Hochschule RheinMain
Fachbereich DCSM - Informatik
Prof. Dr. Georg Hinkel

Verteilte Systeme

SS 2023
LV 4132

Übungsblatt 1
Praktische Übungen
Bearbeitungszeit: 2 Wochen
Abgabe: 06.05.2024, 00:00 Uhr MESZ

Aufgabe 1.1 (Repository, 5 Punkte):

In diesem Praktikum werden die Abgaben in einer hochschuleigenen GitLab-Instanz, insbesondere in einem eigens für die Veranstaltung angelegtem Repository, verwaltet.

`https://gitlab.cs.hs-rm.de/verteiltesysteme/hamster`

Auf dieses Repository haben Sie keinerlei Schreibrechte und müssen es daher erst einmal in Ihr eigenes Profil klonen, bei GitLab nennt sich dieser Vorgang *forken*. Bitte achten Sie darauf, dass sie den Fork als privates Repository in Ihrem eigenem Namensraum einrichten und Ihrem Praktikumsleiter Zugriffsrechte auf Ihren Fork geben. Die Vorlagen beinhalten außerdem auch Dateien, mit denen GitLab Ihre Lösungen automatisch prüfen kann. Um diese Funktionalität zu aktivieren, gehen Sie bitte in die Einstellungen Ihres Forks, haken bitte die Funktionalität CI/CD an und speichern („Save Changes“). Wichtig: Damit Ihr Praktikumsleiter die CI-Pipelines sehen kann, sollten Sie ihm mindestens Developer-Rechte geben.

Achten Sie bitte unbedingt nochmal darauf, dass Sie den Fork auf private Sichtbarkeit gestellt haben. Andernfalls müssen wir die hochschulweite Veröffentlichung Ihrer Lösungen als Täuschungsversuch ahnden.

Weiterführende Hilfe finden Sie dazu im Wiki des Studiengangs und im Internet, z.B.:

`https://doku.cs.hs-rm.de/doku.php?id=gitlab`

Wie in der Vorlesung angekündigt, erlauben wir Ihnen, bei der Bearbeitung der Praktikumsaufgaben zwischen mehreren Programmiersprachen zu wählen. Sie können die Lösungen der einzelnen Praktikumsaufgaben in Java, C oder C# einreichen. Dafür bestehen im Repository verschiedene Branches, die mit Vorlagen und CI-Konfigurationen der jeweiligen Plattform gefüllt sind.

Damit Sie mit dem Git-Repository arbeiten können, müssen Sie es zunächst klonen, das zentrale Repository als zweiten Remote einrichten und für jedes praktische Übungsblatt einen separaten Branch anfertigen. Wenn Sie unbedingt wollen, können Sie dies mit Hilfe grafischer Benutzeroberflächen erledigen, auf lange Sicht ist es aber empfehlenswert, zu lernen Git über die Kommandozeile zu bedienen. Führen Sie daher die folgenden Befehle aus:

- Initialer Checkout der Aufgaben aus dem Repository:

  ```
  $ git clone git@gitlab.cs.hs-rm.de:<account>/hamster.git && cd hamster
  ```

  Hinweis: Bitte ersetzen Sie den Platzhalter

- Bitte erstellen Sie für jedes Übungsblatt einen separaten Branch. Für das erste Praktische Übungsblatt sollte dieser den Namen *1-hamsterlib* tragen:

  ```
  $ git checkout main && git checkout -b 1-hamsterlib-lsg
  ```

  Hinweis: Bitte verwenden Sie nicht den Namen *1-hamsterlib*, sonst riskieren Sie Konflikte da das zentrale Repository bereits Branches enthält, die mit `1-hamsterlib/` beginnen.

- Registrierung des zentralen Git-Repositories als zweiten Remote:

  ```
  $ git remote add vs git@gitlab.cs.hs-rm.de:verteiltesysteme/hamster.git
  ```

- Aktualisierung der Vorlage für die gewählte Programmiersprache (im Beispiel Java, für C oder C# ersetzen Sie bitte den Branchnamen):

  ```
  $ git pull --rebase vs 1-hamsterlib/java
  ```

  Diese Vorgehensweise ist auch für alle weiteren Übungsblätter erforderlich. Sollten während der Bearbeitungszeit dieses Übungsblatts noch Änderungen auftreten, können Sie erneut einen Pull ausführen, um diese Änderungen direkt in Ihre Lösung des Übungsblatts zu übernehmen.

