MODULE NAME:	MODULE CODE:
PROGRAMMING 1A	PROG5121/d
PROGRAMMING 1A	PROGf5111

ASSESSMENT TYPE: POE (PAPER AND MARKING RUBRICS)

TOTAL MARK ALLOCATION: 100 MARKS

TOTAL HOURS: A MINIMUM OF 15 HOURS IS SUGGESTED TO COMPLETE THIS ASSESSMENT

By submitting this assignment, you acknowledge that you have read and understood all the rules as per the terms in the registration contract, in particular the assignment and assessment rules in The IIE Assessment Strategy and Policy (IIE009), the intellectual integrity and plagiarism rules in the Intellectual Integrity Policy (IIE023), as well as any rules and regulations published in the student portal.

INSTRUCTIONS:

1. ***No material may be copied from original sources, even if referenced correctly, unless it is a direct quote indicated with quotation marks. No more than 10% of the assignment may consist of direct quotes.***
2. ***Make a copy of your assignment before handing it in.***
3. *Assignments must be typed unless otherwise specified.*
4. *Begin each section on a new page.*
5. *Follow all instructions on the PoE cover sheet.*
6. *This is an individual assignment.*

Referencing Rubric

Providing evidence based on valid and referenced academic sources is a fundamental educational principle and the cornerstone of high-quality academic work. Hence, The IIE considers it essential to develop the referencing skills of our students in our commitment to achieve high academic standards. Part of achieving these high standards is referencing in a way that is consistent, technically correct and congruent. This is not plagiarism, which is handled differently.

Poor quality formatting in your referencing will result in a penalty **of a maximum of ten percent being deducted from the percentage awarded**, according to the following guidelines. Please note, however, that **evidence of plagiarism in the form of copied or uncited work (not referenced), absent reference lists, or exceptionally poor referencing, may result in action being taken in accordance with The IIE's Intellectual Integrity Policy (0023)**.

Markers are required to provide feedback to students by indicating **(circling/underlining) the information that best describes the student's work.**

Minor technical referencing errors: 5% deduction from the overall percentage – the student's work contains **five or more errors** listed in the minor error's column in the table below.

Major technical referencing errors: 10% deduction from the overall percentage – the student's work contains **five or more errors** listed in the major error's column in the table below**.**

If both minor and major errors are indicated, then 10% only (and not 5% or 15%) is deducted from the overall percentage. The examples provided below are not exhaustive but are provided to illustrate the error

Required: **Technically correct referencing style**	**Minor errors in technical correctness of referencing style** **Deduct 5% from percentage awarded**	**Major errors in technical correctness of referencing style** **Deduct 10% from percentage awarded**
Consistency • The same referencing format has been used for all in-text references and in the bibliography/reference list.	Minor inconsistencies. • The referencing style is generally consistent, but there are one or two changes in the format of in-text referencing and/or in the bibliography. • For example, page numbers for direct quotes (in-text) have been provided for one source, but not in another instance. Two book chapters (bibliography) have been referenced in the bibliography in two different formats.	Major inconsistencies. • Poor and inconsistent referencing style used in-text and/or in the bibliography/ reference list. • Multiple formats for the same type of referencing have been used. • For example, the format for direct quotes (in-text) and/or book chapters (bibliography/ reference list) is different across multiple instances.
Technical correctness Referencing format is technically correct throughout the submission. Position of the reference: a reference is directly associated with every concept or idea. For example, quotation marks, page numbers, years, etc. are applied correctly, sources in the bibliography/reference list are correctly presented.	**Generally, technically correct with some minor errors.** • The correct referencing format has been consistently used, but there are one or two errors. • Concepts and ideas are typically referenced, but a reference is missing from one small section of the work. • Position of the references: references are only given at the beginning or end of every paragraph. • For example, the student has incorrectly presented direct quotes (in-text) and/or book chapters (bibliography/reference list).	**Technically incorrect.** • The referencing format is incorrect. • Concepts and ideas are typically referenced, but a reference is missing from small sections of the work. • Position of the references: references are only given at the beginning or end of large sections of work. • For example, incorrect author information is provided, no year of publication is provided, quotation marks and/or page numbers for direct quotes missing, page numbers are provided for paraphrased material, the incorrect punctuation is used (in-text); the bibliography/reference list is not in alphabetical order, the incorrect format for a book chapter/journal article is used, information is missing e.g. no place of publication had been provided (bibliography); repeated sources on the reference list.
Congruence between in-text referencing and bibliography/ reference list • All sources are accurately reflected and are all accurately included in the bibliography/ reference list.	**Generally, congruence between the in-text referencing and the bibliography/ reference list with one or two errors.** • There is largely a match between the sources presented in-text and the bibliography. • For example, a source appears in the text, but not in the bibliography/ reference list or vice versa.	**A lack of congruence between the in-text referencing and the bibliography.** • No relationship/several incongruencies between the in-text referencing and the bibliography/reference list. • For example, sources are included in-text, but not in the bibliography and vice versa, a link, rather than the actual reference is provided in the bibliography.
In summary: the recording of references is accurate and complete.	In summary, at least **80%** of the sources are correctly reflected and included in a reference list.	In summary, at least **60%** of the sources are incorrectly reflected and/or not included in reference list.

