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

TU Kaiserslautern
Fachbereich Informatik
AG Programmiersprachen

Projekt 1: Programmierpraktikum 2021

Ausgabe: 08. Juni 2021
Abgabe: 29. Juni 2021, 15 Uhr

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 `P1` 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.

Vorlesungsumfrage

Wir sind an Ihrem Feedback interessiert. Bitte nehmen Sie sich die Zeit, die anonyme Vorlesungsumfrage auszufüllen. Studierende mit `@cs.uni-kl.de` Mailadresse haben dazu bereits eine E-Mail erhalten, alle anderen können sich auf https://vlu.cs.uni-kl.de/teilnahme/ z. B. mit ihrer RHRK Mailadresse einen Link anfordern. Ihr Feedback hilft uns, die Vorlesung und die Übungen weiter zu verbessern.

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, 15.06.2021 23:59 Uhr, müssen mindestens die Testfälle von Aufgabe 1 in sinnvollem Umfang erstellt sein.
2. Bis Dienstag, 22.06.2021 23:59 Uhr, soll Aufgabe 1 weitestgehend gelöst sein und von Aufgabe 2 sollte mindestens ein Grundgerüst erkennbar sein.

Einleitung

Ziel dieses Projekts ist die Erstellung eines einfachen Systems zur Verwaltung von Softwarepaketen. Das zu erstellende System ist angelehnt an den Paketmanager Pacman[1] der Linux-Distribution Arch[2]. Die Grundlage für den Paketmanager stellen Package Repositories im Internet dar, in denen zentral Datenbanken mit verfügbaren Softwarepaketen und die Pakete selbst abgelegt sind.

In Arch Linux sind die Softwarepakete im Wesentlichen in drei Repositories unterteilt[3]: Im `core` Repository befinden sich relativ wenige, sehr wichtige Pakete, die dafür benötigt werden, um das System zu booten, sich mit dem Internet zu verbinden, Pakete zu bauen, Dateisysteme zu verwalten und die dabei helfen das System zu installieren. Im Repository `extra` befinden sich alle offiziellen Pakete, die nicht die genannten Kriterien erfüllen, um in `core` enthalten zu sein. Das `community` Repository enthält weitere Pakete, die von sogenannten „trusted users“ stammen.

Um ein Paket auf einem Rechner zu installieren, verwendet man den Befehl `pacman -S paketname`. Im Folgenden ist eine Beispielausgabe für die Installation des Pakets `sqlite` dargestellt:

```
# pacman -S sqlite
resolving dependencies...
looking for conflicting packages...

Packages (1) sqlite-3.35.5-1

Total Download Size:   1.71 MiB
Total Installed Size:  9.48 MiB
Net Upgrade Size:      0.24 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
 sqlite-3.35.5-1-x86_64                          1748.4 KiB  6.42 MiB/s 00:00
    [############################################] 100%
(1/1) checking keys in keyring
    [############################################] 100%
(1/1) checking package integrity
    [############################################] 100%
(1/1) loading package files
    [############################################] 100%
(1/1) checking for file conflicts
    [############################################] 100%
:: Processing package changes...
(1/1) upgrading sqlite
    [############################################] 100%
:: Running post-transaction hooks...
(1/1) Arming ConditionNeedsUpdate...
```

Pacman löst zunächst die Abhängigkeiten des zu installierenden Pakets auf. Eventuell benötigte Abhängigkeiten sind in diesem Beispiel bereits auf dem System installiert, es muss also lediglich noch das Paket `sqlite` heruntergeladen und installiert werden.

[1] https://wiki.archlinux.de/title/Pacman
[2] https://de.wikipedia.org/wiki/Arch_Linux
[3] https://wiki.archlinux.org/title/Official_repositories

