Projektdokumentation

Projekt 04

Entwicklung eines Netzwerkadressenrechners

Abgabetermin: Montag 12. Juni 2017

Gruppenmitglieder:
Jan-Luca Gutsch, Sergej Frank, Valentin Beller,
Marcel Schmidt, Jasmin Teller, Joshua Senkpiel

Inhaltsverzeichnis

1 Einleitung

Im Folgenden wird der Projektrahmen und das Projekt vorgestellt.

1.1 Projektumfeld

Das Projekt ist im Rahmen des Schulunterrichts an der Staatlichen Gewerbeschule ITECH (BS14) entstanden. Im regelmäßigen Projektphasen sollen neu erlernte Kompetenzen angewendet werden.

1.2 Projektziel

Das vorgegebene Ziel des Projektes war ein Netzwerkadressenrechner, der als Rich-Client Anwendung entwickelt werden sollte. Er soll einer fiktiven IT-Firma dazu dienen, ihre Netzwerke effizient und verlässlich zu planen. Als Grundlage dient ein Lastenheft, welches im Moodle Kurs "Projekt 04" hinterlegt ist.

1.3 Projektabgrenzung

Dem Lastenheft zufolge soll keine Datenbankanbindung integriert werden.

2 Projektplanung

Im folgenden zeigen wir auf, wie wir die Durchführung des Projekts geplant haben.

2.1 Projektphasen

Die folgenden Projektphasen wurden von uns in erster Hand festgelegt.

2.1.1 Planungsphase

Die Planungsphase soll uns einen Überblick über das Projekt, die Ziele und unsere Durchführung schaffen. Dabei setzen wir zum Großteil auf Brainstormingrunden, die unsere Planung am effektivsten unterstützen. In dieser Phasen legen wir die in 2.2 aufgezeigte Methodik und die in 2.3 dargestellten Resourcen fest.

2.1.2 Entwurfsphase

In der Entwurfsphase planen wir alle Komponenten der Software. Dabei versuchen wir unter anderem folgende Fragen zu beantworten:

Soll die Software modularisiert werden?
Wenn ja, wie soll die Modularisierung erfolgen?
Auf welcher Basis soll die Netzwerklogik entstehen?
Welche Codekonventionen wollen wir als Grundlage nehmen?
Sollen Tests eingesetzt werden?
Wenn ja, welche Tests und wie ausgiebig?
Wie soll unsere Klassenstruktur aussehen?

Aus dieser Phase entsteht ein Klassendiagramm, welches einen ersten Überblick über unsere Implementierung darstellt. Die Antworten auf diese Fragen bilden die Grundlage für die nächste Phase; die Implementierung.

2.1.3 Implementierungsphase

In der Implementierungsphase beginnen wir damit die Logik und Benutzeroberfläche zu entwickeln. Auf Grundlage der in 2.1.2 festgelegten Klassenstruktur und Modularisierung können wir in einem agilen Entwicklungsprozess (siehe 2.2) beginnen die einzelnen Funktionen und Komponenten zu implementieren.

2.2 Entwicklungsprozess

Wir haben uns für einen Agilen Entwicklungsprozess entschieden, um auf spontane Ideen und Anregungen besser eingehen zu können. Außerdem ermöglicht die Agile Entwicklung mehr Freiraum bei der Entwicklung, was sich positiv auf das Arbeitsklima, sowie

unsere persönlichen Lernerfolge auswirkt. Die agile Softwareentwicklung schützt uns des weiteren auch vor Änderungen im Projektauftrag, die leider häufiger auftreten, da wir diese in der Regel besser umsetzen können.

2.3 Resourcenplanung

Als Entwicklungsumgebung haben wir uns für IntelliJ des Herstellers JetBrains entschieden. Als Versionskontrolle setzen wir auf Git mithilfe eines GitHub Repositories.

3 Projektentwurf

Dieser Abschnitt dokumentiert die Entwurfsphase des Projektes.

3.1 Komponentenentwurf

Zunächst entwarfen wir einen Plan bezüglich der einzelnen Komponenten der Anwendung. Dabei handelt es sich um die Netzwerklogik, sowie die Frontendlogik bzw. das GUI. Wir definierten für beide Komponenten jeweils ein Java Package um diese logisch zu trennen, was sich als unentbehrlich herausgestellt hat um eine klare Übersicht über die Programmstruktur zu behalten und um die Verteilung der Aufgaben sinnvoll innerhalb der Gruppe gliedern zu können.

3.2 Entwurf der Netzwerklogik

Zunächst definierten wir eine Klasse um den elementaren Baustein der Anwendungslogik zu speichern; Die Klasse IPAddress stellt alle Daten einer IPv4 Adresse zur Verfügung, wobei sie keine Logik implementiert. Desweiteren entwarfen wir die Klassen Network und Subnet. Als letztes entwarfen wir die Klasse Host um Endpunkte in den Netzwerken darstellen zu können. Die drei Klassen sollten alle die Klasse IPAddress nutzen um verschiedenste Informationen zu speichern z.B. Netzwerk ID order Subnetmaske. Als Sammlung verschiedenster Funktionen definierten wir die Klasse NetUtils. Dadurch erhalten wir eine bessere Übersicht innerhalb der Klassen, und können diese Funktionen gleichzeitig besser allen Klassen zur Verfügung stellen.

3.3 Entwurf der GUI

Der erste Entwurf der GUI sah ein Tab-Layout vor, wie es in den Beispielen zu sehen ist, die dem Projektauftrag beiliegen. Dabei sollte sich der Workflow nacheinander durch eine Übersicht der Netzwerke, über eine Übersicht der jeweiligen Subnetze, bis zu einer der dazugehörigen Hosts bewegen. Nach einer ersten Einarbeitung in Swings TreeNodes und SplitPanes wurde entschieden diese Aufteilung der GUI zu verwerfen, und stattdessen eine Komplettübersicht über alle Netzwerke, Subnetze und Hosts in einem Fenster mit Baumstruktur und SplitPanes anzuzeigen.