Overall Feedback about the consistency, technical correctness and congruence between in-text referencing and bibliography:

.....

.....

Read this:

Anyone can code and create software however, good software engineers create software that is testable, scalable, and maintainable. Good quality, clean code is a fine art that is not easy to achieve. We would like to start teaching you how to do this from day one. This POE will thus require that you not only create your solution, but also to:

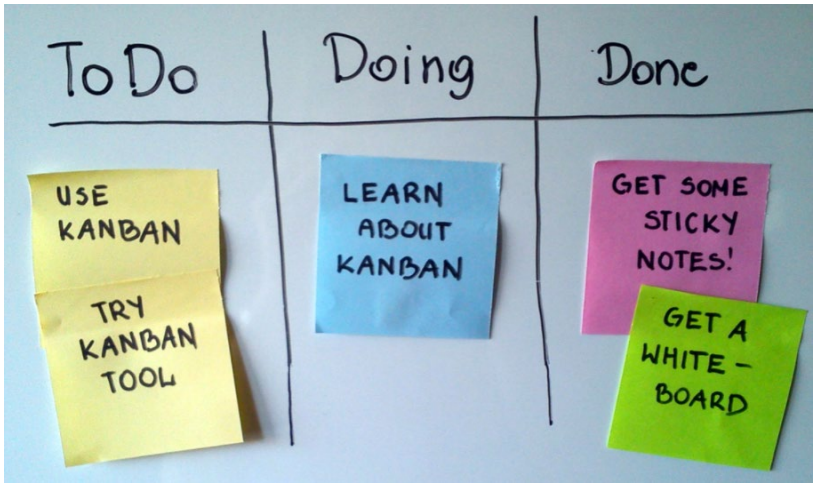
- Make use of version control (git) – we will be using GitHub.
- Test your software using unit tests – we will be using Junit.
- Automate these tests so that your code is tested with every change you make.

You will receive step ty step instructions and video resources to help you make use of the above-mentioned tools.

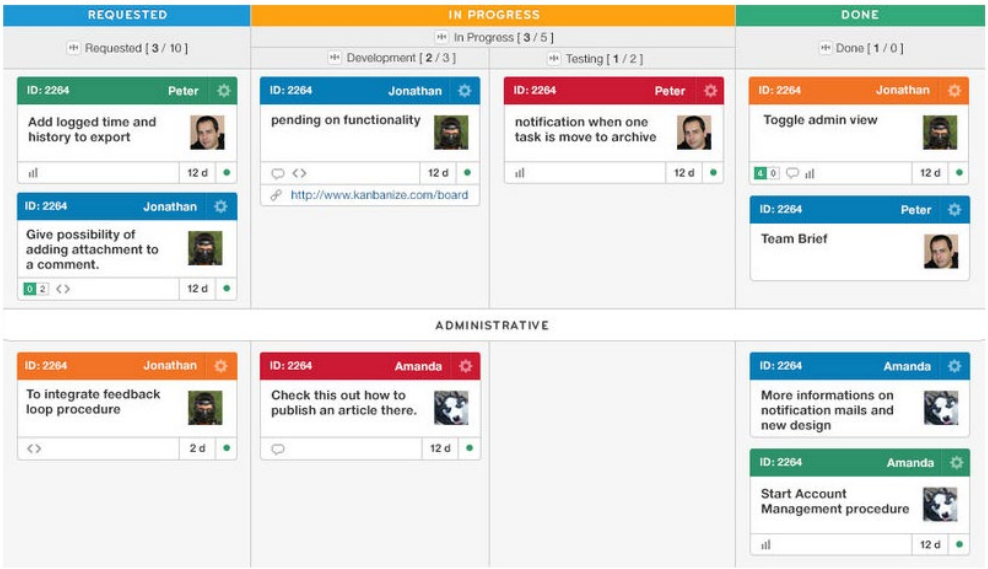
What are we building?

