A core principle of unit testing is

"Any program feature without an automated test simply doesn't exist"

JUnit - is a framework for writing unit tests

- Unit testing is particularly important when software requirements change frequently
  - Code often has to be refactored to incorporate the changes
  - Unit testing helps ensure that the refactored code continues to work

Maven Surefire plugin configuration into pom.xml

To be able to run the tests from the command prompt, your pom.xml configuration file will need to include a JUnit provider dependency for the Maven Surefire plugin

Test fixture - sets up the data (both objects and primitives) that are need for every test
(Eg. If you are testing code that updates an employee record, you need an employee record to test it on)

Unit test - is a test of a single test

test case - tests the response of a single method to a particular set of inputs

test suite - a collection of test cases

test runner - software that runs tests and reports results

The delta parameter

- If the actual value is not equal to the expected value, JUnit throws an unchecked exception, which causes the test to fail

- Most often the delta parameter can be zero. It comes into play with calculations that are not always precise, which includes many floating-point calculations. The delta provides a range factor. If the actual value is within the range expected - delta and expected + delta the test will pass. You may find it useful when doing mathematical computations with rounding or truncating

Note * JUnit creates a new instance of the tests class before invoking each @Test method. This helps ensuring the ~~independence~~ independence between test methods and avoids unintentional side effects in the test code.

Branch Coverage - Covers both the true and false conditions unlike statement coverage

- A branch is the out come of a decision, so a branch coverage simply measures which decision outcomes have been tested

From Section 1 (of this notebook)

Editing Dockerfile

line 1 FROM openjdk:17 // using java 17
2 COPY target/classes/ /tmp , // grab class files that
3 WORKDIR /tmp // container becomes, was generated dump
working directory, into container in location
4 CMD java ie.abu.Hello called tmp

→ Runs java command in Working directory and in this example is running this class file "Hello

By click on the Double green arrows you can edit run configuration

and Edit Imagetag to relate to project (try add versions)

and Edit container name to relate to project

You can use image tage to run image from terminal with intelij and it should automatically use its address

To get a slim down version of openjdk you can look in search in docker desktop and use a different jdk this can increase or decrease size of container

Running a jar file instead of a class file

```
Line 1  FROM openjdk:17
2       COPY target/ /tmp
3       WORKDIR /tmp
4       CMD java -jar HelloExample-1.0-SNAPSHOT.jar
```

Might Require manifest file to know what the main file is

docker run --name ____ -p 3000:8080 ____
(version)

* If it won't let you commit in intelij "detected dubious ownership"

use git config --global --add safe.directory '*'

Spring and JPA Tutorial

@Id
@GeneratedValue(strategy = GenerationType.Auto):
private long count;

// it uses count as a unique number in database called a primary key

@Entity // Used to store info in database
^
above classes

@Table // informing Springboot about the class so that it is a class to the database

import org.springframework.data.jpa.repository.JpaRepository;

public interface PassengerRepo extends JpaRepository
<Passenger, Long> { },

↑
JPA interface pattern where we identify our domain class the unique key

private final PassengerRepo passengerRepo;

Makes Service class (PassengerService) to use the Repo class (PassengerRepo) interface to access our database using JPA specifications

@AllArgsConstructor so that SpringBoots will be responsible for providing access to ~~Passenger~~ Repo (PassengerRepo)

Eg

```
@AllArgsConstructor
@Service
public ~~Service~~ class PassengerService {

    private final PassengerRepo passengerRepo;

    public List<Passenger> getPassengers()
    {
        return passengRepo.findall(); ← List
    }
```

- getPassenger() method use passenger Repo instead of hard coded values "passengerRepo.findall();"

~~Repo~~ application.properties in resources config

spring.datasource.url = ~~Postg~~ jdbc:postgresql://localhost:5432/labs
↓ location of database ↑ name of database

spring.datasource.username = postgres] you need password + username
" . " . password = password] to connect to database

spring.jpa.hibernate.ddl-auto = ~~n~~ create - drop → creates but at the end deletes
↓ This does all the heavy lifting of the main operations

spring.jpa.hibernate.ddl-auto = update
↓ update each time

spring.jpa.show-sql = true (shows it running)

