PROG2004 OBJECT ORIENTED PROGRAMMING

Summary

Title	Assessment 2 – Programming tasks - a theme park management system
Type	Programming
Due Date	Monday 2 December 2024 11:59 pm AEST (Start of Week 6)
Length	NA
Weighting	60%
Academic Integrity (See below for limits of use where GenAI is permitted)	You **may use** GenAI tools to get some ideas or insight about particular problems in code. However, you **MUST NOT** generate a complete solution code. Please refer to the **Academic Integrity section** below to see what you can and cannot do with the GenAI tools.
Submission	You will need to submit the following to the submission link provided for this assignment on the MySCU site: • The JAVA project with all your source code files. • The link to your GitHub repository. • A video explaining your code.
Unit Learning Outcomes	This assessment maps to the following ULOs: • ULO2: apply object-oriented programming principles to solve intermediate problems • ULO3: distinguish between and use advanced collection classes • ULO4: apply various inbuilt mechanisms within the programming languages to handle concurrency and various forms of input and output

Rationale

This assessment aims to assess students' ability to apply object-oriented programming concepts and principles, advanced collection classes, and input/output mechanisms to solve problems. It addresses unit learning outcomes 2, 3 and 4 and relates to modules 3, 4, 5 and 6.

Task Description

In this assignment, your task is to write JAVA codes that manage a theme park and its visitors to the rides (**Park Rides Visitor Management System - PRVMS**). Your solution must apply the object-oriented programming concept and utilise suitable collections to handle the data.

In this assessment, you MUST:

- Use **JAVA language** to develop the solution.
- Submit **JAVA project** with all your source code files to MySCU link.
- **Demonstrate your work progress** by regularly commiting your code to your GitHub repository over the last three weeks.
- **Record a video** to explain your code and demonstrate your understanding.

If you fail to do any of the above, you will lose marks and be required to attend an interview to explain your code. If you cannot explain your code, you will be submitted for academic misconduct.

Task Instructions

The task consists of 8 parts covering Modules 3 to 6.

Part 1 – Classes and Inheritance

To get started:

- Create a new Java project called **username-A2**.
- In the **src** directory, create a new class called **AssignmentTwo**.
- In the AssignmentTwo class, create the **main method**.
- In the **src** directory, create at least four other classes called:
    - Person
    - Employee
    - Visitor
    - Ride

In the PRVMS, you are creating:

- The *Employee* class is used to track the theme park staff who operate rides.
- The *Visitor* class is used to track the theme park visitors.
- The *Ride* class is used to track the rides available at the theme park, e.g., roller coaster, water riders, etc.

In the Person class:

- Add at least 3 instance variables suitable for a person
- Add a default constructor and a second constructor that sets the instance variables using parameters
- Add getters and setters for all Person instance variables

In the Employee class:

- Extend the Person class
- Add at least 2 instance variables suitable for theme park staff
- Add a default constructor and a second constructor that sets the instance variables (Employee and Person) using parameters
- Add getters and setters for all Employee instance variables

In the Visitor class:

- Extend the Person class
- Add at least 2 instance variables suitable for a theme park member
- Add a default constructor and a second constructor that sets the instance variables (Visitor and Person) using parameters
- Add getters and setters for all Visitor instance variables

In the Ride class:

- Add at least 3 instance variables suitable for a Ride. One of these instance variables must be of type Employee, i.e. used to know if the ride is open and there is a ride operator in charge running the ride.

- Add a default constructor and a second constructor that sets the instance variables using parameters.
- Add getters and setters for all Ride instance variables, including the one to assign an Employee to operate the ride.

You may look at Movie World theme park's website (https://movieworld.com.au/attractions?height=150) to get some ideas about the rides.

