Hinweise für die Übungsaufgaben zu Middleware – Cloud Computing

Arbeitsumgebung

- Alle Teilnehmenden bekommen ein eigenes Projektverzeichnis unter

  `/proj/i4mw/<loginname>`

  Dieses ist aus dem CIP-Pool zugreifbar und kann als Workspace für die Übungsaufgaben dienen.

- Hilfestellungen und Vorgaben zu den Übungsaufgaben finden sich im *Pub-Verzeichnis* unter

  `/proj/i4mw/pub/<aufgabe>`

- Die Aufgaben sollen in 3er-Gruppen bearbeitet werden. Dazu erhält jede Gruppe ein eigenes Projekt-Repository im GitLab (`https://gitlab.cs.fau.de/`).

  Wichtig: Bei (zusätzlicher) Verwendung eines eigenen Repositories o.Ä. ist sicherzustellen, dass dieses **nicht von außerhalb der eigenen Gruppe** zugreifbar ist.

- Da die Aufgaben in MW teilweise die selben Bibliotheken (z.B. `jaxrs`) verwenden, bietet es sich an, alle Aufgaben in einem gemeinsamen Projekt zu bearbeiten und keine separaten Verzeichnisse je Aufgabe anzulegen.

Bearbeitung der Aufgaben

- Die Aufgaben können an vielen Stellen in Arbeitspakete aufgeteilt werden, dennoch empfiehlt sich eine enge Zusammenarbeit in der Gruppe. Eine mögliche Arbeitsweise innerhalb der Gruppe ist

  1. Alle Gruppenmitglieder lesen sich **vor** der Bearbeitung das Aufgabenblatt **vollständig** durch
  2. Gemeinsames Treffen zur Absprache (u.A. Überblick Gesamtsystem, Zusammenspiel der einzelnen Komponenten, Aufteilung in Arbeitspakete)
  3. Implementierung der einzelnen Komponenten (in Gruppenarbeit oder einzeln)
  4. *Eventuell: Gemeinsames Treffen zur Integration und Testen der Gesamtfunktionalität*

- Der Code sollte lesbar und nachvollziehbar sein. Komplizierte Teile sollten mit Kommentaren erläutert werden.
- Wenn eine bestimmte Programmstruktur in der Aufgabenstellung verlangt wird, sollte die Lösung diese einhalten:
  - Namen von Klassen und Packages
  - Sichtbarkeit, Namen, Parametertypen und Rückgabewerte von Methoden (und Variablen)
- Die Lösungen müssen eigenständig erstellt worden sein. Verstöße werden geahndet!
- Aufgabenteile, die für die 5-ECTS-Variante optional sind, werden auf dem Aufgabenblatt dementsprechend gekennzeichnet (vgl. Teilaufgabe 1.2.3 bei Aufgabe 1). Für 7.5 ECTS müssen **alle** Aufgabenteile bearbeitet werden.

Abgaben

- Zum Abgabezeitpunkt sollte die finale Version der Lösung im Projekt-Repository eingecheckt sein. Die eigentliche **Abgabe einer Aufgabe erfolgt durch Präsentation** der eigenen Lösung gegenüber einem Übungsleiter.
- Sollte eine Präsentation der eigenen Implementierung bis zum Abgabetermin nicht möglich sein, ist dies **im Voraus** entweder per E-Mail (siehe unten) oder persönlich einem Übungsleiter mitzuteilen.
- Die Abgabe kann entweder während der regulären Rechnerübung erfolgen oder an einem individuell vereinbarten Termin stattfinden. Die Vereinbarung für individuelle Termine erfolgt über einen Online-Terminplaner.

Bei Problemen mit der Aufgabenstellung könnt ihr euch jederzeit an die Mailingliste wenden:

`i4mw@lists.cs.fau.de`

Für organisatorische Fragen dient die folgende Mailingliste:

`i4mw-owner@lists.cs.fau.de`

1 Übungsaufgabe #1: Web-Services

In der ersten Aufgabe sollen Freundschaftsbeziehungen in einem sozialen Netzwerk analysiert und das Ergebnis per Web-Service verfügbar gemacht werden. Von allen Nutzenden des Netzwerks sind dabei folgende Daten bekannt:

- *ID*: eine Zeichenkette, die Nutzende eindeutig kennzeichnet
- *Name*: eine Zeichenkette mit dem (Klar-)Namen eines/-r Nutzenden
- *Freunde*: eine Liste mit den *ID*s der Freundschaftsbeziehungen eines/-r Nutzenden

Bereitgestellt werden diese Daten von einem bereits vorhandenen Web-Service, dem „Facebook-Service“.

