• Class elicitation is a dual process - Class suggestion, class Rejection

Understanding Project Shopping Cart

1. Classess (Suggested List of Class)

Shopping Cart // Maybe Inheritance
Item
- Book
- Calender
- Pen
- Tshirt

2. Reject classes

Shopping Cart
Item - Name, Quantity, Price, ItemType, id
- ~~Book~~ - Book
- ~~Calender~~ - Calender
- Pen
- Tshirt

LineItem

Final list of classes (Potential) Should

1. Shopping Cart
2. Item
3. Line Item

CICD - Using Git Notes

Git - Init to configure Repo (Git Init)

To see if Repo is in a folder go to view and then show hidden items (.git should be filename)

git config user.email "G00380316@atu.ie" (links or tags email)

git config user.name "Enoch Abiodun" " "

It tags all changes to this email or name

git status (Shows files and their status in the repo) ← shows any changes

git add (adds new files in repo to Staging area to get ready to commit)

git commit -m " " (commits files to repo)

↙ ↓

notes change and clears files floating in repo

git log (Shows history such as commits and who and messages of commits in repo)

git checkout (can use it to go back to previous commit)

touch .gitignore (creates file (linux command))

↳ (git ignore file [example "*.bak"] → Extension

Class elicitation

git add . (means add everything to stage)

git checkout -b "feature1" (creates new branch)

git branch (list all your branches)

git checkout [branch name] (to switch branch)

* Never work on master work on alternative branch

git merge [branch name] (merges changes on a different branch with the current branch)

A commit is a record of what files you have changed since the last time you made a commit. Essentially, you make changes to your repo (for example, adding a file or modifying one) and then tell git to put those files into a commit.

Commits make up the essence of your project and allow you to go back to the state of a project at any point

How do you tell git which files to put into a commit? This is where the staging environment or index come in. When you make changes to your repo, git notices that a file has changed but won't do anything with it (like adding it in a commit)

To add a file to a commit, you first need to add it to the staging environment. To do this, you can use the git add <filename> command

Once you've used the git add command to add all the files

you want to the staging environment, you can then tell git to package them into a commit using the git ~~command~~ commit command.

When making a commit and you forget to use -m when committing and end up in the VIM editor, just do the following

1. Type i to switch into insert mode so that you can start editing the file
2. Enter or modify the text with you file
3. Once you're done, press the escape ~~esc~~ key Esc to get out of insert mode and back to command mode
4. Type :wq to save and exit your file

Say you want to make a new feature but are worried about making changes to the main project while developing the feature. This is where git branches come in.

Branches allow you to move back and forth between 'states' of a project. For instance, if you want to add a new page to your website you can create a new branch just for that page without affecting the main part of the project. Once you're done with the page, you can merge your changes from your branch into the master branch. When you create a new branch, Git keeps track of which commit your branch 'branched' off of, so it knows the history behind all the files

Let's say you are on the master branch and want to create a new branch to develop your web page. Here's

what you'll do: Run git checkout -b <my branch name>. This command will automatically create a new branch and then 'check you out' on it, meaning git will move you to that branch, off of the master branch.

After running the above command, you can use the git branch command to confirm that your branch was created

To exit scroll mode in the git bash terminal you can press 'q' to exit.

git config -l (A config check)

git remote add (Telling Git that we want to push code to a cloud based solution)

example " git remote add origin https://github.com/" "/[redacted]t assignment.git"

git remot get — nickname — location in the cloud

↳ For location

git branch -M main (Renames branch to main)

↑ for master because Github uses main instead of master

git push -u origin main (Pushes all code from local to origin to the branch main)

↑ Unified "Keep two guys linked"

* git pull origin - "To pull from Github"

go to github website then go to settings and Develop settings and then Personal access tokens tick all the options and click generate new token

Once you put the token in the prompt it should complete the ~~prompt~~ push and link the

git push -u origin "bug01" (git push -u origin "branch name" is to push local branch to remote repo and link them)

↙ branch

↑ use this for new files aswell

Pull Requests is how you bridge the gap on Github to main and you feature branch in an a controlled environment and its the area that code is reviewed and then if its okay and no conflicts you can then merge feature to main and most likely you will be prompted to delete branch.