In the AssignmentTwo class, add the following code:

```
public class AssignmentTwo {
    public static void main(String[] args) {
    }
    public void partThree(){
    }
    public void partFourA(){
    }
    public void partFourB(){
    }
    public void partFive(){
    }
    public void partSix(){
    }
    public void partSeven(){
    }
}
```

Module 3 - Advanced class design

The following part of the assessment covers the content in Module 3.

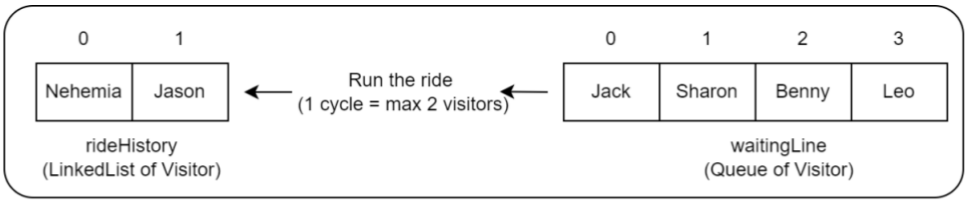
Part 2 – Abstract class and interface

Update the classes you created above to meet these requirements:

- In *Person* class, make the class as **abstract** as the class will never be instantiated (e.g., no object can be created from the class).
- Create ***an interface*** named *RideInterface* so it defines the following methods. Implement this interface on *Ride* class.
  - An interface method named *addVisitorToQueue*: to add a visitor to the queue. It has a parameter of *Visitor*. See Part 3.
  - An interface method named *removeVisitorFromQueue*: to remove a visitor from the queue. See Part 3.
  - An interface method named *printQueue*: to print the list of waiting visitors in the queue. See Part 3.
  - An interface method named *runOneCycle*: to run the ride for one cycle. See Part 5.
  - An interface method named *addVisitorToHistory:* to add a visitor to the ride history. It has a parameter of *Visitor*. See Part 4.
  - An interface method named *checkVisitorFromHistory*: to check whether the visitor is in the ride history. It has a parameter of *Visitor*. See Part 4.
  - An interface method named *numberOfVisitors:* to return the number of *Visitors* in the ride history. See Part 4.
  - An interface method named *printRideHistory*: to print the list of visitors who took the rides. See Part 4.

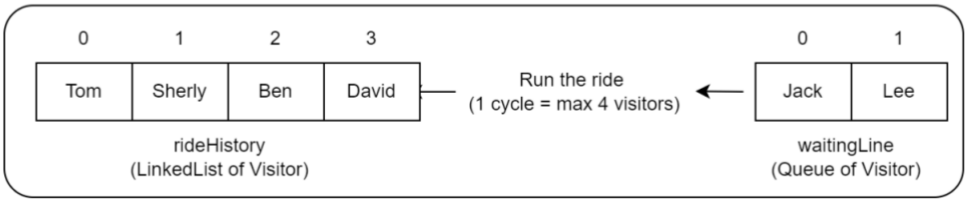
For Parts 3 – 4, please see the illustration below.

rollerCoaster
(Class Ride)



This ride has 4 visitors (Jack, Sharon, Benny, Leo) in the waiting line and 2 visitors (Nehemia, Jason) have took the ride

thunderstorm
(Class Ride)



This ride has 2 visitors (Jack, Lee) in the waiting line and 4 visitors (Tom, Sherly, etc) have took the ride

Part 3 – Queue Interface

Visitors must join the waiting line before taking the ride. The program needs the ability to keep track of *Visitors* who are waiting to take the ride and the order in which they joined the waiting list, i.e., *first in first out*.

There is no maximum slot for the queue as visitors may decide if they want to join the waiting line or not.

For this part of the assignment:

- Using a *Queue*, update the *Ride* class so that a *Ride* can store *Visitors* (i.e., *Visitor* objects) who are waiting to take the *Ride*.