Many development houses make use of Kanban boards to manage their projects. The image below is an example of a very basic Kanban board, these boards are normally divided into three different lanes:

- **To Do:** Which contains new tasks or tasks that have not yet been started.
- **Doing:** For tasks that are in progress.
- **Done:** For completed tasks.



More advanced Kanban boards also indicate which developer is assigned to the task, the duration of the task and a task ID (see below)



You can read up more on Kanban boards here : https://www.atlassian.com/agile/kanban/boards
This Portfolio of Evidence will allow you to build this system by adding one feature at a time in 3 different code sprints (tasks).

Finally:

This POE is practical in nature and relies on the theoretical knowledge you will be gaining while completing the PRLD5121 module which covers the theoretical knowledge of the following coding constructs:

- Variables and variable scope
- Data types
- Classes, methods, and interfaces
- Operators
- Operator precedence
- Decisions
- Loops
- Arrays

You will practically implement each of these constructs in this POE.

Instructions

You will need to make sure that you have the following to start this POE.

1. A GitHub account (Please user your connect account to sign up) – you can sign up here: https://github.com/
2. **Next apply for the GitHub student developer pack here (GitHub will take 7 days to process your application)** https://education.github.com/pack
3. Make sure you have received and accepted the GitHub organization invite from your campus. You can use these instructions to assist you: https://docs.idalko.com/exalate/display/ED/Accept+an+invitation+in+Exalate+for+GitHub
4. Create a private repository in the organization: https://docs.github.com/en/get-started/quickstart/create-a-repo
5. You will be asked to write unit tests to ensure that your code is working correctly – please watch the following video to prepare: https://www.youtube.com/playlist?list=PL480DYS-b_kfHSYf2yzLgto_mwDr_U-Q6

Part 1 — Registration and login feature (Marks: 40)

At the end of this specific task, students should be able to:

- Create classes, methods, and other OOP programming constructs.
- Use decisions
- Produce an application that accepts input and returns output
- (Learning unit 1-4)

Your very first task is to create a registration and login feature. This feature needs to allow users to (Read through the entire task before you start any work):

1. **Create an account by entering username, password, first name and last name.**
   a. The system needs to check that the following conditions are met, and reply with the appropriate output message:

Conditions	Messages	
	True	**False**
Username contains an underscore and is no more than 5 characters long	"Username successfully captured"	"Username is not correctly formatted, please ensure that your username contains an underscore and is no more than 5 characters in length ."
Password meets the following password complexity rules, the password must be: • At least 8 characters long • Contain a capital letter • Contain a number • Contain a special character	"Password successfully captured"	"Password is not correctly formatted, please ensure that the password contains at least 8 characters, a capital letter, a number and a special character."

2. **Login to the account using the same username and password.**
    a. The system should provide the following messages to verify the user's authentication state:

Conditions	Messages	
	True	**False**
The entered username and password are correct, and the user is able to log in.	"Welcome <user first name> ,<user last name> it is great to see you again.	"Username or password incorrect, please try again"

3. You will need to implement a **Login** class with the following methods to ensure that your application meets good coding standards and that the code you write is testable.