If your invited to Collaborate in somebodies else project and have access to their repo on Github make a new folder using a name suitable for project and use Git bash here then use this command

git clone _____ (In the underlined area paste HTTP link from Github.com where the repo is)

Process - Class suggestion, class

git branch -d {local_branch} (use -D to force deleting branch without checking merging status)

git push origin -d {remote_branch} (Removes a remote branch from the server)

CICD - Usen Maven with Github 2022 NOTES

When starting a new Maven Project Look for Advanced Settings or Artifact Coordinates on Intelij

and enter a Group Id, Artifact Id and if prompted Version "1.0-SNAPSHOT"

~~Once the Project~~

Once the project is built we want to enable version control so we go to VCS// at the top of intelij and click the first option enable integration version control// and choose git//

After it is common you must then commit files since you created a new project and files haven't been tracked yet to do this click on Git section and then Commit changes then you should be brought to a tab where it shows untracked files, add them to staging environment

You can do this by right clicking on files and then clicking on the git dropdown and selecting add once you added all necessary files click "commit" you will be prompt for message to describe commit

After writing description a pop up asking for ~~use~~ use

Process - Class suggestion, class

name and email will appear just like Git type in and then uncheck "set globally" to bypass security issues and then click ok and the commit should take place and staging area cleared

Any time you make a class it should be contained in a package (it is used for identification - If two classes uses the same name or if you need to share a particular class it differentiate your class from another persons)

Connect Intelij to Remote Repository

1.
File -> Settings -> Version Control -> Github

2.
Add Accounts and then log in your account is now linked

3
Git -> Manage Remotes -> Click 'add'

Paste in link to the Remote Github repo in the Url section

How to get Another person projects from Github through Intellij

1.
Open Intellij and then click "Get from VCS"

* Class elicitation is a dual process - class -

2.

Paste Repository link you wants to clone in Url Section

Making a new branch with Intelij

Right click any then select Git -> new Branches -> your prompt for a name and then press ok you should be automatically checked out

CICD - Using TDD with Junit

Mvn repository.com can be used to get dependecies for your pom file in your project

Once you have added dependecies and applications such as Junit and the Junit engine you can generate test classes automatically by right-clicking on the class/ method and click generate then test then selects Before and after option
tear

@Test should be used on top of test methods

@BeforeEach ← Makes this method Run before every new test
void setup() {
MyCount = new Count(); ← new instance of Counter made before every test eg.
}

Surefire plugin helps lets you run tests true it and this is by clicking maven section tab and navigating to test in the lifecycle dropdown or

Plugins dropdown and clicking surefire plugin and clicking surefire:test

assertThrows(IllegalArgumentException.class, "")

↑

Test will fail unless Exception thrown

(Testing for failure)

() -> { }

↓

lambda

"Block" of code your trying to execute (Functional Programming)

Test Fixtures

Methods annotated with @Before will execute before every test case

Methods annotated with @After will execute after every test

```
@BeforeEach
public void setUp() {...}
@AfterEach
public void tearDown() {...}
```

Test Cases

Method annotated with @Test are considered to be test cases

```
@Test
public void testAdd() {...}

@Test
public void testToString() {...}
```

Within a test Case

- Call the methods of the class being tested
- Assert what the correct results should be with one of the provided assert methods
- These steps can be repeated as many times as necessary
- An assert method is a JUnit method that peforms a test, and throws an AssertionError if the test fails
  - JUnit catches these exceptions and shows you the results

Test for failure

- We need to test that the application is failing as we expect

- Let assume that there is a feature in the Counter app where you can set the starting point for the counter

```
public Counter (int starting) {
  if (starting <= 0)
     throw new IllegalArgumentException ("must be greater
     than 0");
}
```

Test using Executable (Lambda)

```
public Counter (int starting) {

    if (starting <= 0)
        throw new IllegalArgumentException ("Must
      be greater than 0");
}
```

- To test, we need to generate an illegalArgume

```
@Test
void testConstructorWithValue()
{
   assertThrows (IllegalArgumentException.class, ()->
                 {new Counter (starting: -1);});
}
```

Test Quality & Code Coverage

What is code coverage?

