Informatics Large Practical

Michael Glienecke, PhD
School of Informatics, University of Edinburgh

Document version 4.1.0

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 that are provided to them.

New for 2025

We are trying to keep the course, which already runs for a considerable amount of time, fresh and contemporary year by year. Changes in technology, environment, usage, application, and especially student feedback and needs dictate these changes, which are sometimes disruptive to the past.

Among the many things that are different from the past is the move away from the classical essay - which in the age of Gen-AI is difficult to utilise anyway. As a replacement„ we considered a video submission much more contemporary and realistic. This, coupled with a sub-part of the submission which is entirely the student's choice, leaves ample space for creativity and hopefully fun, which after all is why we all work in the IT sector.

Aside the classical Q&A there will be something new - the so-called BOF sessions (Birds of a Theather) - where I will talk about general contemporary IT topics, which have no direct relation to any course-specific content and are thus totally free and if you miss them, you will have any chance to achieve full marks. Should you attend, there is a high likelihood that you will hear provocative, contemporary, or simply extended background information for your very own pursuit.

It is our expressed wish to keep the impact of these changes on the student at a minimum level, but sometimes things do not work as anticipated, things change, etc.

However, please be patient if some of the things we are trying again do not work as anticipated - it is in your interest to constantly improve and adapt this course so that your learning experience is optimal.

Scope

The Informatics Large Practical (ILP) is an individual practical exercise which consists of one large design and implementation project, with 4 coursework submissions and one oral examination.

The **general aim** of ILP is to familiarise you with the current state-of-the-art techniques and methodologies used in software engineering in distributed environments and to provide you with a solid foundation for your own future software engineering work. To achieve this, you will have lectures where more general information is taught (including some topics you have had in the past as a refresher), and lectures where more programming related things are discussed.

As usual, lectures serve as information provisioning, whereas tutorials and Q&A are there to help you achieve your course work and explain some topics in more detail. As we are dealing with highly complex systems, the best approach to deliver the tutorials and Q&A is via in-person meetings where the teacher, TA and demonstrators can help students.

CW1 involves creating a new project to implement a basic REST service with limited functionality running inside Docker as a container. The main building blocks will be working with coordinates and drone moves, order checking and validation, as well as assignment to drones. This service serves as the foundation for the second assignment. *The main focus is to make sure you have the necessary foundational knowledge for assignment 2 as well as a basic understanding of a container context (docker in our case) and REST services.*

As development is also about the structure and organisation of the source code, these will be marked for CW1, as will the proper use of Java as a language and development environment.

CW2 is the implementation of the entire project.*The main focus is on programming and implementing the required tasks - please check the individual assignment paper for details on these*

Marking will be done using an automarker for CW1 and CW2 and the results for these will be sent to you as an e-mail report.

CW3 *is the implementation of something completely based on your own choice, as long as it has a direct and coherent relation to ILP.* You can literally do whatever you want, as long as it is related to ILP. Languages and frameworks allowed are Java, Python, JavaScript, Go, Rust, C#, React, Angular, or HTML-5.
You have to submit your solution and a description in a PDF with a maximum of 1,000 words describing why you chose the topic, which problem it is supposed to solve, for whom it is, and how you implemented it.
You will be marked on your submission and mainly this PDF-document for: Innovation / Idea / Benefit, Execution / Implementation, Completeness and Style.

CW4 will be a 5-7 min video presentation where you are supposed to demonstrate your solution in addition to explaining which target audience you selected for the solution, which motivation you had and what the background of the solution is.
You will be marked based on: Background elaboration / target audience, Problem statement, Explanation of your choice, Presentation as such.

YOU CAN ONLY BE MARKED FOR CW4 IF YOU SUBMITTED CW3. WITHOUT CW3 NO MARKING WILL BE POSSIBLE FOR CW4

Oral examination will be an interview-style question and answer with the student, lasting around 6 minutes, where 2-3 questions (some easier, some harder) have to be elaborated upon. The questions will be randomly assigned. Often these questions do not have a 100% right or wrong answer, but serve as the basis to see if you understood the knowledge and are aware of implications.

To prevent hidden bias every marking for the oral examination, CW3 or CW4 will be done simulta-

neously be at least 2, mostly 3 markers and the mean value taken as a mark.

Please note that the coursework parts are not equally weighted. There is no ILP exam paper so, to calculate your final score out of 100, just add your marks for the coursework.

Coursework	**Weight**	**Consists of**
Coursework 1	25%	
		Programming Tasks (19%)
		Style & Coding (6%)
Coursework 2	33%	Programming Tasks
Coursework 3	20%	Student's choice implementation with PDF explanation
Coursework 4	12%	Video presentation
Oral examination	10%	

Some words about lectures, marking, auto-testing, and essays

In the last years ILP was mainly a coding course, where you got some specific instructions (this document), implemented some classes, and got some result. This no longer serves the purpose; you should be able to reflect on what you are doing and implement the necessary features, including the unit test.