- Anzeigen der lokalen Änderungen:

  ```
  $ git diff
  ```

- Speichern der lokalen Änderungen im Repository (wichtig!):

  ```
  $ git commit -a
  $ git push
  ```

Kontrollieren Sie frühzeitig, ob das Template vollständig in Ihr Repository eingefügt wurde und Sie das Projekt mit übersetzen können. Wenn es hier Probleme gibt, wenden Sie sich schnellstmöglich an Ihren Praktikumsleiter!

Zum Übersetzen verwenden Sie bitte

- Für Java:
  `$ ./gradlew jar` Leider ist im zentralen Repository das executable-Flag nicht gesetzt. Daher müssen Sie unter Linux erst noch das executabl-Flag setzen. Verwenden Sie hierzu bitte `chmod +x .gradlew`.

  Sollten Sie mit einem HTTP-Proxy arbeiten (beispielsweise auf einem Poolrechner) steht Ihnen auch das Skript `gradle-build.sh` zur Verfügung, dass Sie mit `./gradle-build.sh $HTTP_PROXY jar` aufrufen können. Dieses Skript macht nichts anderes, als den HTTP Proxy für Gradle umzuformatieren.

- Für C:
  `$ make`

- Für C#:
  `$ dotnet build`

Gerade für Continuous-Integration Pipelines ist die Kenntnis der Kommandozeilenbefehle von Buildsystemen mittlerweile unerlässlich geworden. Es empfiehlt sich daher ebenfalls, dass Sie sich frühzeitig an das Ausführen von Builds über die Kommandozeile gewöhnen, denn das ist genau das, was der GitLab CI-Server mit Ihrem Code machen wird. In den Vorlagen ist jeweils auch

eine *.gitlab-ci.yml*-Datei enthalten. GitLab wird diese verwenden und bei jedem Commit, den Sie in Ihren Fork hochladen automatisch Ihre Lösung übersetzen und Tests ausführen. Diese Tests sind typischerweise (zusammen mit dem Code) Grundlage der Bewertung der Übungsblätter. Sie sollten keinerlei eigene Änderungen an der CI-Konfiguration oder den Testskripten machen müssen. Sollten Sie doch Änderungen machen, müssen Sie diese rechtfertigen.

Sollten Sie Fragen haben, wie die Vorlage eines bestimmten Übungsblatts gebaut wird oder wie die Tests ausgeführt werden sollen, schauen am besten immer in der Datei `.gitlab-ci.yml` nach, da steht immer genau drin, welche Aufrufe der CI-Server mit Ihrem Code tätigt.

Bei Problemen mit dem Git-Zugang wenden Sie sich bitte an die Laboringenieure.

Schicken Sie schließlich eine Email an Ihren Praktikumsleiter mit der Angabe, mit wem Sie in welchem Repository (Link) an den Aufgaben zusammenarbeiten möchten. Dieser wird daraufhin die Konfiguration Ihres Forks prüfen und Ihnen wenn alles richtig ist die 5 Punkte geben.

Aufgabe 1.2 (Projekt „Hamsterasyl“, 15 Punkte):

Wer länger unterwegs ist, kann seine Goldhamster bei einem westhessischen Hamsterverwahrungsunternehmen abgeben. Die Verwaltung der Gasthamster soll nun durch ein neues IT-System unterstützt werden. Die Kunden können ihre Hamster abgeben und ihnen dabei optional einen Vorrat an Leckerli mitgeben. Für die Aufnahme eines Hamsters wird einmalig ein Grundbetrag von 17 € fällig. Die Hamster haben nichts besseres zu tun als ständig in ihrem Laufrad zu rennen, wodurch dieses mit einer Drehzahl von 25 Umdrehungen pro Minute rotiert. Diese Umdrehungen werden vom System erfasst und pro 1000 Umdrehungen erhöht sich der am Ende zu zahlende Betrag um 5 €. Die Kunden können anrufen und sich nach dem Wohlergehen ihrer Lieblinge erkundigen. Für jeden Anruf dieser Art wird 1 € berechnet. Ausserdem können die Kunden ihrem Hamster Leckerli geben lassen. Ist dabei der anfängliche Vorrat an Leckerli erschöpft, so stellt das Hamsterasyl natürlich gerne weitere Leckerli zur Verfügung – allerdings zum Stückpreis von 2 €.

Zufällig entdeckt die IT-Abteilung des Hamsterverwahrungsunternehmens die Open-Source-Bibliothek „Hamsterlib“, die exakt die benötigten Funktionen implementiert, und die somit verwendet werden soll. Den Quellcode zu dieser Bibliothek finden Sie in Ihrem Git-Repository (das Verzeichnis ist abhängig von der gewählten Programmiersprache). Den Aufbau des Repositories wird Ihnen das Readme Ihres Repositories verraten.