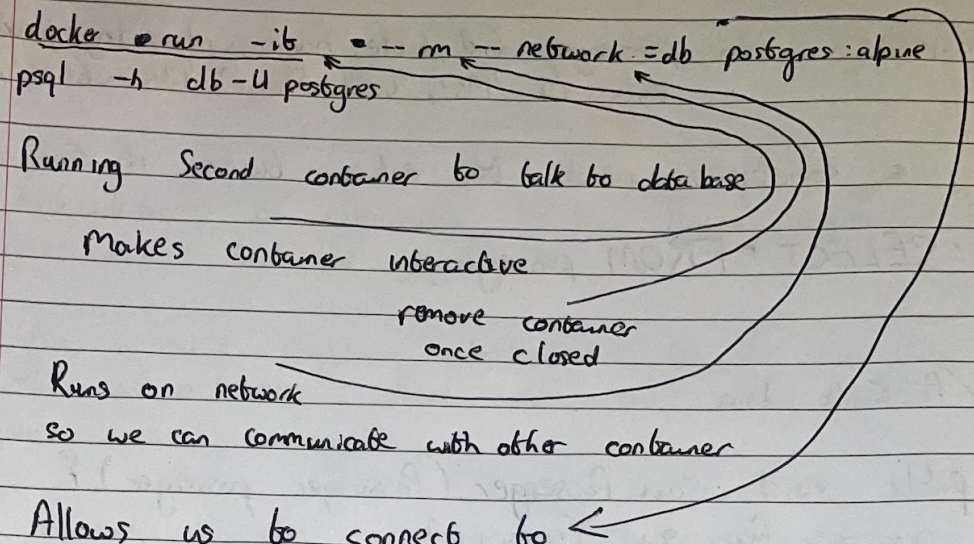
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.Postgre-
" . " . " . " . ↑ format_sql = true, - SQLDialect
How we are gonna communicate to hibernate

Docker command
Docker network create db (Running two containers with Postgres and also a container with terminal so we can see database)
↓
Creates Network
that any container parts of this network can look into other containers in this network

: ↑ ↑
calls container db Expose port and talks to this port

docker run --name db -p 5432:5432 " --network-db -v
"%cd%": /var/lib/postgresql/data" -e ~~POST~~ POSTGRES_PASSWORD-
password -d postgres:alpine
↑ runs "postgres:alpine" Image
stores info in container in this location (current location)

makes container available in network



CMD commands

for postgres admn : \l

(gives list of databases)

- : CREATE DATABASE abj;
- :lq (to quit)
- :\c abu (\c to connect to a certain database)
- :\dt (list database contents such as table provided by postgres)

Example of inserting into a data base

- :insert into passenger (id, title, name, phone, age, count)
  values (1, 'Ms', 'Wallis Shane', '134906488', 4, 1);

  * sometimes need to enter twice if Error on terminal

- : SELECT * FROM passenger;

/Back to Java

public void savePassenger(Passenger passenger){
  passengerRepo.save(passenger);
}
// Saves the entity passenger so Saves our Database

To run everything again in cmd you will have to remove Network

docker network rm db

and then remove containers and re run commands

Using UmL

- UmL is use to break down the system into a use case model and then a class model

- And for object-oriented systems analyse and design

Putting Uml to work

1. A use case diagram, describing how the system is used.

2. A use case scenario. This scenario is a verbal articulation of exceptions to the main behavior described by the primary use case.