To address these issues, several changes have been made to the ILP **lecture structure**, which are described below:

- As REST-services and containerisation are now commonplace, ILP will also be implemented as a REST-service running inside a **docker** container as well.
- To enable you to do that and to create enough background knowledge, many lectures will be provided on these topics. This will include JSON handling and REST server access.
- Unit testing is an essential part of modern software engineering, so we will cover that in depth, including mocking.
- The structure of the source code into classes, services, packages, and the project organisation will be parts as well.

As you can see, **auto-marking** will be a large part of your final mark. The benefit of auto-marking is that there is no unconscious bias, no personal preferences of style or form - just rules which have to be fulfilled. So we have worked hard (learning from the previous courses as well) to make all this as smooth as possible for you.

Specifications for programming tasks and essays

As programming tasks change more often than this more general document, **we decided to separate the detailed specifications for the individual submissions.**
So, for a detailed specification, please access the corresponding Learn page in ILP (usually *Assessment Instructions* under *Assessment*) for any specific information.

Contents

List of Figures

Chapter 1

The Coursework Specification

1.1 Introduction

Assume that we wanted to create a drone-based medication delivery service for emergency situations where the normal pharmacy dispatch system might no longer work.
We would send prepared prescriptions from dispatch points (usually pharmacies) to deliver them to their correspondent receiver(s) using a drone. The delivery is physically done via drones, which have a set of attributes like capacity, availability, cold / hot storage, cost / move, maximum distance, etc., affecting the usability for the deliveries.
The medications can be normal or special in that they require cold / warm storage, emergency supplies, etc., and have an associated packaging size.

The flight drones reside at several central static points spread over the geographic area covered. **They have to start from there and return to the same point after a set of deliveries has been made**. In these points the drones are immediately re-charged by exchanging the battery and thus available for another flight. Due to wear and tear, **not all drones are available all the time and can fail during the day**, so the availability before each flight is dispatched must be checked.

Each flight will be from a central static location to a pick-up point and from there to one or many drop-off points(s), returning to the static location. **It is therefore intended that a drone delivers more than one delivery in one flight to optimise utilisation**.

The problem of submitting prescriptions to the pharmacy and obtaining legitimacy from the receiver before the package is dropped must be solved in other projects and is not of any concern to the system.

We only take care to deliver existing prepared orders as efficiently as possible from a pick-up point to a set of drop-off points.

— ◇ —

The idea of a drone-based medication delivery service has some issues that should be taken into account when implementing the solution.

- Drones - being battery-operated - only have a limited amount of energy they can spend in the air, so optimising the deliveries is paramount. This can be mainly achieved by having several deliveries, which are potentially in one direction, on the same drone and delivering them one by one.
- You cannot fly everywhere, as some areas are (potentially due to public safety or protection) entirely forbidden to cross (the so-called **"no-fly zones"**) and some have restrictions for crossing (the so-called **"restricted zones"**).

— ⋄ —

Ordering the delivery of the medication is relatively straightforward and the School of Informatics is developing an online system to take medication orders and add them to a database of orders to be delivered.
This system will perform the necessary checks so that we can assume that only valid orders as such are present. **This does not imply that each order can be executed as perhaps drones are not available, or not for the order due to special requirements like cooling, etc.**.

This online system will consist of 4 main parts:

- ▷ an order processing back-end which is currently under development [1].
- ▷ a REST-based service, which provides information such as drones available at static points, static points as such, no-fly zones, restricted zones, etc.
  You will only consume the service (in part 2 of the course), yet you will not modify anything. The base URL is like: `https://ilp-rest-2025.XXX.azurewebsites.net/`, where the XXX is replaced by the region of the hosting provider, etc. **As this will change[2] over time the URL has to be taken from an environment variable when your service starts up.**
- ▷ **A REST-based service to calculate the routes for the drone from pick-up to (potentially multiple) drop-off points. This includes checking orders and assigning them to available drones, including optimizations, and simple flight calculations.** [3]
- ▷ A mobile user-interface probably in react-native to cover iOS and android at the same time

The main problem is that orders might have no available matching drone (e.g. requiring cooling or heating), or too many orders are placed, so that path optimisation is critical to guarantee a distribution with as few as possible drones and flights. In addition, it might mean that not the closest central drone location for a drop-off or pick-up can be taken, but one which has availability for the drone in need.

— ⋄ —

In general, the following rules apply:

- You can assume that all orders can and **must** be delivered. Your orders are the only ones to be shipped, so do not consider outside your orders.
- Optimisation is important, but do not over-prioritise it; this will be part of a later project stage. For the moment, it is important to deliver all orders in as few as possible drone usages and moves.
- A major challenge will be to allocate the right drones (and filter out those which cannot be used) from the static locations and to load them optimally.

