Informatics Large Practical

Michael Glienecke and Stephen Gilmore
School of Informatics, University of Edinburgh

Document version 2.2.0

About

The Informatics Large Practical is a 20 point Level 9 course which is available for Year 3 undergraduate students on Informatics degrees. It is not available to visiting undergraduate students or students in Year 4 or Year 5 of their undergraduate studies. It is not available to postgraduate students. Year 4, Year 5 and postgraduate students have other practical courses which are provided for them.

Scope

The Informatics Large Practical is an individual practical exercise which consists of one large design and implementation project, with two coursework submissions. Coursework 1 involves creating a new project and implementing some fundamental components of the project. Coursework 2 is the implementation of the entire project together with a report on the implementation. Please note that the two coursework are not equally weighted. There is no exam paper for the Informatics Large Practical so to calculate your final mark out of 100 for the practical just add together your marks for the coursework.

Coursework	Out of	Weight	Deadline
Coursework 1	25	25%	16:00 on Friday 14th October 2022 *or 16:00 on Friday 21st October 2022 if given seven days extension*
Coursework 2	75	75%	16:00 on Friday 2nd December 2022 *or 16:00 on Friday 9th December 2022 if given seven days extension*

These deadlines are interpreted strictly and penalties are applied automatically to any work which is submitted late. You are advised to submit at least 30 minutes before the deadline in order to avoid problems with uploading your work.

—⋄—

It is essential that you take these deadlines absolutely seriously. The School of Informatics policy requires the course lecturers to record a mark for 0% which is submitted more than seven days late.

Contents

List of Figures

Chapter 0

The Coursework Specification

0.1 Introduction

Have you ever been involved in an all-night hackathon or have been pulling an all-nighter to get super-tough practicals like this one finished on time? If so you will know that there comes a time late at night when nothing else but a pizza will keep you going to get all your work done! Well, not to worry, the School of Informatics is considering creating a service called *PizzaDronz* where students can order a pizza by an app and have it delivered directly by drone to the top of the Appleton Tower where they can collect it and eat it while taking a break from the keyboard. Your midnight feast worries should be a thing of the past when the service launches on 1st January 2023! Of course, pizzas make a great lunchtime snack to share with friends so the service will operate all day, not just during the hours of darkness.

— ◇ —

Having pizzas delivered by drone will minimise the time that busy Informatics students need to spend queueing to buy lunch or dinner and will also speed delivery, because, unlike delivery by car or bike, drones do not need to follow the road layout, stop at red traffic lights, and so forth. The idea is also good for the environment. Fewer deliveries by car means less exhaust pollution generated, and cleaner air for Edinburgh. In addition, the service could be helpful to new students who have just joined the School and have not yet got their bearings and do not know the great pizza restaurants near the University's Central Area.

— ◇ —

The idea for a drone-based food delivery service has some issues, especially when delivering hot food. We will assume that the drone will fly high enough that it will not crash into even very tall buildings such as The David Hume Tower. However, Edinburgh's many seagulls might think that someone has kindly sent them a delicious meal and attack the drone in order to be able to get at the contents inside. In addition, software and hardware errors do happen so we must route the drone as much as possible away from populated areas such as George Square Gardens and Bristo Square, among others. Ideally, the drone should usually be flying over the roofs of buildings. This is for the *safety* reason just mentioned (we don't want to drop hot food or metal drones onto unsuspecting students studying in the sunshine in George Square Gardens). An additional issue is *privacy*. Many drones are fitted with cameras and some students might not like the idea that they might be photographed by a drone flying overhead. If the drone is flying over the roofs of buildings then a more innocent explanation might occur such as the University is surveying the roofs of the buildings and looking for cracked roof tiles or leaks in a roof. In fact, the drone does not have a camera fitted but privacy is important and we don't want to give students unnecessary worries about their privacy. For this reason, student-populated areas will be designated as "no-fly zones". The drone will therefore have to plan its routes so that it does not fly over the no-fly zones.

— ◇ —

Ordering the pizzas is relatively straightforward, but not completely straightforward. The School of Informatics is developing an online system to take pizza orders and add these to a database of orders to be delivered[1]. What is less clear is whether or not it is feasible for the drone to fulfil these orders, given that (i) the service is expected to be popular with a lot of pizza orders being placed each day, (ii) only one drone is available for making the deliveries, (iii) the drone cannot carry more than one order (of a maximum of four pizzas) at a time (to avoid delivering the wrong order to the wrong person, for example a non-vegetarian pizza to a vegetarian), (iv) the drone must avoid populated areas in the no-fly zone, (v) the drone can only fly for a limited time before its battery will run out and it will need to be recharged, and specifically, (vi) the drone can carry between one and four pizzas[2]. We will be interested in how many pizza orders can be delivered before the battery needs to be recharged.

— ⋄ —

Your task is to devise and implement an algorithm to control the flight of the drone as it makes its deliveries while respecting the constraints on drone movement specified in this document. You will be provided with some synthetic test data representing typical pizza orders and other data about the service such as the details of the pizza restaurants which are participating in the scheme, the menus for these shops, and the location of the drop-off point on top of the Appleton Tower. This information will come in the form of a REST-service running on a server, which returns the data in JSON-format[3] *(more detailed information and an example will be provided).*
It is important to stress that the information in the test data which you will be given only represents the current best guess at what the elements of the drone service will be when it is operational, and the service in practice might use different shops or it might even deliver to different drop-off points (such as the top of the Informatics Forum). For this reason, your solution must be *data-driven.* That is, it must read the information from the REST-service and particular shops or particular drop-off points or other details must not be hard-coded in your application, except where it is explicitly stated in this document that it is acceptable to do so.

— ⋄ —

The *PizzaDronz* operators have a set of thermally-insulated boxes which are attached to the drones. When a full insulated box of pizzas lands on the top of the Appleton Tower it is swapped for an empty insulated box and sent off for its next delivery run. The insulated boxes are thoroughly sanitised between uses.

— ⋄ —

You should think that your software is being created with the intention of passing it on to a team of software developers and student volunteers in the School of Informatics who will maintain and develop it in the months and years ahead when the *PizzaDronz* delivery service is operational. For this reason, the *clarity and readability of your code is important*; you need to produce code which can be read and understood by others.

0.1.1 Latitudes and longitudes

In this practical we will be using latitudes and longitudes to identify locations on the map (such as pizza restaurants and the drop-off point on the roof of the Appleton Tower).

- ▷ Longitude is the measurement east or west of the prime meridian.
- ▷ Latitude is the measurement of distance north or south of the Equator.

[1]At present only an alpha version of the pizza ordering system is available, with some known shortcomings in validation which will be fixed in the final release, but at present we will have to deal with these using defensive programming.

[2]It is not possible to order zero or negative numbers of pizzas, or more than four pizzas, half-pizzas, single slices of pizza, or drinks or ice-cream or any other types of snacks. Pizzas are only available in a maximum of 14-inch size. It is not possible to order larger pizzas than 14-inch pizzas, but smaller diameter ones are OK.

[3]https://en.wikipedia.org/wiki/JSON

(The above are National Geographic definitions.) Latitudes and longitudes are measured in *degrees*, so we stay with this unit of measurement throughout all our calculations. Even when we are calculating the *distance* between two points we express this in degrees rather than metres or kilometres to avoid unnecessary conversions between one unit of measurement and another. As a convenient simplification in this practical, locations expressed using latitude and longitude are treated as though they were points on a plane, not points on the surface of a sphere. This simplification allows us to use Pythagorean distance as the measure of the distance between points. That is, the distance between (x_1, y_1) and (x_2, y_2) is just

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

— ⋄ —

In general, it will not be possible to manoeuvre the drone to a specified location exactly. Being *close to* the location will be sufficient, where ℓ_1 is *close to* ℓ_2 if the distance between ℓ_1 and ℓ_2 is strictly less than the *distance tolerance* of 0.00015 degrees.

— ⋄ —

When we write a location as a pair of coordinates in this document we will use the convention (*longitude*, *latitude*) because the language which we use for rendering maps puts longitude first and latitude second. In this project, longitudes will always be negative (~ -3) and latitudes will always be positive ($\sim +56$) because Edinburgh is located at (approximately) longitude 3 degrees West and latitude 56 degrees North.

0.1.2 The movement of the drone

The flight of the drone is subject to the following stipulations:

- the drone can make at most 2000 moves before it runs out of battery;
- the moves are of two types, the drone can either *fly* or *hover*—the drone can change its latitude and longitude when it flies, but not when it is hovering i.e. when it makes a hover move—flying and hovering use the same amount of energy;
- every move when flying is a straight line of length 0.00015 degrees[4];
- the drone *cannot fly in an arbitrary direction*: it can only fly in one of the 16 major compass directions as seen in Figure 1. These are the primary directions North, South, East and West, and the secondary directions between those of North East, North West, South East and South West, and the tertiary directions between those of North North East, East North East, and so forth. We use the convention that 0 means go East, 90 means go North, 180 means go West, and 270 means go South, with the secondary and tertiary compass directions representing the obvious directions between these four major compass directions. The convention that we use for angles simplifies the calculation of the next position of the drone.
- as the drone flies, it travels at a constant speed and consumes power at a constant rate;
- when the drone is hovering, we use the obvious junk value of the Enum object `null` value for the angle, to indicate that the angle does not play a role in determining the next latitude and longitude of the drone;
- the drone *must hover for one move* when collecting a pizza order from a restaurant, and do the same when delivering pizzas to the roof of the Appleton Tower;
- the drone is launched each day from the top of the Appleton Tower at location $(-3.186874, 55.944494)$ and should return *close to* this location before running out of battery energy.

[4] Because of unavoidable rounding errors in calculations with double-precision numbers these moves may be fractionally more or less than 0.00015 degrees. Differences of $\pm 10^{-12}$ degrees are acceptable. Double-precision numbers must be used to represent quantities measured in degrees because of the need for accuracy in specifying locations.

