# Lab Report <Template>

Abgabe <01>

<Jan> <Mühlbacher>

<Sebastian> <Toporsch>

<Tobias> <Weigl>

Repository: <URL>

## Vorgehen

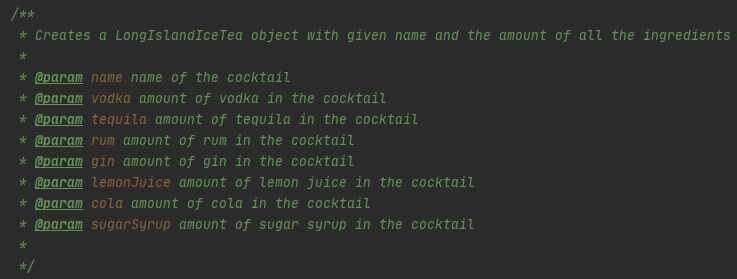
Notieren Sie Ihr Vorgehen in den Übungen, dokumentieren Sie fortlaufend wichtige Schritte, erstellen Sie Screenshots (wenn hilfreich), dokumentieren Sie für Sie wichtige Erkenntnisse, usw.

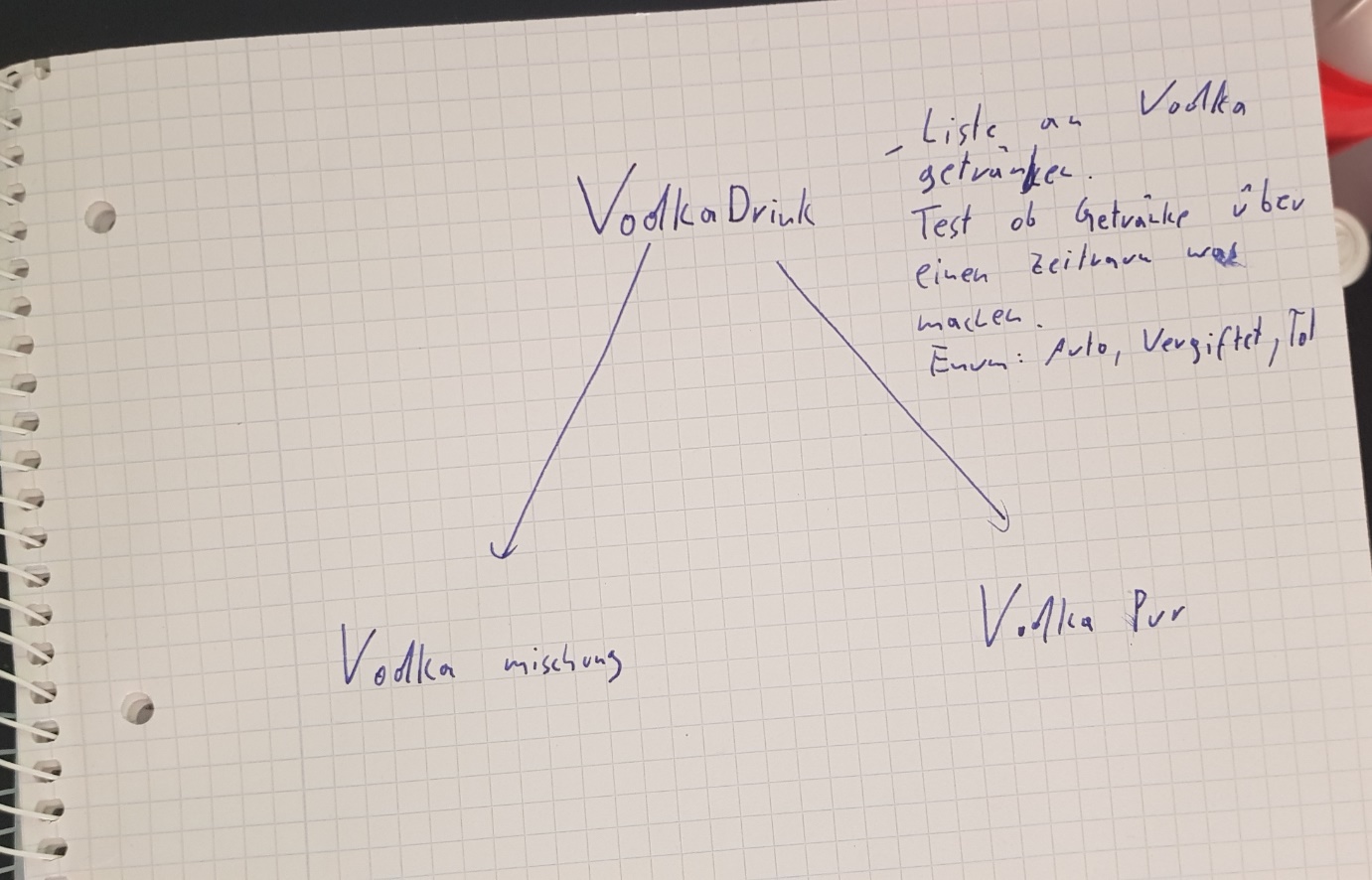
Übung 1 (19.03.2020):