Ein Paket kann von vielen anderen Paketen abhängen. Falls alle oder einige der benötigten Pakte nicht vorhanden sind, müssen diese zunächst installiert werden.

```
# pacman -S brtfs-progs
resolving dependencies...
looking for conflicting packages...

Packages (2) lzo-2.10-3  btrfs-progs-5.12.1-1

Total Download Size:   0.94 MiB
Total Installed Size:  5.49 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
 lzo-2.10-3-x86_64                                  82.5 KiB  4.24 MiB/s 00:00
    [############################################] 100%
 btrfs-progs-5.12.1-1-x86_64                       882.2 KiB  6.95 MiB/s 00:00
    [############################################] 100%
(2/2) checking keys in keyring
    [############################################] 100%
(2/2) checking package integrity
    [############################################] 100%
(2/2) loading package files
    [############################################] 100%
(2/2) checking for file conflicts
    [############################################] 100%
:: Processing package changes...
(1/2) installing lzo
    [############################################] 100%
(2/2) installing btrfs-progs
    [############################################] 100%
Optional dependencies for btrfs-progs
    python: libbtrfsutil python bindings
    e2fsprogs: btrfs-convert [installed]
    reiserfsprogs: btrfs-convert
:: Running post-transaction hooks...
(1/3) Reloading system manager configuration...
  Skipped: Current root is not booted.
(2/3) Reloading device manager configuration...
  Skipped: Device manager is not running.
(3/3) Arming ConditionNeedsUpdate...
```

Im Beispiel hängt das Paket `btrfs-progs` vom Paket `lzo` ab, welches zuvor installiert werden muss.

Die Abhängigkeiten jedes Pakets sind in einer Datenbankdatei hinterlegt, die im entsprechenden Repository enthalten ist. Einige Pakete stellen Teile des Pakets unter einem anderen Namen bereit (`%PROVIDES%` Abschnitte in den Datenbankdateien). Diese nennen wir im Folgenden *virtuelle Pakete*. Hängt also ein Paket von einem virtuellen Paket ab, muss das Paket installiert werden, welches das virtuelle Paket bereitstellt. Zum Beispiel stellt das Paket `ncurses` das virtuelle Paket `libncursesw.so` bereit. Pakete können damit präzise eine Abhängigkeit zu Teilen eines anderen Pakets ausdrücken, z. B. wie hier zur Library `libncursesw.so`.

Die Pakete selbst werden mit Hilfe von PKGBUILD Shell-Skripten gebaut. Diesen Aspekt werden wir im Rahmen des Projektes allerdings nicht beleuchten.

Aufgabe 1 Gerichtete Graphen

In der Projektvorlage finden Sie die Schnittstelle `de.tukl.programmierpraktikum2021.p1.a1.Graph`. Die einzelnen Methoden sind mit Javadoc versehen. Lesen Sie sich den restlichen Aufgabentext sowie die Javadoc Kommentare durch, um die Schnittstelle zu verstehen.

Die Schnittstelle `Graph` modelliert einen gerichteten Graphen. Der Graph besteht aus einer Menge von Knoten (engl. *node*). Zwischen den Knoten gibt es Kanten (engl. *edge*), die jeweils eine Richtung haben. Für zwei Knoten a, b unterscheiden wir also zwischen den Kanten $a \to b$ und $b \to a$. Der Graph kann keine, eine oder beide dieser Kanten enthalten. Dieselbe Kante (also in derselben Richtung) kann jedoch nicht mehrfach enthalten sein. Auch die Kante $a \to a$ ist möglich.

In den Knoten speichern wir Daten (engl. *data*). Die Kanten sind ungewichtet. Die Typparameter `D` der Schnittstelle ist der Typ für die Daten.

Der Benutzer der Schnittstelle muss die Knoten durch eindeutige Identifier vom Typ `String` referenzieren. Mit der Methode `void addNode(String nodeId, D data)` wird ein neuer Knoten mit Identifier `nodeId` im Graph hinzugefügt.

Sollte der Benutzer der Schnittstelle eine Methode wie z. B. `void setData(String nodeId, D data)` mit einem ungültigen Knoten-Identifier aufrufen, so wird die Ausnahme `InvalidNodeException` geworfen.

Kanten werden durch die Identifier der beiden angrenzenden Knoten identifiziert. Der Aufruf

```
graph.addEdge("A", "B")
```

fügt eine neue Kante $A \to B$ hinzu. Falls die übergebenen Knoten-Identifier ungültig sind, wird auch hier eine `InvalidNodeException` geworfen. Falls die Kante (in der angegebenen Richtung) bereits im Graph existiert, so wird eine `DuplicateEdgeException` geworfen.

Die Methode `Set<String> getIncomingNeighbors(String nodeId)` gibt die Menge aller Knoten-Identifier x zurück, für die die Kante $x \to$ `nodeId` im Graph enthalten ist. Umgekehrt gibt die Methode `Set<String> getOutgoingNeighbors(String nodeId)` die Menge aller Knoten-Identifier y zurück, für die die Kante `nodeId` $\to y$ im Graph enthalten ist.