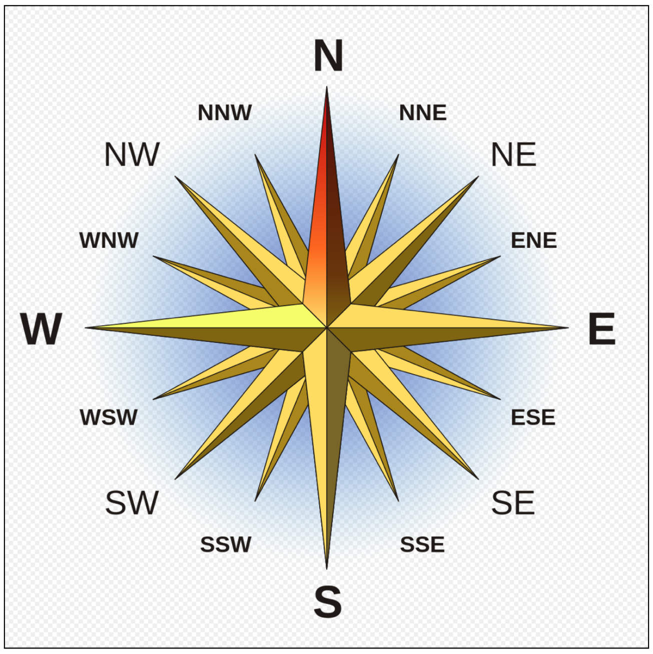


Figure 1: The 16-point compass rose. Original SVG image `https://commons.wikimedia.org/w/index.php?curid=2249878`

0.1.3 The University of Edinburgh Central Area

The University of Edinburgh Central Area is defined to be all locations which have a latitude which lies between 55.942617 and 55.946233. They also have a longitude which lies between −3.184319 and −3.192473. Outside organisations should have litttle objection to the drone being in this area because this is mostly University land containing University buildings. For this reason it is important for the drone to return to the Central Area once it has collected the pizza(s) *in as few moves as possible.* The Central Area is illustrated in Figure 2.

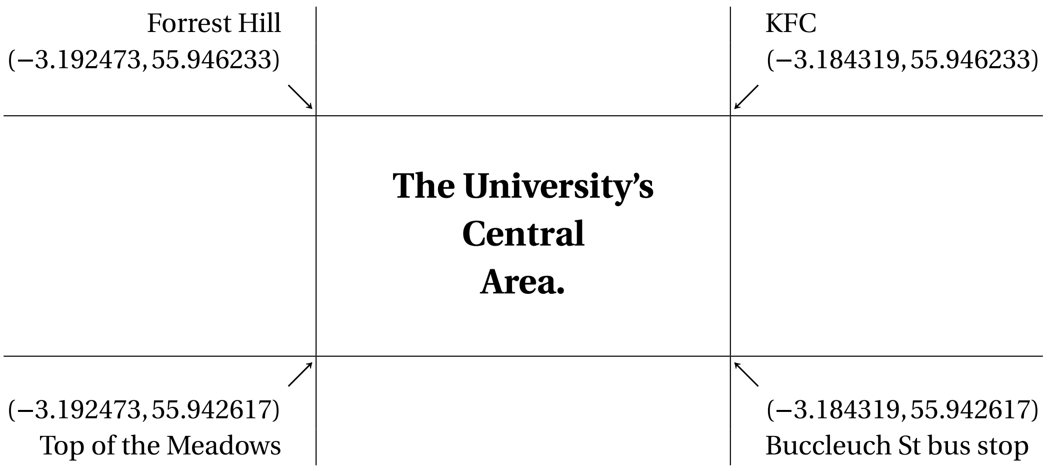


Figure 2: The University of Edinburgh Central Area

The above constants are merely used for representational purpose. To provide a potential for future enlargement the actual central area has to be retrieved by the application from the REST service using the **centralArea** endpoint (more details are provided later).

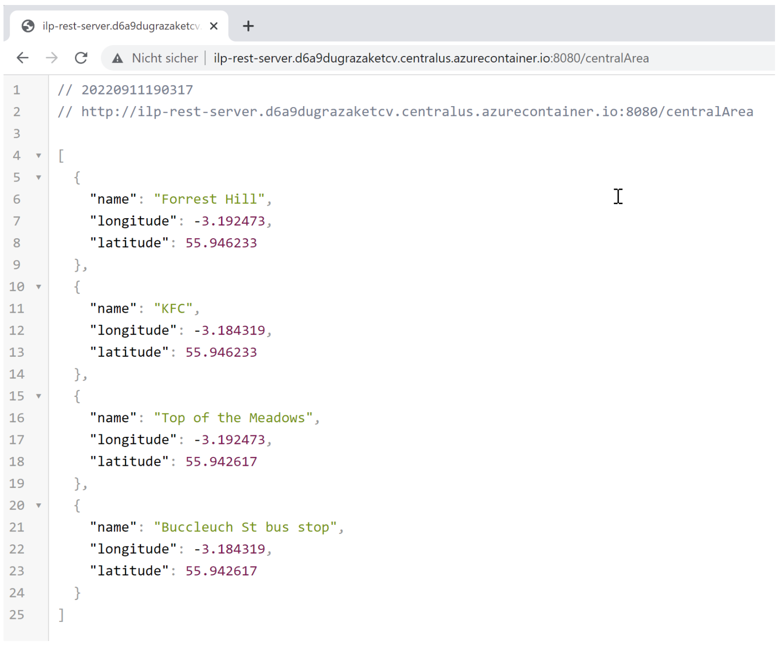
Once the drone has entered the Central Area, *it cannot leave it again* until it has delivered the ordered pizzas to the roof of the Appleton Tower. This rule prevents invalid solutions which attempt to avoid the no-fly zones by leaving the Central Area and returning into it just beside Appleton Tower.

0.1.4 The REST-Server: Dynamic and static content

All dynamic information which the service needs to function is provided by a newly developed centralized REST-server. This server (which actually could be a single machine or a cluster to provide a more secure platform for later business growth) makes the whole system more dynamic and your life much easier as it serves as a central point of intelligence.
The REST-service not only provides a REST-API to use (for the dynamic data), but serves static content (files) as well.
All data needed by the *PizzaDronz* service is coming from the REST-service and has to be retrieved every time the *PizzaDronz* service is started to make sure the latest information is processed. This is very important as otherwise data changes might not be reflected properly.

—⋄—

The current URL to reach the REST-server is: **https://ilp-rest.azurewebsites.net** (hosted on Azure as a docker instance running the Java REST-server)
To test the REST-server you can use your browser and just type the base-URL and then either the filename requested (for static resources) or the REST-endpoint (for dynamic resources). The result of the operation is either a file download (for a static resource) or the display of a JSON data structure (the dynamic data).
This could look as in Figure 3:


ilp-rest-server.d6a9dugrazaketcv
Nicht sicher | ilp-rest-server.d6a9dugrazaketcv.centralus.azurecontainer.io:8080/centralArea

```
// 20220911190317
// http://ilp-rest-server.d6a9dugrazaketcv.centralus.azurecontainer.io:8080/centralArea

[
  {
    "name": "Forrest Hill",
    "longitude": -3.192473,
    "latitude": 55.946233
  },
  {
    "name": "KFC",
    "longitude": -3.184319,
    "latitude": 55.946233
  },
  {
    "name": "Top of the Meadows",
    "longitude": -3.192473,
    "latitude": 55.942617
  },
  {
    "name": "Buccleuch St bus stop",
    "longitude": -3.184319,
    "latitude": 55.942617
  }
]
```

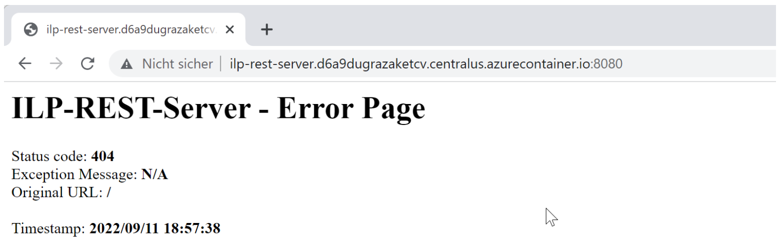
Figure 3: Showing the central area data for the REST-request for *centralArea*

Should an error occur (either by accessing an invalid resource *URL* or a server-side problem) an error display similar to Figure 4 is shown

—⋄—

These URLs could look like:

- https://ilp-rest.azurewebsites.net/test


ilp-rest-server.d6a9dugrazaketcv
Nicht sicher | ilp-rest-server.d6a9dugrazaketcv.centralus.azurecontainer.io:8080

ILP-REST-Server - Error Page

Status code: **404**
Exception Message: **N/A**
Original URL: /

Timestamp: **2022/09/11 18:57:38**

Figure 4: Showing an error as an access was attempted with a resource being specified which results in a HTTP code **404** - Resource not found

- https://ilp-rest.azurewebsites.net/test/echoInput
- https://ilp-rest.azurewebsites.net/orders
- https://ilp-rest.azurewebsites.net/orders/YYYY-MM-DD[5]
- https://ilp-rest.azurewebsites.net/bounding-box.geojson

— ⋄ —

Static files being served

- `all.geojson` - shows all visual elements (bounding box, no-fly-zones and restaurants)
- `bounding-box.geojson` - defines the bounding box
- `no-fly-zones.geojson`
  Within the Central area are four regions where the drone is not allowed to fly; a drone entering these regions will be considered to be malfunctioning. These four regions are known as the *no-fly zones* for the drone. The details are in the file `geojson/no-fly-zones.geojson`. This file is in GeoJSON format, which is a standard way of encoding geographic data structures and map information. For more details on GeoJSON visit `https://geojson.org`. To render GeoJSON maps, visit `https://geojson.io`.

  In the test data there are four no-fly zones of populated areas where the drone is not allowed to fly into or fly over. These cover the George Square Area; the Dr Elsie Inglis Quadrangle; the Bristo Square Open Area; and the Bayes Central Area. The drone cannot move into any of the no-fly zones with a move, and it cannot *pass through* any no-fly zones in moving from one point to the next.

