URIs (15 Points)

In this project you will write tests for a specification in Java and afterwards implement the specification.

1 The Specification

The specification is an adaption and simplification of the Internet standard RFC 3986.[1] This standard defines the syntax for those parts of URIs (Uniform Resource Identifier) that are common with all URI schemes.
Examples for URIs are web addresses with the http/https scheme, but also "mailto:a@b.com", "tel:+49-681-302-0" and many more schemes. We considerably reduced the syntax of the specification so that essentially only URIs similar to the http scheme are still allowed.
The details of the specification can be found in the JavaDoc comments of the interfaces and classes in the package `uri`: `Uri`, `IPv4Address`, `Host`, `UriParser` and `UriParserFactory`. The to be accepted syntax of the URIs in particular is specified in `Uri`. The grammar that is used there consists of rules that each define one non-terminal on the left of the equals sign. This non-terminal can take the possible syntactical forms on the right.
A more comprehensive description of this grammar representation with some examples can be found on Wikipedia.[2]
We additionally use value ranges for literals in the form of `"a"-"f"` and `"0"-"9"`.
If parts of the specification are not clear to you, please do ask in the forum.

2 Assignments

2.1 Write Tests (7 Points) Submission: 14.06.2022 23:59h

Note that the committed state at the time of the respective submission deadline (1 week before the end of the project) is used for the evaluation of your tests.
For this assignment you have to implement JUnit tests. These tests are used to evaluate implementations of the specification – in particular your own. Therefore, we will run your tests using erroneous implementations as well as a correct implementation of the given interfaces and classes. For the erroneous ones, we expect that at least one of your tests fails, while it passes on the correct implementation.
Each erroneous implementation is encapsulated in a distinct JUnit test, which is run as part of the daily tests. By inspecting the results of these tests, you can see whether your tests were successful. The erroneous implementations will expose their deficiencies with short inputs already and also do not have to be tested with an exhaustive search of all possible input strings.
However, remember to use correct and invalid example inputs that test edge cases of the specification and consider possible discrepancies and misinterpretations of the grammar in the documentation for the class `Uri`. Note that there will be more erroneous implementations that will be used for evaluating your tests. Therefore, write your tests against the specification and not exclusively against the daily tests.
Please to not test whether the implementations withstand enormous inputs.

Technical Remarks
All tests that shall be considered for the evaluation of this assignment have to be implemented in the Java package `uri.tests`. It is irrelevant whether you do so in a single or multiple classes. You can find some small exemplary tests in the class `uri.tests.SimpleTests`. Additionally, you will find two helper functions that will help you when working with `IPv4Address` objects.
If you want to implement further helper functions or classes for your tests, you also have to implement those in the `uri.tests` package.

[1] `https://www.ietf.org/rfc/rfc3986.txt`
[2] `https://en.wikipedia.org/wiki/Augmented_Backus%E2%80%93Naur_form`

You must only use the provided interfaces `Uri`, `IPv4Address`, `Host` and `UriParser`, from the package `uri`. Especially your own classes from the package `uri.implementation` and other custom packages cannot be used. Furthermore, an additional class `UriParserFactory` is given by us. You can use this to create instances of our given implementations of the interfaces.
If you implement a custom `UriParserFactory` in the second assignment, you can execute your tests from within Integrated Development Environment. This then tests your own implementation of `UriParser`.
Moreover, you must not use tests provided by your fellow students as the tests are subject to grading.
To support you, there is the test `prog2.tests.daily.TestsCorrect.all` that runs all of your tests on the correct implementation once and outputs the failed tests. Therefore, if some points of the specification are not entirely clear to you, you can write a relevant test that checks whether your interpretation of the specification is in accordance with the reference implementation. After the submission deadline, your tests will continue to be tested by the daily tests, but they will no longer be graded.

2.2 Parser Implementation (8 Points) Submission: 21.06.2022 23:59h

Note that the committed state at the time of the project deadline is used for the evaluation of your implementation!
For this assignment you have to correctly implement the specification yourself. To kick-start you, we already set up the package `uri.implementation` that already contains concrete classes for the `Uri`, `Host` and `IPv4Address` interfaces. These just have to be filled with the implementation and can be used in your parser. Additionally, you find the `UriParserImplementation` class, in which you can implement your parser. Create an instance of this class in `UriParserFactory` and return it.
As you write the tests yourself in the first assignment, we only use additional tests for your implementation for the evaluation at the end. Therefore, it is a good strategy to start by writing the tests, so that you are made aware of issues with your parser early on.
You *must not* use parts of the Java standard library that parse URIs/URLs, neither direct nor indirect. In particular, using the `java.net` library is not allowed for this project.

3 Java Projects – Visual Studio Code

Starting with this project, you will implement the following projects in Java. We recommend that you use Visual Studio Code to work on the projects. Suitable extensions for this editor are already installed in the VM and corresponding configuration files are provided with the projects.
If you do not use the VM, you will need the Java Development Kit (JDK), either the Oracle[3] or the OpenJDK[4] variant. Our servers (and the VM) have OpenJDK version 17 installed. In theory, there should not be a difference if you use the Oracle JDK instead. Utilizing the daily tests, you can ensure that your implementation behaves on our servers as it does for you locally.

3.1 Compiling in Visual Studio Code

Visual Studio Code automatically compiles the projects when you run tests or start with a main method. In contrast to the C projects, you therefore do not have to execute a dedicated build command.

3.2 Testing and Debugging in Visual Studio Code

Running and debugging individual tests is done easiest with the test panel. This panel contains a list of all tests in the project, grouped by the java package and class in which they reside. After a test is executed, an editor window with a result summary opens. This summary shows which tests were passed and it provides additional information on failed tests. Every new test that you create is listed in the test panel.

3.3 Cloning the Repository

To be able to edit the project in Visual Studio Code, you first have to checkout the repository and import the project:

[3] `https://www.oracle.com/java/technologies/downloads/`
[4] `https://jdk.java.net/17/`

1. Clone the project in any folder:
   `git clone https://prog2scm.cdl.uni-saarland.de/git/project4/$NAME /home/prog2/project4`
   where $NAME has to be replaced with your CMS username.

2. Open the cloned directory in Visual Studio Code and confirm that the contents are trusted.