A unifying software framework for vehicle routing and logistics.

Jannik Enenkel , Jannik Geyer , Jan-Niklas Keiner , Johannes Nguyen,
Jan Steulerand and Neil Urquhart
Edinburgh Napier University
School Of Computing
10 Colinton Road
Edinburgh, UK

Motivation

- Involved in a number of research and commercial projects
  - Routing urban milk deliveries
  - Scheduling social care workers
  - Routing food deliveries
  - Multi-objective routing
  - Modelling and optimising commuters

Motivation

- What do all of these have in common?
- They all need to access real-world data
- *The problem:*
  - Differing data sources are accessed in different ways
    - Online APIs
    - Localised data
  - Differing providers produce differing formats of answer
  - Moving applications between data providers becomes difficult

An example

- A research study (Judson, Hart & Urquhart) looked at optimising social care visits
  - Workers could travel by public transport or by car
  - Two scenarios were looked at, the City of Edinburgh and the City of London
- The data sources that we had to work with were
  - The Transport for London API for London area p

An example: 2

- The data sources that we had to work with were
  - The Transport for London API
  - The Travel Line Scotland Journey Planner
  - Open Streetmap / GraphHopper

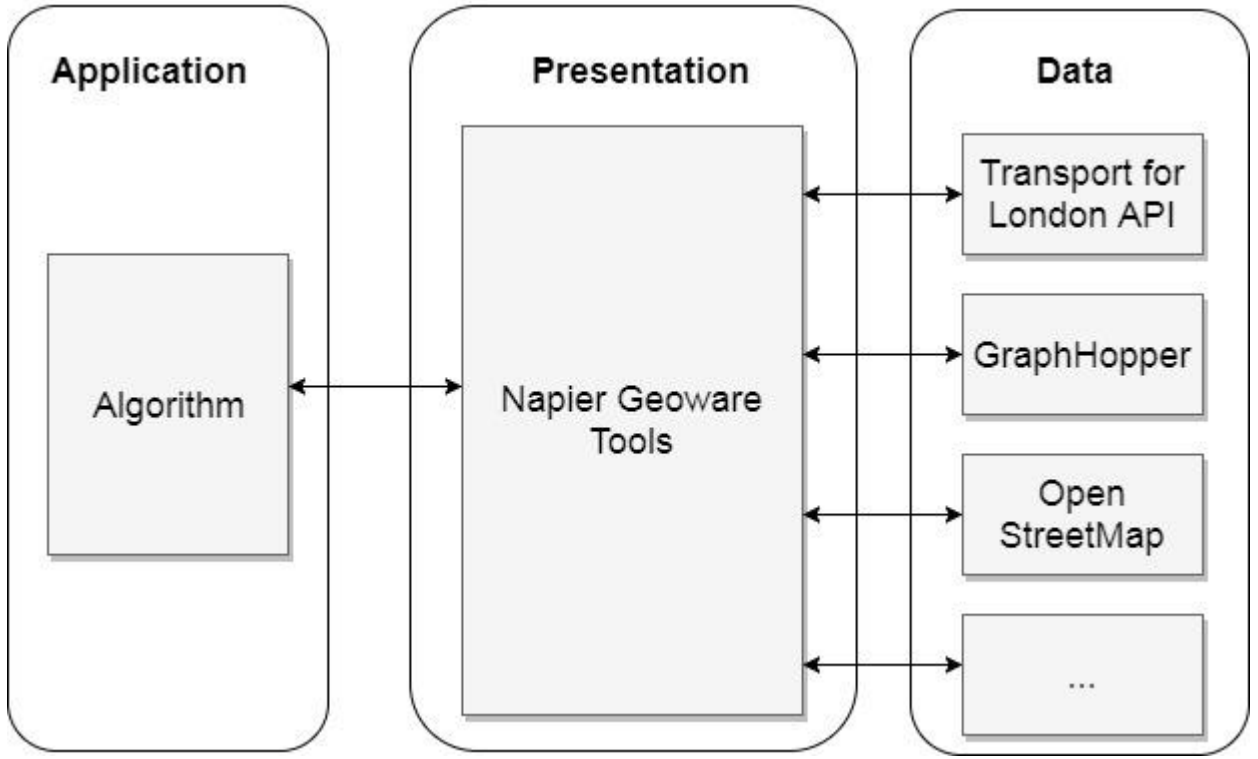
An example: 3

- Although the algorithm (NSGA-II) is the same for all of the examples an extensive amount of modifications had be to be carried out to allow all three data sources to be integrated.