1.1 Clients

In der ersten Teilaufgabe soll ein Web-Service-Client implementiert werden, der den Zugriff auf den Facebook-Service über eine Kommandozeilen-Shell ermöglicht. Hierfür ist es zunächst erforderlich, die in einer Registry veröffentlichte Adresse des Diensts zu ermitteln, um anschließend Anfragen direkt an ihn stellen zu können.

1.1.1 Registry-Client (für alle)

Die im Rahmen dieser Aufgabe genutzte Registry [Folie 1.3 : 2] existiert bereits und ist in der Lage Gruppen, Dienste und Schlüssel-Wert-Paare (jeweils Zeichenketten, siehe Tafelübung) zu verwalten. Jegliche Interaktion mit der Registry soll in einer Klasse `MWRegistryClient` gekapselt werden, die mindestens folgende Methoden umfasst:

```
public class MWRegistryClient {
    public String[] listGroups();
    public String[] listServices(String group);
    public void createService(String group, String service);
    public void deleteService(String group, String service);
    public String[] listKeys(String group, String service);
    public String getValue(String group, String service, String key);
    public void putValue(String group, String service, String key, String value);
    public void deleteValue(String group, String service, String key);
}
```

Die Methode `listGroups()` liefert die Namen sämtlicher registrierter Gruppen zurück. Mittels `listServices()` können die Namen der Dienste einer bestimmten Gruppe erfragt werden. Mit `createService()` lässt sich ein neuer Eintrag für einen Dienst anlegen und mit `deleteService()` wieder entfernen. Die Methode `listKeys()` gibt die Schlüssel aller zu einem bestimmten Dienst verwalteten Einträge zurück. Das Ausgeben, Speichern und Löschen von Schlüssel-Wert-Paaren erfolgt mittels `getValue()`, `putValue()` bzw. `deleteValue()`. Fehler beim Zugriff auf die Registry werden bei allen Methoden durch das Werfen einer `MWWebServiceException` signalisiert.

Aufgabe:

→ Implementierung der Klasse `MWRegistryClient` im Package `mw.client`

Hinweise:

- Unter `/proj/i4mw/pub/aufgabe1/` steht ein Grundgerüst der Klasse `MWRegistryClient` bereit, das es erlaubt, den Client mittels einer Shell zu bedienen. Darüber hinaus enthält das Verzeichnis alle erforderlichen JAX-RS-Bibliotheken sowie eine Beschreibung der Registry-REST-Schnittstelle (`registry-api.readme`).
- Die Adresse der Registry wird unter `/proj/i4mw/pub/aufgabe1/registry.address` bekannt gegeben.
- Für Schreibzugriffe auf die Registry ist eine Authentifizierung erforderlich [Folie 1.3 : 3].
- Bei der Client-Implementierung ist Code-Duplizierung so weit wie möglich (und sinnvoll) zu vermeiden.

1.1.2 Web-Service-Client (für alle)

Unter Verwendung des Registry-Clients lässt sich nun die in der Registry hinterlegte Facebook-Service-Adresse ermitteln (Gruppe: `i4`, Dienst: `facebook`, Schlüssel: `address`). Der eigentliche Zugriff auf den Facebook-Service [Folie 1.1 : 4] ist in einer Klasse `MWWebServiceClient` zu realisieren, die zunächst mindestens folgende Methoden anbietet:

```
public class MWWebServiceClient {
    public String[] search(String string) throws MWWebServiceException;
    public String getName(String id) throws MWWebServiceException;
    public String[] getFriends(String id) throws MWWebServiceException;
}
```

Die Methode `search()` liefert die IDs von Nutzern, deren Namen die Zeichenkette `string` enthält. Mit `getName()` lässt sich der Nutzername zu einer ID erfragen. Ein Aufruf von `getFriends()` stellt die IDs der Freunde eines Nutzers bereit. Fehlersituationen werden in allen Fällen per `MWWebServiceException` signalisiert. Details zur Facebook-Service-REST-Schnittstelle finden sich unter `/proj/i4mw/pub/aufgabe1/facebook-api.readme`.

Für den Zugriff auf den Facebook-Service soll die Klasse `MWWebServiceClient` dem Nutzer, analog zum Registry-Client, eine Kommandozeilen-Shell zur Verfügung stellen. Diese umfasst zunächst zwei Kommandos: Mit „`search <string>`“ soll nach Namen gesucht werden können, die die Zeichenkette `<string>` enthalten; „`friends <id>`“ dagegen listet sämtliche zu `<id>` gehörigen Freunde auf. Zusätzlich zu den vom Facebook-Service zurückgegeben IDs sind in beiden Fällen auch die entsprechenden Namen (→ `getName()`) auf dem Bildschirm auszugeben.

