**Introduction to Object Oriented Programming:**

**COURSEWORK 2**

**Project Writeup and Evaluation**

**Ben Seager – 229063174**

Contents

[**1 - Class Description 2**](#_Toc130220203)

[**2 – UML Diagram 4**](#_Toc130220204)

[**3 – Example Execution 5**](#_Toc130220205)

[**4 – Project Evaluation 7**](#_Toc130220206)

# 1 - Class Description

**MODULE**

An abstract superclass used to initialise object instances for three subclasses: CWOnly, EXOnly and CWEX. Module contains three private attributes: mname, status and finalmark; Attributes are associated with initialising the name, type and final mark (respectively) of a module from the three specified types.

Accessor and Mutator methods are also contained within this class to allow the user to alter and retrieve object values. Methods can be overridden, allowing for differing purposes whilst maintaining the identifier.

**CWONLY**

A concrete subclass that inherits methods from its superclass, Module. Used to initialise instances of a Coursework Only object and alter the values of its four private attributes (with Accessor and Mutator methods): hws, cwm, fp and finalmark;

* hws: An integer array of size three that stores homework values between 0 and 100, inputted by the user.
* fp: An integer that stores the value of the student’s Final Project between 0 and 100, inputted by the user.
* cwm: A double that stores the value of the student’s coursework after the appropriate weightings have been applied, between 0 and 100.
* finalmark: A double that stores the student’s final mark for the Coursework Only module, between 0 and 100.

Two methods inherited from Module are overridden in this class in order to apply the appropriate weightings for cwm and fp, when setting the value of finalmark.

**EXONLY**

A concrete subclass that inherits methods from its superclass, Module. Used to initialise instances of an Exam Only object and alter the value of one private attribute (with a single Accessor and Mutator method):

* exam: An integer that stores the value of the student’s exam between 0 and 100, inputted by the user.

**CWEX**

A concrete subclass that inherits methods from its superclass, Module. Used to initialise instances of a Coursework and Exam object and alter the values of its four private attributes (with Accessor and Mutator methods): hws, cwm, exam and finalmark;

* hws: An integer array of size four that stores homework values between 0 and 100, inputted by the user.
* exam: An integer that stores the value of the student’s exam between 0 and 100, inputted by the user.
* cwm: A double that stores the value of the student’s coursework after the appropriate weightings have been applied, between 0 and 100.
* finalmark: A double that stores the student’s final mark for the Coursework Only module, between 0 and 100.

Two methods inherited from Module are overridden in this class in order to apply the appropriate weightings for cwm and exam, when setting the value of finalmark.

**STUDENT**

A concrete class used to initialise instances of a Student object and alter the values of its six private attributes: name, surname, id, cw001, ex002 and ce003;

* name: A String that stores the students name, inputted by the user.
* surname: A String that stores the students name, inputted by the user.
* id: An integer that stores the student’s id number, taken from the current iteration number of the loop where the student object is initialised.
* cw001: An object of type CWOnly that is stored within the Student class.
* ex002: An object of type EXOnly that is stored within the Student class.
* Ce003: An object of type CWEX that is stored within the Student class.

This class stores objects from other classes due to various methods requiring the ability to access attributes from these objects through their Accessor methods when the method is called (This occurs due to the Student Class containing methods that display values for each of the three module types in a table format).

**STUDENTTEST - MAIN**

This class contains the main method that will be executed.

StudentTest simulates a gradebook scenario, where the user must input various values corresponding to the marks/scores for progress checks and exams that each student whose values are being inputted will have taken, which will then be stored and converted into a final mark for each of three modules: CW001 (Type: CWOnly), EX002 (Type: EXOnly) and CE003 (Type: CWEX), through the use of the Accessor and Mutator methods defined within the respective, previously described classes.

Additionally, this class provides the option to view the results for each student in a tabular format by taking the inputted values and giving the user the ability to choose between which table they view (Individual marks for a single module or all modules in one table).