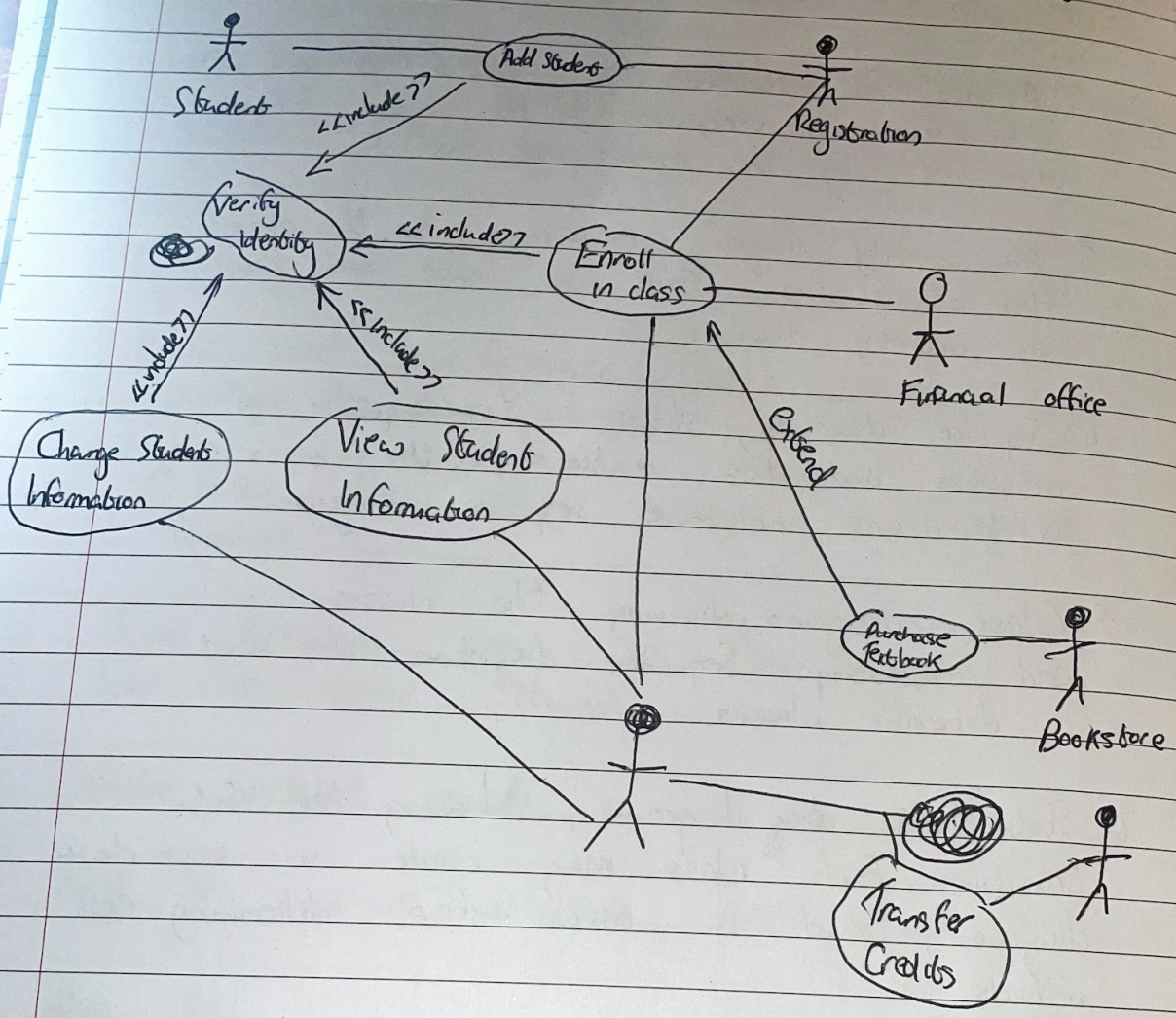
3. An activity diagram, illustrates the overall flow of activities. Each use case may create one activity diagram

4. Sequence diagrams, shows the sequence of activities and class relationships. Use cases may create one or more sequence diagrams.

5 Class diagrams, showing the classes and relationships. Sequence diagrams are used to determine classes.

6. Statechart drag diagrams, showing the state transitions. Each class may create a statechart diagram, which is useful for determining class methods

A Use Case Example of Student Enrollment

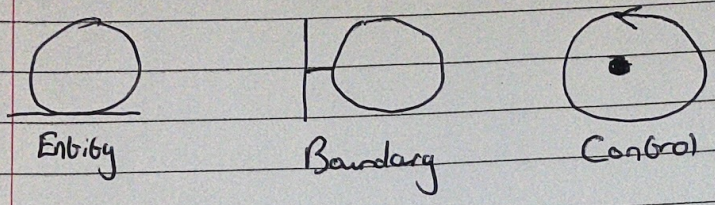


Types of Classes in a Sequence Diagram:

- Presentation layer, what the user sees, corresponding to the interface or boundary classes
- Business layer, containing the unique rules for application, corresponding roughly to control classes

- Persistence or data access layer, for obtaining and storing data, corresponding to the entity classes

Symbols for the class types



*Dot not supposed to be there