Legen Sie Ihre Lösungen der folgenden Aufgaben im Paket `de.tukl.programmierpraktikum2021.p1.a1` ab.

a) Implementieren Sie eine Klasse `GraphImpl implements Graph`. Überlegen Sie sich, wie Sie den Graph in Java repräsentieren möchten. Es gibt verschiedene sinnvolle Möglichkeiten. Achten Sie darauf, dass die Implementierung effizient ist. Sie können für Ihre Implementierung gerne Werkzeuge und Datenstrukturen aus der Java Standardbibliothek benutzen (Listen[4], Arrays, Maps[5], Sets[6], ...). Sie dürfen jedoch keine Bibliothek zur Repräsentation von Graphen verwenden, da dies das Lernziel torpedieren würde!

b) Erstellen Sie JUnit Tests für Ihre `Graph` Implementierung. Wenn Sie in der Testmethode Methoden aufrufen, die Ausnahmen werfen können, dann müssen Sie auch die Testmethode mit `throws GraphException` versehen. Zum Testen einer erwarteten Ausnahme sei auf die JUnit Dokumentation[7] verwiesen.

   Stellen Sie mit Hilfe der Jacoco Reports sicher, dass Sie (auch ohne die Tests von 5) eine Instruction Coverage von mindestens 95% und eine Branch Coverage von mindestens 90% erreichen.

[4] https://docs.oracle.com/javase/8/docs/api/java/util/List.html
[5] https://docs.oracle.com/javase/8/docs/api/java/util/Map.html
[6] https://docs.oracle.com/javase/8/docs/api/java/util/Set.html
[7] https://howtodoinjava.com/junit5/expected-exception-example/

Aufgabe 2 Paketmanager

In der Projektvorlage finden Sie die Schnittstelle `de.tukl.programmierpraktikum2021.p1.a2.Pacman`. Die einzelnen Methoden sind mit Javadoc versehen. Lesen Sie sich den restlichen Aufgabentext sowie die Javadoc Kommentare durch, um die Schnittstelle zu verstehen. Implementieren Sie dann entsprechend die Klasse `class PacmanImpl implements Pacman`. Modellieren Sie die Abhängigkeiten der Pakete mit Hilfe Ihrer Graphenimplementierung aus der vorherigen Aufgabe. Verwenden Sie als Identifier der Knoten den Namen der Pakete und als Typ für die Daten im Graphen `Package`.

In der Projektvorlage ist im Pfad `./src/main/resources/core.db.zip` eine (leicht angepasste) Kopie des core Repositories mit über 200 Paketen enthalten. Wenn Sie möchten, können Sie die Datei entpacken und sich die darin enthaltenen Paketspezifikationen genauer anschauen. Als Beispiel ist im Folgenden die Paketspezifikation für `sqlite` aufgeführt.

```
%FILENAME%
sqlite-3.35.5-1-x86_64.pkg.tar.zst

%NAME%
sqlite

%BASE%
sqlite

%VERSION%
3.35.5-1

%DESC%
A C library that implements an SQL
    database engine

%CSIZE%
1790381

%ISIZE%
9937446

%MD5SUM%
75b6275b30032cc52c88161155af8516

%SHA256SUM%
17377ad1b45240ca74f1de3a2a485abb...

%PGPSIG%
iQEzBAABCAAdFiEErcih/MFeAdRTEEGe...

%URL%
https://www.sqlite.org/

%LICENSE%
custom:Public Domain

%ARCH%
x86_64

%BUILDDATE%
1618901148

%PACKAGER%
Andreas Radke <andyrtr@archlinux.org>

%REPLACES%
sqlite3

%PROVIDES%
sqlite3=3.35.5

%DEPENDS%
readline
zlib

%MAKEDEPENDS%
tcl
readline
zlib
```

Wie in der Einleitung bereits angedeutet, sind für das Projekt lediglich die Abschnitte `%NAME%`, `%VERSION%`, `%DEPENDS%` und `%PROVIDES%` von Interesse. Alles andere, wie optionale Abhängigkeiten (`%OPTDEPENDS%`), Abhängigkeiten zum Bauen der Pakete (`%MAKEDEPENDS%`) und Konflikte (`%CONFLICTS%`) vernachnässigen wir. Das Beispiel definiert ein Paket `sqlite` mit Version `3.35.5-1` sowie ein virtuelles Paket `sqlite3` mit Version `3.35.5`. Das Paket hängt ab von den Paketen `readline` und `zlib`.