— ⋄ —

Your task is to devise and implement a REST-based micro service that provides endpoints to:

- Organize the data to find the proper drones for the job needed (data checks)

[1] and not to be considered further
[2] Currently the URL is: https://ilp-rest-2025-bvh6e9hschfagrgy.ukwest-01.azurewebsites.net/
[3] This is your work and the main subject of the ILP course

- Dispatch several orders using as little drones, with as little movement as possible (considering cost, order and load factors as well as travel distances). This involves to have multiple drop-off from one pick-up
- Perform 2-dimensional flight path calculations for the orders in the item before

— ◇ —

To make the services as available and scalable (vertical vs. horizontal scaling) as possible, they will be run in a Kubernetes environment later. For the development time docker will be utilised as this is a similar container environment with less effort and complexity. Thus, your service will have to run as a docker container inside docker. There are alternatives available to docker, such as podman [4], which solve one of the biggest problems docker has - the background daemon, which is necessary to run the system. As we cannot test every possible option, we stay with docker for the time being, yet this might change at a later time. You can use any other option as well, just make sure that your final image (which you submit) works with docker, as this is the reference (and marking) platform.
A benefit of running as a container inside a docker runtime is that your service can run completely on its own in its own runtime environment, not affected by anything on the outside. It just provides "endpoints" (like /actuator/health), which can be called and return data according to the requirements (coursework specifications).

— ◇ —

You will be provided with data about central drone locations, drone capabilities, pick-up availability, etc. This information will come in the form of a REST-service running on a server, which returns the data in JSON format[5] *(more detailed information and examples will be provided).*

It is important to stress that the information in the test data which you will be given during development 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 data. For this reason, your solution must be *data-driven.* That means **your service must read all the information from the ILP-REST service and no data can be hard coded in your application, except where it is explicitly stated in this document that it is acceptable to do so**.

— ◇ —

Any data which acts as a kind of "instruction" (like the orders to process) will be passed into your service via the endpoint call in the corresponding data exchange areas.

— ◇ —

As environments are dynamic and URLs can change, everything which you consume (like the ILP-REST-service URL) must be coming from a configuration. Normally, this would be a configuration server, an LDAP directory, or anything else, yet for our means environment variables serve the purpose wonderfully. You can set them in your development environment before you run your service and the auto-marker will set the necessary ones as well during loading / starting of your provided image.

— ◇ —

[4]https://podman.io/
[5]https://en.wikipedia.org/wiki/JSON

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 at the School of Informatics who will maintain and develop it in the months and years ahead when the *MedSupplyDrones* delivery service is operational. For this reason, *clarity and readability of your code is important*; you need to produce code that can be read and understood by others.

— ◇ —

1.2 Latitudes and longitudes

In this course, we will be using latitudes and longitudes to identify locations on the map (such as pick-up and drop-off points).

▷ Longitude is the measurement east or west of the prime meridian.

▷ Latitude is the measurement of distance north or south of the Equator.

(The above are National Geographic definitions.) Latitudes and longitudes are measured in *degrees*, so we stay with this unit of measurement throughout all of our calculations. Even when we calculate *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 course, **locations expressed using latitude and longitude are treated as if they were points on a plane, not points on the surface of a sphere**. This simplification allows us to use the Pythagorean distance as a 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 accurately move the drone to a specified location. 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 (we use 5 digits of precision** [6]).

— ◇ —

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.

— ◇ —

As drones have limited capacity, the geographic area covered by drones will be limited to a reasonable size and all source-destination pairs will be reachable for a single charge for a drone.

[6] For the geographical position of Edinburgh this equates to roughly 100cm in latitude, which is a reasonable resolution for a drone

1.3 Drone movement specifications and definitions

The flight of the drone and its movement is subject to the following stipulations:

- the moves are of two types, the drone can **either** *fly horizontally* or *fly vertically* — the drone can change its latitude and longitude when it flies horizontally, but not when it is flying vertically, which only alters its flight altitude — flying and hovering use the same amount of energy;
- The flight altitude is not considered in the flight
- **Hovering is represented by two moves at the same coordinate**
- every move when flying is a straight line of length 0.00015 degrees[7];
- As a technical side-note one would normally use BigDecimal in Java instead of double to get around many precision-loss and rounding problems.
- the drone *cannot fly in an arbitrary horizontal direction*: it can only fly in one of the 16 major compass directions as seen in Figure 1.1.

  These are the primary directions North, South, East and West, and the secondary directions between those of NE, NW, SE and SW, and the tertiary directions between those of NNE, ENE, and so forth.

  These 16 directions are all 22.5° apart from each other

  We use the convention that:

  - 0 means go East
  - 90 means go North,
  - 180 means go West,
  - 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.

  So, as an example to get from A to B this movement sequence might look like the following picture:
- as the drone flies, it travels at a constant speed and consumes power at a constant rate;

[7] 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.

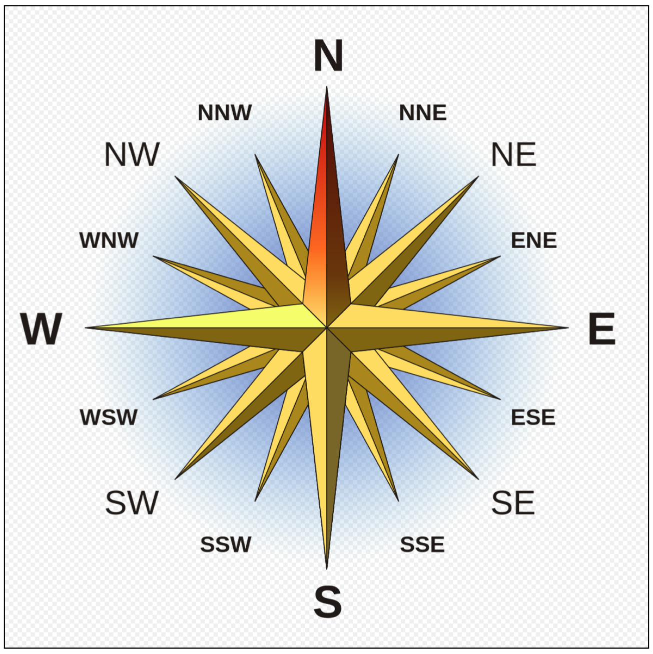


Figure 1.1: The 16-point compass rose `https://commons.wikimedia.org/w/index.php?curid=2249878`