  The no-fly zones are illustrated in Figure 5, together with the limits of the Central area.
- `restaurants.geojson` - shows all participating restaurants

These files are only used for your reference and testing and **must not** be used as a data source for the *PizzaDronz* service. Any necessary data for the *PizzaDronz* service has to be retrieved dynamically from the REST-service (for details see REST-Service Endpoints for data).

REST-Service Endpoints for data

- `test` - can be used to test the service.
  Either simply the default result of the `test` endpoint is displayed, or by passing an echo-string after

[5]retrieve all orders for a specific date in exactly the format which is passed. If no records are found an empty JSON array is returned

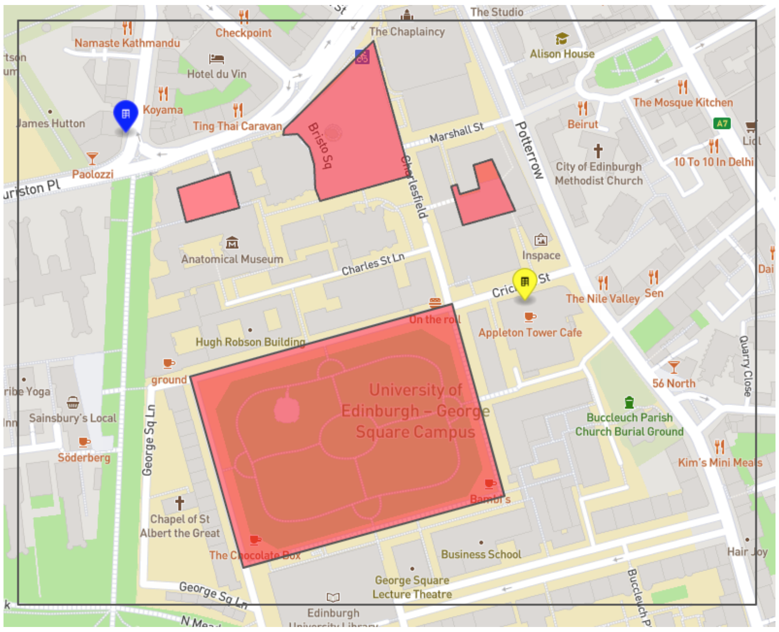


Figure 5: A GeoJSON map of the University's Central Area rendered by the website http://geojson.io/. The map shows the Central area (the outer grey rectangle), the four no-fly zones (the semi-transparent red polygons), the landing point on the top of Appleton Tower (in yellow) and in the upper left corner the one pizza restaurant which is inside the Central Area (*Civerinos Slice*, with the blue teardrop-shaped marker).

the endpoint (like `test/myEchoString`) the calculated result from the service (which includes the echo string passed in the request)

- `centralArea` - returns the central area corner points (currently 4)
- `noFlyZones` - lists all defined no-fly zones as a vector of objects with a name and polygon coordinates
- `restaurants` - lists all defined restaurants with their coordinates and the pizzas on offer (including a price for each pizza)
- `orders` - lists all pending orders in the system (valid and invalid) and will be used as a data source to calculate the necessary flight tracks for each day
- `orders/YYYY-MM-DD` - lists all pending orders **for a specific date** in the system (valid and invalid) and will be used as a data source to calculate the necessary flight tracks for each day

0.1.5 Accessing the REST-Server

To make life a bit easier for you, enclosed are some code samples which show how to access the REST-Server for dynamic data structures (JSON-format) or static content (files).

Retrieving JSON-data

Below is a simple client application which uses the */test* endpoint and passes the supplied data to server and reads the response.

```
package uk.ac.ed.inf;

import com.fasterxml.jackson.databind.ObjectMapper;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
```

```
/**
 * A very simple client to GET JSON data from a remote server
 */
public class TestClient
{
    public static void main(String[] args )
    {
        if (args.length != 2){
            System.err.println("Testclient␣Base-URL␣Echo-Parameter");
            System.err.println("you␣must␣supply␣the␣base␣address␣of␣the␣ILP␣REST␣Service\n" +
                    "␣e.g.␣http://restservice.somewhere␣and␣a␣string␣to␣be␣echoed");
            System.exit(1);
        }

        try {
            String baseUrl = args[0];
            String echoBasis = args[1];

            if (! baseUrl.endsWith("/")){
                baseUrl += "/";
            }

            // we call the test endpoint and pass in some test data which will be echoed
            URL url = new URL(baseUrl + "test/" + echoBasis);

            /**
             * the Jackson JSON library provides helper methods which can directly
             * take a URL, perform the GET request convert the result to the specified class
             */
            TestResponse response = new ObjectMapper().readValue(
                    new URL(baseUrl + "test/" + echoBasis), TestResponse.class);

            /**
             * some error checking - only needed for the sample (if the JSON data is
             * not correct usually an exception is thrown)
             */
            if (! response.greeting.endsWith(echoBasis)){
                throw new RuntimeException("wrong␣echo␣returned");
            }

            System.out.println("The␣server␣responded␣as␣JSON-greeting:␣\n\n"
                    + response.greeting);

        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

JSON-data wrapping (using de/serialization)

The data returned is just a sequence of characters and by using *Jackson*[6] this is converted to a class instance[7].
To make this possible the data class which receives the JSON data has to be created and annotated accordingly. In the sample this has been done with *TestResponse*

```
package uk.ac.ed.inf;

import com.fasterxml.jackson.annotation.JsonProperty;

public class TestResponse {
    @JsonProperty("greeting")
    public String greeting;
}
```

Retrieving static files

Below is a simple client application which downloads the file from the specified address and stores it under the same name

```
package uk.ac.ed.inf;

import java.io.BufferedInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;

/**
 * Simple download application to retrieve a file from the REST server
 *
 */
public class Download
{
    public static void main( String[] args )
    {
        if (args.length != 2){
            System.err.println("Download␣Base–URL␣Filename");
            System.err.println("you␣must␣supply␣the␣base␣address␣of␣the␣ILP␣REST␣Service" +
                    "␣e.g.␣http://restservice.somewhere␣and␣a␣filename␣to␣be␣loaded");
            System.exit(1);
        }

        URL finalUrl = null;
        String baseUrl = args[0];
```

[6]For detailed information please visit: https://stackabuse.com/definitive-guide-to-jackson-objectmapper-serialize-and-deserialize-java-objects/

[7]This process of converting the textual data to objects is called deserialization

```
        String filenameToLoad = args[1];
        if (! baseUrl.endsWith("/")){
            baseUrl += "/";
        }

        try {
            finalUrl = new URL(baseUrl + filenameToLoad);
        } catch (MalformedURLException e) {
            System.err.println("URL␣is␣invalid:␣" + baseUrl + filenameToLoad);
            System.exit(2);
        }

        try (BufferedInputStream in = new BufferedInputStream(finalUrl.openStream());
             FileOutputStream fileOutputStream =
                     new FileOutputStream(filenameToLoad, false)) {
            byte[] dataBuffer = new byte[4096];
            int bytesRead;
            while ((bytesRead = in.read(dataBuffer, 0, 1024)) != -1) {
                fileOutputStream.write(dataBuffer, 0, bytesRead);
            }

            System.out.println("File␣was␣written:␣" + filenameToLoad);
        } catch (IOException e) {
            System.err.format("Error␣loading␣file:␣%s␣from␣%s␣->␣%s",
                    filenameToLoad, finalUrl, e);
        }
    }
}
```

0.1.6 The makeup of an order

We haven't said much so far about the nature of a pizza order so let's discuss that now. As you might imagine, there is a limit on the weight that the drone can lift. The maximum number of items in an order has been fixed so that the drone will always be able to lift the order, even if it consists of the heaviest pizzas which can be ordered by the drone service. There are other constraints also, as listed below.

1. An order can have a minimum of one pizza, and a maximum of four.
2. Every order is subject to a fixed delivery charge, which is £1.

The restaurants which participate in the service are notified when the drone will arrive; they start cooking the pizza(s) and then when the drone arrives they place the pizzas in an insulated box, and fix the box to the drone when it is hovering close to the location of the restaurant/pizza shop. We imagine that the insulated box is hanging down from the drone so that the drone is always hovering some safe height above the user's head.

—⋄—

The box can contain pizzas up to 14 inches in diameter, but not larger than this. None of the pizzas returned by the REST-request to `restaurants` are larger than this.

—⋄—

We will not be very concerned here with the system which sends web or text message order notifications to the shops; the architects of the drone service already have a system in place for this. However, due to a misunderstanding between the web front end back-end team, each thought that the other was responsible for validating the data entered by the customer and as a result neither team has implemented this. This means that there will be invalid orders in the REST-responses from `orders` which you must detect and filter out and not attempt to deliver. We will give examples later.

0.1.7 The structure of the orders

The information about each day's orders is returned in the REST-request to `orders` and is delivered as an array of JSON-elements like (each one representing one order):

```
{
    "orderNo": "1234",
    "orderDate": "2023-03-01",
    "customer": "Michael Glienecke",
    "creditCardNumber": "12342222",
    "creditCardExpiry": "05/24",
    "cvv": "123",
    "priceTotalInPence": 1400,
    "orderItems": [
      "Pizza1",
      "Pizza2",
      "Pizza3"
    ]
  }
```

—◇—

From this we learn that order number 1234 was for three items. All items in an order *must* come from the same pizza restaurant and are valid pizzas[8]

—◇—

These orders are returned for the entire two-year period beginning on 2023-01-01 and ending on 2023-05-31 (every day has orders) and contain invalid orders as well.
So, you have to check card number, expiration, order date and items among other details (as described in this document) as well.
Your task will be to create a Java object (named `Order`) for these orders where the corresponding data types for members can be derived from the JSON format. For example, orderNo is provided as a string (using ""), which implies that member `orderNo` in class `Order` is of type `String` as well.

0.1.8 The outcome of an order

Because some orders may be invalid the drone must not deliver these orders. We will use the following `enum` to classify order outcomes

