SWE1 – Bericht Übungsbesprechung

In diesem Bericht wird auf die Implementierung und den Aufbau des Navigation Plugin eingegangen. Die Aufgabe des Navigation Plugin besteht darin, eine Datei im XML-Format zu parsen und dabei eine interne Straßen-Ort Zuordnung zu erstellen. Die XML-Datei wurde mithilfe von *Osmosis*, einer Java Applikation zur Verarbeitung von OSM-Dateien, erstellt. Die OSM-Datei deckt Wien sowie Teile Niederösterreichs und des Burgenlands ab. Bei dem Erstellen der XML-Datei wurden nur die relevanten Informationen berücksichtigt, also die Straße und der zugehörige Ort. Dafür wurde folgendes Kommando verwendet:

```
C:\Users\Leo>osmosis --read-xml data.osm --tf accept-nodes --node-key keyList=addr:street --node-key keyList=addr:city --write-xml data_.xml
```

Es werden also nur Node-Tags mit vorhanden Werten zu den Keys *addr:street* und *addr:city* akzeptiert. Das Ergebnis ist eine XML-Datei mit etwa 50 MB, wodurch die spätere Indexierung nicht zu viel Zeit beansprucht.

Das Navigation Plugin übernimmt die Bearbeitung von Requests mit der URL *localhost/navigation?navigation_plugin=true.* Dabei handelt es sich um eine POST Request mit den Wertepaaren ***value*** *(gibt den Straßennamen an)* und ***load*** *(kennzeichnet ob die Karte neu aufbereitet werden soll)*. Die Wertepaarzuweisung erfolgt über das Textfeld (*value*) und die Buttons, (*load*) welche eine Ajax Post Request initialisieren.

User Interface – Navigation Plugin

Erfolgt eine dementsprechende Request, so wird die Funktion *handle()* der Klasse *PluginNavigation* ausgeführt. Wie im Code Ausschnitt zu sehen ist werden zu Beginn benötigte Objekte initialisiert, darunter ein Objekt der Klasse *StreetCollection*.

```
@Override
public Response handle(Request req) {
    resp = new ResponseImpl();
    contentString = req.getContentString();
    streetCollection = StreetCollection.newStreetCollectionInstance();
    boolean isInitialized = true;
```

Initialisierungen in der handle() Funktion – Navigation Plugin

StreetCollection: Die Klasse StreetCollection verwaltet und speichert die Straßen-Ort Zuordnung in einer HashMap mit Straßennamen (String) als Key und den Ortsnamen als Wert (LinkedList). Um diese für mehrere Threads zugänglich zu machen, wird lediglich eine Instanz erstellt und diese auf Anforderung zurückgegeben. Eine weiteres Klassenattribut ist ein Lock der zur Synchronisation

paralleler Threads dient, da während des Prozesses der Straßenaufbereitung keine anderen Anfragen bearbeitet werden können.

```
public class StreetCollection {
    private static Map<String, LinkedList<String>> StreetColl;
    private static StreetCollection streetCollection = null;
    private static Lock lock = new ReentrantLock();
```

Klassenattribute StreetCollection

Ein Thread der eine neue Straßen-Ort Zuordnung erstellen soll, ruft die Methode *setLock()* auf, um den alleinigen Zugriff darauf zu erlangen. Ist der Lock bereits besetzt, check mithilfe von *tryLock()*, wird eine Exception geworfen. Damit erhalten Threads die Information, ob eine Straßen-Ort Abfrage durchgeführt werden kann, oder ob gerade eine neue Aufbereitung stattfindet.

```
public static synchronized boolean setLock() throws IllegalAccessException {
    if (lock.tryLock()) return true;
    else throw new IllegalAccessException("busy"); // if lock is already owne
}
```

Im folgenden Ausschnitt ist zu sehen, wie die Erstellung der Straßen-Ort Zuordnung erfolgt. Ein Objekt der Klasse *Node* enthält den Straßennamen und zugehörigen Ort, und wir mit *handleNode()* der HashMap entsprechend hinzugefügt. Ist der Straßenname noch nicht enthalten, wird ein neues Wertepaar erzeugt, ansonsten wird der Ort der zugehörigen Liste hinzugefügt. Die Methode *getCitiesOfStreet()* gibt eine LinkedList mit den zu einer Straße zugehörigen Ortsnamen zurück, oder *Null* wenn diese nicht existiert.

```
public synchronized void handleNode(Node n) {
    String city = n.getCity();
    String street = n.getStreet();
    if (StreetColl.containsKey(street)) {
        if (StreetColl.get(street) != null && !StreetColl.get(street).contains(city)) {
            StreetColl.get(street).add(city);
        }
    } else {
        StreetColl.put(street, new LinkedList<String>());
        StreetColl.get(street).add(city);
    }
}

public LinkedList<String> getCitiesOfStreet(String street) {
    if (StreetColl.containsKey(street)) {
        if (StreetColl.get(street) != null) {
            return StreetColl.get(street);
        }
    }
    return null;
}
```