Aufgabe:

→ Implementierung der Klasse `MWWebServiceClient` im Package `mw.client`

1.2 Web-Service

In dieser Teilaufgabe soll ein eigener Web-Service („Pfad-Dienst“ [Folie 1.1 : 6]) entwickelt werden, der basierend auf den Facebook-Service-Daten die kürzeste Verbindung zwischen zwei Nutzern ermittelt. Beispiel: „Woher ‚kennen' sich A und Z?“ → „A ist befreundet mit B, B ist befreundet mit C,..., Y ist befreundet mit Z.“

1.2.1 Pfad-Dienst (für alle)

Der zu erstellende Web-Service verfügt über eine REST-Schnittstelle und bietet seinen Dienst bei HTTP-`GET`-Anfragen auf den Pfad `/path/{startID}/{endID}` an. Die Pfadparameter `{startID}` und `{endID}` repräsentieren dabei die IDs der Nutzer zwischen denen der kürzeste Pfad zu ermitteln ist. Implementiert werden soll der Pfad-Dienst in einer Klasse `MWPathServer`. Als Hilfestellung liegt im Pub-Verzeichnis die Klasse `MWDijkstra` bereit, die den Dijkstra-Algorithmus über folgende statische Methode anbietet:

```
public class MWDijkstra {
    public static String[] getShortestPath(String startID, String endID,
                                           Map<String, Collection<String>> friendships);
}
```

Im Parameter `friendships` erwartet `getShortestPath()` alle zu berücksichtigenden Freundschaftsbeziehungen. Der Wert eines Map-Eintrags (`Collection<String>`) wird dabei als Menge der IDs aller Freunde des zugehörigen Schlüssels (`String`, ebenfalls ID) interpretiert. Je kleiner die Anzahl der übergebenen Freundschaftsbeziehungen, desto schneller terminiert der Algorithmus. Vor dem Aufruf von `getShortestPath()` ist es also erforderlich, eine Vorverarbeitung durchzuführen, um `friendships` zusammenzustellen und dabei möglichst klein zu halten. Hierbei kann beispielsweise folgendermaßen vorgegangen werden (Pseudo-Code):

```
for(int i = 1; true; i++) {
    StartFreundeskreis := alle Nutzer, die von startID in i Schritten erreichbar sind;
    EndFreundeskreis   := alle Nutzer, die von endID   in i Schritten erreichbar sind;
    if(StartFreundeskreis ueberschneidet sich mit EndFreundeskreis) {
        friendships := StartFreundeskreis vereinigt mit EndFreundeskreis;
        break;
    }
}
```

Ausgehend von `startID` und `endID` werden (mittels Facebook-Service) die erweiterten Freundeskreise der beiden Nutzer so lange vergrößert, bis sie sich überschneiden. Ist dies der Fall, soll `friendships` die Vereinigung der beiden erweiterten Freundeskreise bilden; im Regelfall handelt es sich hierbei um weniger als 10.000 Knoten.

```
public class MWPath {
    String[] path;
    int numberOfIDs;
    int numberOfCalls;
}
```

Das Ergebnis der anschließenden Pfadbestimmung soll über ein `MWPath`-Objekt bereitgestellt werden und neben dem Pfad (`path`) auch Informationen darüber enthalten, wie viele verschiedene IDs sich letztendlich in `friendships` befanden (`numberOfIDs`) bzw. wie viele Facebook-Service-Aufrufe (`numberOfCalls`) nötig waren.

Aufgaben:

→ Implementierung der Klassen `MWPathServer` und `MWPath` im Package `mw.path`
→ Veröffentlichung der Pfad-Dienst-Adresse in der Registry (eigene Gruppe, Dienst: `path`, Schlüssel: `address`)

Hinweis:

- Die Aufbereitung der Daten kann bei weiter auseinander liegenden IDs (z. B. ab einer Distanz von mehr als sieben Schritten) einige Zeit dauern. Zum Testen sollten daher zunächst Start- und End-IDs gewählt werden, von denen (z. B. durch Ausprobieren) bekannt ist, dass sie näher beieinander liegen.

1.2.2 Erweiterter Web-Service-Client (für alle)

Als nächstes ist der Web-Service-Client aus Teilaufgabe 1.1.2 so zu erweitern, dass er über das Kommando „`path <startID> <endID>`“ einen Zugriff auf den Pfad-Dienst anbietet. Bei der Ausgabe eines ermittelten Pfads soll dabei sowohl die ID eines jeden Pfadknotens als auch der dazugehörige Klarname angezeigt werden. Darüber hinaus sind ebenfalls die im `MWPath`-Ergebnisobjekt enthaltenen Statistiken auszugeben.