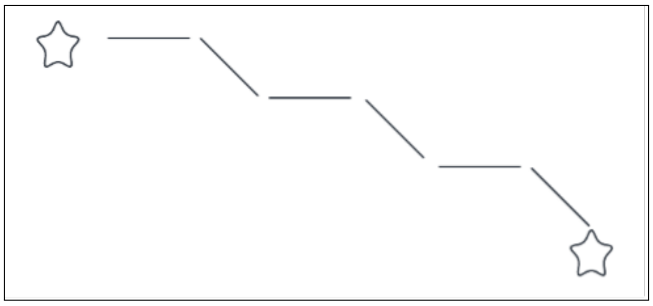


Figure 1.2: Stepwise moves of the drone to reach a target

1.4 Additional requirements

The following additional requirements are given:

- The general requirement is that each drone has to start at the point where it is operated (the central static drone points), fly to a medical dispatch (a pick-up) point to receive the medication(s), deliver the medication to one or several drop-off point(s), and return to the base where it started. All this has to be done within the maximum flight distance which is defined per drone.
- Each point must be hit with precision. To test for a hit consider a circle with 0.00015° radius around any point you use - if the coordinate on your flightpath is inside the circle, you are **"close"**. This closeness test is very important as it is used extensively by the auto-marker and will come up in CW1 explicitly
- You are not allowed to cross corners of no-fly-zones (or enter the zone in general). This will result in flight paths like the one depicted below:

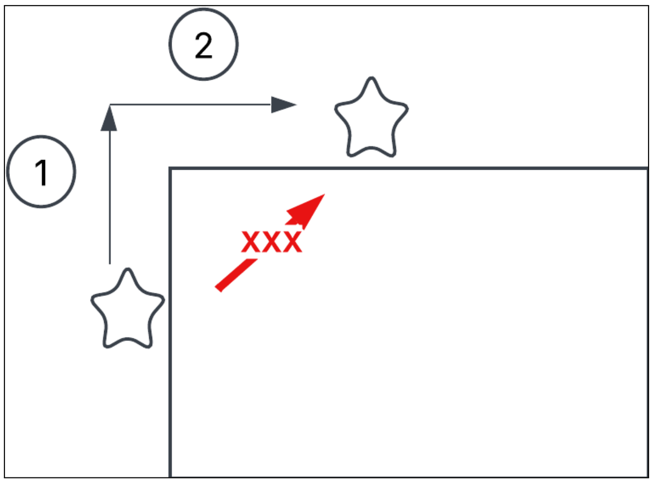


Figure 1.3: A correct and ann invalid flight path crossing a corner

- Restricted or No-Fly-Zones are always rectangular, defined by 4 points. Every zone has an implicit width of at least 0.00015° or the current width defined by the coordinates. So, even if two opposite sides are next to each other, it is no line, but a rectangle.
- Drones have cost / move (as a larger drone might carry more, but costs a lot more / move) plus a fixed cost for each start / stop when they take off and return to their bases (which is again dependent on the drone). So, for any flight your total cost can be calculated like:

  Total Cost for flight = Constants for take-off and return + moves on path * cost / move

  The constants are specific per drone
- You must be as efficient as possible in terms of total moves and total cost as **both factors will be considered in the marking**
- Your total execution time for solving the individual tasks may never exceed 30 seconds per task. As the marking system is very fast, this should equate to roughly 45 - 60 seconds on normal machines and only be used as an upper boundary.

  Based on previous years' experience, the bulk of execution time was between 1 and 10 seconds. So, if your algorithm is much more (say 45 seconds, etc), you have room for improvement - the question is simply if benefit merits the extra effort. It should be stated that sometimes extra calculations are needed to produce the most optimal routing, especially with multiple drop-off points.

  However, do not focus too much on the performance of the algorithm, as long as it stays within the defined dimensions, as it will not be considered for marking. ***The overall architecture, as well as order retrieval, processing, and especially optimisation, are much more important and should receive your main attention.***
- The programming language to be used for your service is Java (preferably version 21[8])

  You are free to use any needed libraries and Java-versions as long as you can produce a valid docker image.

[8]as you are running as a docker image all this is hidden from the auto-marker, so all is your choice; just Java 21 has many nice features you might like to use

If there is one book which would help you tremendously for this course and in general for you later programming work, I would recommend: "Cloud native Spring in Action" - for details please see the library section on the Learn page where the necessary online links can be found. *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.*[9]

If there is one book which would help you tremendously for this course and in general for you later programming work, I would recommend: "Cloud native Spring in Action" - for details please see the library section on the Learn page where the necessary online links can be found.

