Übung 9:
Buildprozesse / Coverage

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Software Engineering
WS 2018/19 - Dr. Michael Eichberg

Abgabe

Die Übung wird als *sbt*-Projekt (`http://www.scala-sbt.org`) bereitgestellt. *sbt* kann verwendet werden, um Java (und Scala) Projekte einfach auszuführen und zu testen. Installieren Sie `sbt` auf Ihrem Rechner und stellen Sie sicher, dass ein Java SDK (min. Java 8) installiert ist. Überprüfen Sie, dass der Java Compiler `javac` auf der Kommandozeile ausführbar ist, auch wenn Sie sich nicht im Verzeichnis mit der ausführbaren Datei befinden. Wenn nicht, passen Sie Ihren `PATH` entsprechend an, Hinweise wie dies bei Ihrem Betriebssystem möglich ist, finden Sie im Internet.
IntelliJ erlaubt das Importieren von *sbt*-Projekten direkt, dafür muss allerdings das Scala Plugin installiert sein. Um ein Eclipse-Projekt zu erzeugen, kann `sbt eclipse` im Projektverzeichnis (das Verzeichnis, dass die Datei build.sbt enthält) ausgeführt werden, danach kann das Projekt mit `Datei > Importieren > Vorhandene Projekte in Arbeitsbereich` importiert werden. Wenn Sie eine `main`-Methode geschrieben haben, können Sie Ihr Programm mit `sbt run` ausführen, Tests können Sie unter `src/test/java/` anlegen und mit `sbt test` ausführen.
Das Ausführen des Kommandos `sbt` im Projektverzeichnis startet den interaktiven Modus von *sbt*. Im interaktiven Modus können Kommandos ausgeführt werden, ohne jedes Mal `sbt` eingeben zu müssen.
Um Ihre Lösung abzugeben, melden Sie sich zunächst unter

`https://submission.st.informatik.tu-darmstadt.de/course/se18`

an und erzeugen Sie ein *submission token*. Wenn Sie den interaktiven Modus von *sbt* verwenden, führen Sie dann folgendes Kommando aus:

`submit <ihreTUId> <submissionToken>`

wobei *<ihreTUId>* Ihre eindeutige Identifikationsnummer an der TU Darmstadt (**nicht Ihre Matrikelnummer!**) und *<submissionToken>* das zuvor generierte Token ist. Geben Sie die spitzen Klammern nicht mit an, der Befehl sollte etwa so aussehen: `submit ab01cdef 01234567`. Wenn Sie nicht den interaktiven Modus verwenden, muss das Kommando in Anführungszeichen gesetzt werden, also `sbt "submit <ihreTUId> <submissionToken>"`. Sie können (innerhalb der Abgabefrist) beliebig oft eine Lösung einreichen, allerdings wird nur die zuletzt eingereichte Lösung bewertet. Die letzte Lösung, die ein Gruppenmitglied eingereicht hat, wird zur Bewertung für die ganze Gruppe herangezogen. Koordinieren Sie sich daher in Ihrer Gruppe, wer Ihre gemeinsame Lösung einreicht.
Stellen Sie sicher, dass Sie das zur Verfügung gestellte Template nutzen und die Namen und Signaturen der vorgegeben Klassen und Methoden nicht verändern sowie dass von Ihnen hinzugefügte Klassen und Methoden die geforderten Namen und Signaturen verwenden. Ändern Sie außerdem nichts an der vorgegebenen Datei `build.sbt`. Andernfalls wird das System Ihre Lösung nicht bewerten können. Beachten Sie, dass der Zugriff auf die Abgabeplattform nur im **internen Netz der Universität** möglich ist. Für einen Zugriff von außerhalb benötigen Sie daher eine VPN-Verbindung. Überprüfen Sie, ob Ihre Abgabe erfolgreich war, indem Sie sie von der Abgabeplattform herunterladen. Überprüfen Sie dabei auch, ob alle Dateien in der Abgabe enthalten sind.

Einführung

In dieser Übung befassen Sie sich mit Buildprozessen und Test Coverage. Bearbeiten Sie die folgenden beiden Aufgaben und erstellen Sie eine einzelne PDF-Datei mit Ihren Antworten. Legen Sie diese Datei in den Ordner `solution` in dem Template, das Ihnen zur Verfügung gestellt wurde, und verwenden Sie zur Abgabe wie oben angegeben `sbt submit`.

Problem 1 Buildprozesse — 10P

In dieser Aufgabe werden Sie einen voll automatisierten Buildprozess erstellen. Hierzu wird GitHub[1] als Git-Repository Provider und Travis CI[2] als Continous Integration Service genutzt.

