CAB302 Software Development Special Topic Lecture Assignment 2

Faculty of Science and Engineering

Semester 1, 2018

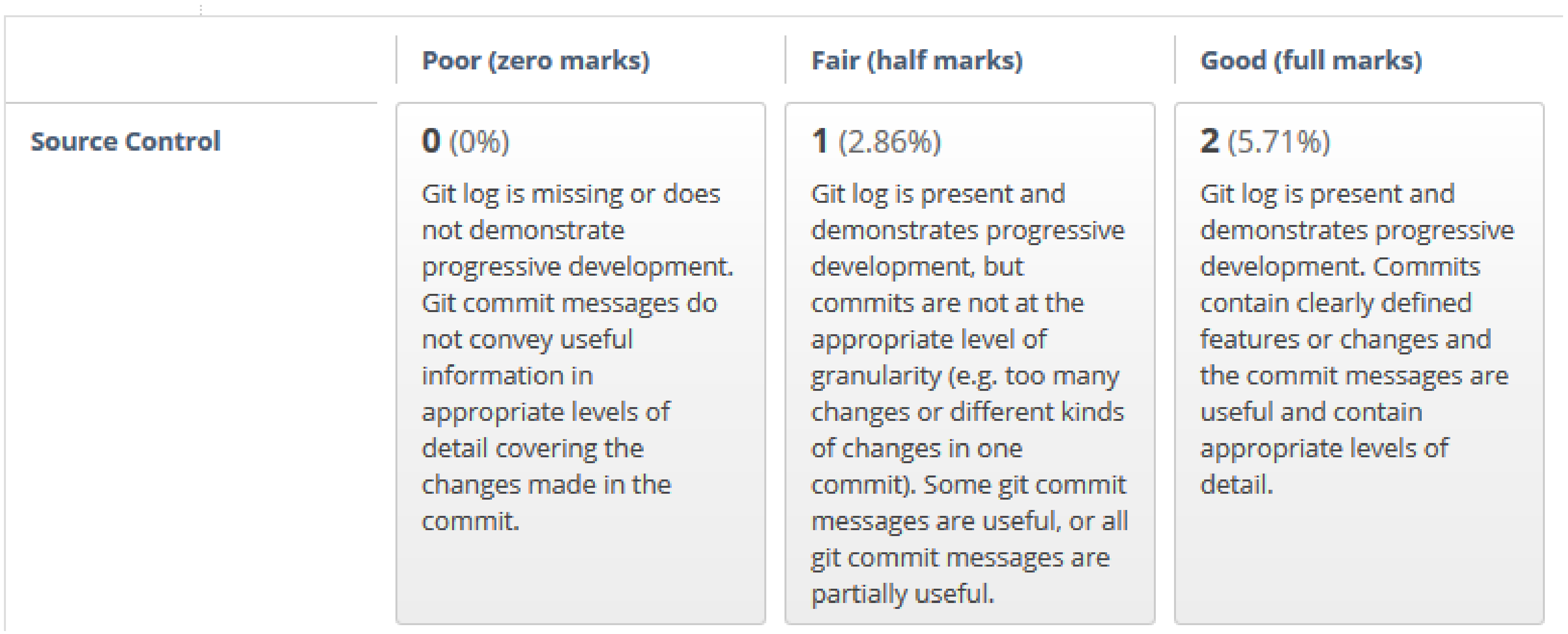
Timothy Chappell

Assignment 2

- Task: To implement an Inventory Management System
  - Specification is on Blackboard
- Due date: 25/05/18
  - Friday of Week 12
- Group size: 2
  - Enrol in a group on Blackboard (under Assignment 2 Groups)
  - You will need to join a group before you can submit the assignment

Marking

- Weighting: 35%
- Marked via rubric (available on Blackboard)

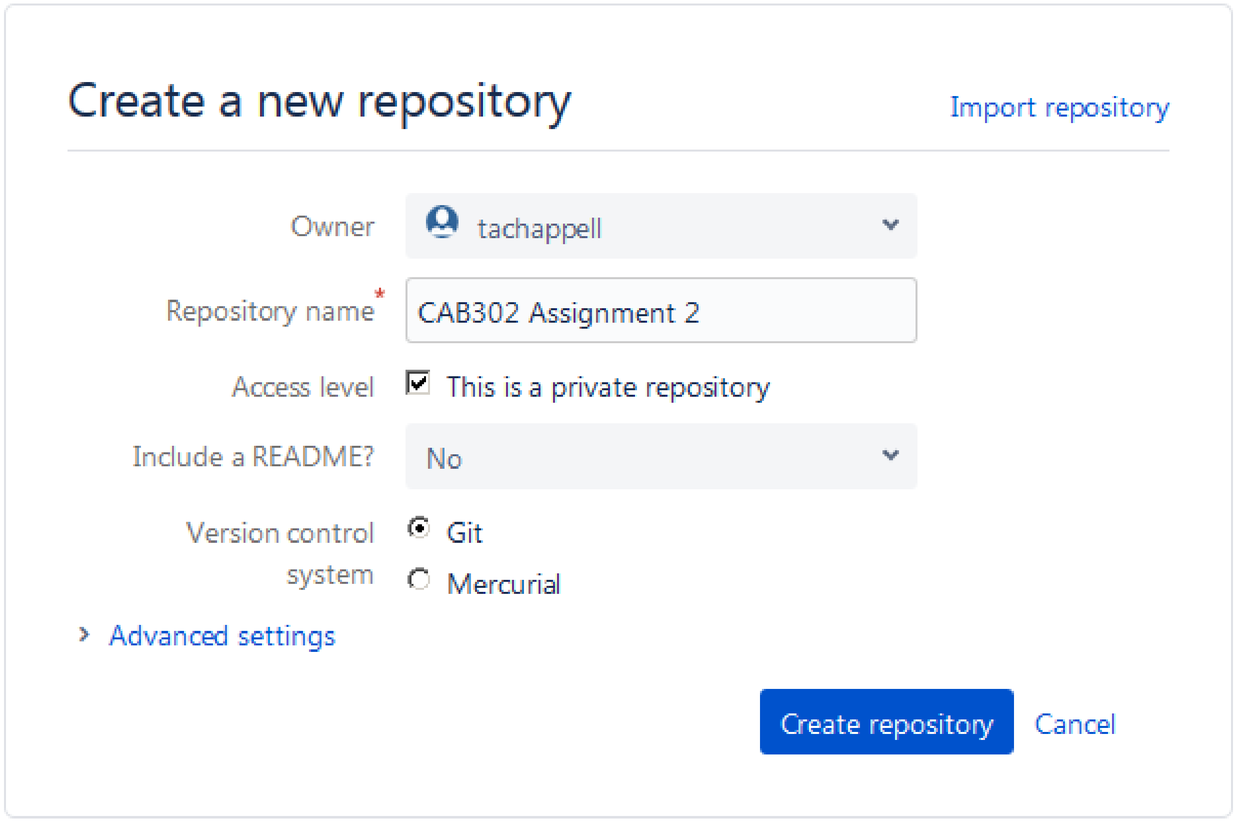


	Poor (zero marks)	Fair (half marks)	Good (full marks)
Source Control	**0** (0%) Git log is missing or does not demonstrate progressive development. Git commit messages do not convey useful information in appropriate levels of detail covering the changes made in the commit.	**1** (2.86%) Git log is present and demonstrates progressive development, but commits are not at the appropriate level of granularity (e.g. too many changes or different kinds of changes in one commit). Some git commit messages are useful, or all git commit messages are partially useful.	**2** (5.71%) Git log is present and demonstrates progressive development. Commits contain clearly defined features or changes and the commit messages are useful and contain appropriate levels of detail.