4 Implentierung

Die größte Phase unseres Projektes bestand darin, die nun geplante Anwendung zu implementieren. Die folgenden Schritte dokumentieren diesen Vorgang. Die einzelnen Vorgänge sind chronologisch geordnet.

4.1 Logik I - Implementierung

Als erstes implementierten wir die Logik der IPv4 Adresse. Dabei stellten sich uns viele Probleme. So hatten wir größere Probleme mit der Sprache Java, da diese im Umgang mit Zahlen und binären Operationen mehrere Schwächen aufweist. So fehlen zum Beispiel die unsigned Datentypen, wodurch viele Operationen zunächst sehr provisorisch implementiert werden mussten.

Aufbauend auf der IPv4 Adresse wurden die Klassen Network und Subnet implementiert. Während der Implementierung dieser, entschieden wir uns dafür beide von einer gemeinsamen Basisklasse NetworkBase erben zu lassen um viele gemeinsame Eigenschaften und Methoden nicht doppelt implementieren zu müssen. Dieser Schritt ersparte uns auch in vielen weiteren Bereichen Arbeit, da es für viele Operationen irrelevant ist, ob es sich um ein Netzwerk, oder ein Subnetz handelt.

4.2 GUI I - Implementierung

Durch unsere Trennung der Logik und GUI wurde es uns ermöglicht beides getrennt von einander zu entwickeln. So konnte ein Großteil der GUI ohne Funktion erstellt und erst nachträglich mit der Logik verbunden werden.

4.3 Tests

Um unsere Implementierung der Methoden zum Parsen von IP Adressen und Netzwerken zu testen entwickelten wir erste Tests. Diese halfen uns entscheidend bei der Suche nach Fehlern, sowie bei der Pflege des Codes. Um eine möglichst stabile Software zu entwickeln, erstellen wir weitere Tests für die meisten Funktionen im Netzwerk Logik Bereich.

4.4 GUI II - Neuentwurf

Bei der Implementierung der Logik kamen weitere Ideen zur Umsetzungen auf und nach einiger Recherche legten wir ein neues Konzept zugrunde. In diesem sollten alle Netzwerke und Subnetze ein einer Baumstruktur abgebildet werden. Dies führte zu einer deutlich aufgeräumteren und verständlicheren Oberfläche. Die Oberfläche wurde schnell umgestaltet und nach anfänglichen Problemen mit der Datenbindung funktionierten die grundlegeneden Funktionen schon nach kurzer Zeit und ohne großen Aufwand.

4.5 Logik II - Zusammenführung von Network und Subnet

Durch die Entwicklung einer Baumstruktur stellen wir fest, dass eine rekursiv aufgebaute Logik der Netzwerke dafür deutlich besser geeigent war als die aktuelle Trennung in Netzwerke und Subnetzen. Außerdem ist eine solche rekursive Implementierung deutlich eleganter und ebenso realitätsnäher als die vorhandene. Die Zusammenführung gestaltete sich durch unsere bereits vorhandene NetzwerkBase Klasse einfach.

4.6 Logik III - Implementierung von IPv6

Teil der Aufgabenstellung war es, allen Netzwerken und Hosts IPv6 Adressen zuweisen zu können. Die Entwicklung einer IPv6Address Klasse gestaltete sich zunächst als schwierig, da wir nur wenig Erfahrung im Umgang mit IPv6 hatten und viele Probleme zunächst recherchiert werden mussten.

Im Laufe der Implementierung von IPv6 benannten wir die schon bestehende IPv4 Adress-Klasse in IPv4Address, da nun beide Adress-Klassen von einer gemeinsamen Basisklasse IPAddress erben sollten. Die Network Klasse versahen wir ebenso mit den IPv6 Eigenschaften, wie die Host Klasse.

4.7 Fazit

Das in 2.2 festgelegte agile Vorgehen haben wir in vielen Situationen genutzt. Die Wahl hat sich dabei defintiv ausgezahlt. Da wir von Anfang an einen sehr modularen und abstrakten Ansatz verfolgt haben, mussten wir selbst bei größeren Änderungen nur wenig Probleme lösen. So ließ sich beispielsweise die Zusammenführung von Network und Subnet ohne Änderungen der Logik in der GUI bewerkstelligen. Die Änderungen beschränkten sich auf das Umbennen der Klassennamen.

5 Dokumentation

5.1 Java Dokumentation

Zur Dokumenation des Java Sourcecodes nutzen wir die Java Doc Funktion. Allen Klassen, Feldern, Eigenschaften und Methoden sind spezielle Kommentare vorgestellt, die die Funktion, Rückgabewerte und alle Parameter beschreiben. Dadurch stellen wir sicher, dass der Quellcode für alle verständlich und nachvollziehbar bleibt. Außerdem erleichtert dies die allgemeine Wartung bzw. Ergänzungen des Codes.

Die Nachverfolgbarkeit der Änderungen und Zwischenstände ist durch die Versionskontrolle (Git) garantiert.

5.2 Projektdokumentation

Die Projektdokumentation wurde in LaTeX geschrieben und ebenfalls mit Git versioniert.

5.3 Sourcecode

Die Java Quelldateien, sowie die Quelldateien dieser Dokumentation stehen öffentlich unter MIT Lizenz auf GitHub zur Verfügung. `https://github.com/SergejFrank/IT5LSubnetting`