- Code coverage is a way of ensuring that your tests are

actually testing you code.

- When you run your tests you are presumably checking that you are getting the expected results
- Code coverage will tell you how much of your code you exercised by running the test

Coverage Tool - JaCoCo

- The JaCoCo Maven plugin provides the JaCoCo runtime agent to your tests and allows basic report creation

Code coverage metrics

Number of different metrics are used determine how well exercised the code is.

- Statement coverage
- Branch coverage
- Path coverage

Statement Coverage

- The statement coverage is also known as line coverage
- The statement coverage covers only the true conditions
- Through statement coverage we can identify the statements

executed and where the code is not executed because of blockage.

- It does not understand the logical operators

- Statement coverage is the most basic form of code coverage

The benefit of statement coverage is its ability to identify which blocks of code have not been executed (Does not identify bugs from the control flow constructs in source code like compound conditions or consecutive switch labels)

Using Code Coverage Tutorial

- To run tests with coverage

1. Click run dropdown on the toolbar of Intellij
2. Click "Run '....' with Coverage"

* Note If it doesn't run you might need to run it normally first

Jacoco runs is a test coverage plugin

"clean" in lifecycle for maven gets rid of target folder and cleans everything

To get a report outside intelij using Jacoco after installing the plugin

you run the lifecycle package that should generate a new target folder if you had just used the clean lifecycle but this is not necessary as by running package it runs prevous lifecycles
Next navigate to site in the target folder then jacoco in the site folder then open index.html in browser

* If its not green it doesn't exist

Creating a yaml file for github actions

- make a new folder in project called ".github"
- make a sub folder called "workflows" then make a file with the extension "yml"

Using Micro Services Video

https://spring.io -> https://start.spring.io

- If your getting a port build fail for your new project you can open applications properties and change the port

server.port = "8081"
↑
Example default is 8080

Tomcat is the webserver that will run microservice

"Endpoint" - Communicating with application to do task with

@RestController - informs springboot lets springboot know to listen for calls from the class

@GetMapping - Marks what methods activates when certain calls come in

@RequestMapping - Directs where GetMapping should map to

@GetMapping("{passengerID}")
↑ Captures information e.g pathVariable
if you can use this to call methods

Anything at the end of url triggers in micro service video

with Ultimate you can generate Http requests without browser by left clicking on the global signal or by going to Services tab & Dropping down Http Requests

@Service ^ Describes porpuse of class as a Service (Dealing with logic

@AllArgsConstructors Using Lombok it creates code for you like Getter & Setters

Eg @AllArgsConstructor
@Getter
@Setter
@NoArgsConstructor

*Note when building classes or anything for a Microservices in terms of testing it is bad practice for the Microservices to be tightly coupled because now you can't do Unit test without a integral test

Eg (PassengerService myService = new PassengerService(); ← Passenger Application check tutorial file
(return myService.getPassengers)

Using Docker lab 2022

To access cmd for your folder for Docker navigate to folder clear address bar and type cmd

eg you can run "docker run hello-world"
then "docker image ls" ↑ lists all images
" "docker ps" ↑ lists all images running

pulls an image that was already made by someone else in an container (pulls image)

Container port

Eg Docker run --name website -d -p 3000:80 nginx ↑ Image

↑ Names whatever your running website

↑ detaches so it doesn't run in cmd

maps anything that happens on your machine (pc) on port 3000 to port 80 (communication)

↑ Container that has a server set up inside it

Dock stop " " (To stop image running)

* Any time a container stops you lose all the information

docker run --name website1 -v "%cd%:/usr/share/nginx/html" -d -p 3000:80 nginx

↑
Maps all the infomation contained within the container and you can make a copy of all the infomation thats inside container and put it to somewhere in my local computer

-v (Set up volume "storage area")

store usr grab image directory
"%cd%:/user/share/nginx/html/"

on my local computer using my current directory maps everything I do in the location (location is inside the the image)

You can create prettier website by grabbing a free website from https://startbootstrap.com/

- Unzip the website in your volume location that you setup
- Reload browser and you should see website

container name
docker exec -it "" bash if you want to peep inside running container

Docker ~~pal~~ plugin / Extension can be found in services section in Intelij Services
on intelij (U)

To create a new Dockerfile, right-click on your root project and select new Dockerfile or new file and call it ("Dockefile")

Lecture notes on Maven

What is Maven?