Source Control

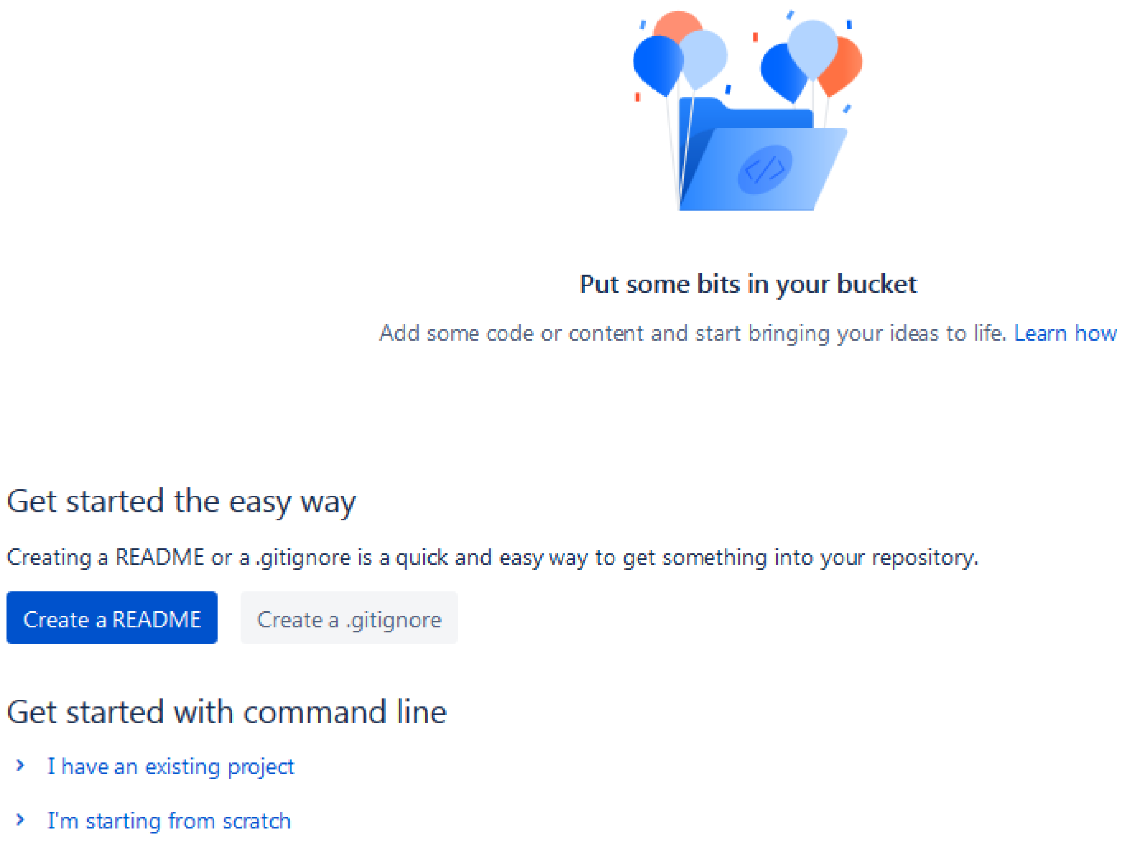
- To allow both you and your group partner to work on the project, create a remote repository on a site like Bitbucket or Github
- Make sure the repository is **private**.
- Add your group partner to the project so that they can make commits. You will both need accounts on Bitbucket/Github.
- The remote repository will also act as a backup for your assignment, allowing you to recover files if anything goes wrong
  - I recommend keeping your report in the repository as well

Source Control


Create a new repository

Owner	tachappell
Repository name*	CAB302 Assignment 2
Access level	☑ This is a private repository
Include a README?	No
Version control system	[✓ Git] Mercurial

Viewing the newly-created repository



Put some bits in your bucket

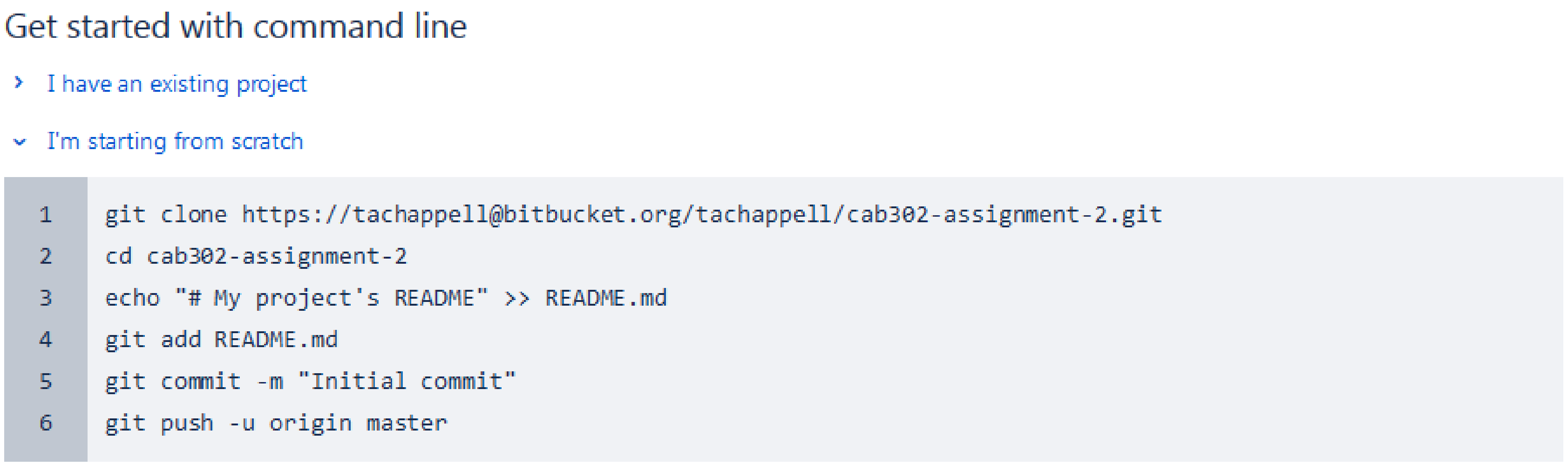
Add some code or content and start bringing your ideas to life. Learn how

Get started the easy way

Creating a README or a .gitignore is a quick and easy way to get something into your repository.