Zurück zur Methode *handle()* in der Klasse *PluginNavigation.*

```
boolean isInitialized = true;
try {
    if (contentString != null) {
        contentString = URLDecoder.decode(contentString, StandardCharsets.UTF_8.name());
        String[] bodyVal = contentString.split( regex: "&");
        String[] value = bodyVal[0].split( regex: "=");
        String load = bodyVal[1].split( regex: "=")[1];
        boolean isLoad = load.equals("True");
```

Im ersten Schritt wird geprüft, ob die Request einen Body/Content enthält oder nicht. Da das NavigationPlugin POST Requests abarbeitet sollte bei einer validen Request der Content nicht *Null*

sein. Im nächsten Schritt wird der Content, welcher für die Übertragung URL-kodiert wurde, in UTF-8 Format übersetzt. Darauffolgend werden die beiden Wertepaare *value* und *load* ausgelesen, um festzustellen ob es sich um eine Straßen-Ort Abfrage oder eine neue Straßenaufbereitungsanfrage handelt.

Die Variable *isLoad* ist dann *True*, wenn die Straße neu aufbereitet werden soll, ansonsten *False*. Im letzteren Fall wird geprüft ob ein Straßenname als Wert zum *value* Key übergeben wurde. Ist dies der Fall, so wird mittels *streetCollection.isInitialized()* geprüft, ob bereits eine Straßen-Ort Zuordnung vorhanden ist. Falls diese nicht existiert, wird in der Response eine Fehlermeldung als Content gesetzt und eine Flag *isInitalized* gesetzt. Im anderen Fall wird mittels *setLock()* nun geprüft ob die Straßen-Ort Zuordnung gerade neu aufbereitet wird, wodurch entweder eine Exception geworfen wird oder die Abfrage stattfindet. Im weiteren Ablauf wird mit *getCitiesOfStreet()* die Ortsabfrage durchgeführt und eine LinkedList mit den Ortsnamen zurückgegeben.

```
boolean isLoad = load.equals("True");
if (!isLoad) {
    if (value.length > 1) {
        if (streetCollection.isInitialized()) {
            // check if lock is owned by another thread
            if (streetCollection.setLock()) {
                // if not - release it since no parsing here
                streetCollection.unLock();
                LinkedList<String> cities = streetCollection.getCitiesOfStreet(value[1]);
                if (cities != null) {
                    isValid = true;
                    prepareResponse(cities, value[1]);
                }
            }
        } else if (!streetCollection.isInitialized()) {
            resp.setContent("Error: Straßen sind nicht verfügbar");
            isInitialized = false;
        }
    }
} else if (isLoad) {
```

War die Suche erfolgreich werden die Ortsnamen mittels *prepareResponse()* der Reihe nach dem Response Content hinzugefügt.

```
    public void prepareResponse(LinkedList<String> cities, String street) {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append("Street: " + street);
        stringBuilder.append(System.getProperty("line.separator"));
        for (String s : cities) {
            stringBuilder.append(s);
            stringBuilder.append(System.getProperty("line.separator"));
        }
        resp.setContent(stringBuilder.toString());
    }
```

Ist die Variable *isLoad* auf *True* gesetzt, so handelt es sich um eine neue Straßenaufbereitungsanfrage. Dementsprechend wird mittels *parseXML()* eine neue Straßen-Ort Zuordnung erstellt und zurückgegeben.

```
} else if (isLoad) {
    isInitialized = false;
    streetCollection = XMLParserSAX.parseXML();
    if (streetCollection != null) {
        resp.setContent("Straßen neu aufbereitet");
    }
```

XMLParserSAX: Bei der *parseXML()* handelt es sich um eine statische Methode der Klasse *XMLParserSAX*, welche die Initialisierung des Parsing Prozesses übernimmt. Dafür wird eine *SAXParserFactory* Instanz erstellt, welche es ermöglicht einen *SAXParser* zu erhalten. Genauso wird der Dateipfad der XML-Datei angegeben, sowie ein neues *NavStreetHandler()* Objekt erstellt, wessen Methoden später von der *SAXParser* Instanz während des Parsens aufgerufen werden.

```
public static StreetCollection parseXML() throws ParserConfigurationException, SAXException, IOException, IllegalAccessException {
    SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
    String path = Thread.currentThread().getContextClassLoader().getResource( name: "").getPath() + System.getProperty("file.separa
    SAXParser saxParser = saxParserFactory.newSAXParser();
    handler = new NavStreetHandler();
    // try to lock collection
    handler.tryLock();
    LOGGER.log(Level.INFO,  msg: "Initializing XML parsing");
    saxParser.parse(new File(path), handler);
    //print employee information
    LOGGER.log(Level.INFO,  msg: "XML parsing finished");
    handler.unLock();
    return handler.getsColl();
}
```