```
public enum OrderOutcome {
    Delivered,
    InvalidCardNumber,
    InvalidExpiryDate,
    InvalidCvv,
    InvalidTotal,
    InvalidPizzaNotDefined,
```

[8]The restaurants and their pizzas can be retrieved using the `restaurants` REST endpoint

```
    InvalidPizzaCount,
    InvalidPizzaCombinationMultipleSuppliers,
    Invalid
}
```

0.1.9 Files to be used during testing

In addition to the dynamic data returned from the various JSON requests while your application is running, you can retrieve some GeoJSON files[9] which have been prepared for you to use when you are *testing* your application.

— ⋄ —

When the *PizzaDronz* service is operational the server content will be kept up-to-date but the GeoJSON files prepared for testing purposes will not; they only relate to the synthetic data that you are given to test your application, not the lunch orders received each day when the service is operational. This means that your application should not read these GeoJSON files, you should only load them on the http://geojson.io/ website when checking the flightpath that you have generated for the airborne drone.

— ⋄ —

Unlike, for example, the GeoJSON file of the no-fly zones, the content of the database is not held in a graphical format so the purpose of the GeoJSON test files is to allow us to produce a visualisation of the information in the database, to help with understanding the important locations which are in the synthetic data. One file, `all.geojson`, includes representations of all of the locations of interest in the drone Central area. This is shown in Figure 6. This file will make a convenient background when rendering your drone flightpath.

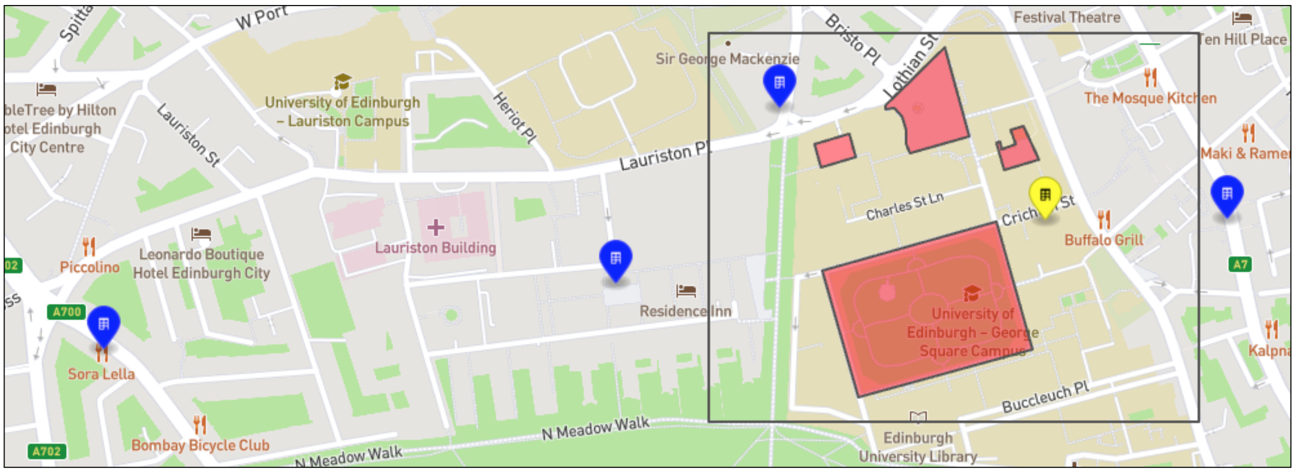


Figure 6: The contents of the file `all.geojson` rendered by the website http://geojson.io/. This file contains all of the features that we have seen in Figure 5 plus the initial location of the drone (the yellow placemarker, on top of Appleton Tower), and the four pizza restaurants which are participating in the scheme according to the website content `restaurants.geojson` (the blue placemarkers, with a building symbol). The semi-transparent red polygons are the no-fly zones.

0.1.10 An illegal flight path

Once back inside the Central Area, the drone must not leave again until it has delivered the pizzas to the roof of Appleton Tower. The flightpath shown in Figure 7 from Sora Lella restaurant to Appleton Tower is completely illegal, as well as being sub-optimal in taking more moves than necessary.

[9] see above for details or just download the files using the browser and keep them locally

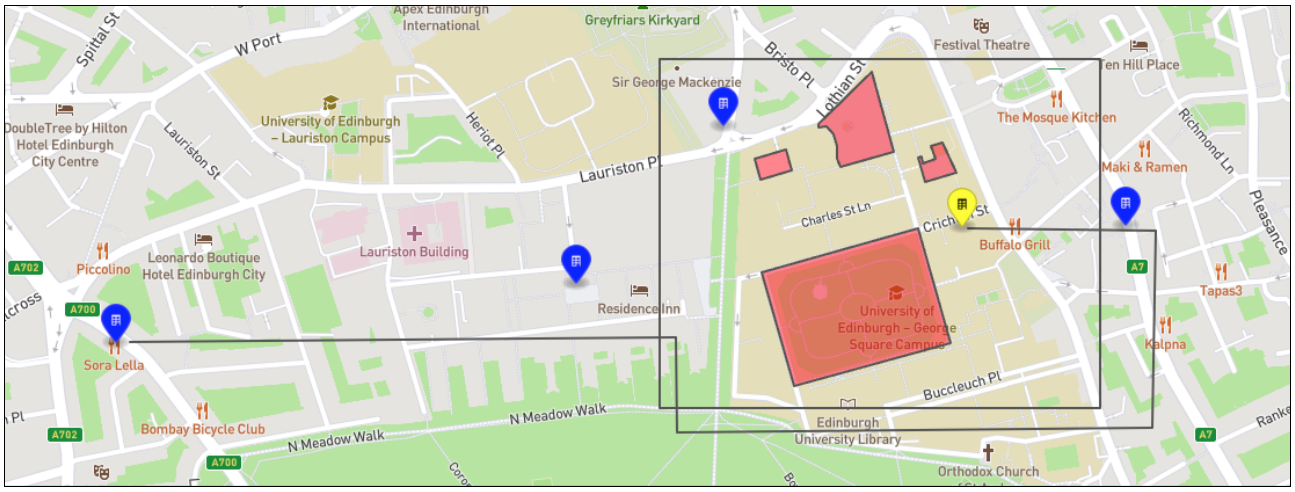


Figure 7: An illegal flightpath which leaves the Central Area again after having entered it.

0.1.11 The participating restaurants

The restaurants participating in the trials of the *PizzaDronz* service can be retrieved using the `restaurants` REST request, which returns coordinates as well as the menu they offer.

0.1.12 The runtime of your code

Your application to plan and plot the flightpath of the drone should aim to have a runtime of *60 seconds or less*. You need to bear this restricted runtime in mind when designing the algorithm that you will use to generate the flightpath of the drone.

—⋄—

Runtimes which are much longer than 60 seconds, for example an average of 10 or 20 minutes, would obviously be problematical because they are delaying the launch of the drone considerably. Delays in the collection and delivery of their pizza(s) are sure to be unpopular both with the restaurants and with hungry students so an algorithm with a runtime of 10 or 20 minutes would be quite unsatisfactory.

—⋄—

When the drone delivery service is operational the *über JAR* of your application[10] will be deployed on a server running Ubuntu 20.04 (focal) DICE or a similar version, as found on `student.compute.inf.ed.ac.uk`. The precise machine to host the service has not been purchased yet but its specifications will be similar to or better than the machine `student.compute.inf.ed.ac.uk`. A good way to check whether your application will be able to deliver the required level of performance in deployment would be to time the runtime of your *über JAR* on the machine `student.compute.inf.ed.ac.uk` when it is lightly-loaded[11].

0.1.13 Use of randomness

You may find it useful for your algorithm to make use of (controlled) randomness in your algorithm. This is fine, but there should only be a single source of pseudo-random numbers in the application, which *must* be initialised by a seed which is passed in via the Java command line.

[10] The über JAR is the relocatable compiled version of your code packaged together with all of the libraries that it depends on.
[11] For example, when the `who` command lists fewer than ten users using the machine.

0.1.14 Judging the viability of the service

It is accepted that the service may not be able to deliver every order every day depending on the number of orders placed. An important metric to be used in determining the viability of the drone delivery service will be the *sampled average number of pizza orders delivered by the service before the battery is exhausted*. You should have this metric in mind when developing your algorithm for controlling the drone.

—⋄—

The sampled average number is calculated by taking a random sample of days (say 7, 12, 24, or 31 days) and computing the average of the number of pizza orders delivered on each of those days. A small sample of days will give an approximate idea of the viability of the service; larger sample sizes will give a more accurate idea.

0.2 Files to be created in CW 2

As the main part of this practical exercise, you are to develop a *Java 18* application which when given a date (*passed in as a runtime argument to the application*) calculates a flightpath for the drone which delivers the pizza orders placed for that date as best it can before it returns close to its initial starting location. As previously stated, the number of moves made by the drone[12] should be 2000 or fewer.

—⋄—

The date format **YYYY-MM-DD** (which is actually the ISO 8061 format) has to be used in every file which is generated .

So e.g. flightpath-2023-04-13.json or drone-2023-04-13.geojson or deliveries-2023-04-13.json.

The format is always xxxx-**YYYY-MM-DD** where xxxx is the relevant file (drone, flightpath or deliveries)
Note: Please use hyphens, not underscores, in the filenames and use only lowercase letters. A filename of `drone-2023-04-15.geojson` is acceptable; a filename like `Drone_2023-04-15.GEOJSON` is not. Please note that your filename must use two digits for the day and the month; a filename of `drone-15-9-2023.geojson` is not acceptable because it does not match the pattern for the filename of `drone-`*`YYYY-MM-DD`*`.geojson`.

—⋄—

Your application should record the drone's behaviour by creating three local files in JSON-format to record information on the day's deliveries, which have to have the *.json* extension and have to be submitted with the application.

- The first file *(deliveries-YYYY-MM-DD.json)* records both the deliveries and non-deliveries made by the drone
- The second file *(flightpath-YYYY-MM-DD.json)* records the flightpath of the drone move-by-move
- The third file *(deliveries-YYYY-MM-DD.geojson)* is the drone's flightpath in GeoJSON-format

—⋄—

Assuming that your project is named `PizzaDronz` then when compiled with the Maven build system your Java application will produce an *über* JAR file in the `target` folder of your project which will be named `PizzaDronz-1.0-SNAPSHOT.jar`. If you run this JAR file with the command

