Matthias Michael Hauf 1327763
Jonathan Brenner 1329488

Getting Started

Für die Aufgabe 1 ist eine Microservice verfügbar. Dieser kann mit "lein run" im Verezichnis ./efp-server_aufgabe_one auf der Kommandozeilen gestartet werden. Der Client ist über ./web_client/index.html Verfügbar.

Spezifikation der Schnittstelle

HTTP-Verb:	POST
body:	text/plain
response:	application/json
port:	http 5000
	ssl 8443
	Aufgabe1
url	localhost

Im body wird der Inhalt der Java-Klasse erwartet, der für die Validierung benutzt wird.

Ablauf der Analyse und verwendete Design Patterns
Der Inhalt des Java-File wird an den Server gesendet. Dieser ruft die Methode check auf und entfernt zuerst alle Java-Kommentare aus dem String. Dieser kommentarlose String wird dann aufgeteilt in seine Klassen parts. Dazu wird jeder wichtige Part erkannt und in ein Object hinzugefügt. so kann der Nutzer nach der Aufteilung leicht auf Klassen Namen, Imports und die Funktionen zugreifen. Dann wird spezifisch für die Aufgabe nach Fehlern geschaut. Bei Klassen Name und Package ist dies dank der Aufteilung sehr einfach. Um die requestQueue zu überprüfen wird rückwärts gesucht. Es beginnt bei der suche des des Customer Creations diese muss nämlich mit der Request Queue aufgerufen werden. Dadurch erhalten wir den Namen der Variable und können von dieser nach ihrer Instanziierung Ausschau halten. Dort lösen wir dann die static final Variable auf und vergleichen es mit dem gewünschte jndi.propertie.

Verwendete Design Patterns

Tail Recursion wird verwendet, um bei mehreren Funden über einen String zu iterieren. Es wird auch im ganzen Code durchgehend verwendet.

```
(defn get-all-matches
  [javastring regex]
  (loop [matcher (re-matcher regex javastring) matches []]
    (if-let [match (re-find matcher)]
      (recur matcher (conj matches match))
      matches)))
```

Matthias Michael Hauf 1327763
Jonathan Brenner 1329488

Hier benutzen wir Replacing Iterator um über die von uns definierten Suchbegriffe zu iterien und fügen das Ergebnis einer neuen Map hinzu.

```
(reduce
        (fn [new-map [key val]]
          (assoc new-map key (re-find (re-pattern (str "(?<=" val ")\\w+")) split-class-line)))
        {}
        {:class-name "public class " :extends "extends " :implements "implements "})
```

Replacing Template Method wird verwendet, um unterschiedliche Notifikation bei der Analyse zu erstellen. So werden leichte Fehler nur als Warnung erkannt und schwerwiegende als Error.

```
(defn buildAddNotifcation [type]
  (fn [result name comment]
    (into result
      [{:type type
        :name name
       :description comment}])
   )
  )

(def addWarning (buildAddNotifcation "Warnung"))
(def addError (buildAddNotifcation "Error"))
```

Fehlerklassen

Name	Beschreibung	Gewichtung
ClassNameError	Der Name der Klasse entspricht nicht den Anforderungen aus der Aufgabenstellung.	Warning
PackageNameError	Der Packagename entspricht nicht den Anforderungen aus der Aufgabenstellung.	Warning
PropertiesError	Der Pfad für die Properties ist nicht korrekt angegeben, wie in der Aufgabenstellung gefordert.	Error
JMSTextMessageError	Die JMS TextMessage konnte nicht gefunden werden.	Error

Gewichtung

Warning	Leichtgewichtiger Fehler
Error	Schwergewichtige