Machen Sie sich anhand der Dokumentation und dem Quellcode mit den API-Funktionen der Bibliothek vertraut. Insgesamt bietet die Bibliothek sieben Funktionen.

In der Vorlage finden Sie ein einfaches Hauptprogramm zum Testen der Bibliotheksfunktionen. Erstellen Sie daraus ein Programm `hamster`, das mit einem Kommandoparameter und (je nach Kommando) evtl. weiteren Parametern aufgerufen wird, und das folgende Funktionen unterstützt:

- `hamster list`
  Gibt eine Liste mit dem gesamten Hamsterbestand (Namen, Preise und Leckerli-Vorrat) tabellarisch aus.

- `hamster list Meier`
  Gibt eine Liste mit den Hamstern des Besitzers „Meier“ aus.

- `hamster add Schmidt Pausbackenbube`
  Fügt einen neuen Datensatz für den Hamster „Pausbackenbube“ des Besitzers „Schmidt“ mit dem Standard-Anfangspreis 17 € (= Vollpension zzgl. Laufradbenutzung) hinzu.

- `hamster add Schmidt Pausbackenbube 55`
  Wie oben, jedoch erhält „Pausbackenbube“ einen Vorrat von 55 Leckerli mit auf den Weg.

- `hamster feed Bilbo Baggins 3`
  Verfüttere 3 Leckerli an den Hamster „Baggins“ des Besitzers „Bilbo“. Falls der Leckerli-Vorrat von „Baggins“ erschöpft ist, erhöht sich der Preis um 2 € je Leckerli.

- `hamster state Dirk Dickbacke`
  Zustandsabfrage des Hamsters „Dickbacke“ des Besitzers „Dirk“. Geliefert wird die Anzahl der Laufradumdrehungen, die Größe des Leckerlivorrats und der aktuelle Preis, der sich durch die Abfrage um jeweils 1 € erhöht.

- `hamster bill Bigspender`
  Gibt den vom Besitzer „Bigspender“ zu zahlenden Gesamtbetrag (Summe über alle seine Hamster) aus und löscht alle Datensätze von „Bigspender“.

Allgemeine Hinweise:

- Falsche oder fehlende Benutzereingaben müssen abgefangen und mit einer Fehlermeldung zurückgewiesen werden. Ihr Programm sollte in solchen Fällen eine kurze Bedienungsanleitung („RTFM-Text“) ausgeben.

- Falsche Parameter (z.B. zu lange Besitzer- oder Hamsternamen) werden von der Bibliothek zurückgewiesen. Ihr Programm sollte in solchen Fällen eine qualifizierte Fehlermeldung ausgeben.

- Jede Kombination aus Hamster- und Besitzernamen darf nur ein Mal vorkommen. Versuche, dieselbe Namenskombination ein weiteres Mal einzutragen werden von der Bibliothek erkannt und müssen mit einer qualifizierten Fehlermeldung zurückgewiesen werden.

- Im Verzeichnis `scripts` finden Sie ein Shell-Script `testhamster.sh`, mit dem Sie Ihr Programm gründlich testen können. Es ruft Ihr Programm mit verschiedenen Optionen auf und vergleicht die erhaltene Ausgabe mit der Ausgabe der Referenz-Implementierung. Dieses Skript läuft nur unter Linux (arbeiten Sie wenn nötig mit WSL) und wird auch für die Bewertung Ihrer Abgabe verwendet.

Hinweise Java

- Es kursiert eine Vielzahl von Bibliotheken, mit denen man unter Java Kommandozeilenparameter parsen kann. Leider ist uns keine Bibliothek bekannt, die mit Verben (so etwas wie list, add, feed, state und bill) umgehen kann. Wenn Sie eine gute Bibliothek kennen, sagen Sie uns bitte Bescheid, dann können wir diese Ihren Komilitonen auch zur Verfügung stellen.

Hinweise C

- Verwenden Sie die Funktion `strtoul(3)` um die Zeichenketten aus der Kommandozeile in `int`-Werte umzuwandeln.

- Leider ist uns in C keine Bibliothek bekannt, die mit Verben (so etwas wie list, add, feed, state und bill) umgehen kann. Wenn Sie eine gute Bibliothek kennen, sagen Sie uns bitte Bescheid, dann können wir diese Ihren Komilitonen auch zur Verfügung stellen.

Hinweise C#

- Verwenden Sie am besten die Bibliothek *CommandLineParser* (`https://www.nuget.org/packages/CommandLineParser`), die sie am bequemsten per NuGet-Referenz in Ihr Projekt einbinden können.