```
java -jar target/PizzaDronz-1.0-SNAPSHOT.jar  2023-04-15
       https://ilp-rest.azurewebsites.net cabbage
```

it should read the lunch orders for the date 2023-04-15 from the REST service, together with the restaurants and menus, which is located at the provided base address. It should use the hashcode of the final word on the command line (the example here is "cabbage") as the seed to initialise the random-number generator (if used). Your application should check that the command line parameters are valid (in the sense of being a valid date and IP address[13]).

—⋄—

Your application may write any diagnostic messages that it likes to the standard output stream provided that the total size of these messages does not exceed 1Mb.

—⋄—

[12]Refer to Page 7 for the definition of a move.
[13]here you might simply catch the corresponding exception if there is an error

The results computed by your algorithm to control the drone will be written to three local files (one of them a GeoJSON file).

The outfile `deliveries-YYYY-MM-DD.json` In this file you have an array of JSON records, each with attributes for:

- `orderNo` — the eight-character hexadecimal string assigned to this order in the `orders` REST service endpoint;
- `outcome` — the `OrderOutcome` value for this order, as a string; and
- `costInPence` — the total cost of the order, including the standard £1 delivery charge.

Please ensure that you use exactly the attribute names `orderNo`, `outcome`, and `costInPence` for this file and that you write an array [] of JSON records, one entry for each processed order.

If the file already exists overwrite it, please.

The file `flightpath-YYYY-MM-DD.json` In this file you have an array of JSON records, each with attributes for a single move[14]:

- `orderNo` — the eight-character order number for the pizza order which the drone is currently collecting or delivering[15];
- double value `fromLongitude` — the longitude of the drone at the start of this move;
- double value `fromLatitude` — the latitude of the drone at the start of this move;
- int value `angle` — the angle of travel of the drone in this move, as a string[16];
- double value `toLongitude` — the longitude of the drone at the end of this move;
- double value `toLatitude` — the latitude of the drone at the end of this move and
- long `ticksSinceStartOfCalculation` — the elapsed ticks since the computation started for the day - every record will have a higher value than the previous one and records the duration this move calculation took

If the file already exists overwrite it, please.

Please ensure that you use exactly the attribute names `orderNo`, `fromLongitude`, `fromLatitude`, `angle`, `toLongitude`, `toLatitude` and `ticksSinceStartOfCalculation` for this file and that you write an array [] of JSON records, one entry for each processed order.

The output file `drone-YYYY-MM-DD.geojson` This file is generated in the current working directory when the application is run. It is a text file in GeoJSON format[17]. It should contain a FeatureCollection which consists of exactly one Feature. That Feature must be of type LineString. The LineString contains a list of coordinates which illustrate the flightpath of the drone. These coordinates should be approximately equal to the corresponding longitudes and latitudes stored in the `flightpath` table of the database. They will not be exactly equal because by default GeoJSON documents use single-precision floating-point values for longitudes and latitudes whereas the values stored in the database are double-precision. This means that we will take the values in the database to be the definitive flightpath of the drone with the values in the GeoJSON file providing a reasonable approximation for the purposes of visualisation.

— ⋄ —

[14]a detailed record of every move made by the drone while making the day's lunch deliveries is the final result **in the order the moves happened**

[15]The contents of this field should be the 8-character string "`no-order`" when the drone is making the flight back to the top of the Appleton Tower when all of the day's orders have been delivered.

[16]Refer to Page 7 of this document for the allowable values which this field can take.

[17]Please see `http://geojson.org` for details of the GeoJSON format.

When it is rendered by the website `http://geojson.io` along with the contents of the testing file `all.geojson`, your `drone-`*`YYYY-MM-DD`*`.geojson` file should produce a visualisation similar to that in Figure 8 with a grey line showing the flightpath of the drone. The shape of this line will depend on the date being plotted and on the drone control algorithm which you devise.

—⋄—

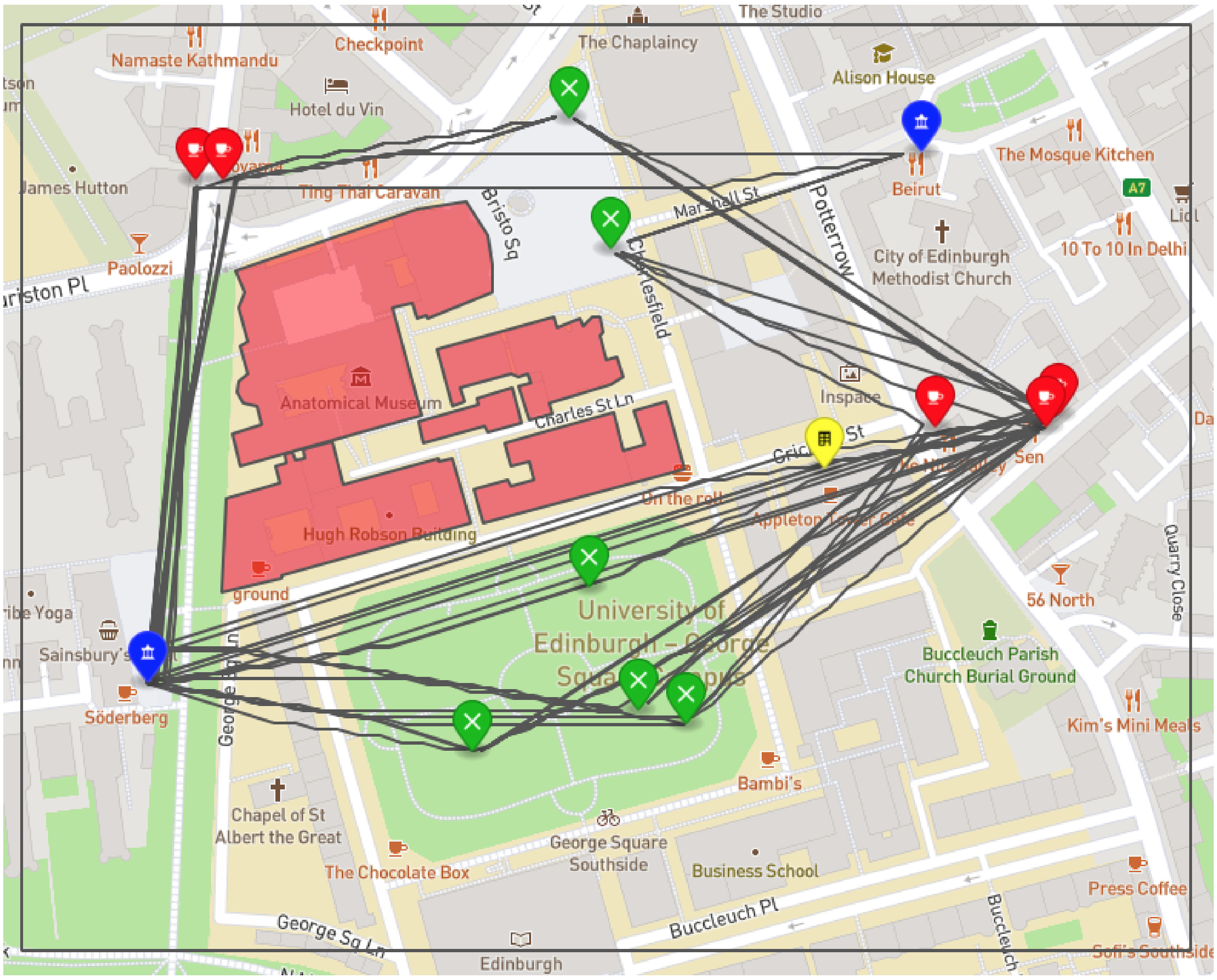


Figure 8: The contents of the file `all.geojson` (from Figure 6) overlaid with an output GeoJSON file with the drone's flightpath, rendered together by the website http://geojson.io/. The grey squiggly line connecting the initial location (the yellow marker), the pizza restaurants (the blue markers), and the final location (the initial location again) is a graphical representation of the flightpath of the drone which your algorithm will generate. Note that the drone may leave the Central Area (the outer rectangle) and it never enters the no-fly zones (the semi-transparent red polygons such as those over the Bristo Square and George Square gardens in the centre).

0.3 Development environment

We will provide support for the use of IntelliJ IDEA Community Edition as your development environment for this project. You may already be familiar with IntelliJ IDEA from the *Informatics 1 – Object-Oriented Programming* course. IntelliJ IDEA CE is available for download from `https://www.jetbrains.com/idea/`.

It is the free edition of the IntelliJ IDEA platform and supports the programming language and the build system which we use (Java and Maven respectively). Downloads of IntelliJ IDEA are available for Windows, macOS, and Linux. IntelliJ IDEA is pre-installed on DICE Ubuntu and is accessed via the command `ideaIC`.

0.4 Programming language: Java

The programming language to be used for your software is Java. The architects of the drone delivery service have chosen Java version 18 as the version of Java which will be used throughout the project. You should ensure that the code which you submit can be compiled and run on a Java 18 installation. This is essential because the service will use the `jwebserver` application which was added in Java version 18. Earlier versions of Java cannot be used to complete this practical.

— ◇ —

This version of Java has been selected because it provides local variable type inference and streams, and other helpful features such as records and pattern matching. You are expected to use these Java features in order to have an up-to-date implementation. One of the course lectures will be devoted to explaining how and when to use these new Java language features.

— ◇ —

Java has been chosen as the development language for the service because the specifics of the hardware which will be purchased to run the service are not yet known so it is important to chose a language which is portable between different operating systems. Any libraries which you include in your project must similarly be implemented in Java for maximum portability. You must check this by finding the source code of the libraries that you use.

— ◇ —

If you plan to work on your own machine instead of DICE you must install Java 18. This version of Java is available for download from `https://jdk.java.net/18/`. You are downloading the OpenJDK JDK 18.0.1.1 General-Availability Release.

— ◇ —

We expect you to be already familiar with Java. If you are not, or if you would benefit from a refresher on Java, we recommend the textbook *Java Precisely* by Peter Sestoft, third edition published by MIT Press in 2016, as providing a concise and clear introduction to Java.[18]

0.4.1 New features available in Java 18

The following features did not exist or were not used in the version of Java that you were taught with in the *Informatics 1: Object-Oriented Programming* course two years ago. If your code is to fit well with the other code being written for the project (which is also in Java 18) then you will need to use these features or your code will need to be re-written by another member of the development team, which is obviously wasteful and pointless.

Streams

In Year 1, there were two separate courses, *Informatics 1: Functional Programming* and *Informatics 1: Object-Oriented Programming*. The OOP course avoided using streams because these are a functional programming concept and so were rightly the province of the FP course. We're past that now so we need to learn how to use streams in Java. Read `https://www.baeldung.com/java-8-streams` and/or watch `https://youtu.be/t1-YZ6bF-g0` or one of the many other tutorials on the Web about streams in Java.

[18]Many Java textbooks are available so if you cannot get a copy of Java Precisely please feel free to choose another textbook. Alternatively, many tutorials on Java are available online, including the (slightly dated but still very useful) Java Tutorial from Oracle at `https://docs.oracle.com/javase/tutorial/`. The Java real-eval-print loop `jshell` can also provide a useful refresher.

Local variable type inference

Java has a form of type inference for local variables in methods which allows the type of an object to be *inferred* which means that rather than writing