NavStreetHandler: Die *NavStreetHandler* Klasse erweitert die *ContentHandler* Klasse und überschreibt gewisse Parse-Methoden. Die Klasse beinhaltet eine *StreetCollection* Instanz, welche während des Parsens erweitert und schließlich zurückgegeben wird. Genauso eine Methode *tryLock()* welche prüft ob der Lock bereits von einem Thread belegt ist.

```
public class NavStreetHandler extends DefaultHandler {
    private String elementValue;
    private String city;
    private String street;
    private int attrCount = 0;
    private Node node;
    private boolean isNode;
    private StreetCollection sColl;

    public NavStreetHandler() throws IllegalAccessException {
        this.sColl = StreetCollection.newStreetCollectionInstance();
    }

    public void tryLock() throws IllegalAccessException {
        sColl.setLock(); // try to lock for parsing
    }
```

Die Methode *startElement()* wird aufgerufen wenn der Parser einen Start-Tag findet. Damit können wir feststellen um welche Art von Tag es sich handelt, in diesem Fall sind die Tags *Node* und *tag* relevant. Der Tag Name wird über die Variable *qName* abgerufen. Da das *tag* Element den Straßen – und Ortsnamen enthält, wird geprüft ob diese existieren und dementsprechend abgerufen.

```java
@Override
// Methode wird aufgerufen wenn der Parser zu einem Start-Tag kommt
public void startElement(String uri, String localName, String qName,
                         Attributes atts) throws SAXException {
    if (qName.equals("node")) {
        isNode = true;
    }
    if (isNode && qName.equals("tag")) {
        String key = atts.getValue( qName: "k");
        if (key.equals("addr:city")) {
            attrCount++;
            city = atts.getValue( qName: "v");
        }
        if (key.equals("addr:street")) {
            attrCount++;
            street = atts.getValue( qName: "v");
        }
    }
```

Sind beide Attribute innerhalb einer Node enthalten, wird in der Methode *endElement()* eine neue *Node* Instanz erstellt und mit *handleNode()* der Straßen-Ort Zuordnung hinzugefügt.

```java
@Override
// Methode wird aufgerufen wenn der Parser zu einem End-Tag kommt
public void endElement(String uri, String localName, String qName)
        throws SAXException {
    if (qName.equals("node")) {
        // check if city and street belong to same node
        if (attrCount == 2) {
            node = new Node(city, street);
            sColl.handleNode(node);
        }
        attrCount = 0;
        isNode = false;
    }
}
```

Node: Die Klasse enthält lediglich den Straßen- und Ortsnamen sowie *Get()* und *Set()* Methoden.

```java
public class Node {
    private String city;
    private String street;

    public Node(String city, String street) {
        this.city = city;
        this.street = street;
    }

    public void setCity(String city) { this.city = city; }

    public void setStreet(String street) { this.street = street; }

    public String getCity() { return this.city; }

    public String getStreet() { return this.street; }
}
```

Ist der Parsing Prozess abgeschlossen, wird die StreetCollection Instanz nun vom NavStreetHandler zurückgegeben und der Lock freigegeben.

```
            saxParser.parse(new File(path), handler);
            LOGGER.log(Level.INFO, msg: "XML parsing finished");
            handler.unLock();
            return handler.getsColl();
        }
```

Das Navigation Plugin erhält damit die neue Straßen-Ort Zuordnung und setzt dementsprechend den Response Content. Im unteren Teil des Code Ausschnitts ist zu sehen, dass im Falle einer *IllegalAccessException*, d.h. wenn während einem versuchten Zugriff gerade der Parsing-Prozess stattfindet, dementsprechend eine Fehlermeldung gesetzt wird, um den Client darüber zu informieren.

```
                } else if (isLoad) {
                    isInitialized = false;
                    streetCollection = XMLParserSAX.parseXML();
                    if (streetCollection != null) {
                        resp.setContent("Straßen neu aufbereitet");
                    }
                }
            }
            if (!isValid && isInitialized) {
                resp.setContent("Not found: Keine Städte gefunden");
            }
        } catch (UnsupportedEncodingException e) {
            LOGGER.log(Level.SEVERE, msg: "Unexpected error: " + e.getMessage(), e);
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            resp.setContent("Warning: Parser is busy right now");
            resp.setStatusCode(200);
            return resp;
        } finally {
            // try to release a lock
            if (parser != null) parser.releaseLock();
        }
        resp.setContentType("text/plain");
        resp.setStatusCode(200);
        return resp;
    }
```