Outline Concept

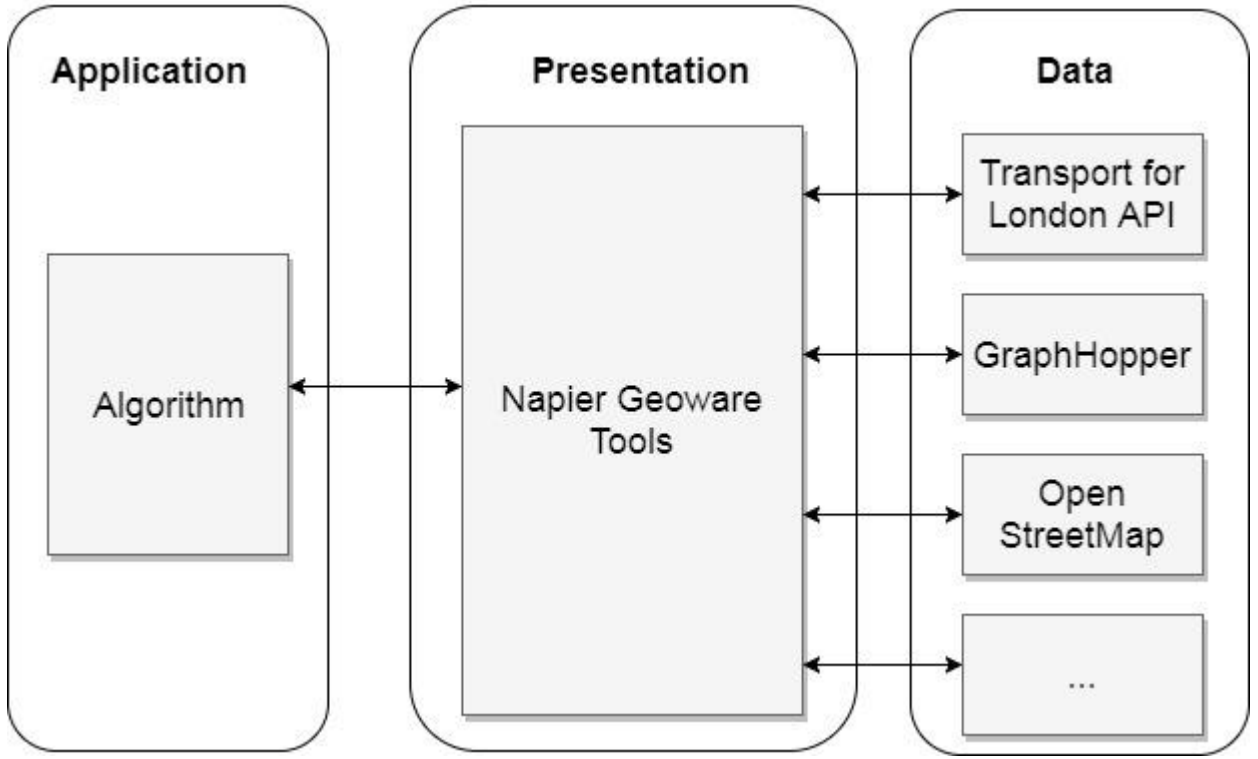
- Can we produce a framework that makes it simple to work with multiple sources of Geospatial information?
- Base the framework on design patterns such *Façade* and *Factory* to hide the detail.
- The application programmer can concentrate on writing the application and not worry about the nature of the data sources

Architecture



- If our interest is in problem solving then we only have to write the Application layer and understand the presentation layer.
- If our application needs to access differing data sources it only requires minimal changes at the application level

Architecture



- The presentation layer (NGT) is open source (hopefully others will add to it!)
- The presentation layer provides an Interface based around a set of generic classes for routes, locations etc.
- When objects are passed from the presentation to application layers the source of the data should be recorded.