Method Name	Method Functionality
Boolean: checkUserName()	This method ensures that any username contains an under score (_) and is no more than
Boolean: checkPasswordComplexity()	This method ensures that passwords meet the following password complexity rules, the password must be: • At least eight characters long. • Contain a capital letter • Contain a number • Contain a special character
String registerUser()	This method returns the necessary registration messaging indicating if: • The username is incorrectly formatted • The password does not meet the complexity requirements. • The two above conditions have been met and the user has been registered successfully.
Boolean loginUser()	This method verifies that the login details entered matches the login details stored when the user registers.
String returnLoginStatus	This method returns the necessary messaging for: • A successful login • A failed login

4. It is good practice to never push code that has not been tested, you will need to create the following unit tests to verify that your methods are executing as expected:

Test: (assertEquals)	Test Data and expected system responses.
Username is correctly formatted: The username contains an underscore and is no more than 5 characters long	**Test Data: "kyl_1"**
	The system returns: "Welcome <user first name> ,<user last name> it is great to see you."
Username incorrectly formatted:	**Test Data: "kyle!!!!!!!"**

The username does not contain an underscore and is no more than 5 characters long	**The system returns:** "Username is not correctly formatted, please ensure that your username contains an underscore and is no more than 5 characters in length."
The password meets the complexity requirements	**Test Data:** "Ch&&sec@ke99!"
	The system returns: "Password successfully captured"
The password **does not** meet the complexity requirements	**Test Data:** "password"
	The system returns: "Password is not correctly formatted, please ensure that the password contains at least 8 characters, a capital letter, a number and a special character."
Test (assertTrue/False)	
Login Successful	**The system returns:** True
Login Failed	**The system returns:** False
Username correctly formatted	**The system returns:** True
Username incorrectly formatted	**The system returns:** False
Password meets complexity requirements	**The system returns:** True
Password does not meet complexity requirements	**The system returns:** False

5. Watch the following video to help you create the necessary unit tests in NetBeans: https://www.youtube.com/watch?v=2EIUHHoVfmU [22 February 2022]

***** Make sure to use the test data detailed in the table for assertEquals as this will be used to mark your task.***

6. Finally, developers make use of Continuous Integration and Continuous Deployment (CI/CD) pipelines to iteratively build systems and to test not only the functionality but also the quality of their code. It is good practice to start working with at least CI in mind. We will be implementing GitHub actions to:
    a. Automate the tests we have written to run whenever we updated our code.
        i. Make sure you have signed up for the GitHub student developer pack
        ii. Follow the steps detailed below to automate your tests using GitHub Actions: https://www.youtube.com/watch?v=b3cIRsVPLR4&t=282s [Accessed 22 February 2022].

Part 2 — Adding Tasks feature (Marks: 55)

At the end of this specific task, students should be able to:

- *Create and work with Loops*
- *Handle and manipulate strings*
- *(Learning Units 4 and 5).*

It is good practice to leave your main branch as your long live branch, this means that the code on this branch is always in perfect working order and tested. We make use of feature branches in order to ensure that any code we push to GitHub does not break our main branch. You can create a feature branch by running the following command.

`git checkout -b KhanbanTasks` (you can use any branch name)

***** You are welcome to make use of GitHub desktop or your IDE to push code to GitHub if you are not comfortable with using the command line.***

You can now add the following functionality to your application:

1. The users should only be able to add tasks to the application if they have logged in successfully.
2. The applications must display the following welcome message: "**Welcome to EasyKanban".**
3. The user should then be able to choose one of the following features from a numeric menu:

a. Option 1) Add tasks

b. Option 2) Show report - this feature is still in development and should display the following message: **“Coming Soon”.**

c. Option 3) Quit

4. The application should run until the users selects quit to exit.
5. Users should define how many tasks they wish to enter when the application starts, the application should allow the user to enter only the set number of tasks.
6. Each task should contain the following information:

Task Name	The name of the task to be performed: “Add Login Feature”
Task Number	Tasks start with the number 0, this number is incremented and autogenerated as more tasks are added .
Task Description	A short description of the task, this description **should not exceed 50 characters** in length. The following error message should be displayed if the task description is too long: ***“Please enter a task description of less than 50 characters”*** *OR* ***“Task successfully captured”*** if the message description meets the requirements.
Developer Details	The first and last name of the developer assigned to the task.
Task Duration	The estimated duration of the task in hours. This number will be used for calculations and should make use of an appropriate data type.
Task ID	The system must autogenerate a TaskID which contains the first two letters of the Task Name, a colon (:), the Task Number, a colon (:) and the last three letters of the developer

	assigned to the task's name. The ID should be displayed in all caps: AD:0:INA
Task Status	The user should be given a menu to select the following task statuses from: • To Do • Done • Doing

7. The full details of each task should be displayed on the screen (using JOptionPane) after it has been entered and should show all the information requested in the table above in the following order: Task Status, Developer Details, Task Number, Task Name, Task Description, Task ID and Duration;
7. The total number of hours across all tasks should be accumulated and displayed once all the tasks has been entered.