In addition to adding a *Queue*, you need to add the following methods to the *Ride* class that work with the *Queue*:

- A method named *AddVisitorToQueue* to add a *Visitor* to the *Queue* (see the interface you created in Part 2).
- A method named *RemoveVisitorFromQueue* to remove a *Visitor* from the *Queue* (see the interface you created in Part 2).
- A method named *PrintQueue* that prints all the details for all *Visitors* in the *Queue* in the order they were added (see the interface you created in Part 2).

Note: Make sure all the above methods print suitable success/failure messages.

Demonstration

In the *partThree* method in the *AssignmentTwo* class:

- Create a new *Ride* object.
- Using the methods you created:
  - Add a minimum of 5 *Visitor* to the *Queue*.

- Remove a *Visitor* from the *Queue*.
- Print all *Visitors* in the *Queue*.

Module 4 - Advanced collection

The following part of the assessment covers the content in Module 4.

Part 4A – Collection class (LinkedList)

The *Ride* class is missing the ability to store a collection of *Visitors* who have taken the ride. Once a visitor takes the ride, **the visitor will be removed from the queue and needs to be added to a collection** so the management can understand how many people have taken the ride.

For this part of the assignment:

- Using a LinkedList collection, update *Ride* so that a *Ride* object can store a collection of *Visitors* (i.e. datatype *Visitor*) who have taken the *Ride*.

In addition to adding the *collection*, you need to add the following methods to *Ride* that work with the *collection*:

- A method named *addVisitorToHistory* to add a visitor to the ride history (see the interface you created in Part 2).
- A method named *checkVisitorFromHistory* to check whether the visitor is in the ride history (see the interface you created in Part 2).
- A method named *numberOfVisitors* to return the number of *Visitors* in the ride history (see the interface you created in Part 2).
- A method named *PrintRideHistory* to print the details of all *Visitors* who have taken the *Ride* (**you must use an *Iterator,* or you will get no marks**). See the interface you created in Part 2.

Note: Make sure all the above methods print suitable success/failure messages.

Demonstration

In the *partFourA* method in the *AssignmentTwo* class:

- Create a new *Ride* object.
- Using the methods you created:
  - Add a minimum of 5 *Visitors* to the collection.
  - Check if a *Visitor* is in the collection.
  - Print the number of *Visitors* in the collection.
  - Print all *Visitors* in the collection.

Part 4B – Sorting the collection

There is no way to sort the *Visitors* who have taken a *Ride*. For this part of the assignment:

- Create a class (you can choose the name) that implements the *Comparator* interface. When you implement the *compare* method from the *Comparator* interface, you must use a minimum of two of the instance variables in your comparison.
- Create a method in the *Ride* class that sorts the collection using the sort (e.g., List list, *Comparator* c) method in the *Collections* class.

Note: You MUST use the *Comparator* interface. You CAN NOT use the *Comparable* interface.

Demonstration

In the *partFourB* method in the *AssignmentTwo* class:

- Create a new *Ride* object.
- Using the methods you created:
  - Add a minimum of 5 *Visitors* to the collection.
  - Print all *Visitors* in the collection.
  - Sort the collection
  - Print all *Visitors* in the collection again to show that the collection has been sorted.

Part 5 – Run a ride cycle

A ride is usually run in cycles where each cycle will take a number of visitors from the queue line to take the ride.

The *Ride* class is missing the ability to run the ride. Once a visitor takes the ride, **the visitor will be removed from the queue** (see Part 3) **and needs to be added to the collection** (Part 5)**.**

In *Ride* class, you need to add the following properties:

- A property named *maxRider* to identify how many visitors a ride can take in one cycle. At least 1 visitor is required to run the ride.
- A property named *numOfCycles* to identify how many times the ride is run. By default, it is 0. Increase by 1 every time the ride is run.

In *Ride* class, you need to implement the *RunOneCycle* method (see the interface you created in Part 2):

- If no ride operator is assigned to the ride, the ride cannot be run, and a message is printed out.
- If there are no waiting visitors in the queue, the ride cannot be run, and a message is printed out.
- A number of visitors in the queue (based on the *'maxRider'*) will be removed from the queue and then added to the collection.