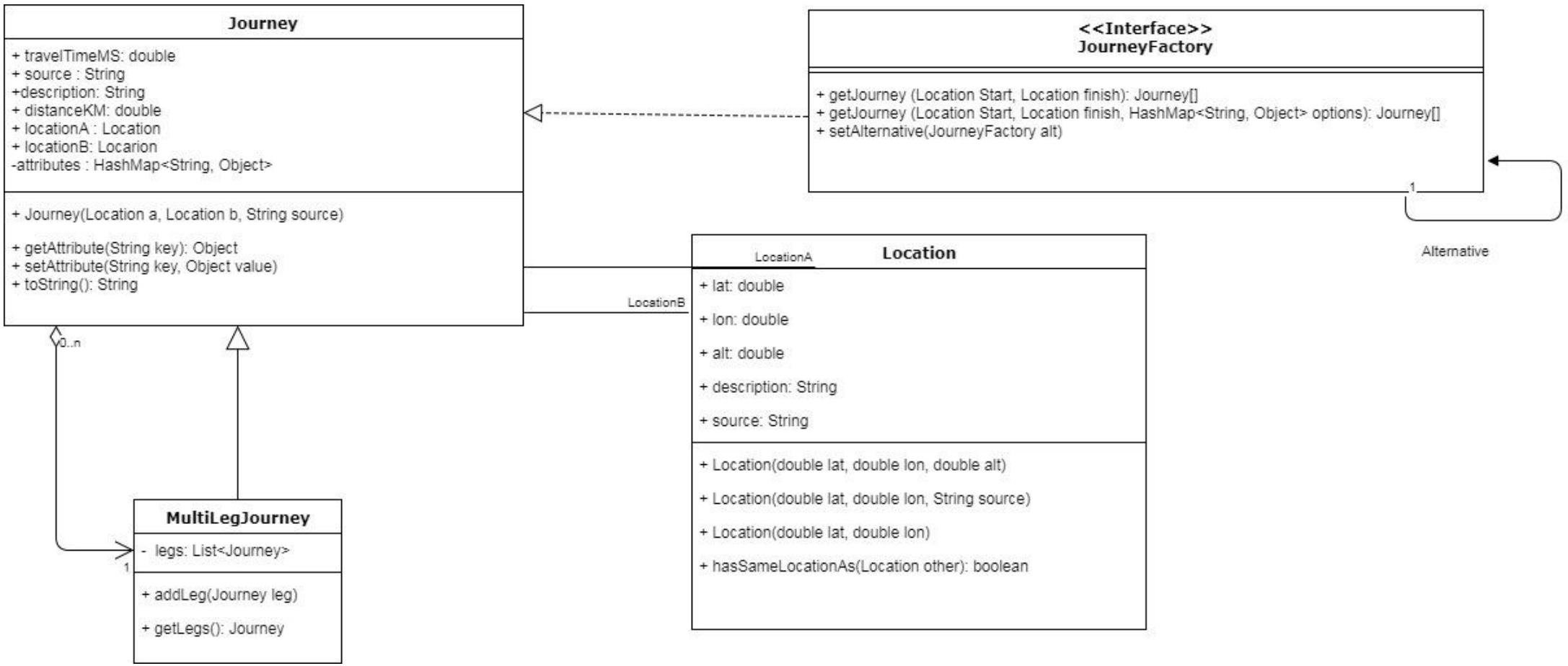
Overview

- In order to speed up various projects it was decided to implement the architecture as *Napier Geoware Tools*.
- The initial set of tools would be constructed by a group of students as final year projects:
  - Jannik Enenkel
  - Jannik Geyer
  - Jan-Niklas Keiner
  - Johannes Nguyen
  - Jan Steulerand
- All were exchange students from Technische Hochschule Mittelhessen