Aufgaben:

→ Erweiterung der Klasse `MWWebServiceClient`
→ Was ist die kürzeste Verbindung zwischen den Nutzern mit den IDs 1694452301 und 100000859170147?
→ Veröffentlichung der Statistiken für die oben genannte Verbindung in der Registry (eigene Gruppe, Dienst: `path`, Schlüssel: `number-of-ids` bzw. `number-of-calls`)

1.2.3 Verbesserter Pfad-Dienst (optional für 5,0 ECTS)

Die aktuelle Implementierung des Pfad-Diensts aus Teilaufgabe 1.2.1 benötigt eine gewisse Zeit zur Bestimmung der kürzesten Verbindung zwischen zwei Nutzern. Um die Antwortzeit des Diensts verringern zu können, ist es zunächst erforderlich, sich klar zu machen, welcher der Ausführungsschritte (Senden der Anfrage an den Pfad-Dienst, Zusammenstellung der zu berücksichtigenden IDs, Ausführung des Dijkstra-Algorithmus, Zurücksenden der Antwort) jeweils wie viel Zeit in Anspruch nimmt. Von besonderem Interesse ist hierbei auch die Anzahl der Aufrufe am Facebook-Service, die eine einzelne Pfadberechnung nach sich zieht, schließlich bedeutet jeder dieser Aufrufe eine zusätzliche Interaktion mit einem entfernten Rechner.

Um die Anzahl der zu sendenden und zu empfangenden Nachrichten zu reduzieren, bieten viele Web-Service-APIs die Möglichkeit, mehrere Aufrufe in einer einzelnen Anfrage zu bündeln; diese Technik wird als „Batching“ bezeichnet. Auch der Facebook-Service verfügt über zwei Methoden, die diesen Ansatz verfolgen (siehe REST-Schnittstellenbeschreibung in der Datei `/proj/i4mw/pub/aufgabe1/facebook-api.readme`). Diese sollen im Rahmen dieser Teilaufgabe durch eine Erweiterung des Web-Service-Clients [Folie 1.1 : 5] nutzbar gemacht werden:

```
public class MWWebServiceClient {
    public String[] getNames(String[] ids) throws MWWebServiceException;
    public Map<String, HashSet<String>> getFriends(String[] ids) throws [...];
}
```

Ein Aufruf der Methode `getNames()` liefert alle zugehörigen Namen der im Parameter `ids` übergebenen IDs; die i-te Stelle des Rückgabewerts gehört dabei zur i-ten Stelle im `ids`-Array. Mit Hilfe von `getFriends()` lassen sich die Freundschaftsbeziehungen mehrerer Nutzer auf einmal vom Facebook-Service abfragen. Das Ergebnis ist hierbei genauso zu interpretieren wie die `friendships`-Map von `getShortestPath()` aus Teilaufgabe 1.2.1.

Nicht nur für den Web-Service-Client bringen gebündelte Aufrufe Vorteile mit sich, sondern auch für den Pfad-Dienst. So lässt sich mit Hilfe dieser Technik etwa die Anzahl der Facebook-Service-Aufrufe während des Zusammenstellens der IDs für den Dijsktra-Algorithmus drastisch reduzieren, beispielsweise indem alle Freundschaftsabfragen für den `StartFreundeskreis` in einem Schritt zusammengefasst werden (siehe Teilaufgabe 1.2.1, `EndFreundeskreis` analog). Dies führt in der Konsequenz zu einer erheblichen Effizienzsteigerung des Pfad-Diensts und soll abschließend umgesetzt werden.

Aufgaben:

→ Analyse der Latenzen bei der Pfadbestimmung im aktuellen Pfad-Dienst
→ Erweiterung der Klasse `MWWebServiceClient`
→ Bereitstellung einer Implementierung zur Pfadbestimmung, die auf gebündelte Aufrufe zurückgreift
→ Analyse des verbesserten Pfad-Diensts (Wie groß ist der Zeitgewinn?)

Hinweis:

- Dem Web-Service-Client soll es in der verbesserten Implementierung des Pfad-Diensts möglich sein festzulegen, welcher Ansatz bei der Pfadbestimmung zu verwenden ist (d. h. einzelne oder gebündelte Aufrufe). Um dem Dienst diese Entscheidung zu übermitteln, bietet sich das Hinzufügen eines Anfrageparameters zur bereits bestehenden REST-Schnittstelle des Pfad-Diensts an (z. B. „`?batching=<false|true>`“).

Abgabe: am 06.11.2024 in der Rechnerübung