```
java.util.GregorianCalendar gc = new java.util.GregorianCalendar();
```

in the body of a method you can instead write

```
var gc = new java.util.GregorianCalendar();
```

and save yourself some time and some unnecessary syntactic clutter. The two lines of Java code above have exactly the same meaning but the second is obviously shorter and clearer.

Records

Java now includes records which are data classes which behave like final objects, containing immutable data. A record class is declared like this where the "..." represents methods of the class.

```
public record LngLat(double lng, double lat){ ... };
```

The following code produces the output below.

```
var appletonTower = new LngLat(-3.186874, 55.944494);
System.out.println(appletonTower);
```

```
LngLat[lng=-3.186874, lat=55.944494]
```

Note that we did not have to write the `toString` method for the `LngLat` record. This, and other convenience methods such as `equals` are provided for us. A full explanation is available from Lead Java Software Engineer John Marty at `https://youtu.be/gJ9DYC-jswo`.

Pattern matching

Java 18 introduced quite complex but useful pattern matching. An example use is below. This is not a typical use of pattern-matching, but just a compact example to show the syntax.

```
public record Date(int day, int month, int year){};
   ...
var today = new Date(15, 9, 2022);
Object o = (Object) today;
if (o instanceof Date d){
   System.out.println("Today is " + d.day() + "/" + d.month() + "/" + d.year());
}
```

This produces as output *Today is 15/9/2022.* A fuller explanation of pattern matching is available at `https://docs.oracle.com/en/java/javase/18/language/pattern-matching.html`.

0.4.2 Documentation for your code

Your code should contain documentation in JavaDoc syntax. This is a Java comment syntax which can be used to generate human-readable documentation in the form of HTML pages. A description of JavaDoc is online at `https://www.oracle.com/uk/technical-resources/articles/java/javadoc-tool.html`. The HTML format of your JavaDoc can be generated in IntelliJ IDEA using Tools → Generate JavaDoc ....

0.5 Project management

The build system to be used for your project is Apache Maven, a Java-based build system which manages all of the project dependencies in terms of Java libraries which you use, and enables you to build your system into a single self-contained JAR file (the *über JAR*) for deployment. This JAR file has all of your code and all of the libraries that you use together in one place. IntelliJ IDEA CE comes with Maven installed so you do not need to download Maven separately.

— ◇ —

The course lectures will explain use of the Maven build system; we do not expect you to be already familiar with Maven.

0.5.1 Using third-party software and libraries

This practical exercise allows you to use free software, but not commercial software which you must pay for or license. One free software development kit (SDK) which you should find useful is the Mapbox Java SDK which provides classes and methods for parsing and generating GeoJSON maps. Instructions for adding the Mapbox Java SDK to your project are available at `https://docs.mapbox.com/android/java/overview/`.

Chapter 1

Informatics Large Practical: Coursework One

Michael Glienecke and Stephen Gilmore
School of Informatics, University of Edinburgh

1.1 Introduction

This coursework and the second coursework of the ILP are for credit; weighted 25%:75% respectively. Please now read Appendix A for information on good scholarly practice and the School's late submission policy.

— ⋄ —

In this project you are creating a Java application which is built using the Maven build system. We will begin by using IntelliJ IDEA to create the project structure.

1.2 Getting started

If you are working on your own laptop you should begin by downloading IntelliJ IDEA, if you do not already have it. Download it from `https://www.jetbrains.com/idea/download/`. On DICE, IntelliJ IDEA is available via the `ideaIC` command.

— ⋄ —

Next, create a new Maven project in IntelliJ IDEA by choosing File → New → Project . . . , and choosing Maven Project as the option. If you have downloaded Java 18 but not yet used it in IntelliJ then use the Project SDK dropdown and choose Add JDK . . . to add it now. This will set Java 18 as the value for Project SDK.

— ⋄ —

Check the option "Create from archetype . . . " and choose org.apache.maven.archetypes:maven-archetype-quickstart. (There will be other archetypes in the list named quickstart; be sure to get the one which has the prefix org.apache.maven.archetypes.)

— ⋄ —

On the next page, edit the Artifact Coordinates and fill in the options as shown below:

Group Id:	`uk.ac.ed.inf`
Artifact Id:	`PizzaDronz`
Version:	`1.0-SNAPSHOT`

On the next page leave the values as they are and click “Finish". Your project will be created.

— ◇ —

You should now have a working Maven project structure. Note that there are separate folders for project source and project tests. Note that there is an XML document named `pom.xml` where you can place project dependencies. Two Java files have been automatically generated for you: `App.java` and `AppTest.java`.

1.3 Setting up a source code repository

(This part of the practical is not for credit, but it strongly recommended to help to protect you against loss of work caused by a hard disk crash or other laptop fault.)

— ◇ —

In the Informatics Large Practical you will be creating Java source code files and Maven project resources such as XML documents which will form part of your implementation, to be submitted both here and in Coursework 2. We recommend that these resources be placed under version control in a source code repository. We recommend using the popular Git version control system and specifically, the hosting service *GitHub* (`https://github.com/`). GitHub supports both public and private repositories. You should create a *private* repository so that others cannot see your project and your code.

— ◇ —

Check your current Maven project into your GitHub repository. Commit your work after making any significant progress, trying to ensure that your GitHub repository always has a recent, coherent version of your project. In the event of a laptop failure or other problem, you can simply check out your project (e.g. into your DICE account) and keep working from there. You may have lost some work, but it will be a lot less than you would have lost without a source code repository. A tutorial on Git use in IntelliJ is here: `https://www.jetbrains.com/help/idea/set-up-a-git-repository.html`

1.4 The implementation task

For this coursework we will implement some fundamental Java classes and methods which will be useful also for Coursework 2. The functions which we will implement are concerned with the movement of the drone and with the delivery cost for items. Your implementation will be judged on three criteria: *correctness*, *documentation*, and *code readability*.

(a) Your implementation must have a Java package named `uk.ac.ed.inf` with a class named `LngLat` for representing a point (*you can use a record in Java as well, which would actually be more suitable*[1]). The constructor for this class should accept two double-precision numbers, the first of which is a longitude and the second of which is a latitude. The class should have two public double fields named `lng` and `lat`. (This part is a freebie; see above.)

(b) The `LngLat` record should have a no-parameter method called `inCentralArea` which returns `true` if the point is within the Central area and `false` if it is not.

(c) The `LngLat` record should have a method `distanceTo` which takes a `LngLat` object as a parameter and returns the Pythagorean distance between the two points as a value of type `double`.

[1]https://www.baeldung.com/java-record-keyword

(d) The `LngLat` class should have a method `closeTo` which takes a `LngLat` object as a parameter and returns `true` if the points are close to each other in the sense given on page 7 and `false` otherwise.

(e) The `LngLat` record should have a method `nextPosition` which takes a compass direction as a parameter and returns a `LngLat` record which represents the new position of drone if it makes a move in the direction of the compass direction, following the definition of a move given on page 7.

(f) Your project should have three classes named `Restaurant`, `Menu` and `Order` in the package `uk.ac.ed.inf` where Restaurant and Menu represent the results from the REST-request to the `restaurants` endpoint after de-serialization[2]. Restaurant should have a member `getMenu()` which returns the `Menu` objects as an array, which are defined for the restaurant. In addition the class should have a `static Restaurant[] getRestaurantsFromRestServer(URL serverBaseAddress)` method which returns an array of Restaurants which are defined (including the menus).

This static method acts as a factory method so you can call:

```
Restaurants [] participants =
Restaurant.getRestaurantsFromServer(new URL("https://...")
```

(g) The `Order` class should have a method `getDeliveryCost` which accepts a vector of the participating restaurants (including their menus) and a variable number of strings for the individual pizzas ordered. It returns the `int` cost in pence of having all of these items delivered by drone, including the standard delivery charge of £1 per delivery.

Should you find a combination where the ordered pizza combination cannot be delivered by the same restaurant ***this should be considered an invalid combination*** *and an exception* ***InvalidPizzaCombinationException*** *be thrown.*

(h) **You will continue working on these classes in coursework 2 in far more detail**

1.5 Allocation of marks

A total of 25 marks are allocated for Coursework 1 according to the following weighting.

Correctness (15 marks): Your application should correctly implement the classes and methods described in the list above.

Documentation (5 marks): The methods which you implement should have JavaDoc comments which provide brief but clear descriptions of the purpose of the method and its return value and the role of each parameter passed to the method.

Code readability (5 marks): Your Java code should be well-structured and clear with idiomatic use of version 18 of the Java language. You should consider the readability of your code, thinking that it will be passed on to the developers of the drone delivery service to extend and maintain as their needs change.

1.6 Preparing your submission

Make a compressed version of your `ilp` project folder using ZIP compression. Your `ilp` project folder is normally found in the folder `~/IdeaProjects`.

- On Linux systems use the command `zip -r ilp.zip ilp`.

[2] conversion from the textual representation to the object-model

- On Windows systems use Send to > Compressed (zipped) folder.
- On Mac systems use File > Compress "ilp".

You should now have a file called `ilp.zip`. In order to streamline the processing of your submissions, and help avoid lost submissions, please use exactly the filename `ilp.zip`. The archiving format to be used is ZIP only; do not submit TAR, TGZ or RAR files, or other formats.

1.7 How to submit

Ensure that you are LEARN-authenticated by visiting `http://learn.ed.ac.uk`. Go to the ILP LEARN page. Click on the *Assessment* link in the left-hand margin bar and then the link that says *Coursework 1*. Use the *Browse My Computer* option to find and upload your `ilp.zip` file. When finished, make sure that you click *Submit*.

—◇—

This submission mechanism should allow you to make multiple submissions. Later submissions will overwrite earlier ones. Submissions which arrive after the coursework deadline will be subject to the School's late submission penalties as detailed at `http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`. Extension Rule 1 will be applied for submissions for Coursework 1 and Coursework 2. This states that "Extensions are permitted (7 days) and Extra Time Adjustments (ETA) for extensions are permitted." The complete statement of this rule is available at the URL above.

Chapter 2

Informatics Large Practical: Coursework Two

Michael Glienecke and Stephen Gilmore
School of Informatics, University of Edinburgh

2.1 Introduction

As noted above, Coursework 1 and Coursework 2 of the ILP are for credit; weighted 25%:75% respectively. Information on good scholarly practice and the School's late submission policy is provided in Appendix A.

— ◇ —

You are now to extend your project from Coursework 1 into a complete implementation of the drone delivery service as described in this document. You can re-use any code which you produced for Coursework 1 including the code in the Maven XML document named `pom.xml`. You are free to edit, refactor, or delete any code from Coursework 1, keeping only the code that you find useful.

— ◇ —

This coursework consists of the report, the implementation, and a collection of output files written by your implementation. The code which you submit for assessment should be readable, well-structured, and thoroughly tested.

2.2 Report on your implementation

You are to submit a report documenting your project containing the following. Your report should have two sections as described below.

1. *Software architecture description.* This section provides a description of the software architecture of your application. Your application is made up of a collection of Java classes; explain why you identified *these classes* as being the right ones for your application.

2. *Drone control algorithm* This section explains the algorithm which is used by your drone to control their flight around the locations of interest and back to the start location of their flight, while avoiding

all of the no-fly zones and trying to maximise the drone's score on the *sampled average percentage monetary value* metric described on page 18.

This section of your report should contain two graphical figures (similar to Figure 8 in this document) which have been made using the `http://geojson.io` website, rendering the flights of your drone on two dates of your choosing on top of the background provided by the file `testing/all.geojson`.

The maximum page count of your project report is 10 pages, with title pages, references, appendices, and all other material included in the page total. Given the length, your report does not need a table of contents. You cannot submit reports which are over 10 pages in length, but shorter submissions will be accepted. The choice of font, font size, and margins is up to you but please consider the readability of your submission, and avoid very small font sizes and very small margin sizes. Reports of few pages tend to attract few marks; consider 10 pages to be a *goal*, as well as a limit. Your report must be in PDF format in a file named `ilp-report.pdf`.

2.3 Source code of your application

You are submitting your source code for review where your Java code will be read by a person, not a script. You should tidy up and clean up your code before submission. This is the time to remove commented-out blocks of code, tighten up visibility modifiers (e.g. turn `public` into `private` where possible), fix and remove TODOs, rename variables which have cryptic identifiers, remove unused methods or unused class fields, fix the static analysis problems which generate warnings from IntelliJ[1], and refactor your code as necessary. The code which you submit for assessment should be well-structured, readable and clear.

2.4 Result files of the algorithm

In addition to submitting your source code for assessment, your submission should include 12 output files giving the results of trying your drone **on 12 different days** as well as 12 flightpath and 12 deliveries files for these days as well. For details please refer to section: 0.2.

As the REST-service only returns order data for a certain data range ***(currently 2023-01-01 until 2023-05-31)*** *your chosen dates have to be within this range.*

These 36 files should be in a directory resultfiles below the root level of your project directory[2].
[3]

```
drone-2023-01-01.geojson
drone-2023-02-02.geojson
          ⋮
drone-2023-05-18.geojson
```

All of the files submitted should have been generated by the version of your application which you submit for assessment.

2.5 Things to consider

- Your submitted Java code will be read and assessed by a person, not a script. It should contain helpful comments in JavaDoc format, documenting your intentions. Your submitted code should be readable and clear.

[1] normally there has to be a very good reason why any warning should still be issued after this clean-up phase

[2] So, if /ilp is your root, then the files are supposed to be in /ilp/resultfiles

[3] Another option might have been to write back the results to the REST-service, yet for the sake of lowering complexity and making CW 2 easier to process this approach was **not** taken

- Your code will be compiled and executed starting from the same REST-service as you use during development. It should generate and populate the necessary files as described in the section of this document starting on page 19. The generated tables will be processed by a script so they must have the table names, column names, and types specified above.
- Logging statements and diagnostic print statements (using `System.out.println` and friends) are useful debugging tools. You do not need to remove them from your submitted code; it is fine for these to appear in your submission. You can write whatever you find helpful to `System.out`, but the content of your output files must be as specified above. An excessive level of logging can be counter-productive, causing the user not to read the log output, thereby defeating the purpose. Consider what should be logged, and log sparingly.
- Error messages should be written to `System.err`, not `System.out`.
- Your application should be robust. Failing with a `NullPointerException`, `ClassCastException`, `ArrayIndexOutOfBoundsException` or other run-time error will be considered a serious fault.

2.6 Allocation of marks

A total of 75 marks are allocated for Coursework 2 according to the following weighting.

Report (20 marks): You are to provide a document describing your implementation. Your document should be a clear and accurate description of your implementation as described in Section 2.2 above. The two sections of the report are equally weighted, with 10 marks for each section.

JavaDoc documentation (10 marks): You are to document your Java code using the JavaDoc format. Your documentation should be informative and useful, clearly describing the classes, fields and methods of your project. Your JavaDoc code should compile to give a valid HTML document.

Implementation (30 marks): Your submission should faithfully implement the drone behaviour described above, hosted in a framework which allows the drone to make a maximum of 2000 moves on any day. Your application should be usably efficient, without significant stalls while executing. Your code should be readable and clear, making use of private values, variables and functions, and encapsulating code and data structures.

Correctness and effectiveness (15 marks): The flightpaths in the `flightpath` files generated by your application will be tested to ensure that the moves made by the drone are legal according to the description given above, considering the drone Central area and the no-fly zones described above (and delivered from the REST-service). The drone's score on the *sampled average percentage monetary value* metric described on page 18 will be considered: the higher the score the better the quality of the drone.

2.7 Before you submit

You are creating a Java application which is built using the Maven build system. Follow these steps to ensure that your submission builds successfully with the Maven build system.

1. In IntelliJ, open the Maven panel in the top right-hand corner and choose the Maven Lifecycle option "package" as shown in Figure 2.1. (If you are not using IntelliJ as your IDE you can instead run the command `mvn package`, or use whatever means your IDE provides for running this command.)

   This must produce a JAR file in the `target` directory named `ilp-1.0-SNAPSHOT.jar`. Check that your JAR has this exact name. If it does not, modify your project settings or your Maven `pom.xml` file to fix this. Submissions which cannot build a runnable JAR file from the project's `pom.xml` file should anticipate losing marks for implementation correctness here.

2. Run your JAR file with the command

```
java -jar target/PizzaDronz-1.0-SNAPSHOT.jar  2023-04-15
  https://ilp-rest.azurewebsites.net cabbage
```

This must produce the three output files in the current working directory

- `drone-2023-04-15.geojson`
- `flightpath-2023-04-15.json`
- `deliveries-2023-04-15.json`

Check that the GeoJSON output file has this exact name. If it does not, modify your Java code to fix this. Submissions which write wrongly-named files or otherwise fail to create output files should anticipate losing marks for implementation correctness here.

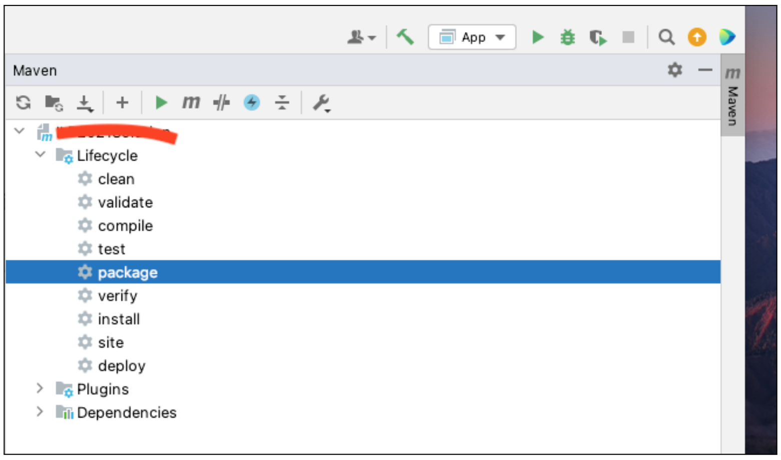


Figure 2.1: Issuing the Maven lifecycle `package` command from the Maven panel in IntelliJ. This is done to build the *über JAR* file for the project.

2.8 Running your project on DICE

(It is not compulsory that you test your project on DICE before you submit but if you wish to here are the sort of commands which you will need.)

—◇—

First, copy your *über JAR* from your own machine onto DICE with a secure copy command like

```
scp target/ilp-1.0-SNAPSHOT.jar s1234567@student.ssh.inf.ed.ac.uk:/home/s1234567
```

This will copy your *über JAR* into your home directory on DICE. Log into your DICE account and the `student.compute` server like this.