- *Your entire service must run inside the submitted docker image.*

  Given previous years' experience, please make sure you test your final image in a docker environment with proper test data and processes

1.5 Exemplary and schematic flight paths

To help you imagine the flight paths, you will have to generate, the image below shows a simple 2D-flight path.

Figure 1.4 shows 2 drone locations with 2 pick-up points and some drop-off points. The big questions for you (considering total moves and cost) will be if the middle point is included in the left or right path.

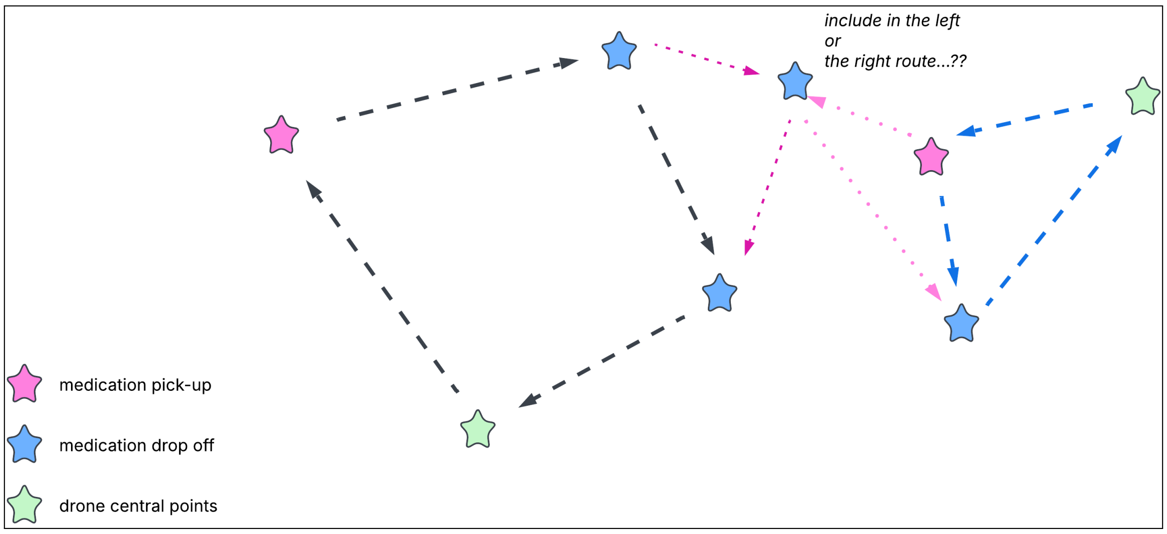


Figure 1.4: Creating 2D flight paths with some optimization

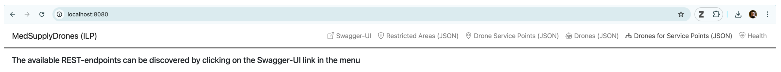
[9]Many Java textbooks are available so if you cannot get a copy of Java Precisely please feel free to choose another textbook. Alternatively, many tutorials on Java are available online, including the (slightly dated but still very useful) Java Tutorial from Oracle at `https://docs.oracle.com/javase/tutorial/`.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

1.6 The ILP-REST-Service

All dynamic information needed by the *MedSupplyDrones* service is provided by a newly developed centralised REST service (base URL at `https://ilp-rest-2025.XXX.azurewebsites.net/`[10]). This service [11] makes the whole system more dynamic and provides dynamic data, which allows you to always get up-to-date information about the data items provided, etc.
You have to retrieve the data every time before you calculate the paths or perform other operations to ensure that the latest information is processed, which could have changed in the meantime. This is very important because otherwise the changes in the data may not be properly reflected.[12]

— ◇ —

To test the REST-server you can use your browser and just type the base-URL for the base page or the REST-endpoint (for dynamic resources).
The Base-URL will produce the following image:



The available REST-endpoints can be discovered by clicking on the Swagger-UI link in the menu