Create a **Task class** that contains the following messages:

Method Name	**Method Functionality**
Boolean: checkTaskDescription()	This method ensures that the task description is not more than 50 characters.
String: createTaskID()	This method creates and returns the taskID
String: printTaskDetails()	This method returns the task full task details of each task.
Int: returnTotalHours()	This method returns the total combined hours of all entered tasks.

8. Please use the following the following test data to create unit tests.

Test Data:	
Num Tasks	2
Test Data for Task 1	
Task Name	"Login Feature"
Task Number	Auto generated.
Task Description	"Create Login to authenticate users"
Developer Details	Robyn Harrison
Task Duration	8hrs
TaskID	Auto generated
Task Status	To Do

Test Data for Task 2	
Task Name	"Add Task Feature"
Task Number	Auto generated.
Task Description	"Create Add Task feature to add task users"
Developer Details	Mike Smith
Task Duration	10hrs
TaskID	Auto generated
Task Status	Doing

9. Create the following unit tests:

Test AssertEquals:	
Task Description should **not be** more than 50 Characters	Test for both success and failure **<u>The system should return:</u>** **<u>Success</u>** ***"Task successfully captured"*** **<u>Failure:</u>** ***"Please enter a task description of less than 50 characters"***

TaskID is correct	**The system should return:** AD:1:BYN When supplied with the data from Test case 1 The system should test the remainder of the TaksIDs in a loop (refer to video): CR:0:IKE, CR:1:ARD, CR:2:THA, CR:3:ND
Total Hours Correctly accumulated in loop	**Additional test data:** 1)Test Data for Task1 and Task2. 2: **Num Tasks:** 5, **Durations:** 10,12,55,11,1
	The system should return: 1) 18 on the last iteration of the loop 2) 89 for the additional data

Part 3 — Store Data and Display Task Report (Marks: 65)

At the end of this specific task, students should be able to:

- Handle and manipulate strings
- Create and work with Arrays

You will now add the final features to your app , write and automate the unit tests and submit your final project. Extend your application to allow for the following:

1. Users should be able to use to populate the following arrays:

Array	Contents
Developer	Contains the names of all the developers assigned to tasks
Task Names	Contains the names of all the created tasks
Task ID	Contains the generated taskID's for all tasks
Task Duration	Contains the Duration of all tasks
Task Status	Contains the Status of all tasks

2. Users should be able to use these arrays to:
    a. Display the Developer, Task Names and Task Duration for all tasks with the status of done.
    b. Display the Developer and Duration of the class with the longest duration.
    c. Search for a task with a Task Name and display the Task Name, Developer and Task Status.
    d. Search for all tasks assigned to a developer and display the Task Name and Task Status.
    e. Delete a task using the Task Name.
    f. Display a report that lists the full details of all captured tasks.

3. Use the following test Data for your unit tests and to populate your arrays:

Test Data Task 1	
Developer	Mike Smith
Task Name	Create Login
Task Duration	5
Task Status	To Do

Test Data Task 2	
Developer	Edward Harrison
Task Name	Create Add Features
Task Duration	8
Task Status	Doing

Test Data Task 3	
Developer	Samantha Paulson
Task Name	Create Reports
Task Duration	2
Task Status	Done

Test Data Task 4	
Developer	Glenda Oberholzer
Task Name	Add Arrays
Task Duration	11
Task Status	To Do