Get started with command line

Bitbucket “Getting started” instructions

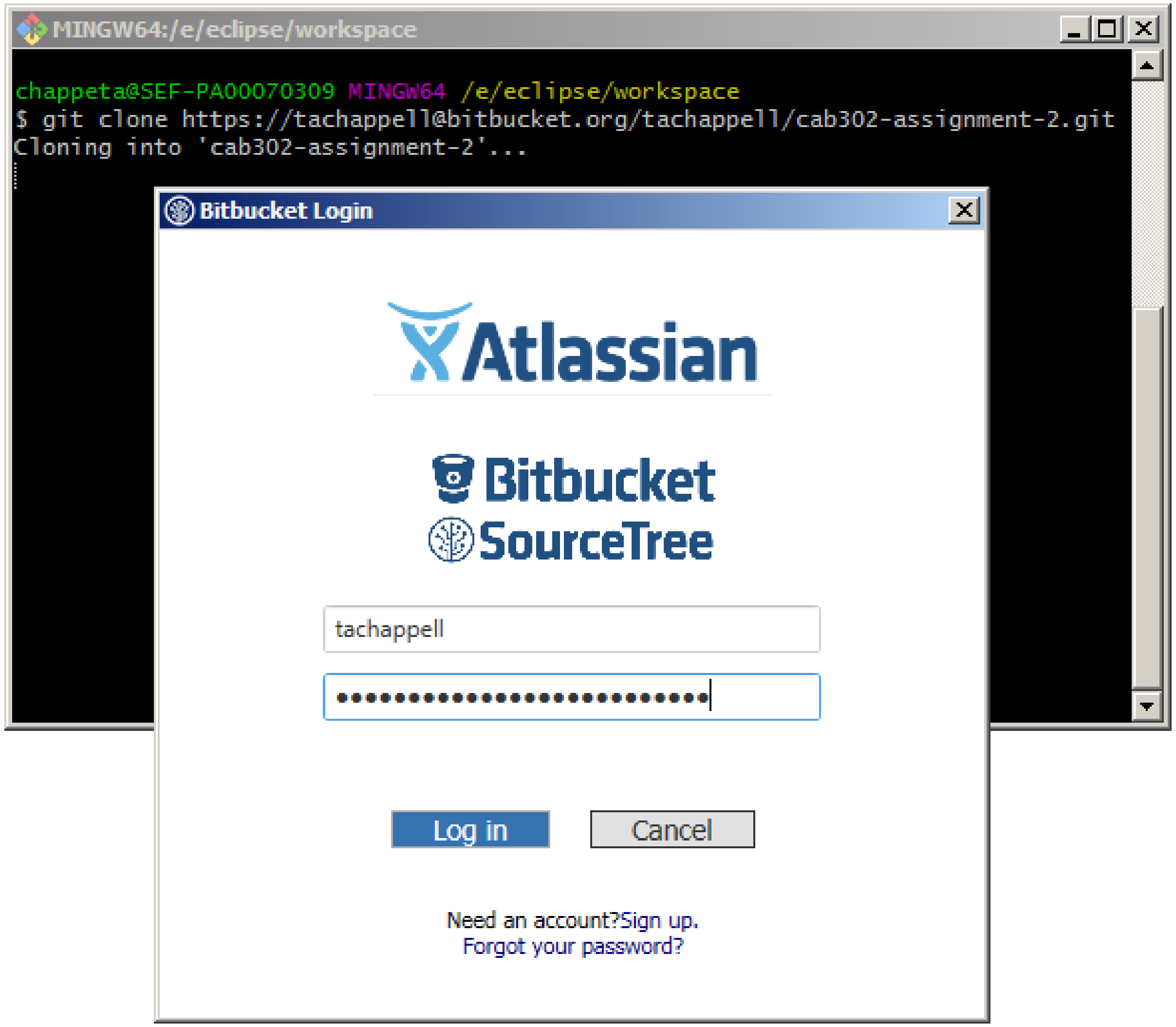

Get started with command line

› I have an existing project

⌄ I'm starting from scratch

```
git clone https://tachappell@bitbucket.org/tachappell/cab302-assignment-2.git
cd cab302-assignment-2
echo "# My project's README" >> README.md
git add README.md
git commit -m "Initial commit"
git push -u origin master
```

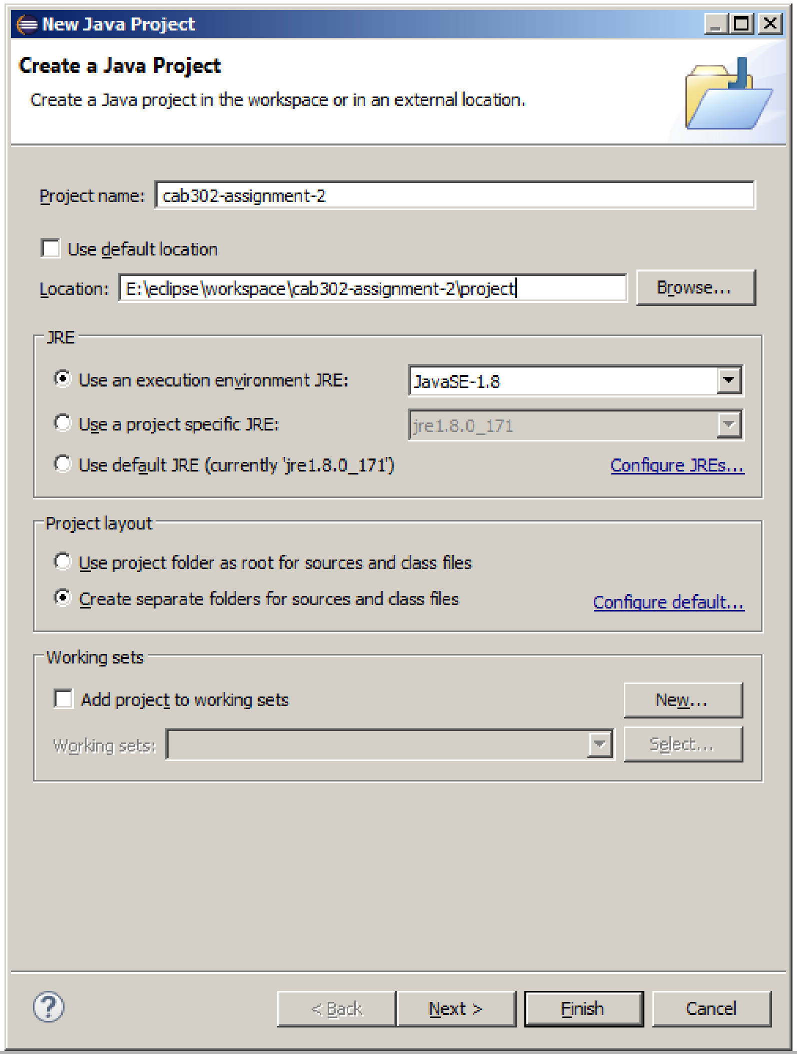
Initial clone



MINGW64:/e/eclipse/workspace

```
chappeta@SEF-PA00070309 MINGW64 /e/eclipse/workspace
$ git clone https://tachappell@bitbucket.org/tachappell/cab302-assignment-2.git
Cloning into 'cab302-assignment-2'...
```

Creating a new Java project in the repository



Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: cab302-assignment-2

[ ] Use default location

Location: E:\eclipse\workspace\cab302-assignment-2\project

JRE

[● Use an execution environment JRE:] JavaSE-1.8

( ) Use a project specific JRE: jre1.8.0_171

( ) Use default JRE (currently 'jre1.8.0_171')

Project layout

( ) Use project folder as root for sources and class files

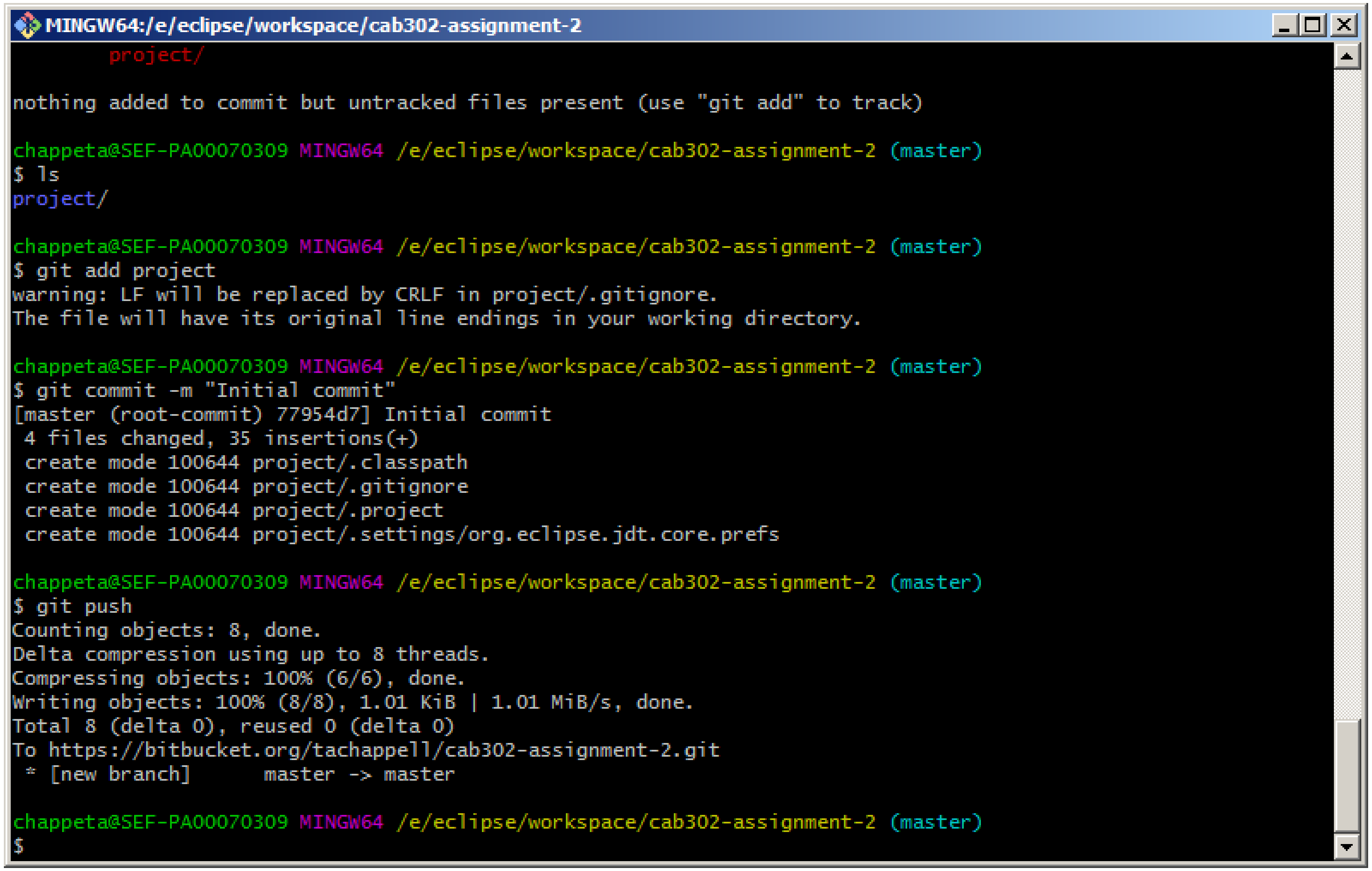
[● Create separate folders for sources and class files]