Das im Template vorhandene Programm kennen Sie bereits aus der letzten Übung. Dieses Programm soll zunächst mittels Gradle[3] gebaut werden. Wichtig ist hierbei, dass Sie das vorhandene build.sbt Skript **nicht** verändern! Diese Aufgabe soll ausschließlich im dafür vorgesehenen Unterordner `flashcards` bearbeitet werden.
Um Gradle zu verwenden, befolgen Sie das entsprechende Tutorial[4]. Beachten Sie dabei, dass Sie natürlich keinen neuen Ordner erstellen müssen, sondern die Befehle im bestehenden Template-Ordner `flashcards` ausführen. Als Testframework wählen Sie `junit`.
Ist das Tutorial richtig befolgt worden, sollte der Gradle-Befehl `build` das Programm nun korrekt bauen und testen. Fügen Sie Ihrem Gradle Buildskript nun das Plugin für Checkstyle[5] hinzu, welches Sie schon in der vorherigen Übung kennen gelernt haben. Denken Sie daran, Checkstyle so zu konfigurieren, dass es die Datei `checkstyle.xml` als Konfiguration nutzt. Ist alles richtig konfiguriert, wird Checkstyle automatisch bei jedem Build-Befehl mitausgeführt. Die Konfigurationsdateien für Checkstyle müssen im Ordner `flashcards` verbleiben, verschieben Sie sie nicht!
Nun soll dem Gradle Buildskript auch das Plugin für SpotBugs[6] hinzugefügt werden. SpotBugs ist die Weiterentwicklung des Ihnen schon bekannten FindBugs aus der letzten Übung. Konfigurieren Sie SpotBugs so, dass es keine Fehler ignoriert (`ignoreFailures = false`). Beachten Sie, dass SpotBugs Java 8 oder 9 voraussetzt (nicht 10 oder höher) sowie dass Probleme auftreten können, wenn im Dateipfad zu ihrem Projekt Sonderzeichen wie Umlaute existieren.
Konfigurieren Sie außerdem den jar-Task[7], damit Sie eine einzige, ausführbare JAR erhalten. Dafür müssen Sie das Attribut `'Main-Class'` von `manifest` auf `de.tud.cs.se.flashcards.Main` setzen.
Erstellen Sie nun ein *öffentliches* GitHub Projekt und pushen Sie mittels git den Inhalt des Ordners `flashcards` in das Projekt.
Binden Sie Travis ein, indem Sie eine `.travis.yml` Datei in dem Ordner erstellen und diese mit dem richtigen Inhalt füllen, damit Travis den `build` Befehl von Gradle nach jedem Push in das GitHub Projekt ausführt. Zusätzlich müssen Sie Ihr GitHub Projekt zu Travis hinzufügen. Ist Travis richtig konfiguriert, wird bei jedem Push ein neuer Build erstellt und ausgeführt. Der letzte Build vor der Abgabe der Hausübung sollte erfolgreich sein.
Fügen Sie Ihrem Projekt zuletzt auch eine `Readme.md` Datei hinzu, die den aktuellen Build-Status des `master` Branchs mittels eines Badges[8] anzeigt.
Wichtig: Geben Sie in Ihrer Lösungs PDF die URL zu Ihrem öffentlichen GitHub Projekt an, da ansonsten keine Bewertung stattfinden kann.

Problem 2 Test Coverage — 6P

Betrachten Sie die folgende Methode, die den Index eines Strings in einem Array von Strings zurückgibt. Wird der String nicht gefunden, gibt die Methode -1 zurück.

```
1  public static int indexOf(String[] arr, String str){
2    int i = 0;
3    while(i < arr.length){
4      if(str == null){
5        if(arr[i] == null)
6          return i;
7      } else if(str.equals(arr[i])) {
8        return i;
9      }
10     ++i;
11   }
12   return -1;
13 }
```

Zum Testen der Methode liegen folgende Testfälle vor:

1 https://github.com/
2 https://travis-ci.com/
3 https://gradle.org/
4 https://guides.gradle.org/building-java-applications/
5 https://docs.gradle.org/current/userguide/checkstyle_plugin.html
6 https://plugins.gradle.org/plugin/com.github.spotbugs
7 https://docs.gradle.org/current/userguide/java_plugin.html#sec:jar
8 https://docs.travis-ci.com/user/status-images/

Nr.	Beschreibung	Wertebelegung	Erwartetes Ergebnis
1	String an Index 1	arr = **new** String[]{ "abc", "def"}; str = **new** String("def");	0
2	String nicht in Array	arr = **new** String[]{}; str = "abc";	-1
2	null an Index 0	arr = **new** String[]{ **null** }; str = **null**;	1

Analysieren Sie die Testabdeckung der gegebenen Testfälle in Hinblick auf Bedingungsabdeckung (engl. Condition Coverage). Geben Sie für jeden Testfall alle Zeilen der ausgewerteten Bedingungen an und zu welchen Werten diese Bedingungen auswerten (z.B. "7 – true, 8 – false, . . . "). Geben Sie außerdem an, ob durch alle Testfälle zusammen eine vollständige Bedingungsabdeckung erreicht wird. Ist dies nicht der Fall, geben Sie die Bedingung(en) an, die unzureichend abgedeckt sind und definieren Sie eine minimale Zahl weiterer Testfälle, um vollständige Bedingungsabdeckung zu erreichen.
Ist mit vollständiger Bedingungsabdeckung nun sichergestellt, dass die Methode korrekt arbeitet? Identifizieren und nennen Sie ein Problem und geben Sie einen weiteren Testfall an, mit dem dieses Problem erkannt werden kann. (Hinweis: Können Exceptions auftreten?)