Als erstes wurde besprochen welche Aufgaben jeder übernehmen soll. Wir einigten uns darauf, dass Tobias Weigl und Jan Mühlbacher sich um die Klassen und Unterklassen kümmern und Sebastian Toporsch die README und die Tests schreibt. Danach wurde besprochen wie jeder seine Aufgaben realisieren will und danach startete jeder mit seiner Arbeit.

#### Detailliertes Vorgehen Tobias Weigl:

Nachdem das GitHub Repository von Herrn Toporsch angelegt wurde, habe ich zuerst die Repository lokal auf meinen Rechner gecloned und dann angefangen meine Subklasse mitsamt der zwei Subklassen dieser Klasse zu konzeptionieren. Ich habe mich hier für Vodka basierte Getränke entschieden. Zuerst habe ich auf Papier ein ganz simples Diagramm erstellt um die Klassenvererbung zu visualisieren und um festzustellen, welche Klassen überhaupt erzeugt werden müssen. Als Überklasse habe ich „VodkaDrink“ gewählt und als Subklassen „Vodka mischungen“ und „Vodka Pur“. Um das Ganze ins Englische zu bringen habe ich die Subklassen im Nachhinein in „VodkaLongdrink“ und „VodkaShot“ umbenannt.

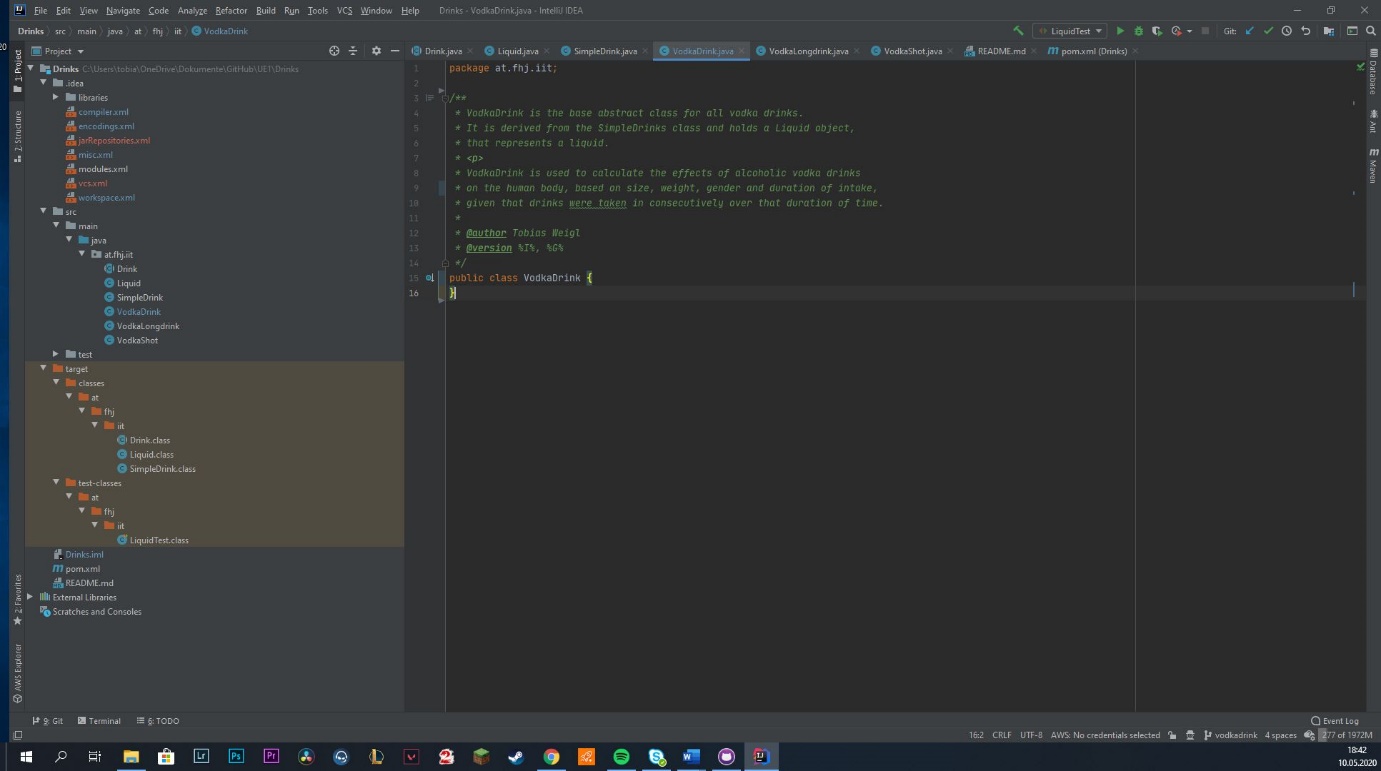




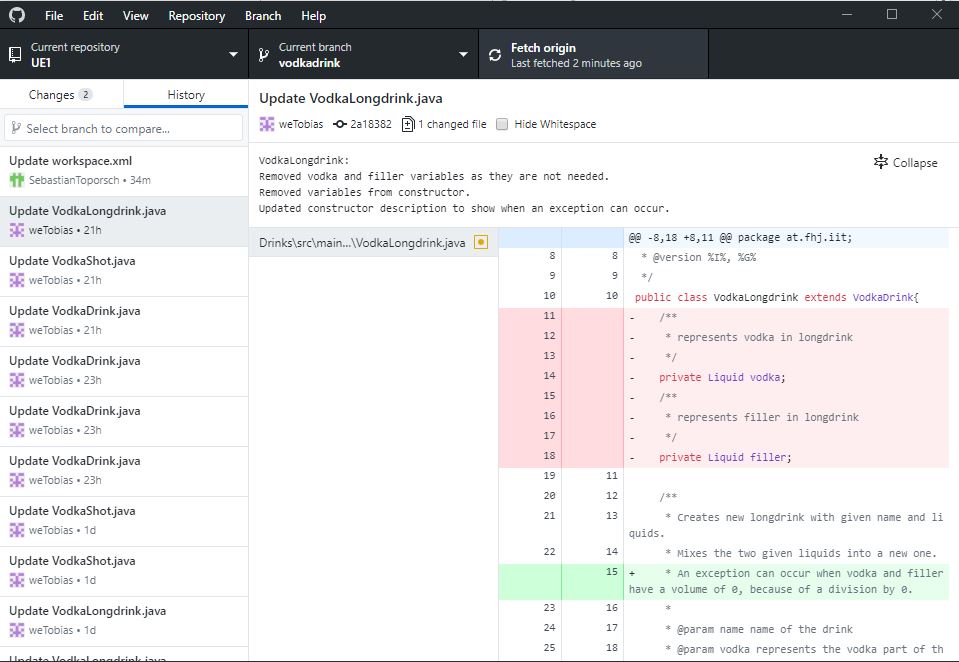
„VodkaDrink“ sollte in diesem Konstrukt einen Sammelbegriff für alle Drinks darstellen, die Vodka beinhalten. „VodkaLongdrink“ sollte dagegen ein Mischgetränk aus Vodka und einem alkoholfreien Getränk und „VodkaShot“ puren Vodka darstellen.  
Als nächsten Schritt habe ich mir überlegt, was dieses Konstrukt als Funktionalität mitbringen sollte und bin schlussendlich auf die Idee gekommen, einen Blutalkoholgehalt Rechner zu programmieren, oder besser gesagt einen Rechner, der dir basierend auf deinem Blutalkoholgehalt deinen aktuellen Körperstatus zurückgeben soll. Dies soll verwirklicht werden, indem man durch eine Methode in der „VodkaDrink“ Klasse eine Liste aus „VodkaDrink“ Objekten einliest, und daraus basierend auf Geschlecht und Gewicht der zu berechnenden Person, sowie der Zeit der Einnahme einen Blutalkoholgehalt errechnet und dann einen Körper Status zurück gibt.  
Die Rückgabe war zuerst als Enum konzipiert, ich habe mich letzten Endes aber für einen String entschieden.