Working sets

[ ] Add project to working sets

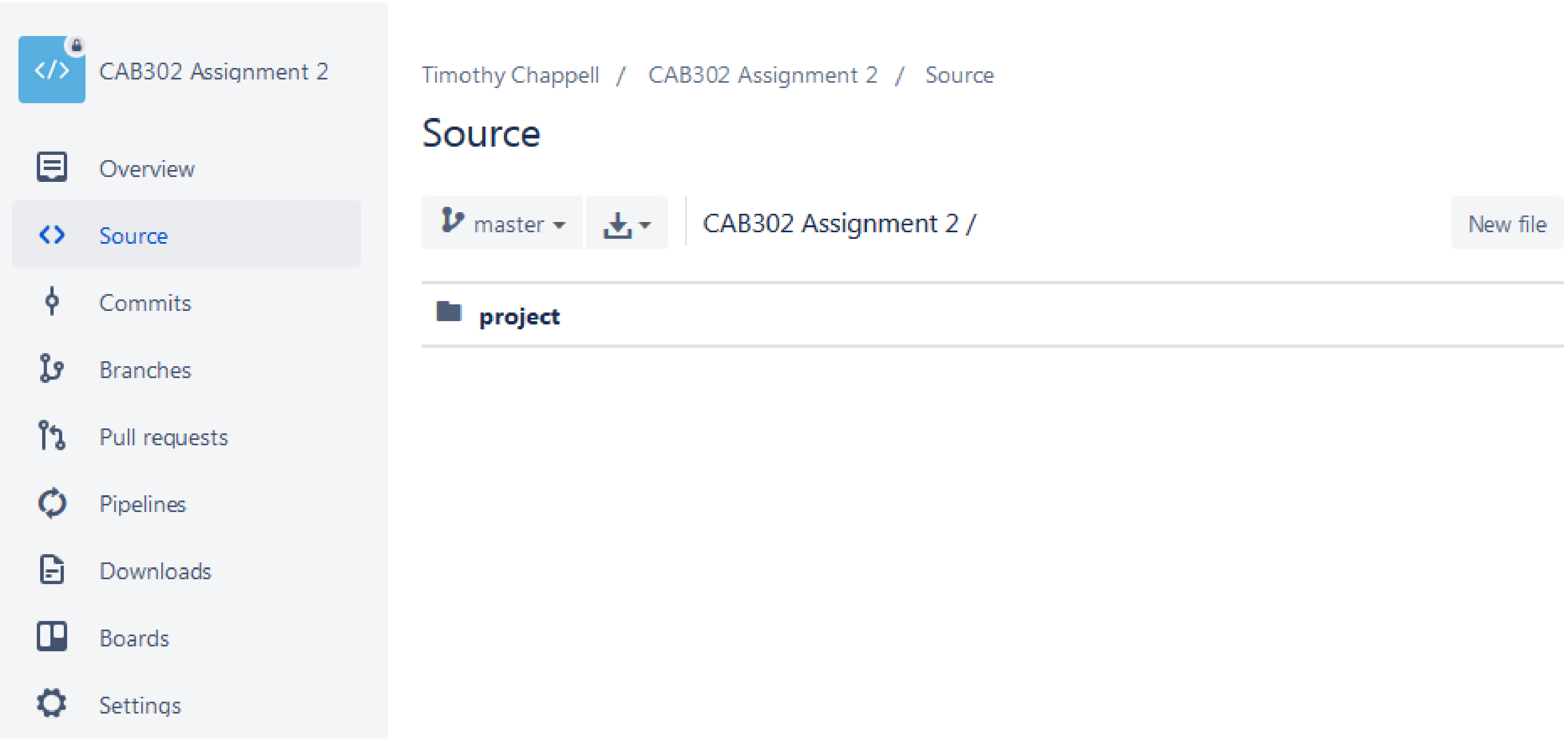
Working sets:

