Prof. Dr. Ralf Hinze
Markus Heinrich, M.Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Programmiersprachen

Projekt 2: Programmierpraktikum 2021

Ausgabe: 29. Juni 2021
Abgabe: 20. Juli 2021, 15 Uhr

Tutor*in für das Programmierpraktikum 2022

Ihnen hat das Programmierpraktikum Spaß gemacht, Sie programmieren gerne in Java und Sie würden Ihr Wissen gerne an andere Studierende weitergeben? Dann bewerben Sie sich gerne nach Ihrer letzten Projektabnahme als Tutor*in für das Programmierpraktikum für das Sommersemester 2022. Schreiben Sie eine kurze E-Mail an markus.heinrich@cs.uni-kl.de.

GitLab Team Repositories

Wir verwenden auch für die Projekte wieder die GitLab Repositories. Führen Sie den Befehl `git pull` aus, damit die Vorlagen heruntergeladen werden. Sie sollten nun einen neuen Ordner `P2` sehen, den Sie wie gewohnt in IntelliJ IDEA öffnen können. Es handelt sich wieder um ein Gradle Projekt, sodass Abhängigkeiten automatisch heruntergeladen werden.

Ihre Lösungen müssen Sie in das lokale Repository committen (`git commit`) und dann auf den GitLab Server übertragen (`git push`). Sprechen Sie sich zur Bearbeitung der Aufgaben mit Ihren Abgabepartner*innen ab, damit Sie unnötige Konflikte vermeiden.

Bearbeitungshinweise

Sie haben für die Bearbeitung des Projekts drei Wochen Zeit, d. h. es ist entsprechend umfangreich. Beginnen Sie also rechtzeitig mit der Bearbeitung und nutzen Sie die Sprechstunden!

Wir erwarten, dass folgende Meilensteine erreicht werden (d. h. Code befindet sich auf GitLab):

1. Bis Dienstag, 06.07.2021 23:59 Uhr, müssen mindestens die Testfälle von Aufgabe 1 in sinnvollem Umfang erstellt sein.
2. Bis Dienstag, 13.07.2021 23:59 Uhr, soll Aufgabe 1 weitestgehend gelöst sein und von Aufgabe 2 sollte mindestens ein Grundgerüst erkennbar sein.

Jacoco

Mit Hilfe des Jacoco Plugins werden im Rahmen des Gradle `test` Tasks Coverage Reports im HTML-Format erstellt. Diese finden Sie nach dem Ausführen des `test` Tasks innerhalb Ihres Projektordners im Unterverzeichnis `build/reports/jacoco/test/html/index.html`. Außerdem werden die Reports auch von GitLab mit Hilfe einer Pipeline automatisch bei jedem Push erzeugt (sofern Ihr Code kompiliert) und als Artefakt archiviert (die Konfiguration dieser Pipeline erfolgt mittels der in der Projektvorlage mitgelieferten Datei `.gitlab-ci.yml`). Sie können die Reports z. B. unter dem Menüpunkt „CI/CD“ → „Jobs“ → Download Icon ganz rechts des entsprechenden Jobs („Download artifacts“) herunterladen.

Einleitung

In dieser Aufgabe erweitern wir die Graph-Implementierung und den Paketmanager vom letzten Projekt um weitere Funktionalität. Die Aufgabenstellung verzichtet an vielen Stellen bewusst auf konkrete Implementierungsvorgaben. Dadurch haben Sie größere Freiheiten wie Sie die Implementierung angehen möchten und Sie können selbstständig Designentscheidungen treffen oder verschiedene Ansätze ausprobieren.

Kopieren Sie zunächst Ihre Implementierung von Projekt 1 in die entsprechenden Pakete unter `de.tukl.programmierpraktikum2021.p1`. Legen Sie anschließend die Implementierungen für Projekt 2 im Paket `de.tukl.programmierpraktikum2021.p2` ab. In den Klassen von Projekt 1 müssen Sie ggf. Zugriffsmodifier einiger Attribute von **`private`** auf **`protected`** ändern und evtl. sogar weitere Anpassungen und Restrukturierungen vornehmen. Funktionalität, die im Rahmen dieses Projekts hinzukommt, sollte möglichst nur in den Klassen von Paket `de.tukl.programmierpraktikum2021.p2` implementiert werden.