Nach der Konzeption habe ich eine kurze Zusammenfassung meines Konzepts geschrieben und als „Description\_VodkaDrinks.txt“ auf die GitHub Repository geladen, damit Herr Toporsch sie in die „readme.md“ implementieren kann.  
Im Anschluss habe ich in der Repository eine neue Branch namens „vodkadrink“ geöffnet, in der ich von nun an meine updates gepushed habe um sie zum Schluss mit der master Branch zu vereinigen.

Jetzt habe ich das Projekt in IntelliJ importiert indem ich ein neues Projekt aus bestehenden Quellen erstellt habe und hierfür die „pom.xml“ angewählt habe, um es auch wirklich als Maven Projekt zu importieren. Angefangen habe ich die Implementierung meines Konzeptes, indem ich zuerst für alle Klassen festgestellt habe, welche Methoden und Konstruktoren sie benötigen. Diese habe ich dann als Signatur implementiert und mithilfe von JavaDoc beschrieben.



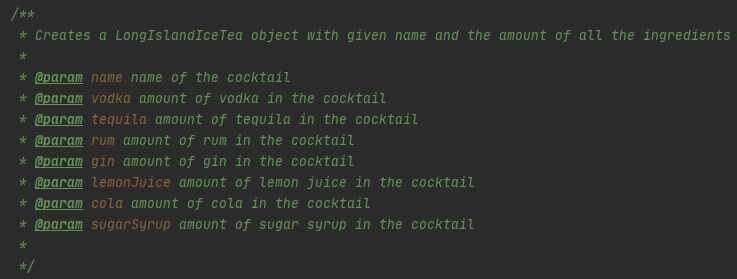
Als Solches habe ich die Klassen auch das erste Mal auf die GitHub Repository gepushed. Danach habe ich die Methoden und Konstruktoren implementiert, damit sie ihre Funktionalität erhalten. Im Verlauf des Programmierens haben sich manche Ideen leicht abgeändert und es wurden Bearbeitungen durchgeführt, diese sind aber immer einher gegangen mit einer Bearbeitung der Beschreibung, sodass immer klar ist, was der Code gerade macht. Die Änderungen habe ich regelmäßig auf die Repository gepushed und mit Nachrichten versehen, die beschreiben was geändert wurde.



#### Detailliertes Vorgehen Jan Mühlbacher:

Zu aller erst habe ich mir ein Thema für meine 3 Klassen überlegen müssen. Da mir Herr Weigl bereits von seiner Idee berichtet hatte, wollte ich eine ähnliche Route einschlagen. Ich plante eine genaue Berechnung der Kalorien eines bestimmten Cocktails ermöglichen und nannte die Subklasse von Drinks somit “CocktailDrinks”. Die Subklassen dieser Klasse repräsentieren zwei sehr bekannte Cocktails - “LongIslandIceTea” und “PinaColada”.