Pushing the initial commit to Bitbucket

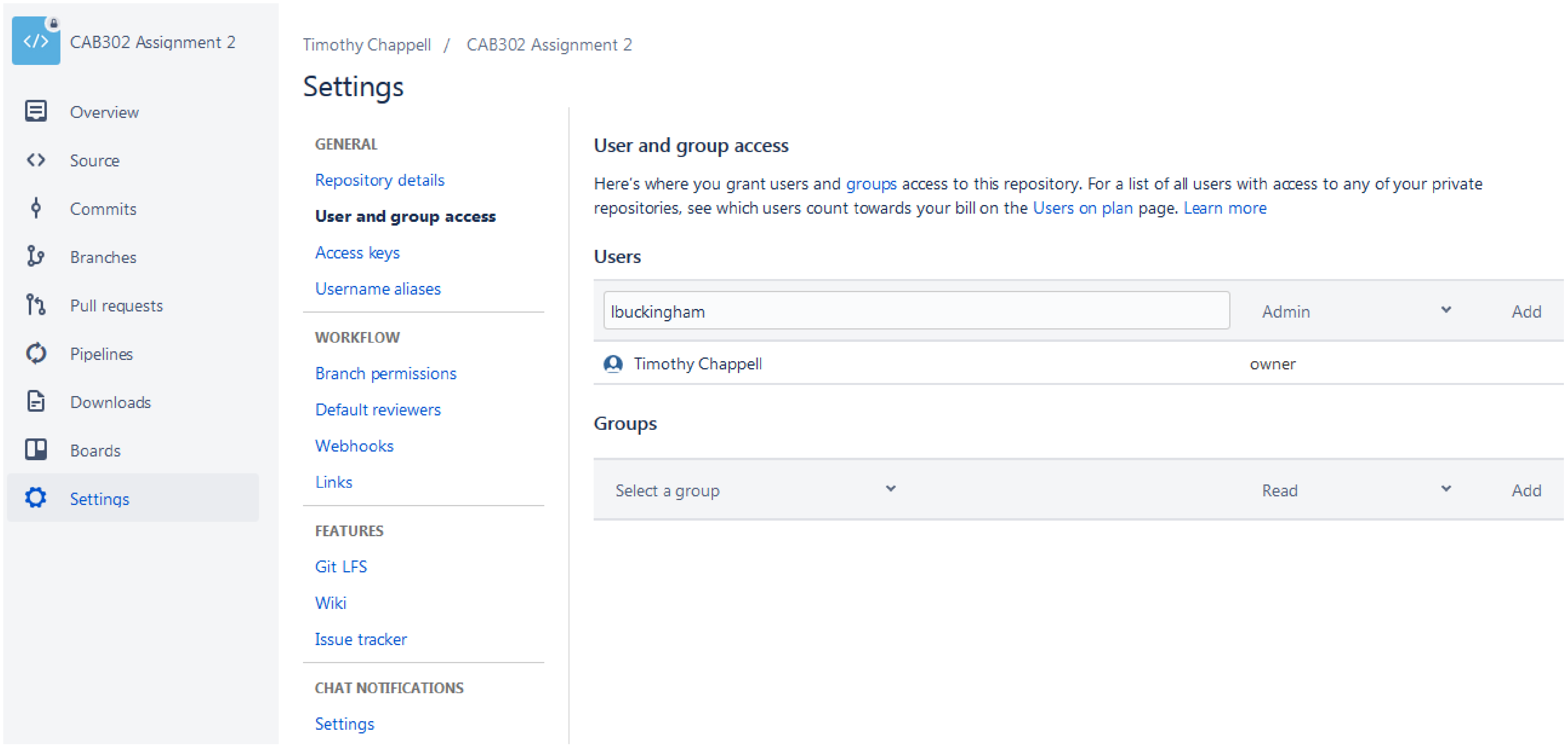

MINGW64:/e/eclipse/workspace/cab302-assignment-2

```
        project/

nothing added to commit but untracked files present (use "git add" to track)

chappeta@SEF-PA00070309 MINGW64 /e/eclipse/workspace/cab302-assignment-2 (master)
$ ls
project/

chappeta@SEF-PA00070309 MINGW64 /e/eclipse/workspace/cab302-assignment-2 (master)
$ git add project
warning: LF will be replaced by CRLF in project/.gitignore.
The file will have its original line endings in your working directory.

chappeta@SEF-PA00070309 MINGW64 /e/eclipse/workspace/cab302-assignment-2 (master)
$ git commit -m "Initial commit"
[master (root-commit) 77954d7] Initial commit
 4 files changed, 35 insertions(+)
 create mode 100644 project/.classpath
 create mode 100644 project/.gitignore
 create mode 100644 project/.project
 create mode 100644 project/.settings/org.eclipse.jdt.core.prefs

chappeta@SEF-PA00070309 MINGW64 /e/eclipse/workspace/cab302-assignment-2 (master)
$ git push
Counting objects: 8, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 1.01 KiB | 1.01 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To https://bitbucket.org/tachappell/cab302-assignment-2.git
 * [new branch]      master -> master

chappeta@SEF-PA00070309 MINGW64 /e/eclipse/workspace/cab302-assignment-2 (master)
$
```

Checking the files on Bitbucket



Adding others to the repository



CAB302 Assignment 2

Overview
Source
Commits
Branches
Pull requests
Pipelines
Downloads
Boards
Settings

Timothy Chappell / CAB302 Assignment 2

Settings

GENERAL
Repository details
User and group access
Access keys
Username aliases

WORKFLOW
Branch permissions
Default reviewers
Webhooks
Links

FEATURES
Git LFS
Wiki
Issue tracker

CHAT NOTIFICATIONS
Settings

User and group access

Here's where you grant users and groups access to this repository. For a list of all users with access to any of your private repositories, see which users count towards your bill on the Users on plan page. Learn more

Users

lbuckingham	Admin	Add
Timothy Chappell	owner	

Groups

Select a group	Read	Add

Test-Driven Development

- Use JUnit to create a comprehensive unit test suite
- Create a JUnit test class for each class in your application that can be tested
  - e.g. the Item class would have an associated ItemTest JUnit test class
- Create the test class before you write the class under testing (Test-Driven Development)
- You and your group partner should test and implement different classes, e.g.:
  - If you wrote the test case for the Stock class, your partner should implement that class from your test cases

JavaDoc

- Each class (not including JUnit test classes) should contain JavaDoc-style comments, both for the class itself and for each method in the class
- Describe the overall purpose of the class/method, as well as any parameters and return values
- Start typing /** and Eclipse will give you a template to work with
- Make sure that the author of each class and method is clearly stated in the comments (use the JavaDoc @author tag)
- Then generate JavaDoc for the project. If you change the comments later, you will need to regenerate the JavaDoc before you submit.
  - See the Week 2 lecture and Week 3 practical for information

Report

- You will need to produce a report with the following structure:
  - 1 page. Title page containing the names and student numbers of both students.
  - 1 – 2 pages. A technical description of your program architecture, drawing reference to object-oriented design concepts such as polymorphism and abstraction. You may want to use a diagram to illustrate your type hierarchies and interactions between classes.
  - 2 – 5 pages. A GUI test report that demonstrates the full range of functionality of the application, including ex

Report

- Title page
  - Self-explanatory

CAB302
Assignment
Report

By

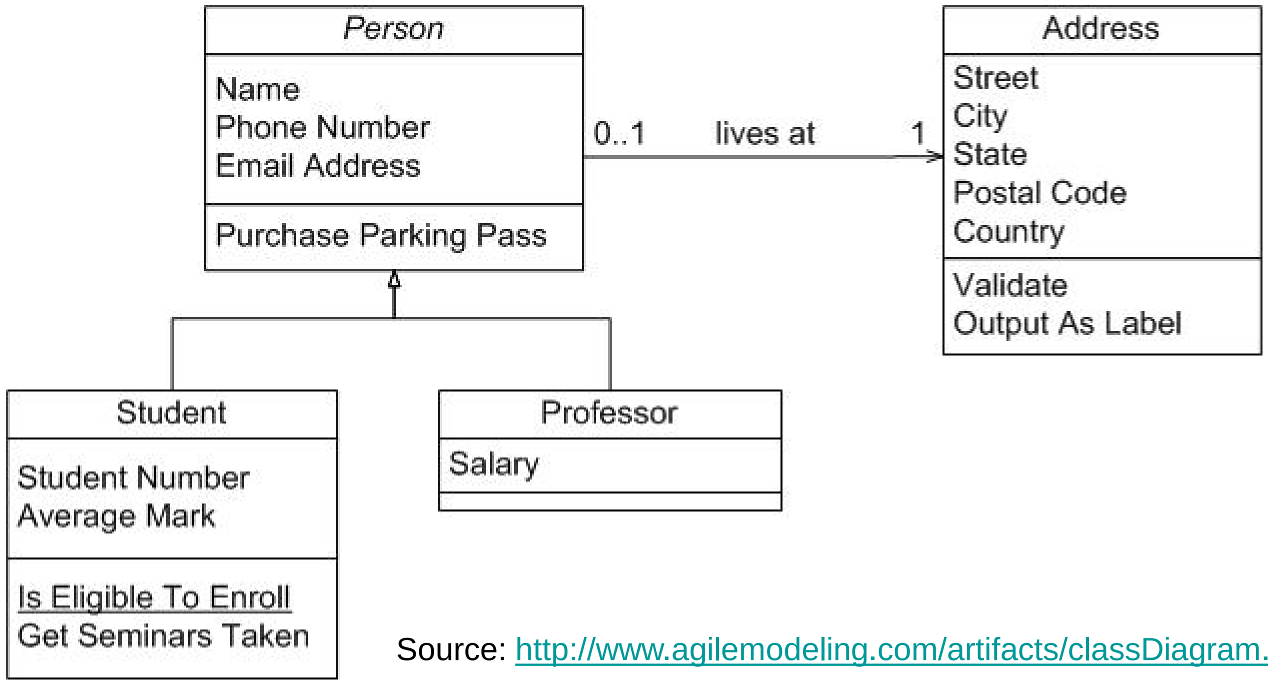
Firstname Lastname (n1234567)
Firstname Lastname (n2345678)

Report

- Program architecture description
  - Include a section for every class in your application, describing each one at a high level, talking about its methods and how it interacts with other classes
  - Draw attention to object-oriented design concepts where they are used, e.g.:
    - When classes that inherit from a shared parent are used to enable dynamic functionality, this is **polymorphism**
  - Draw attention to design patterns when they are used, e.g.:
    - If a class is part of the **observer** pattern, mention this
  - If it helps, you can use a class diagram
  - Unit tests are **not** to be documented in this report

Report

- Class diagram
  - Not necessary, but you may find it useful in writing your report
  - Search for 'class diagram' for inspiration



Source: http://www.agilemodeling.com/artifacts/classDiagram.htm

Report

- GUI test report
  - This is a 2-5 page (or longer- make this as long as you need) report showing every item of functionality, using screenshots to assist
  - Show both normal behaviour and abnormal behaviour- e.g.:
    - What happens if you load a file that isn’t in the correct format?
    - What happens if the sales log is greater than your inventory?
  - (Hint: on Windows use Alt+Printscrn to take screenshots of individual windows. If you need to get multiple windows in the one screenshot, take a fullscreen screenshot with Printscrn and crop it in a paint program like MsPaint)