Hinweis: Projekt 2 baut auf eine funktionierende Implementierung von Projekt 1 auf. Sofern Teile Ihrer Lösung für Projekt 1 noch Mängel aufweisen, bessern Sie diese unbedingt im Rahmen von Meilenstein 1 nach. Sie können in den Sprechstunden bei Bedarf selbstverständlich auch noch Fragen zum ersten Projekt stellen.

Aufgabe 1 Gerichtete Graphen

Die Schnittstelle `GraphExtended` erweitert die Schnittstelle `Graph` aus dem letzten Projekt um weitere Methoden (siehe auch Javadoc). Schreiben Sie eine Klasse `GraphExtendedImpl`, welche die Klasse `GraphImpl` aus Projekt 1 erweitert und die Schnittstelle `GraphExtended` implementiert:

```
public class GraphExtendedImpl<D> extends GraphImpl<D> implements GraphExtended<D>
```

Wie bereits im letzten Projekt, dürfen Sie wieder die Java Standardbibliothek verwenden, jedoch keine Bibliothek zur Repräsentation von Graphen.

Hinweis: Recherchieren Sie in der Literatur oder im Internet wie das algorithmische Vorgehen in den folgenden Teilaufgaben aussehen könnte.

Legen Sie Ihre Lösungen der folgenden Aufgaben im Paket `de.tukl.programmierpraktikum2021.p2.a1` ab.

a) Die Methode `isConnected` prüft, ob der Graph zusammenhängend ist. Wir bezeichnen unseren gerichteten Graphen als (schwach) zusammenhängend, wenn der ungerichtete Graph, der dadurch entsteht, dass alle gerichteten Kanten durch ungerichtete Kanten ersetzt werden, zusammenhängend ist. Ein ungerichteter Graph ist zusammenhängend, wenn es zwischen je zwei beliebigen Knoten einen Weg gibt. Entsprechende Beispiele finden Sie auf Seite 4.

b) Die Methode `isCyclic` prüft, ob der Graph zyklisch ist. Wir bezeichnen unseren gerichteten Graphen als zyklisch, wenn er mindestens einen Zykel enthält, wenn es also einen nichtleeren gerichteten Pfad gibt, dessen Anfangs- und Endknoten identisch sind. Entsprechende Beispiele finden Sie auf Seite 4.

   Zusatzinformation: Ein Baum ist ein zyklenfreier zusammenhängender Graph.

c) Die Methode `breadthFirstSearch` führt eine Breitensuche (*engl. breadth-first search*) durch, um einen bezüglich der Anzahl an Zwischenknoten kürzesten gerichteten Pfad zwischen zwei Knoten zu finden. Gibt es keinen solchen Pfad, ist die Ergebnisliste leer, ansonsten enthält diese die Ids aller Knoten, die durchlaufen werden, um vom Startknoten zum Endknoten zu gelangen.

d) Erstellen Sie JUnit Tests für Ihre `GraphExtended` Implementierung. Stellen Sie dabei sicher, dass Sie eine Instruction Coverage von mindestens 95% und eine Branch Coverage von mindestens 90% erreichen.

Aufgabe 2 Paketmanager

In dieser Aufgabe werden wir den Paketmanager Pacman aus Projekt 1 um weitere Funktionen erweitern. Implementieren Sie dazu die Schnittstelle `de.tukl.programmierpraktikum2021.p2.a2.PacmanExtended`. Lesen Sie sich den restlichen Aufgabentext sowie die Javadoc Kommentare durch, um die Schnittstelle zu verstehen. Implementieren Sie dann entsprechend die Klasse `class PacmanExtendedImpl extends PacmanImpl implements PacmanExtended`.

Es sollen nun auch Konflikte zwischen Paketen bei der Installation beachtet werden. Verwenden Sie dazu die Methoden `Set<String> getAllConflictingPackages()` und `List<String> getConflicts(String pkgs)` aus der `Util` Klasse. Die Methoden liefern symmetrische Ergebnisse: Wenn ein Paket A einen Konflikt mit Paket B hat, liefert `getConflicts` das jeweils andere Paket.[1] In unserer kleinen core-Paketdatenbank stehen lediglich die Pakete iptables und iptbles-nft sowie systemd-resolvconf und openresolv in Konflikt miteinander.