Maven is a build automation tool used primarily for Java projects

Maven addressess three aspects of building Software:

1. it describes how software is built
2. it describes its dependencies
3. it uses conventions for the build procedure

An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins

A Maven POM file (Project Object Model) is an XML file that describe the resources of the project. (ex directories of source code and test source and what external dependecies (Jar files) your project has

The POM file describes what to build, but most often not how to build it. Maven build phases and goals deal with that.

Maven Overview

Maven → Reads pom.xml

1. Reads pom.xml
2. Downloads dependencies into local repo
3. Executes life cycle, build phases and/or goals.
4. Executes plugins

All executed according to selected build profile

Pom file - Maven local Repository

Build Life Cycles
- Phases
  - Goals

Dependecies (JARS)

Build Plugins

Build Profiles

Maven Project Artifacts

Artifact is a file (JAR) that gets deployed to Maven repository.

A Maven build produces one or more artifacts, such as a compiled JAR and a "sources" JAR.

Each artifact has a group ID (usually a reversed domain name, like .com.example.foo), an artifact ID (just a name), and a version string.

<groupId> ie.gmit </groupId>
<artifactId> maven-example </artifactId>
<version> 1.0 </version>

Maven uniquely identifies an artifact using:

- group ID: Arbitrary project grouping identifier (ie.gmit)
- artifact ID: Arbitrary name of project (maven-example)
- version: Version of project
  Format {Major}.{Minor}.{Maintenance}
  Add '-SNAPSHOT' this means it is in development

GAV Syntax: groupId:artifactId:version

Maven features - Build Tool

The advantage of automating the build process is that you minimize the risk of humans making errors while building the software manually.
Additionally, automated build tool is typically faster than human performing the same steps manually.

Maven features - Dependencies

- One of the first goals Maven executes is to check the dependencies needed by your project.
- Dependecies are external JAR files (Java libraries) that your project uses.

If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository.

Maven Features - Dependencies (cont'd)

- Download project dependencies from centralized repositories (ex. Hibernate libraries)
- Automatically resolve the libraries required by projects dependencies

Project ⇔ Local in /m2 ⇔ Central
others...

Maven Features - Dependencies

Dependencies consist of:
- GAV
- Scope: compile, test, provided (default = compile)
- Type: jar, pom, war, ear, zip (default = jar)

```
<project>
...
    <dependencies>
        <dependency>
            <groupId> javax.servlet </groupId>
            <artifactId> servlet-api </artifactId>
            <version> 2.5 </version>
            <scope> compile </scope>
        </dependency>
    </dependencies>
</project>
```

Maven Features - Standardised Builds

- Uniformity across projects through patterns
- Consistent path for all projects

Maven Build Lifecycle

Maven has three lifecycles - default, clean and site

Each lifecycle is made up of lifecycle. When a lifecycle phase is invoked, all preceding phases are executed sequentially one after another

lifecycle phases by themselves doesn't have any capabilities to accomplish some task and they rely on plugins to carryout the task

Depending on project and packaging type, Maven binds various plugin goals to lifecycle phases and goals carryout the task entrusted to them

Maven Build Lifecycle

When default lifecycle is used,

1. Validate the project
2. Compile the sources
3. Runs those (compiles) against the tests
4. Package the binaries (e.g jar)
5. Run integration tests against that package
6. Verify the integration tests
7. Install verified package to the local repository
8. Deploy the installed package to a remote repo

* Use the build tools within Maven rather than the build tools in the IDE, as this is how it will behave when we deploy to a server

