wie bei Ihnen. In der VM ist Eclipse bereits vorinstalliert, wir empfehlen dennoch Eclipse außerhalb der VM zu benutzen da es in der VM zu Performanceproblemen kommen kann.
Starten Sie dazu zunächst Eclipse. Wenn Sie nach einem „Workspace“ gefragt werden übernehmen Sie den Standardpfad (z.B. `/home/prog2/workspace/`) und aktivieren Sie die „Use this as the default ...“ Box bevor Sie mit `OK` bestätigen.
Um das Projekt in Eclipse bearbeiten zu können, müssen Sie erst das Depot auschecken und das Projekt importieren.

1. Klonen Sie das Projekt in einen beliebigen Ordner:
   `git clone https://prog2scm.cdl.uni-saarland.de/git/project4/$NAME /home/prog2/project4`
   wobei Sie $NAME durch ihren CMS-Benutzernamen ersetzen müssen.
2. Benutzen Sie *Import*, ein Unterpunkt des *File* Menüeintrags in Eclipse, um den Importierdialog zu öffnen.
3. Wählen Sie „Existing project into workspace“ aus und benutzen Sie den neuen Dialog um den Ordner des Projekts (siehe Punkt 1) auszuwählen.

[3] `http://de.wikipedia.org/wiki/Eclipse_(IDE)`
[4] `https://github.com/pellaton/eclipse-cheatsheet`
[5] `https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html`
[6] `https://jdk.java.net/11/`