Figure 1.5: Showing the start page of the ILP REST Service

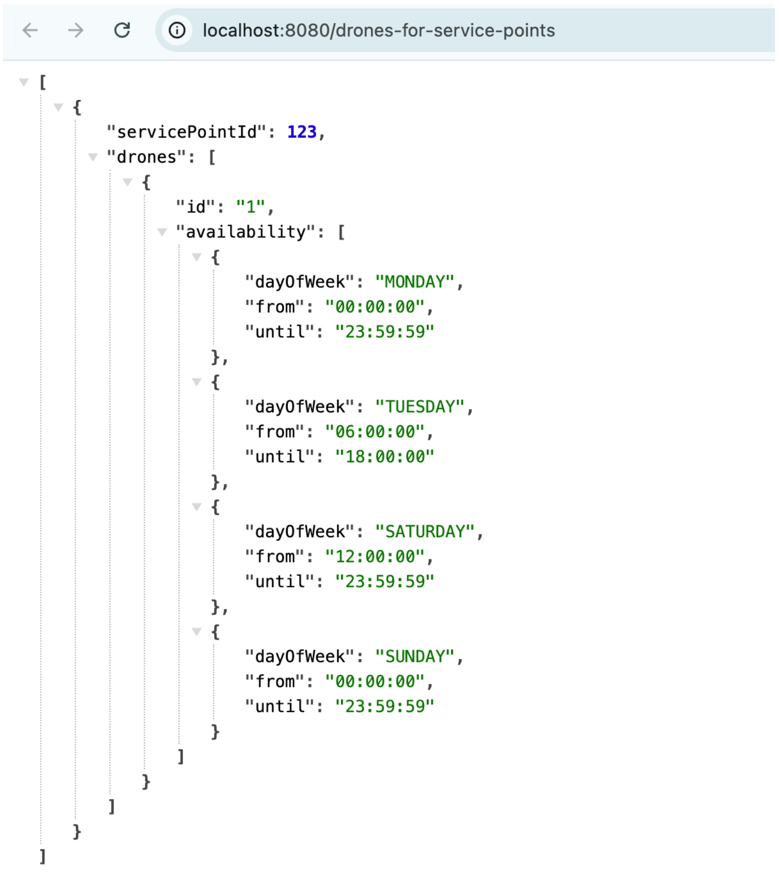
— ◇ —

[10]just be aware to use an environment variable **ILP_URL** for this as it for sure will change

[11]which actually could be a single machine or a cluster to provide a more secure platform for later business growth

[12]During marking it could happen that server-data changes between 2 calls, so even the same environment with the same order could suddenly have a different data basis and thus produce a different result

The result of the operation is always rendered in your browser; either as html (for the base page) or as the display of a JSON data structure (the dynamic data).
This could look as in Figure 1.6 (for the available drones for a service point):

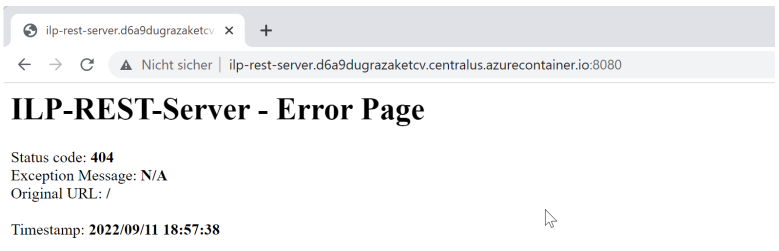


```
[
  {
    "servicePointId": 123,
    "drones": [
      {
        "id": "1",
        "availability": [
          {
            "dayOfWeek": "MONDAY",
            "from": "00:00:00",
            "until": "23:59:59"
          },
          {
            "dayOfWeek": "TUESDAY",
            "from": "06:00:00",
            "until": "18:00:00"
          },
          {
            "dayOfWeek": "SATURDAY",
            "from": "12:00:00",
            "until": "23:59:59"
          },
          {
            "dayOfWeek": "SUNDAY",
            "from": "00:00:00",
            "until": "23:59:59"
          }
        ]
      }
    ]
  }
]
```

Figure 1.6: 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 1.7 is shown


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

— ◇ —

These URLs could look like this:

- `https://ilp-rest-2025.XXX.azurewebsites.net/`

  Represents the base page

- `https://ilp-rest-2025.XXX.azurewebsites.net/actuator/health/livenessState`

  Indicates the state of the system - should be "UP"

- `https://ilp-rest-2025.XXX.azurewebsites.net/restricted-areas`

  Returns information about restricted and no-fly zones

- `https://ilp-rest-2025.XXX.azurewebsites.net/drones`

  Returns information about the capabilities of the drones in the system by drone category.

- `https://ilp-rest-2025.XXX.azurewebsites.net/service-points`

  Returns information about service points used by drones.

- `https://ilp-rest-2025.XXX.azurewebsites.net/drones-for-service-points`

  Returns the available drones for the service points

As these information items can change even between calls to your service, you must retrieve them every time anew

— ◇ —

1.6.1 JSON-data wrapping (using de/serialization)

Any data used in the system for information exchange are JSON based, which is just a sequence of characters. By using *Jackson*[13] *or Gson*[14] this is converted to class instances[15].
Sometimes (like when data are passed into methods of a service) this conversion takes place by the Spring runtime (should you use it, which is highly recommended), or you have to explicitly convert yourself.

1.7 JSON data objects

Inside the ILP- and *MedSupplyDrones* services, the following JSON objects are being used.
Be aware - and this is valid for every JSON we use in ILP - that you should ignore parts of JSON you do not understand (like the alt below in LngLatAlt)[16]

- **LngLatAlt** which defines a point with "lng" and "lat" as longitude and latitude in °, as well as an altitude in moves above 0. If the alt part is not specified (thus **null** in Java), it can also be omitted.