What is a Maven Plugin

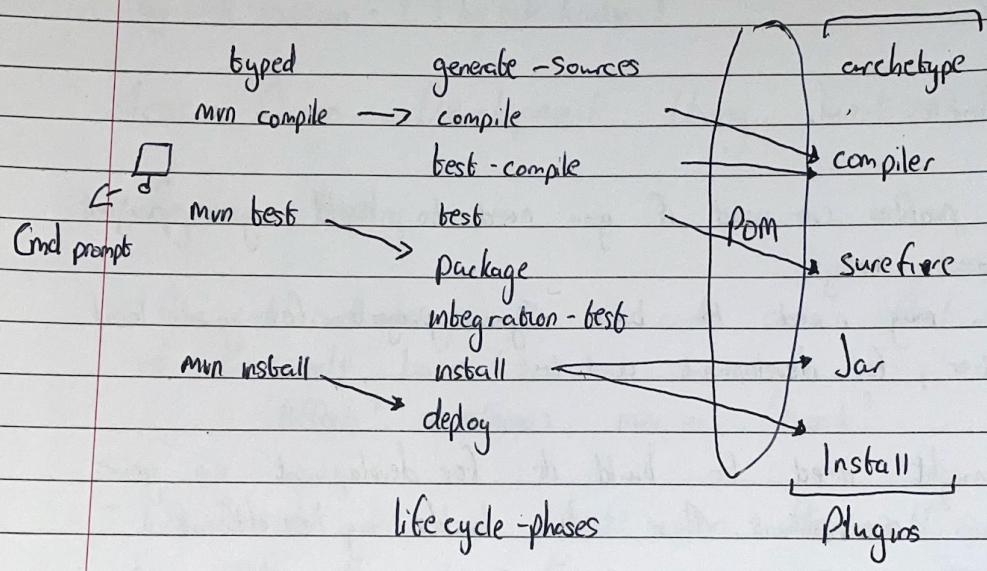
"Maven" is really just a core framework for a collection of Maven Plugins.

Plugins are where much of the real action is performed, plugins are used to:

create jar files
" war "
compile code
unit test code
create project documentation

Example Maven Goals

To invoke a Maven build you set a lifecycle "goal" (eg. install)



Add a plugin

Edit the Pom File to include the Surefire Plugin (eg!

•

→ Next Page

```
<build>
  <pluginManagement>
  <plugins>
    <plugin>
      <groupId> org.apache.maven.plugins </groupId>
      <artifactId> maven-surefire-plugin </artifactId>
      <version> 2.20.1 </version>
    </plugin>
  </plugins>
  </pluginManagement>
</build>
```

Build Profiles

- Build profiles are used if you need to build your project in different ways
(eg You may need to build your project for your local computer, for development and test)

- You might need to build it for development on your production environment. These two builds may be different.

- To enable different builds you can add different build profiles to your POM files. When executing Maven you can tell which build profile to use

- resources / config.properties
- A properties file, later Maven will map the value depend on the profile id

```
# Database Config
db.driverClassName = ${db.driverClassName}
db.url = ${db.url}
db.username = ${db.username}
db.password = ${db.password}

# Email Server
email.server = ${email.server}

# Log Files
log.file.location = ${log.file.location}
```

Test - Driven Development Using Junit CICD NOTES

- Regression testing

- New code and changes to old code can affect the rest of the code base
  - 'Affect' sometimes means 'break'

- We need to run tests on the old code, to verify it works - these are regression tests

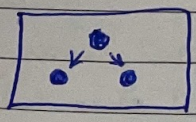
- Regression testing is required for a stable, maintainable code base

What should be tested?

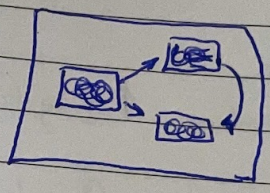
- Test for boundary conditions
- Test for both success and failure
- Test for general functionality
- ETC...

Types of Tests

- Unit
  - Individual classes or types
- Component
  - Group of related classes or types



- Integration
  - Interaction between classes



Unit test - Examines the behaviour of a distinct unit of work. (Within a Java application, the "distinct unit of work" is often (but not always) a single method.

TDD - is a technique whereby you write your test cases before you write any implementation code

Note
- An indication of "intent"
- Tests provide a specification of "what" a piece of code actually does
- Some might argue that tests are part of the documentation.

TDD Stages

1. Write a single test
2. Compile it. (it should not compile because you have not written the implementation code)
3. Implement just enough code to get the test to compile
4. Run the test and see it fail
5. Implement just enough code to get the test to pass
6. Run the test and see it pass
7. Refactor for clarity and "once and only once"
8. Repeat