4. Create the following unit tests:

Test: (assertEquals)	**Test Data and expected system responses.**
Developer array correctly populated: The developer array contains the expected test data.	**<u>Test Data:</u> Developer entry for Test data for tasks 1-4**
	<u>The system returns:</u> "Mike Smith", "Edward Harrington" , "Samantha Paulson", "Glenda Oberholzer"
Display Developer and Duration for task with longest duration.	**<u>Test Data:</u> Task 1-4**
	<u>The system returns:</u> **"**Glenda Oberholzer, 11;
Search for tasks	**<u>Test Data:</u> "**Create Login
	<u>The system returns:</u> " Mike Smith, Create Login"
Search all tasks assigned to Developer	**Test Data:** Samantha Paulson
	<u>The system returns:</u> Create Reports
Delete Task from array	**Test Data:** "Create Reports"
	<u>The system returns:</u> Entry "Create reports" successfully deleted
Display Report	
	<u>The system returns:</u>

Assessment Sheet (Marking Rubric)

Please note: Tear off this section and **attach** it to your work when you submit it/ If this is an online submission, then this information needs to be included in the online submission.

MODULE NAME:	**MODULE CODE:**
PROGRAMMING 1A	**PROG5121/d**
PROGRAMMING 1A	**PROGf5111**

STUDENT NAME:	
STUDENT NUMBER:	

Part 1	**Levels of Achievement**			**Student Result**	**Feedback**
In order to be awarded full marks for these elements of Part 1, students need to have:	**Excellent**	**Good**	**Developing**		
	Score Ranges Per Level (½ marks possible)				
Registration Feature: * Username contains under score and is no more than 5 characters long.	**4—5** Feature excellently implemented.	**2—3** Feature working mostly with some bugs**.**	**0—1** Feature not implemented or very buggy, or the app does not compile.		
Registration Feature: * Password meets complexity requirements	**4—5** Feature excellently implemented.	**2—3** Feature working mostly with some bugs**.**	**0—1** Feature not implemented or very buggy, or the app does not compile.		

Login Feature: ***** Appropriate decision structure is used to verify the user Authentication.	**4—5** Feature excellently implemented.	**2—3** Feature working mostly with some bugs**.**	**0—1** Feature not implemented or very buggy, or the app does not compile.		
Login Feature: ***** The system responds with appropriate confirmation and error messages	**4—5** Feature excellently implemented.	**2—3** Feature working mostly with some bugs**.**	**0—1** Feature not implemented or very buggy, or the app does not compile.		
Unit Tests: ***** Unit tests are created and correctly tests for functionality	**4—5** Feature excellently implemented.	**2—3** Feature working mostly with some bugs**.**	**0—1** Feature not implemented or very buggy, or the app does not compile.		
Automated Tests : *****The TestJava.yml file is created and runs the necessary unit tests in GitHub	**4—5** Feature excellently implemented.	**2—3** Feature working mostly with some bugs**.**	**0—1** Feature not implemented or very buggy, or the app does not compile.		
Coding Standard and Code Complexity	**10—7** Good variable names, low complexity, no redundant code , comments class files , efficient code . .	**7-4** Sufficient variable names, acceptable complexity, low code redundancy, Comments present , leans to towards efficient code	**0—3** Poor variable names, code is overly complex, redundant code present, poor commenting, code is not efficient.		

MODULE NAME:	MODULE CODE:
PROGRAMMING 1A	PROG5121/d
PROGRAMMING 1A	PROGf5111

STUDENT NAME:	
STUDENT NUMBER:	

Part 2	Levels of Achievement			Student Results	Feedback
In order to be awarded full marks for these elements of Part 2, students need to have:	Excellent	Good	Developing		
	Score Ranges Per Level (½ marks possible)				
Welcome Message and Menu Feature: Welcome message displays correctly. Menu option displays correctly	**4—5** A numeric menu is displayed using JOptionPane which allows the user to choose an option using the numbers 1-3	**2—3** A numeric menu is displayed, users are able to choose an option	**0—1** The menu does not display (0) or throws errors when user tries to enter a numeric option (1 max)		
Add TasksFeature: While loop	**4—5** The application loop runs correctly and quits on correct input	**2—3** The loop is present but does not quit on correct input (2 max) or does not run for the duration of the application	**0—1** No loop present (0) , infinite loop (1)		