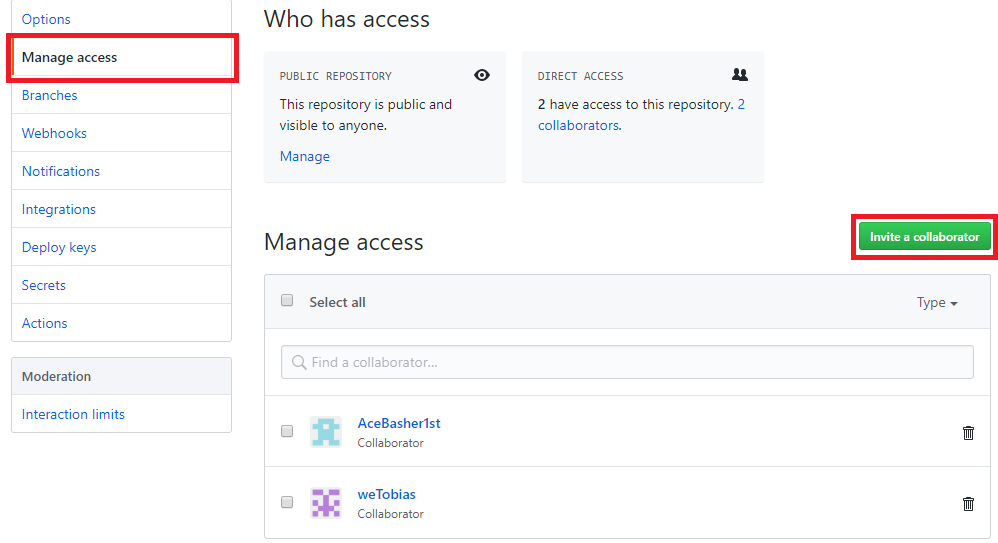
Nach der Idee kam die Umsetzung und so kopierte ich das working-directory auf meinen PC um darin arbeiten zu können. Weiters erschuf ich den Pfad “Drinks”, da wir uns darauf geeinigt hatten, dass ich für die Umsetzung der Drinks-Subklassen zuständig bin. Darufhin schrieb ich zuerst eine txt-Datei und hielt darin fest, was ich plante in meinen Klassen zu implementieren. Daraufhin erstellte ich meine drei Klassen mit logikfreien Methoden und Variablen. Diese dokumentierte ich natürlich alle und pushte meine Änderungen auf den Drinks-Pfad mit einer kleinen Beschreibung.

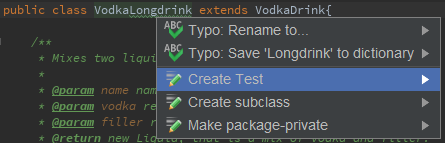


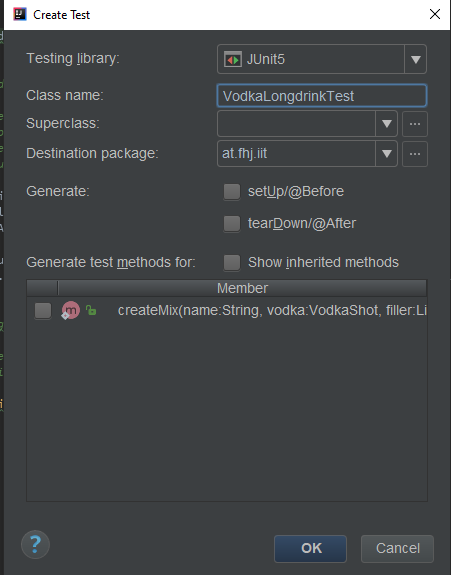
Um eine möglichst genaue Berechnung der Kalorien zu gewährleisten, habe ich mich auf der Seite fatsecret, nach den Kalorienangaben der Zutaten, informiert. Diese Seite erschien mir sehr genau was dieses Thema betrifft, da andere Seiten auch ähnliche Werte anzeigten. Um das Programm jedoch einheitlich zu halten, habe ich meine Kalorien-Daten lediglich von dieser Seite. Um den Alkoholgehalt so genau wie möglich berechnen zu können, habe ich mich von mehreren Seiten beraten lassen, doch die meisten schienen sehr ähnliche Werte wie Wikipedia zu beinhalten, also entschloss ich mich Wikipedia als einheitliche Quelle für diesen Wert heranzuziehen.

Als Nächstes ging ich an die Umsetzung der Funktionen, damit diese auch testbar sind. Im Laufe dieser Arbeiten habe ich die Änderungen immer wieder dokumentiert auf meinen Pfad gepushed. Als ich damit fertig war, fiel mir auf, dass ein paar weitere Funktionalitäten die Klassen deutlich verbessern würden und fügte diese samt Dokumentation - unter Absprache mit meinen beiden Kollegen - hinzu.