Pakete in den offiziellen Repositories werden laufend aktualisiert, alte Versionen werden vom Repository entfernt. Es wird sichergestellt, dass die Pakete innerhalb eines Repositories stets kompatibel sind (Sie müssen in den Aufgabenteilen also nicht prüfen, ob eine bestimmte Version vorhanden ist).

Verwenden Sie die `Util` Klasse, um auf den Inhalt der mitgelieferten Paketdatenbank zuzugreifen.

Legen Sie Ihre Lösungen der folgenden Aufgaben im Paket `de.tukl.programmierpraktikum2021.p1.a2` ab.

a) Implementieren Sie die Methode `void buildDependencyGraph()`, die den Abhängigkeitsgraphen der Pakete aufbaut. Fügen Sie zunächst normale Pakete als Knoten hinzu, dann die virtuellen Pakete und schließlich die Abhängigkeiten zwischen den Paketen in Form von (gerichteten) Kanten.

b) Implementieren Sie die Methode `Set<String> whoRequires(String pkg) throws InvalidNodeException`, welche die Namen aller Pakete zurückgibt (sowohl installierte als auch nicht installierte), welche das Paket `pkg` als direkte Abhängigkeit benötigen.

   Beispiel: Das Paket `sqlite` wird direkt von den Paketen `gnupg`, `sqlite-analyzer`, `sqlite-tcl`, `elfutils` und `nss` benötigt.

c) Die Methode `String transitiveDependencies(String pkg)` soll die transitiven Abhängigkeiten des Pakets `pkg` in einer Baumdarstellung ausgeben. Verwenden Sie geeignete ASCII Zeichen (z.B. `"+"`, `"|"` und `"-"`).

```
sqlite-3.35.5-1
├── zlib-1:1.2.11-4
│   └── glibc-2.33-5
│       ├── linux-api-headers-5.12.3-1
│       ├── tzdata-2021a-1
│       └── filesystem-2021.05.31-1
│           └── iana-etc-20210202-1
└── readline-8.1.001-1
    ├── ncurses provides libncursesw.so-6-64
    │   └── ncurses-6.2-2
    │       ├── gcc-libs-11.1.0-1
    │       │   └── glibc-2.33-5
    │       │       ├── linux-api-headers-5.12.3-1
    │       │       ├── tzdata-2021a-1
    │       │       └── filesystem-2021.05.31-1
    │       │           └── iana-etc-20210202-1
    │       └── glibc-2.33-5
    │           ├── linux-api-headers-5.12.3-1
    │           ├── tzdata-2021a-1
    │           └── filesystem-2021.05.31-1
    │               └── iana-etc-20210202-1
    ├── glibc-2.33-5
    │   ├── linux-api-headers-5.12.3-1
    │   ├── tzdata-2021a-1
    │   └── filesystem-2021.05.31-1
    │       └── iana-etc-20210202-1
    └── ncurses-6.2-2
        ├── gcc-libs-11.1.0-1
        │   └── glibc-2.33-5
        │       ├── linux-api-headers-5.12.3-1
        │       ├── tzdata-2021a-1
        │       └── filesystem-2021.05.31-1
        │           └── iana-etc-20210202-1
        └── glibc-2.33-5
            ├── linux-api-headers-5.12.3-1
            ├── tzdata-2021a-1
            └── filesystem-2021.05.31-1
                └── iana-etc-20210202-1
```

Das Beispiel zeigt eine mögliche Rückgabe der Methode für das Paket `sqlite` (die Reihenfolge der Pakete innerhalb einer Ebene ist für uns unerheblich). Es sollen stets alle Abhängigkeiten eines Pakets zurückgegeben werden, auch wenn diese bereits installiert sind.

d) Die Methode `List<Package> buildInstallList(String pkg) throws InvalidNodeException` soll eine Liste aller Pakete zurückgeben, die installiert werden müssen, um das Paket `pkg` zu installieren. Beachten Sie, dass die Elemente der Liste wie folgt angeordnet sein müssen: Pakete, die von anderen Paketen in der Liste abhängen, müssen nach diesen Paketen in der Liste stehen. Pakete, die bereits installiert sind, sollen nicht mehr in der Liste aufgeführt werden.

   Die Methode `void install(String pkg) throws InvalidNodeException` soll das Ergebnis von `buildInstallList(pkg)` verwenden, um die benötigten Pakete zu installieren. Installieren meint hier lediglich, dass Sie sich mit einem geeigneten Attribut merken sollen, welche Pakete installiert sind (es soll keine wirkliche Installation auf Ihrem System stattfinden).

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