  What a typical JSON would look like

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

[14] https://github.com/google/gson/

[15] This process of converting the textual data to objects is called de-serialisation

[16] JSON has a feature to be "forgiving" for undefined tokens or "strict" as a default, which throws an exception if something not expected is in the data.

```
{
  "lng": -3.192473,
  "lat": 55.946233,
  "alt": 0
}
```

- **RestrictedArea** defines an array of points (vertices) that form a closed [17] polygon. To distinguish each restricted area, they can have a name. If the limits element is missing, this restricted area is to be considered like a "no-fly zone" with lower = 0 and upper = -1.

```
{
    "name": "George Square Area",
    "id" : 1,
    "limits": {
      "lower": 0,
      "upper": -1
    },
    "vertices": [
      {
        "lng": -3.190578818321228,
        "lat": 55.94402412577528
      },
      {
        "lng": -3.1899887323379517,
        "lat": 55.94284650540911
      },
      {
        "lng": -3.187097311019897,
        "lat": 55.94328811724263
      },
      {
        "lng": -3.187682032585144,
        "lat": 55.944477740393744
      },
      {
        "lng": -3.190578818321228,
        "lat": 55.94402412577528
      }
    ]
  }
```

  This defines 5 points (where the last one closes the polygon) with lng / lat as no altitude is needed here. The limit for the restricted areas is given in the limits subelement

- **Drone** which defines a drone and all its attributes:

  What a typical JSON would look like

```
{
```

[17]as the last point will lead back to the start

```
    "id": 1,
    "name": "Drone 1",
    "capability": {
      "cooling": true,
      "heating": false,
      "capacity": 15.5,
      "maxMoves": 6000,
      "costPerMove": 0.03,
      "costInitial": 3.4,
      "costFinal": 4.5
    }
}
```

- **ServicePoint** defines a drone service point with a name and which drones are available.

  What a typical JSON would look like

```
   {
    "name": "Appleton Tower",
    "id": 123,
    "location": {
      "lat": 55.9445,
      "lng": 3.1867,
      "alt": 50
    }
  }
```

- **DroneForServicePoint** defines which drones are available at a service point. What a typical JSON would look like

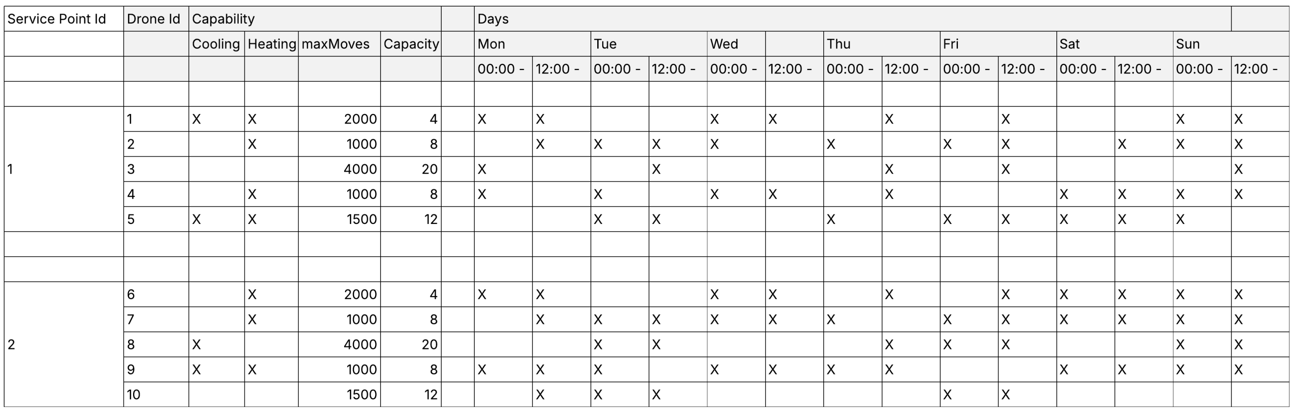
```
  {
    "servicePointId": 123,
    "drones" : [
      {
        "id": 1,
        "availability": [
          {
            "dayOfWeek": "MONDAY",
            "from": "00:00:00",
            "until": "23:59:59"
          },
          {
            "dayOfWeek": "TUESDAY",
            "from": "06:00:00",
            "until": "18:00:00"
          },
          {
            "dayOfWeek": "SATURDAY",
            "from": "12:00:00",
            "until": "23:59:59"
          },
          {
            "dayOfWeek": "SUNDAY",
            "from": "00:00:00",
            "until": "23:59:59"
          }
        ]
      }
    ]
  }
```

This would define a service point (#123) that has 1 available drone (#1) on different days of the week with the capabilities specified in drone #1.

Using this approach drones can move between service points, easily taken off the rota, recharged / repaired and brought back into the service. As these data must be read for each dispatching operation, any change in data will be used the next dispatching occurs.

Further necessary derived JSON-types can be easily constructed. For details of the actual data types to use, check the result of the endpoints of the ILP service or the corresponding CW-specification.