Packaging your completed assignment

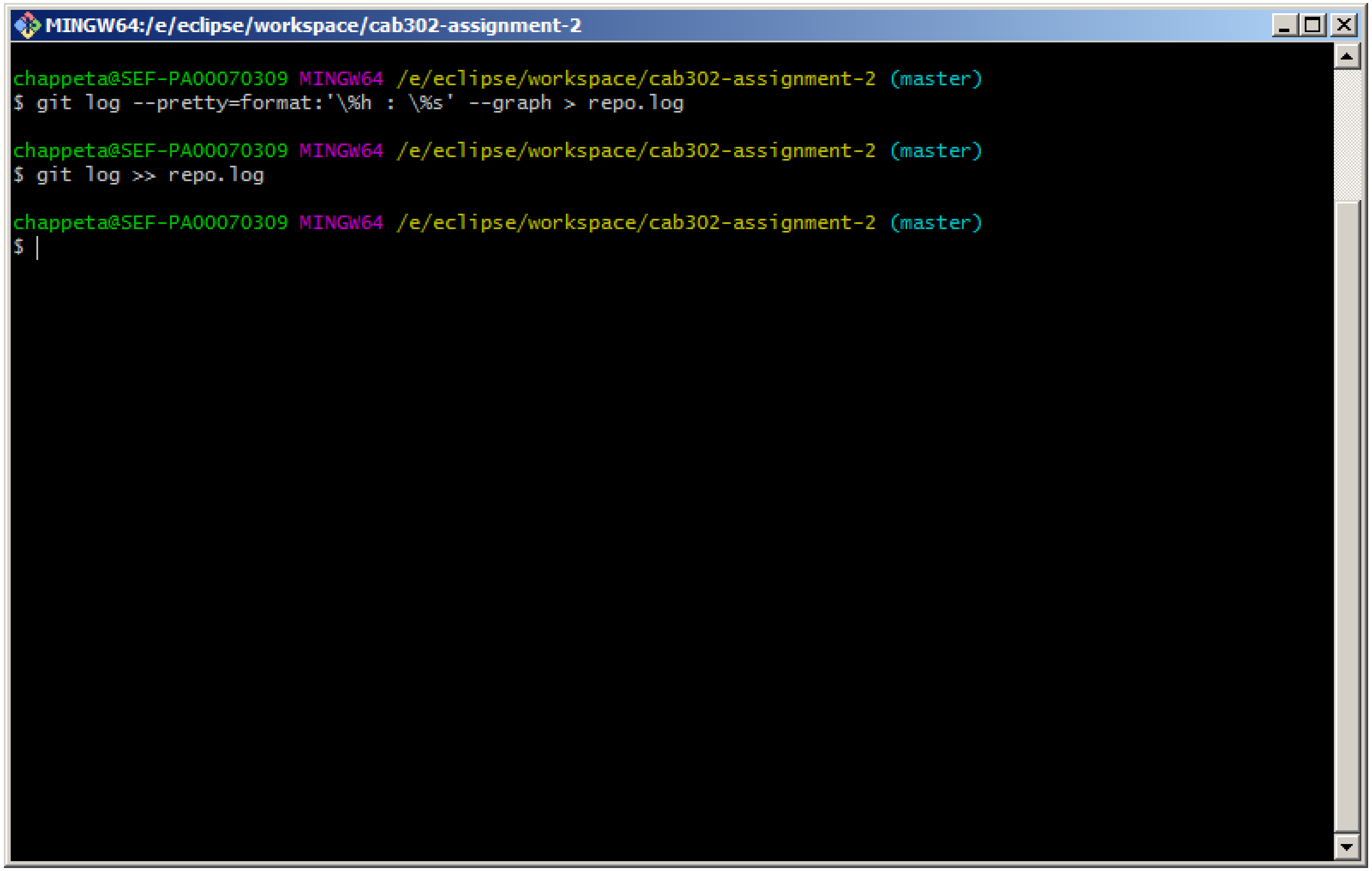
- You must submit the following things to Blackboard:
  - Your report as a PDF
    - Microsoft Word should have the ability to save as PDF or export to PDF
    - OpenOffice / LibreOffice has the same
    - If you use other tools you may need to find a converter
  - A file system export of your source code with accompanying JavaDoc
  - A git log that demonstrates usage of consistent version control.

Producing the Git log

- In the Git Bash shell, or from a terminal in Linux / Mac, navigate to your repository and issue the following commands:
  - git log --pretty=format:'\%h : \%s' --graph > repo.log
  - git log >> repo.log

- This will create a text file called repo.log containing a log of your Git commit history
- You can paste these lines into Git Bash by copying them to your clipboard, then pressing shift+enter
- If you make more commits later, you will need to run these two commands again

Producing the Git log

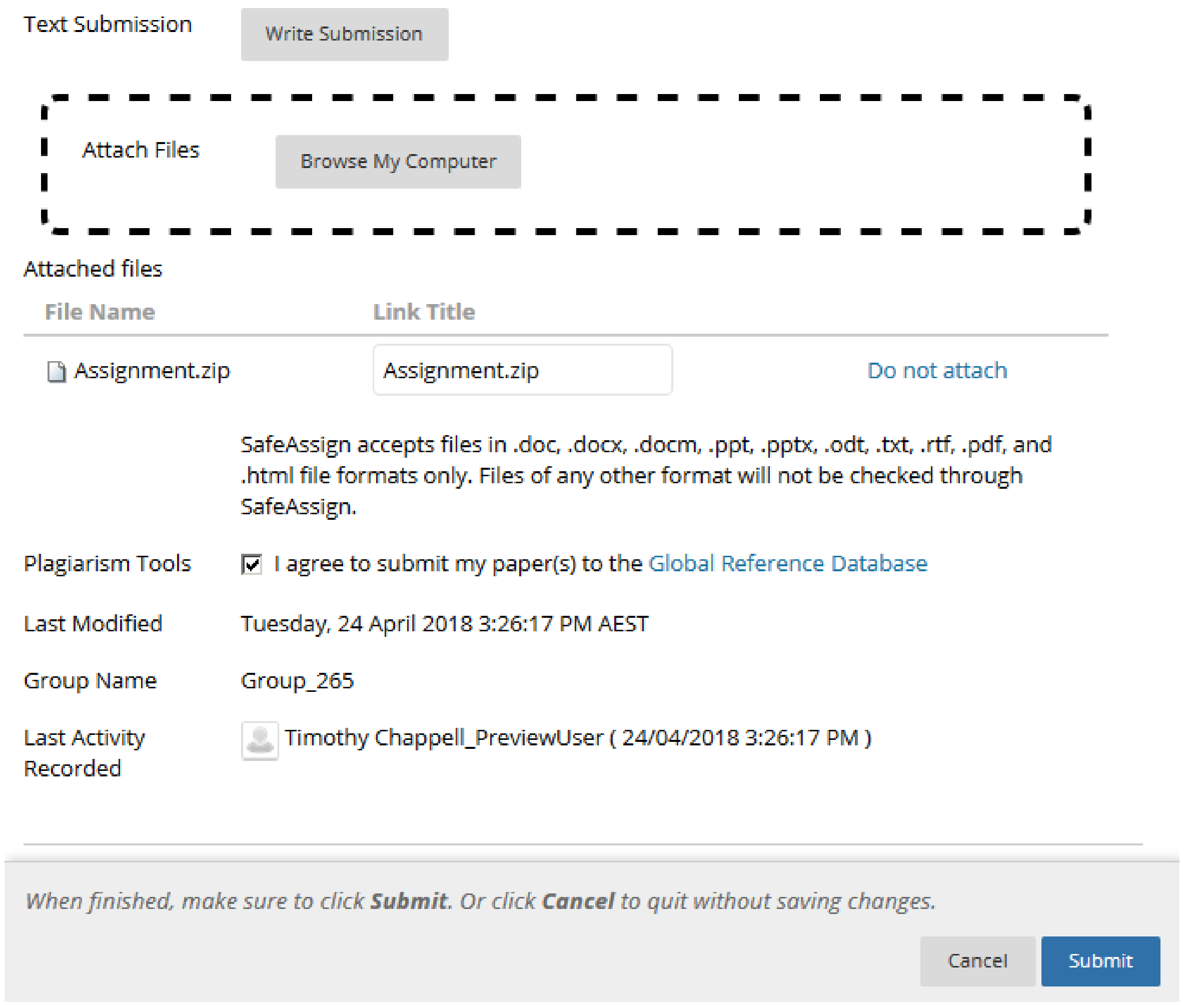


```
chappeta@SEF-PA00070309 MINGW64 /e/eclipse/workspace/cab302-assignment-2 (master)
$ git log --pretty=format:'\%h : \%s' --graph > repo.log

chappeta@SEF-PA00070309 MINGW64 /e/eclipse/workspace/cab302-assignment-2 (master)
$ git log >> repo.log

chappeta@SEF-PA00070309 MINGW64 /e/eclipse/workspace/cab302-assignment-2 (master)
$ |
```