```
ssh s1234567@student.ssh.inf.ed.ac.uk

ssh student.compute
```

(It is necessary to log on to `student.compute` because the `student.ssh` machine does not have Java.) Edit your `~/.bashrc` file to define the `DERBY_HOME` environment variable as shown on page **??**.

—◇—

Finally you will be able to run your *über JAR* on `student.compute.inf.ed.ac.uk` with

```
java -jar target/PizzaDronz-1.0-SNAPSHOT.jar  2023-04-15
        https://ilp-rest.azurewebsites.net cabbage
```

2.9 Packaging your submission

- Run your code twelve times to generate the necessary files as described above. These files must then be moved (or copied) to a folder resultfiles below the top-level of your project structure.
- Make a compressed version of your `ilp` project folder using ZIP compression.
  - On Linux systems use the command `zip -r ilp.zip ilp`.
  - On Windows systems use Send to > Compressed (zipped) folder.
  - On Mac systems use File > Compress "ilp".

  You should now have a file called `ilp.zip`.

2.10 How to submit

Ensure that you are LEARN-authenticated by visiting `http://learn.ed.ac.uk`. Go to the ILP LEARN page. Click on the *Assessment* link in the left-hand margin bar and then the link that says *Coursework 2*. Use the *Browse Local Files* option to find and upload your files

- `ilp-report.pdf` and
- `ilp.zip`

In order to streamline the processing of your submissions, and help avoid lost submissions, please use exactly these filenames. The submission format for your report is PDF only; do not submit DOCX files, TXT files, Markdown files, or other formats. The archive format for compressed files is ZIP only; do not submit TAR, TGZ, or RAR files, or other formats. When finished, make sure that you click *Submit*.

—◇—

This submission mechanism should allow you to make multiple submissions. Later submissions will overwrite earlier ones. Submissions which arrive after the coursework deadline will be subject to the School's late submission penalties as detailed at `http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`. Extension Rule 1 will be applied for submissions for Coursework 2. This states that "Extensions are permitted (7 days) and Extra Time Adjustments (ETA) for extensions are permitted." The complete statement of this rule is available at the URL above.

Appendix A

Coursework Regulations

A.1 Good scholarly practice

Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page:

`https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

This also has links to the relevant University pages. You are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work in a source code repository then you must set access permissions appropriately, limiting access to at most yourself and members of the ILP course team.

— ◇ —

The Informatics Large Practical is not a group practical, so all work that you submit for assessment must be your own, or be acknowledged as coming from a publicly-available source such as Mapbox GeoJSON sample projects, answers posted on StackOverflow, or open-source projects hosted on GitHub, GitLab, BitBucket or elsewhere.

A.2 Late submission policy

It may be that due to illness or other circumstances beyond your control that you need to submit work late. Submissions which arrive after the coursework deadline will be subject to the School's late submission penalties as detailed at

`http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/`
`coursework-projects/late-coursework-extension-requests.`

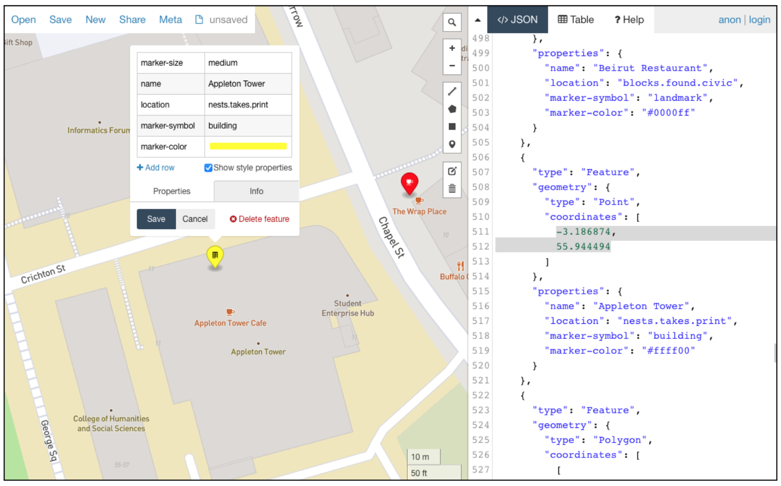
Extension Rule 1 will be applied for submissions for Coursework 1 and Coursework 2. This states that "Extensions are permitted (7 days) and Extra Time Adjustments (ETA) for extensions are permitted." The complete statement of this rule is available at the URL above.

Appendix B

Constants defined in the coursework specification

This appendix contains a summary of the constant values introduced in this document. These values will not change when the service is operational, thus they can be defined as constants of the appropriate type in your Java code.

Constant	**Value**	**Type**	**Defined**
Distance tolerance in degrees	0.00015	double	Page 7
Maximum number of moves a drone can make	2000	int	Page 7
Length of a drone move in degrees	0.00015	double	Page 7
Angle value when hovering	`null`	Enum	Page 7
Appleton Tower longitude	−3.186874	double	Page 7
Appleton Tower latitude	55.944494	double	Page 7
ForrestHillLongitude	−3.192473	double	Page 8
ForrestHillLatitude	55.946233	double	Page 8
KFC longitude	−3.184319	double	Page 8
KFC latitude	55.946233	double	Page 8
Top of the Meadows longitude	−3.192473	double	Page 8
Top of the Meadows latitude	55.942617	double	Page 8
Buccleuch St bus stop longitude	−3.184319	double	Page 8
Buccleuch St bus stop latitude	55.942617	double	Page 8



marker-size	medium
name	Appleton Tower
location	nests.takes.print
marker-symbol	building
marker-color	🟨

[✓ Show style properties]

```
      },
      "properties": {
        "name": "Beirut Restaurant",
        "location": "blocks.found.civic",
        "marker-symbol": "landmark",
        "marker-color": "#0000ff"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -3.186874,
          55.944494
        ]
      },
      "properties": {
        "name": "Appleton Tower",
        "location": "nests.takes.print",
        "marker-symbol": "building",
        "marker-color": "#ffff00"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
```

Figure B.1: The location of the Appleton Tower as specified in `https://ilp-rest.azurewebsites.net/all.geojson`

Appendix C

Using the Piazza Forum

Details

The Informatics Large Practical has a discussion forum on Piazza. You can register yourself to this forum at `https://piazza.com/ed.ac.uk/fall2022/infr0905120223ss1sem1` — please enrol with your own name, not a pseudonym or screen name. If you don't have one already, you will need to create a Piazza account.

C.1 Guidelines

Subscribing to the Informatics Large Practical Piazza forum is optional, but strongly encouraged. Questions posted to the forum may be answered by the course lecturers or by another student on the course. Please read the following notes to ensure that you have the best experience with the forum. These guidelines are based on several years of experience with course fora, where issues such as those below have arisen.

- Discussions and questions on the ILP Piazza forum must relate to the content of the ILP practical. Questions about BitCoin or Elon Musk (for example) will be deleted by the Forum administrators without hesitation.
- Questions asking for students' private information are completely forbidden. These include asking for students' phone numbers, home addresses, or their grades on Coursework 1 or 2.
- Anonymous and pseudonymous posting on the forum is not allowed so please enrol for the course Piazza forum with your own name. Please be aware that, however they may appear to you, posts on the forum are not anonymous to the course lecturers. The forum is available only to students who are enrolled on the ILP this year. The course lecturers reserve the right to delete the enrolment of anyone who is not (or appears not to be) registered for the ILP.
- Especially when commenting on another student's work, please consider the feelings of the person receiving your message. Please refrain entirely from comments criticising the progress of another student. Each of us works at our own pace and there are many different possible orders in which to tackle the work of the ILP. Perhaps you finished implementing something in Week 4, but that does not mean that everyone did.
- If you find some content on the forum helpful, or think that it is making a useful contribution to the course, please acknowledge this by clicking "Good question" or "Good answer" as appropriate; this encourages continued participation in the forum. The course lecturers will endorse answers which they believe to be helpful.

- Forum postings which intend to correct factual errors or resolve ambiguities in the practical specification are welcome. If necessary, the course lecturers will update this coursework document to correct the error/resolve the ambiguity.

- When asking for help with fixing a run-time error, such as an exception, please include what seems to be the most relevant part of the diagnostic error message that you receive, but please include as little of your code as possible. The course lecturers may edit or delete your post if you include too much program code. For the purposes of this practical, the Maven `pom.xml` file is regarded as program code.

- The Informatics Large Practical is an individual programming project so you are not allowed to share your code with others. Please bear this in mind when answering questions on the forum; do not post your solution as an example for someone else to borrow from. *Piazza is not StackOverflow:* please do not post minimal working examples for others to copy and use.

- Many questions on Piazza tend to be of the form “Do we need to do V for Coursework 1?” or “Are we expected to do W for Coursework 2?”. You already have the answers to these questions. This document, the one you are reading right now, contains the definitive statement of what is required for each coursework. It is the *coursework specification.* If this document does not say that it is necessary to do V for Coursework 1, or to do W for Coursework 2, then you do not need to do those things.

- Forum postings which ask for part of the solution to the practical are strongly discouraged. Examples in this category include questions of the form “What is the best way to implement X?" and “Can I have some hints on how to do Y?”

- Questions about the marking scheme for the practical are strongly discouraged. Examples in this category include questions of the form “Which of the following alternatives would get more marks?" and “How much detail is required for Z?” This document already gives you all of the information that you need about the marking scheme, in the sections “Allocation of marks” for each coursework.

- Questions of the form “Can we assume A?” should expect to receive either the reply “No, you cannot make that assumption,” or “Yes, the coursework specification already allows you to make that assumption.” If you were allowed to make an additional assumption A then this document should have told you that you were allowed to make that assumption.