If you think about it graphically, you can easily construct a matrix like this one:



Service Point Id	Drone Id	Capability					Days													
		Cooling	Heating	maxMoves	Capacity		Mon		Tue		Wed		Thu		Fri		Sat		Sun	
							00:00 -	12:00 -	00:00 -	12:00 -	00:00 -	12:00 -	00:00 -	12:00 -	00:00 -	12:00 -	00:00 -	12:00 -	00:00 -	12:00 -
	1	X	X	2000	4		X	X			X	X		X		X			X	X
	2		X	1000	8			X	X	X	X		X		X	X		X	X	X
1	3			4000	20		X			X				X		X				X
	4		X	1000	8		X		X		X	X		X			X	X	X	X
	5	X	X	1500	12				X	X			X		X	X	X	X	X	
	6		X	2000	4		X	X			X	X		X		X	X	X	X	X
	7		X	1000	8			X	X	X	X	X	X		X	X	X	X	X	X
2	8	X		4000	20				X	X				X	X	X			X	X
	9	X	X	1000	8		X	X	X		X	X	X	X			X	X	X	X
	10			1500	12			X	X	X					X	X				

Figure 1.8: Example distribution matrix of drones in service points over the week

Chapter 2

Informatics Large Practical: Considerations for Programming Task Submissions

Michael Glienecke, PhD
School of Informatics, University of Edinburgh

Document version 4.1.0

This chapter only provides general information. For details, please refer to the ILP Learn page for the corresponding programming task requirements

2.1 What has to be submitted

Any programming task submission consists of a ZIP file of the entire working directory structure of your project, as well as the exported docker image you created as a tar file in the root directory.

From this, the auto-marker will extract the tar file and import it into a docker container. The sources are necessary proof and evidence of your independent work, as well as should something fail with the tar-file.

If no sources or tar files are present, marking cannot happen and manual marking must take place. If manual re-construction of the image is necessary, a penalty of 10 points will be applied, so make sure your image is runnable in docker.

2.2 Source code of your service

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 prior to 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.

2.3 Unit testing your service

Your service needs to be tested properly, and unit tests (covered in the lectures) are a major part of that and considered the "internal" test (compared to any outside test using Postman, etc.).

2.4 No files are created in assignments

The service only returns JSON-data and / or http status codes from the exposed API. Therefore, no direct files are returned, yet can be generated by saving the response into a corresponding file.

2.5 Your very own implementation

As written before, for CW3 you are totally free to choose any topic you want to implement, as long as it is related to ILP. Marking will be done on the provided PDF (which should thus be of very good quality) and your submitted sources.
If your solution does not run inside the same docker image, you have to put the sources in the directory under the same root as ILP and provide instructions in a README.MD file (Markup format) on how to build / start / run your implementation. Please make sure this works, as otherwise marks will be taken away.

You submit the sources and PDF in the corresponding Learn submissions.

As marking considers innovation and use case and application, a simple UI to just visualise data is not recommended, as too easy to achieve (you would thus not reach full points).

Please don't overdo it and always consider the points given for the task. You should normally expect to work between 8-12 hours on this at maximum.

The use of AI for inspiration is explicitly allowed, just don't use it to generate all the code - there should be some parts of your very own contribution.

As this is very much prototyping, it could also happen that it does not work in all paths in the end. Nothing to worry about; as long as the idea is identifiable and something works, things are still quite fine.

2.6 Video submission

Your own work has to be presented professionally - this is what the video is for. Please explain what you did, why, for whom and what you tried to solve. In addition, give a short demo so that the audience can properly understand how your tool works.

Just upload the finalized video onto Learn.

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

2.7 Things to consider

- **Please make sure that your submitted tar-file (the docker-image) really runs properly. This can be easily tested by simply importing it into your local docker system and running it plus accessing with postman or curl**
- Your code will be executed "as is" inside the target docker environment (with 8080 as assumed local port) via the provided tar-image.
  Therefore, it can be assured that anything you implemented for your service in your development system is 100% identically executed on the target system.
- Should you produce initialisation exceptions (e.g. wrong Java runtime, etc.), then the service will not work / run and cannot be tested (resulting in a potential manual rebuild - see above).
- 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 if necessary. Your submitted code should be readable, clear, and well structured according to several potential ways to structure the code-base (will be discussed in the lectures as well).
- Logging statements 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.

  An excessive level of logging can be counterproductive, causing the user not to read the log output, thereby defeating the purpose. Consider what should be logged and log sparingly with the appropriate log-level.
- Error messages should be written to `System.err`, not `System.out`.
- Your service should be robust and always return only the defined http codes, regardless of the call issued. Failing with any kind of exception or other run-time error inside the docker environment will be considered a serious fault.

2.8 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 corresponding link for the CW1 or CW2 programming task. Use the *Browse Local Files* option to find and upload your files

- `sXXXXXX.zip` - here XXXXXXX is your student id

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 and the last submission available when the marker is run will be marked.
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`.

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 on the School page:

> `https://informatics.ed.ac.uk/taught-students/all-students/your-studies/academic-misconduct`

This also has links to the relevant University pages. You are required to take reasonable measures to protect your assessed work from unauthorized 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 Use of Gen-AI

For ILP CW3 and CW4 allow explicitly the usage of AI to generate ideas. Not code, not videos - ideas... You can take these as the basis for your own work, yet please make sure that your own work is the majority.

A.3 Late submission policy

It may be 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

> `https://web.inf.ed.ac.uk/infweb/student-services/taught-students/information-for-students/information-for-all-students/your-studies/late-coursework-extension-requests`.