Packaging your completed assignment

- Once all the files are ready, create a zip containing the repository with the source code, JavaDoc, git log and report, then submit this
- Eclipse can create an archive (File→Export→General→Archive File), however you will need to make sure it contains the git log and report.
  - You can add these after if necessary
- Then upload the assignment to Blackboard
  - Only one group member needs to upload the assignment

Uploading your assignment



Text Submission — Write Submission

Attach Files — Browse My Computer

Attached files

File Name	Link Title	
Assignment.zip	Assignment.zip	Do not attach

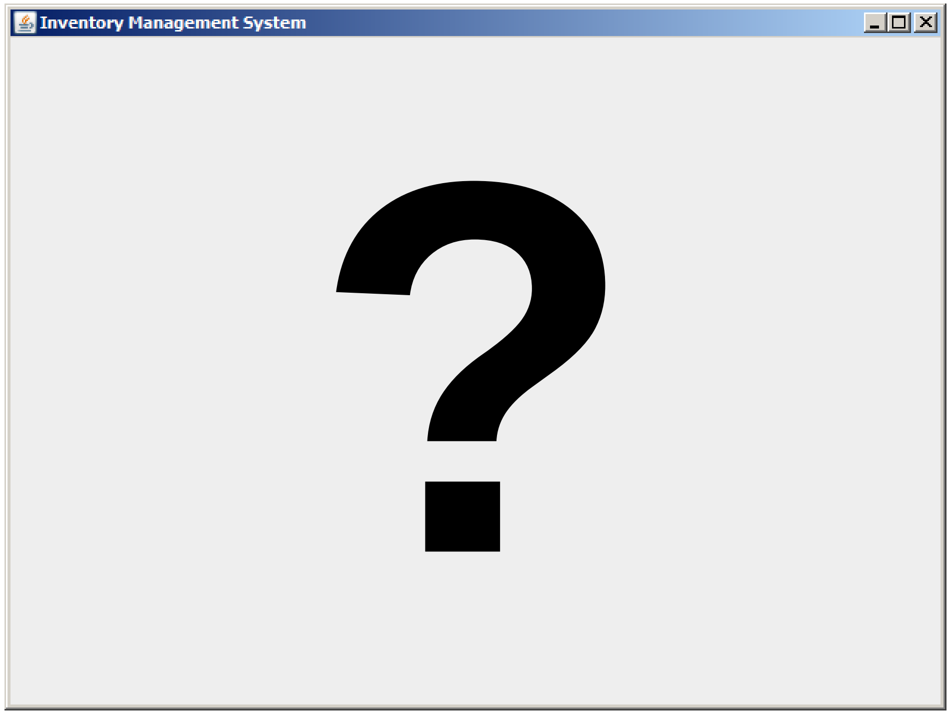
SafeAssign accepts files in .doc, .docx, .docm, .ppt, .pptx, .odt, .txt, .rtf, .pdf, and .html file formats only. Files of any other format will not be checked through SafeAssign.

Plagiarism Tools	☑ I agree to submit my paper(s) to the Global Reference Database
Last Modified	Tuesday, 24 April 2018 3:26:17 PM AEST
Group Name	Group_265
Last Activity Recorded	Timothy Chappell_PreviewUser (24/04/2018 3:26:17 PM)

*When finished, make sure to click **Submit**. Or click **Cancel** to quit without saving changes.*

Cancel Submit

GUI Design

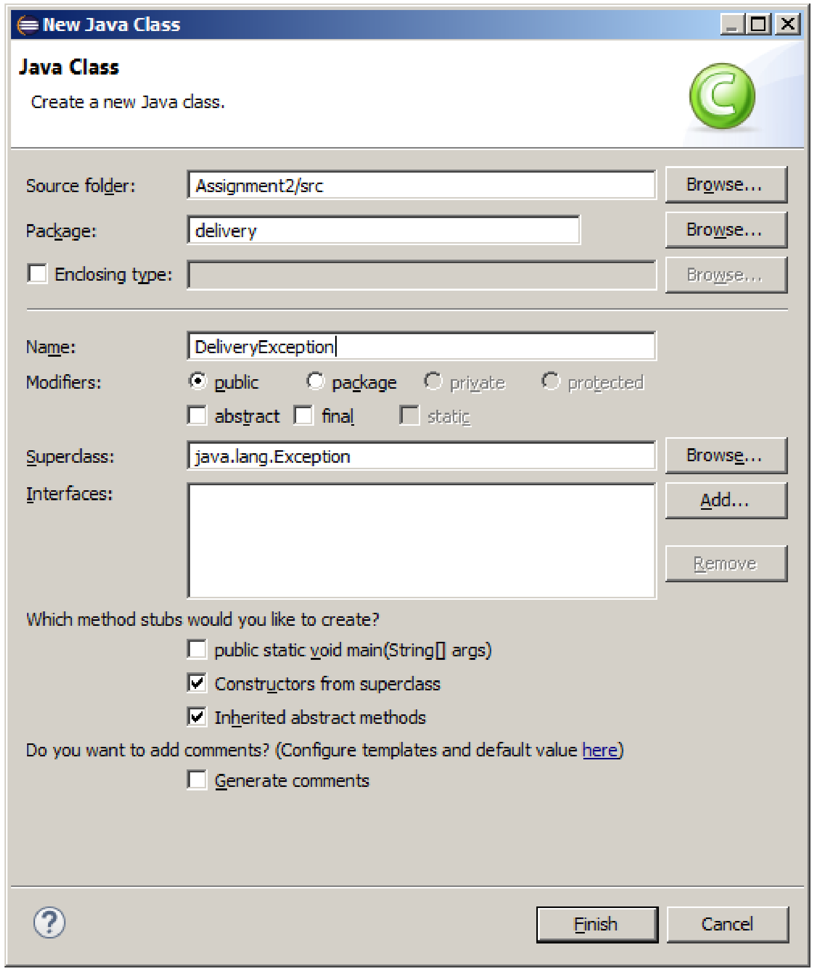


- Entirely up to you and your group partner!
- However, you will be marked on the quality of your design
- Must be implemented using Swing (covered in the lecture and practical next week)

Exceptions

- You will need to implement 3 custom exception types:
  - CSVFormatException
  - DeliveryException
  - StockException
- If it makes sense to do so, you can create other types beyond these three, but you must create at least these three
- Use them in appropriate places, e.g.:
  - CSVFormatException if the CSV file being read is not formatted correctly
  - DeliveryException if there is a problem processing the shipping manifest (e.g. it contains items not in the database)
  - StockException if there is a problem related to processing stock, for example if a sales log records more items sold than in inventory

Exceptions



New Java Class

Java Class

Create a new Java class.

Source folder:	Assignment2/src
Package:	delivery
☐ Enclosing type:	
Name:	DeliveryException
Modifiers:	[✓ public] package private protected
	☐ abstract ☐ final ☐ static
Superclass:	java.lang.Exception
Interfaces:	

Which method stubs would you like to create?