# 2 – UML Diagram

***Diagram

Description automatically generated***

# 3 – Example Execution

How many Students? 2 // Determines the number of iterations that the program must occur for

// Initialises student object and stores to students[i] where i is the iteration value

// CWOnly Object cw001 is initialised

Enter the Student's First Name: Ben // Stored with student.setName()

Enter the Student's Surname: Seager // Stored with student.setSurname()

Enter Homework 1 score for CW001: 88 // Stored with cw001.setHW()

Enter Homework 2 score for CW001: 77

Enter Homework 3 score for CW001: 999 // Input is out of range, prompts a correct input

Input must be between 0 and 100. Try again: 100

Enter the Student's Final Project score for CW001: 0 // Stored with cw001.setFP()

// cw001 is stored to student with stude.setCWOnly(cw001)

CW001 DATA ENTERED

// EXOnly object ex002 is initialised

Enter the Student's EX002 Exam score: 55 // Stored with ex002.setFM()

// ex002 is stored to student with student.setEXOnly(ex002)

EX002 DATA ENTERED

// CWEX Object ce003 is initialised

Enter Homework 1 score for CE003: 66 // Stored with ce003.setHW()

Enter Homework 2 score for CE003: 77

Enter Homework 3 score for CE003: 444 // Input too high, prompts a correct input

Input must be between 0 and 100. Try again: 89

Enter Homework 4 score for CE003: 99

Enter the Student's CE003 Exam score: 100 // Stored with ce003.setExam()

// ce003 is stored to student with student.setCWEX(ce003)

CE003 DATA ENTERED

// Iteration two begins

Enter the Student's First Name: Person

Enter the Student's Surname: Two

Enter Homework 1 score for CW001: -1 // Input is too low, prompts a correct input

Input must be between 0 and 100. Try again: 10

Enter Homework 2 score for CW001: 55

Enter Homework 3 score for CW001: 66

Enter the Student's Final Project score for CW001: 55

CW001 DATA ENTERED

Enter the Student's EX002 Exam score: 78

EX002 DATA ENTERED

Enter Homework 1 score for CE003: 888

Input must be between 0 and 100. Try again: 77

Enter Homework 2 score for CE003: 44

Enter Homework 3 score for CE003: 0

Enter Homework 4 score for CE003: 88

Enter the Student's CE003 Exam score: 77

CE003 DATA ENTERED

0 - CW001

1 - EX002

2 - CE003

3 - All Modules

4 - End Program

Which Table of Results would you like to view? Choose a number: 0

// Input = 0; student.displayCW001() is called to display CW001 values in table format

Marks for CW001:

Name Surname HWs Project Final Mark

Ben Seager 88.0% 0% 44.0%

Person Two 43.0% 55% 49.0%

Which Table of Results would you like to view? Choose a number: 1

// Input = 1; student.displayEX002() is called to display EX002 values in table format

Marks for EX002:

Name Surname Final Mark

Ben Seager 55.0%

Person Two 78.0%

Which Table of Results would you like to view? Choose a number: 2

// Input = 2; student.displayCWEX() is called to display CWEX values in table format

Marks for CE003:

Name Surname CW Exam Final Mark

Ben Seager 82.0% 100% 92.80%

Person Two 52.0% 77% 67.0%

Which Table of Results would you like to view? Choose a number: 3

// Input = 3; student.displayAll() is called to display final mark values in table format

Number of Students: 2

Name Surname CW001 EX002 CE003

Ben Seager 44.00% 55.0% 92.80%

Person Two 49.00% 78.0% 67.00%

Which Table of Results would you like to view? Choose a number: 4

END OF PROGRAM

// Input = 4; Program is terminated

# 4 – Project Evaluation

|  |  |
| --- | --- |
| **Successes** | **Difficulties** |
| Eventually, I successfully managed to write effective code that worked how I intended after multiple restarts and functionality contradictions. | I found it difficult to understand the initial fundamentals of the project (i.e., The class hierarchy) and how certain functionalities should be implemented, such as overriding methods from the module class. |
| Creating the UML Diagram was a thought-provoking process as we had to consider relations and multiplicities, but I think it turned out ok. | I was not sure how to add exception handling, so I ended up implementing while loops to enforce some version of an input range. My project could have been better if this was successfully included. |
| Coming up with a successful implementation of the given scenario after misunderstanding the hierarchy and needs of each class. | Looking back I could have been more successful with time management. I completed the code relatively early and left the writeup to the final day to complete. |

**What have I learnt?**

* I gained experience handling abstract classes and overriding methods to implement different functionalities when using the same identifier.
* It was interesting to play around with the formatting syntax for Java and attempt to correctly output a table at the end of the project.
* Better knowledge concerning subclass inheritance from a superclass and constructor implementation.
* Overall increase in confidence when writing in Java due to the use of more complicated concepts.

**END OF PROJECT WRITEUP**