#### Detailliertes Vorgehen Sebastian Toporsch:

Ich machte ein neues Repository auf und lud meine Teammitglieder ein. Dafür ging ich bei meinem Repository auf „Settings“ und dann auf „Manage access“. Danach musste ich die beiden noch nach entweder ihrer Email oder ihrem Nutzernamen fragen und dann bei „Invite a collaborator“ die Daten, die sie mir angesagt hatten, eingeben.

Nachdem sowohl Tobias als auch Jan ihre Klassen und Subklassen erstellt und gepusht hatten, habe ich die Tests gemacht. Dafür drückte ich bei angewählter Klasse ALT+Enter um ein Dialogfenster zu öffnen und wählte die Option: „Create Test“ aus. 

Nachdem ich dies tat öffnete sich ein neues Fenster in dem ich noch auswählen sollte ob ich beim Erstellen auch noch Tests für bestimmte Klassen hinzufügen will. 

Danach wurde die neue Test Klasse direkt in den Ordner mit den Tests geschoben und ich schrieb die Tests für die im vorhinein besprochenen Klassen. Ich erstellte für jede Klasse (auch getter und setter wenn vorhanden Tests) und besprach immer wieder mit meinen Teammitgliedern wie die Klasse funktionieren soll bzw. was die Rückgabe mancher Methoden ist, falls es nicht ersichtlich war.

Als letzten Schritt habe ich die README aktualisiert und wichtige Code Schnipsel hinzugefügt.

**Kapitel aus der Lehrveranstaltung**

**Source Code Versionierung mit Git:**

Versionierung dient der Speicherung jeder Veränderung, die während der Prozess des Zusammenarbeitens getätigt wird. Dies dient zur einfacheren Zusammenarbeit und vielen anderen Verbesserungen, die das Leben eines Entwicklers erleichtern.

Mit Git kann man zum Beispiel:

* Aufzeichnungen über jede Veränderung aller Dateien sehen
* Zugriff auf bestimmte Versionen die über die Zeit entstehen
* Speichert Veränderungen gemacht hat
* Zurück zu einer bestimmten Person gehen
* Veränderungen vergleichen

Bei Git gibt es drei Zustände:

* Modified
  + Dateien verändern
* Staged
  + Veränderungen fertig und bereit neue Version zu machen
* Comitted
  + Das repository hat eine neue Version

Folgende Befehle gibt es hierbei die besonders wichtig sind:

* git clone <URL>
  + Kopiert ein bestehendes repository
* git push origin master
  + „Schiebt“ die Daten auf den Server und überschreibt den Branch den man auswählt hier „pushed“ man auf den master Branch
* git pull
  + „Zieht“ die Daten vom Server wo das repository ursprünglicherweise geklont wurde

Git commit messages:

Dienen der Rückverfolgung der Änderungen und dass man genau nachvollziehen kann was in welchem Schritt gemacht worden ist.

**Test Automatisierung:**

## Links / Literatur

Dokumentieren Sie hilfreiche Links zu Onlineliteratur bzw. welche Literatur Sie zur Erarbeitung der Abgabe/Übung verwendet haben.

Apache Maven Standard Directory Layout:

<https://www.baeldung.com/maven-directory-structure>

Markdown Syntax:

<https://www.markdownguide.org/basic-syntax/>

Kalorien:

<https://www.fatsecret.com/>

Alkoholgehalt:

<https://de.wikipedia.org/>

* Adaptieren Sie das Template für Ihre Bedürfnisse.  
  Beachten Sie dabei folgende Angaben.
* Entfernen Sie diese Checkliste
* Verwenden Sie folgendes Schema zum Speichern Ihres Lab Reports  
  <JAHR>-Abgabe<Nr>-<Nachname1>-<Nachname2>.pdf  
  **zB: 19-Abgabe01-Ulm-Hofer.pdf**
* Verwenden Sie dieselbe Namenskonvention für Ihre Abgaben bzw. das Repository  
  zB: https://<url-git-server>/19-Abgabe01-Ulm-Hofer/
* Min. 1 A4 für das jeweilige Kapitel in der Lehrveranstaltung, Dokumentieren Sie in einem sinnvollen Maß sodass Sie und StudienkollegInnen Ihre Dokumentation nachvollziehen können!