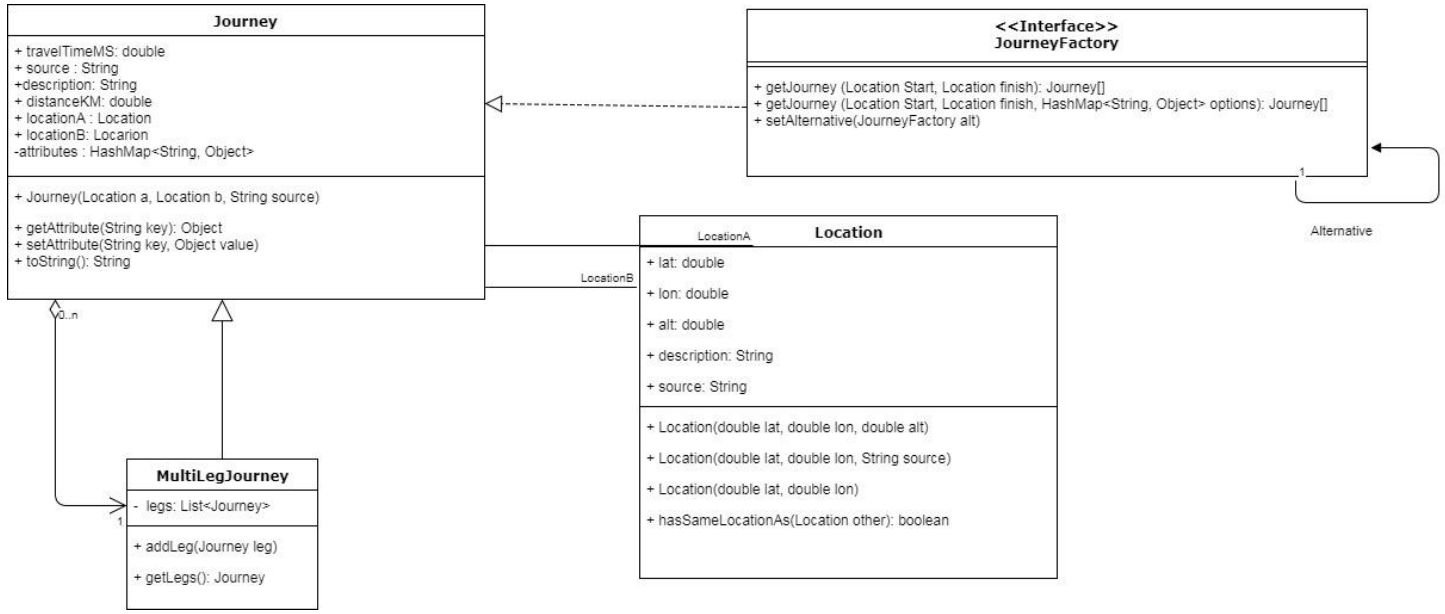
NGT – Contents

- OSM Querying
- GraphHopper routing
- Transport for London routing
- KML writer
- Events/Diary API
- Some utility classes for basic geospatial and OSM data operations

Detailed Design



Detailed Design



- Useful features
  - Use of “Alternative” router
  - MultiLegJourney

Example

- A contrived example (this is available online along with other examples)
  - From a location in Central London, find all of the pubs with 1 square mile
  - Order them in nearest neighbour order
  - Turn that order into a route using TfL journeys
  - Create a calendar with an appointment at each pub (times based on journey times)
  - Print out the instructions
  - Print out a map of the route

Example 1

```
Location currentLocation = new Location(51.507115,-0.127437);
OSMFileUtils.setOsmFilter("/usr/local/Cellar/osmfilter/0.7/bin/osmfilter ");
OSMFileUtils.extractData(currentLocation, "greater-london-latest.osm.pbf", "centralLondon.osm",true);
QueryOsmFacade facade = new QueryOsmFacade();
facade.parseOSMFile("centralLondon.osm");
//need to be able to get OSM data
System.out.println("Found: ");
ArrayList <Location> pubs= facade.findLocationsWithRegEx("pub", currentLocation);

for(Location l : pubs){
        System.out.println(l);
}

EasyKMLCreator kml = new EasyKMLCreator();
kml.addPlacemarks(pubs);
```