Add Task Feature: For Loop	**4—5** A for loop is used to allow the user to enter the assigned number of tasks. The loop runs and exits correctly.	**2—3** A for loop is used for 1 off errors , still allows user to enter tasks	**0—1** No loop (0) Infinite loop (1)		
Add Task: Enter Task Details	**4—5** The application provides adequate output which allows the user to enter the necessary task data. Appropriate variables exist.	**2—3** The feature allows the user to add to task data	**0—1** Not all data can be entered , the variables are not stored correctly etc.		
Add Task :Task Number created in loop (using loop counter)	**4—5** The task number increments correctly as the loop runs	**2—3** One off errors.	**0—1** The loop does not run correctly and does not generate task numbers.		
Add Task :Task ID displays correctly and created with string manipulation (substring)	**4—5** The Task ID is created using string manipulation and loop counters and is displayed correctly for all the looped tests.	**2—3** The task ID is created using string manipulation and counters but contains one off or other errors.	**0—1** TaskID is hard coded or incorrect.		
Add Task : Task details displays correctly	**4—5** The task details are displayed in the correct order and the tests pass	**2—3** Majority of the Task details displays correctly	**0—1** Task details to not display (0) or displays with multiple errors (1)		
Unit Tests	**4—5** Unit tests are created, passed and adequately test the functionality	**2—3** Unit test are created but does not adequately test the functionality **.**	**0—1** No Test (0) , Very little tests with errors in the test class (1)		

Automated Unit Tests	**4—5** TestJava. yml file is updated to run Task Class tests and executes in GitHub . All tests are present and passed	**2—3** ThestJava.yml is created but does not contain all tests for the Task class. Executes in GitHubwith some test failures	**0—1** No tests (0). Multiple Errors (1)		
Coding Standard and Code Complexity	**10—7** Good variable names, low complexity, no redundant code , comments class files , efficient code . .	**7-4** Sufficient variable names, acceptable complexity, low code redundancy, Comments present , leans to towards efficient code	**0—3** Poor variable names, code is overly complex, redundant code present, poor commenting, code is not efficient.		

MODULE NAME:	MODULE CODE:
PROGRAMMING 1A	PROG5121/d
PROGRAMMING 1A	PROGf5111

STUDENT NAME:	
STUDENT NUMBER:	

POE	Levels of Achievement				Feedback
In order to be awarded full marks for these elements of the final POE, students need to have:	Excellent	Good	Developing		
	Score Ranges Per Level (½ marks possible)				
Arrays correctly populated	**4—5** Arrays are created and populated correctly.	**2—3** Arrays are created but not populated correctly.	**0—1** No arrays (0). Arrays with multiple errors (1)		
Display details for task with longest duration	**4—5** The application successfully searches the parallel arrays and displays the correct output .	**2—3** The system searches parallel arrays, incorrect output is displayed	**0—1** Feature omitted (0). Feature contains multiple errors (1)		
Search Array for tasks assigned to developer	**4—5** The application successfully searches the parallel arrays and displays the correct output .	**2—3** The system searches parallel arrays, incorrect output is displayed	**0—1** Feature omitted (0). Feature contains multiple errors (1)		

Delete task using Task Name	**4—5** Array is successfully searched and the appropriate value is removed	**2—3** Array is searched, incorrect value or no value is removed.	**0—1** Feature not implemented (0) or contains multiple errors (1)		
Display Task Report.	**4—5** Report displays correctly with the necessary information	**2—3** Report displays but omits some information.	**0—1** Feature not implement (0) or with multiple errors (1)		
Unit Tests	**4—5** Unit tests are created, passed and adequately test the functionality	**2—3** Unit test are created but does not adequately test the functionality.	**0—1** No Test (0) , Very little tests with errors in the test class (1)		
Automated Unit Tests	**4—5** TestJava. yml file is updated to run Task Class tests and executes in GitHub. All tests are present and passed	**2—3** ThestJava.yml is created but does not contain all tests for the Task class. Executes in github with some test failures	**0—1** No tests (0). Multiple Errors (1)		
Coding Standard and Code Complexity	**10—7** Good variable names, low complexity, no redundant code, comments class files, efficient code.	**7-4** Sufficient variable names, acceptable complexity, low code redundancy, Comments present , leans to towards efficient code	**0—3** Poor variable names, code is overly complex, redundant code present, poor commenting, code is not efficient.		