Mit Konflikten sind die Einträge in den Paketdatenbankdateien im Abschnitt `%CONFLICTS%` gemeint. Es handelt sich nicht um Versionskonflikte, sondern Pacman verhindert damit, dass Pakete auf einem System installiert werden, die nicht sinnvoll zusammen installiert werden können, z. B. weil diese eine ausführbare Datei mit demselben Namen bereitstellen, die aber unterschiedlich implementiert ist.

Der Konstruktor der `Util` Klasse erwartet nun auch einen Pfad zur Paketdatenbank. Für die Aufgabe werden zwei verschiedene Datenbankdateien unter dem Pfad `./src/main/resources/` bereitgestellt: `core.db.zip` enthält keine `core-cycle.db.zip` enthält zyklische Abhängigkeiten. Achten Sie darauf in Ihrer Implementierung einen relativen Pfad anzugeben, also z.B. `new Util("./src/main/resources/core.db.zip")`, damit Ihr Code auch bei Ihren Teampartner*innen und in der GitLab Pipeline lauffähig ist.

Weiterhin soll Pacman unterscheiden, welche Pakete explizit und implizit installiert wurden. Ein Paket `pkg` ist explizit installiert, wenn explizit `install(pkg)` aufgerufen wurde. Wenn es jedoch nur als Abhängigkeit eines anderen Pakets installiert wurde, ist es implizit installiert.

Legen Sie Ihre Lösungen der folgenden Aufgaben im Paket `de.tukl.programmierpraktikum2021.p2.a2` ab.

a) Erweitern Sie die Methode `install` dahingehend, dass Pakete, die mit bereits installierten Paketen in Konflikt stehen, nicht installiert werden können. Geben Sie in diesem Fall eine entsprechende Fehlermeldung auf der Konsole aus und brechen Sie die Installation ab.

   Falls ein Paket installiert werden soll, das eine zyklische Abhängigkeit hat, soll eine Warnung ausgegeben werden, aus der hervorgeht welche Pakete Teil des Zyklus sind. Die Installation soll jedoch trotzdem durchgeführt werden. Brechen Sie den Zyklus dazu an einer geeigneten Stelle auf, um eine Endlosschleife zu verhindern und installieren Sie die Pakete.

   Testen Sie Ihre Implementierung mit der oben genannten Paketdatenbank, die eine zyklische Abhängigkeit enthält. Welche Pakete sind daran beteiligt?

b) Die Methode `remove(String pkg)` entfernt das Paket `pkg`.

   Wenn das zu löschende Paket noch von einem anderen installierten Paket als Abhängigkeit benötigt wird, kann es nicht entfernt werden. Geben Sie eine entsprechende Fehlermeldung auf der Konsole aus.

   Sofern das Paket gelöscht werden kann, sollen auch die implizit installierten Abhängigkeiten des Pakets gelöscht werden. Allerdings nur, wenn diese ihrerseits nicht von anderen installierten Paketen benötigt werden.

c) Erweitern Sie die Methode `String transitiveDependencies(String pkg)` dahingehend, dass bei Paketen mit zyklischen Abhängigkeiten keine Endlosschleife auftritt. Brechen Sie den Zyklus dazu an einer geeigneten Stelle ab.

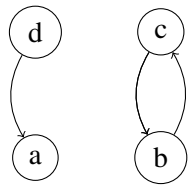
d) Erstellen Sie JUnit Tests für Ihre `PacmanExtended` Implementierung. Stellen Sie dabei sicher, dass Sie eine Instruction Coverage von mindestens 95% und eine Branch Coverage von mindestens 90% erreichen.

[1] Die Symmetrie ist in der original Paketdatenbank meist nicht explizit hinterlegt. Sie wird jedoch durch die Util-Klasse hergestellt.

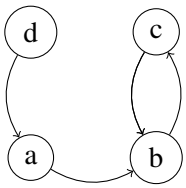
Beispiele

(nicht) zusammenhängende Graphen

nicht zusammenhängend



zusammenhängend

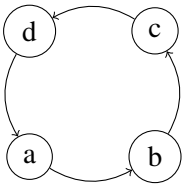


(nicht) zyklische Graphen

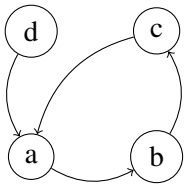
zyklisch

zyklisch

zyklisch



zyklisch

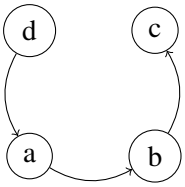


---

nicht zyklisch

nicht zyklisch

nicht zyklisch



nicht zyklisch