Example

```
JourneyFactory router = new SimpleRouter(20);
ArrayList<Location> ordered = nearestNeighbour(currentLocation, pubs,router);
//Setup routers
router = new TransportForLondonFacade("You need to obtain a TfL API key for here....");
JourneyFactory gh = setupHopper();
router.setAlternative(gh);
gh.setAlternative(new SimpleRouter(20));

//Create journey
MultiLegJourney trip = new MultiLegJourney();
Location prev = currentLocation;
for (Location next : ordered){
        Journey[] myJourney = router.getJourney(prev, next);
        trip.addLeg(myJourney[0]);
        prev = next;
}
kml.addJourney(trip);
```

Note the use of alternative JourneyFactories…. If the TfL API fails to return a route, then GraphHopper is tried, if that fails then SimpleRouter is tried as a last resort

Example

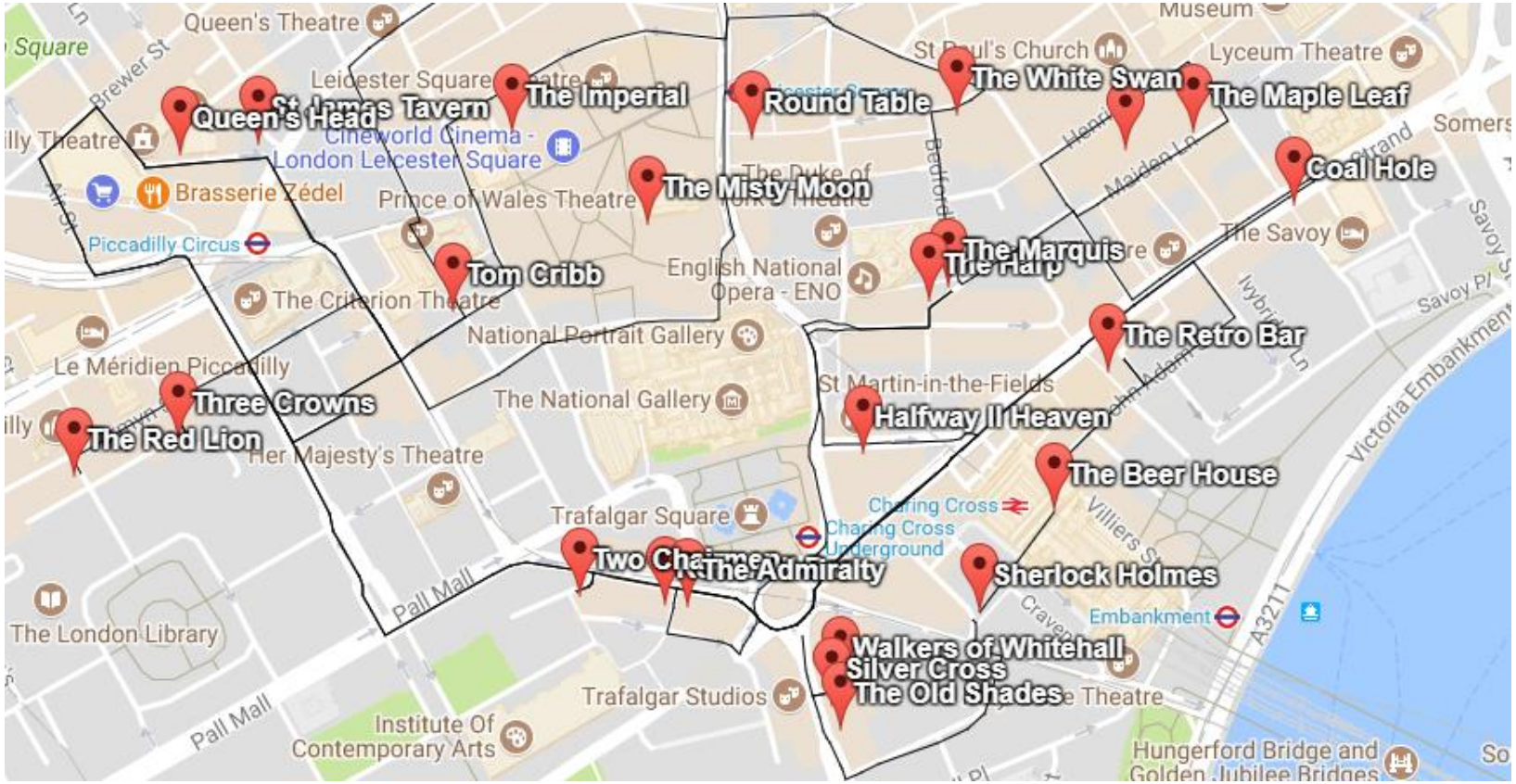
```
try{
kml.saveKMLDocument("london1.kml");
}catch(Exception e){
e.printStackTrace();
System.out.println("Error saving KML");
}
//Create a schedule of calendar events
DiaryFacade schedule = new DiaryFacade();
schedule.deleteAllEvents();//Remove any previous events loaded from file

try{
int count = 0;
LocalDateTime myTime = LocalDateTime.of(2017, 6, 8, 11, 00);

for (Journey j : trip.getLegs()){
String description = "Visit " + count;
if (count==0) description = "Start";
LocalDateTime leave = myTime.plusMinutes(15);
CalendarEntry ce = schedule.createCalendarEntry(myTime, leave, description, j.getPointA().getDescription(), j.getPointA(), null);
j.putAttribute(Journey.FROM_CALENDAR_ENTRY, ce);
int s = (int)j.getTravelTimeMS()/1000;
myTime = leave.plusSeconds(s);
count++;
}
}catch(Exception e){
System.out.println(e);
}
```