- ☐ public static void main(String[] args)
- ☑ Constructors from superclass
- ☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

- ☐ Generate comments

Exception Handling

- Exceptions need to be handled at the appropriate place, which in most cases will mean the GUI
- This means that your methods that aren’t part of the GUI should use a ‘throws’ declaration to pass the method up until it gets to a method that can handle it
- Exceptions should be handled with a popup error dialog or something similar, not written to the console (e.g. with System.out.print)

File I/O

- Your program will need to be able to read and write from CSV-format text files
- There are multiple different formats used and these are described in the specification and the example files available on Blackboard
- All are CSV (Comma-separated variable) files, that consist of lines of text strings and/or numbers separated by commas. This is a popular format as it can be read and written by many pieces of software, including Microsoft Excel
- You do not need to perfectly support the CSV format- just the simple format described in the spec and used by the example files is sufficient. No need to check for quotes or escape characters.

File I/O - Reading

- Create a FileReader from a path, then create a BufferedReader from the FileReader
- You can then read lines from that file

```java
public static void ReadTest() throws IOException {
  FileReader reader = new FileReader("filename.txt");
  BufferedReader bufferedReader = new BufferedReader(reader);
  String line = bufferedReader.readLine();
  if (line == null) {
    System.out.println("The file was empty");
  } else {
    System.out.println("The first line of the file was:");
    System.out.println(line);
  }
  bufferedReader.close();
}
```

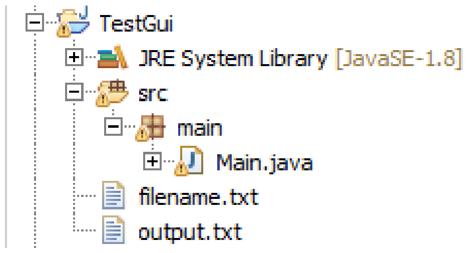
File I/O - Writer

- Create a FileWriter from a path
- You can the write lines from the file

```
public static void WriteTest() throws IOException
{
  FileWriter writer = new FileWriter("output.txt");
  writer.write("This is a newly-created text file\n");
  writer.write("It contains lines of text\n");
  writer.close();
}
```

File I/O - Locations

- When running the program from Eclipse, file locations are relative to the project directory
- If you write a file that did not previously exist, it will not appear instantly- you will need to refresh Eclipse
- If you use a file selection dialog in Swing, the user will be able to specify a file anywhere on their file system and you will get an absolute path you can pass to file I/O functions.



File I/O - Exceptions

- File I/O methods tend to throw checked exceptions (like java.io.IOException)
- This is because many unexpected things can happen when opening files (e.g. file does not exist, you don’t have permissions to read/write a particular file, file closes unexpectedly while you are reading it)
- You will need to handle these appropriately, at the correct level (e.g. a missing or incorrectly-formatted file should display a dialog box with an error message in it)

File I/O – Reading and Writing CSVs

?

- Left as an exercise for the student
- Writing is easy, but reading is more difficult
- There are String methods that you will find useful for breaking a line you just read from a file up- consult the Java API documentation.

Functionality

- Functionality marks are awarded through running your program, not looking at your code
- If your program runs and does everything it is supposed to, you will get the marks in the functionality section of the rubric regardless of what the code looks like
- Conversely, if your program does not run or operates incorrectly, you will lose those marks, even if you've written code handling that functionality and it is totally bug-free

Functionality

- Required functionality for this assignment is expressed in terms of features requested by the client
- These are *non-technical feature requests* – the client has not considered things like exceptional behaviour because they expect you to do the right thing
- How well your application implements these features will determine your Functionality grade

Functionality

- Client Requirement 1:
  View store capital in dollars
  - Self-explanatory
  - Should be formatted with groups of three digits separated by commas, with two decimal places and a $ in front
  - e.g. $100,000.00

Functionality

- Client Requirement 2:
  View store inventory in tabular format

Name	Quantity	Cost ($)	Price ($)	Reorder point	Reorder quantity	Temperature (°C)
rice	252	2	3	225	300	N/A
milk	397	2	3	300	425	3
cheese	418	4	7	375	450	3
ice cream	209	8	14	175	250	-20

Functionality

- Client Requirement 3:
  Load in item properties
- Your program should load a CSV file of the Item Properties format

```
rice,2,3,225,300
beans,4,6,450,525
pasta,3,4,125,250
biscuits,2,5,450,575
```

- An example file in this format is available on Blackboard (item_properties.csv)
- This data will be used to initialise the store's item database with different items, their prices, required temperatures and reorder points
- Note that some items have a maximum temperature, while others do not. This is needed for creating shipping manifests

Functionality

- Client Requirement 4:
  Export manifests based on current inventory
- When the user elects to export a manifest, your program should go through the entire store inventory, find all items where the amount in stock is less than the reorder point and create a shipping manifest ordering the reorder quantity of each item.

Name	Quantity	Reorder point	Reorder quantity
rice	0	225	300
beans	0	450	525
pasta	0	125	250
biscuits	0	950	1075

$\longrightarrow$

```
>Ordinary
rice,300
beans,525
pasta,175
>Ordinary
pasta,75
biscuits,925
>Ordinary
biscuits,150
```

Functionality

- The manifest file should be saved to a location of the user's choice and must follow the format described by the specification and example documents

```
>[truck type]
[item],[quantity]
[item],[quantity]
[item],[quantity]
>[truck type]
[item],[quantity]
[item],[quantity]
[item],[quantity]
```

Functionality

- There are two types of truck that can be used in a shipping manifest, ordinary and refrigerated trucks
- Ordinary trucks:
  - Capacity of 1000 items
  - Priced based on the quantity of items in them:
    - Price = 750 + 0.25*q* where *q* is the number of items in the truck
  - Cannot hold items that have a temperature requirement (e.g. ice cream, frozen dinners)
- Refrigerated trucks:
  - Capacity of 800 items
  - Priced based on the temperature:
    - Price = $900 + 200 * 0.7^{t/5}$, *t* is the truck's temperature (°C)

Functionality

- Ordinary trucks are always cheaper than refrigerated trucks, even at minimum capacity
- Refrigerated trucks are priced based on the coldest item in them as the entire truck needs to be kept at this temperature
- Items that **do not** need refrigeration can be stored in a refrigerated truck or an ordinary truck
- Items that **do** need refrigeration can only be stored in a refrigerated truck
- You should generate a shipping manifest to cost the company the least amount of money necessary

Functionality

- Client Requirement 5:
  Load shipping manifests
- When the user elects to load a manifest, they should be presented with a file selection dialog that allows them to load a shipping manifest file of the format exported by Client Requirement 4.
- The shipping manifest file must be read in, the price of each of the trucks and the wholesale cost of the goods within calculated to determine how this order will reduce the store's capital
- Then the items delivered by the trucks will be added to the store's inventory
- If the cost of the shipping manifest exceeds the store's capital, the store will go into debt, but this is okay (SuperMart has a good relationship with the local bank)

Functionality

Capital: $100,000

Name	Quantity	Reorder point	Reorder quantity
rice	0	225	300
beans	0	450	525
pasta	0	125	250
biscuits	0	950	1075

+

```
>Ordinary
rice,300
beans,525
pasta,175
>Ordinary
pasta,75
biscuits,925
>Ordinary
biscuits,150
```

=

Capital: $91612.50

Name	Quantity	Reorder point	Reorder quantity
rice	300	225	300
beans	525	450	525
pasta	250	125	250
biscuits	1075	950	1075

Functionality

- Client Requirement 6:
  Load sales logs
- When the user elects to load a sales log, they should be presented with a file selection dialog that allows them to load a sales log of the CSV format specified in the specification and sample files

```
[item],[quantity]
[item],[quantity]
[item],[quantity]
```

- This sales log will work in the opposite way to a manifest – the store's capital will increase by the sale price of the goods, while the inventory is decreased by the same amount
- If the store's inventory does not have all the items in the list in sufficient quantity, it should throw an exception and **not** change the store's capital or inventory

Functionality

Capital: $91612.50

Name	Quantity	Reorder point	Reorder quantity
rice	300	225	300
beans	525	450	525
pasta	250	125	250
biscuits	1075	950	1075

+

```
rice,300
beans,525
pasta,250
biscuits,1075
```

=

Capital: $102,037.50

Name	Quantity	Reorder point	Reorder quantity
rice	0	225	300
beans	0	450	525
pasta	0	125	250
biscuits	0	950	1075

Questions

- Any questions?

Thank You

Dr. Timothy Chappell
EECS | Data Science
cab302@qut.edu.au