Example

```
try{
        kml.saveKMLDocument("london1.kml");
}catch(Exception e){
        e.printStackTrace();
        System.out.println("Error saving KML");
}
```



Example

```
//Create a schedule of calendar events
DiaryFacade schedule = new DiaryFacade();
schedule.deleteAllEvents();//Remove any previous events loaded from file
try{
        int count = 0;
        LocalDateTime myTime = LocalDateTime.of(2017, 6, 8, 11, 00);
        for (Journey j : trip.getLegs()){
                String description = "Visit " + count;
                if (count==0) description = "Start";
                        LocalDateTime leave = myTime.plusMinutes(15);
                CalendarEntry ce = schedule.createCalendarEntry(myTime, leave, description, j.getPointA().getDescription(), j.getPointA(),
null);
                j.putAttribute(Journey.FROM_CALENDAR_ENTRY, ce);
                int s = (int)j.getTravelTimeMS()/1000;
                myTime = leave.plusSeconds(s);
                count++;
        }
}catch(Exception e){
        System.out.println(e);
}
```

Example

```
System.out.println("My Trip \n" );
schedule.printAllCalendarEntries();

//Show directions
for (Journey j : trip.getLegs()){
        System.out.println("Appointment " + j.getAttribute(Journey.FROM_CALENDAR_ENTRY));
        System.out.println("Journey:  From " + j.getPointA().getDescription());
        System.out.println("To " + j.getPointB().getDescription());
        System.out.println("Instructions : " + j.getAttribute(Journey.INSTRUCTIONS));
        System.out.println("Journey data source : " + j.getSource());
}
```

Get involved

- https://github.com/NeilUrquhart/NapierGeowareTools
- Use it…. Contribute to it…..
- Give some feedback

- More documentation to follow
  - Feel free to contact n.Urquhart@napier.ac.uk for support

Future Work

- More JourneyFactory classes for useful APIs
- Possibly integrate with a Javascript library such as Leaflet
- Much depends on
  - What I need to support my own research
  - A supply of students (and others) who are willing to add to the library