Note: Make sure all the above methods print suitable success/failure messages.

Demonstration

In the *partFive* method in the *AssignmentTwo* class:

- Create a new *Ride* object.
- Using the methods you created:
  - Add a minimum of 10 *Visitors* to the *Queue*.
  - Print all *Visitors* in the queue.
  - Run one cycle.
  - Print all *Visitors* in the queue after one cycle is run.
  - Print all *Visitors* in the collection.

Module 5 – Input/output

The following part of the assessment covers the content in Module 5.

An important part of many programs is the ability to back up data to a file and then restore it as needed. In this section of the assignment, we will add this ability to our program.

Hint for exporting and importing data

A common way to store data in a file that needs to be imported later is to use comma-separated values (csv). This means that we store a record on a single line, and we separate values using a comma (,). For example, imagine an object for a class called Animal has the following information:

- species: Dog
- breed: Poodle
- colour: Brown
- name: Fido
- age: 7

You could store the *Animal* object in the file on a single line like: Dog,Poodle,brown,Fido,7.

When you read the file, each line in the file will contain the details for a single *Animal* object. You can then use the *split*() method from the *String class* to split the line into individual values and then use the values to create a new *Animal* object.

Part 6 – Writing to a file

The *Ride* class is missing the ability to back up the *Visitors* who have taken the *Ride* (see Part 3). For this part of the assignment:

- Add a method named *exportRideHistory* to the *Ride* class that writes the details of all of the *Visitors* that have taken the *Ride* (i.e. stored in the LinkedList) to a file. The details for each *Visitor* should be written on their own line.
- You must make sure to add all appropriate exception handling and error messages.
- You **DO NOT** need to back up the queue (Part 2).

Demonstration

In the *partSix* method in the *AssignmentTwo* class:

- Create a new *Ride*.
- Add a minimum of 5 *Visitors* to the *Ride* (i.e., the LinkedList).
- Export the *Visitors* to a file.

Part 7 – Reading from a file

The *Ride* class is also missing the ability to restore visitors who have taken the *Ride*. For this part of the assignment:

- Add a method named *importRideHistory* to the *Ride* class that can read the file that was created in the previous section.
- When reading the file, you need to sign up all visitors for the *Ride* (i.e., add them to the LinkedList).

You must make sure to add all appropriate exception handling and error messages.

Note: If you cannot enrol the *Visitor* in the *Ride* class (i.e., add them to the LinkedList), you will still get marks for reading the file.

Demonstration

In the *partSeven* method in the *AssignmentTwo* class:

- Create a new *Ride*.
- Import the file you created in the previous part of the assignment.
- Print the number of *Visitors* in the LinkedList to confirm that the correct number of *Visitors* was imported.
- Print all *Visitors* in the LinkedList to confirm that the details of each *Visitor* were imported correctly.

Use GitHub

You must **create a repository on GitHub** to store your project work with all files and documents. You **must show your work progress** in this assignment by regularly committing your project to the GitHub repository. In each commit, you need to provide a clear explanation of what changes you have made in this commit. **Failing to show the correct work progress will fail the assignment.**

Create a video

In your video, ensure you explain:

- how you implemented inheritance, polymorphism and interface in Parts 1-2,
- how you implemented Queue and LinkedList to handle the queue line and visitors who took the ride in Parts 3-5, and
- how you worked with files to write and read data in Parts 6-7.

Your video does not need to be long or go into much detail. You should be able to do all the above in <= 5 minutes; however, the video must demonstrate that you understand the code you are submitting and that you did not use ChatGPT or a similar GenAI tool to generate it.

Upload your video to your SCU OneDrive and create a sharable link to it.

Resources

Use the following resources to support you when working on this assessment.

- Study modules 3 to 6 materials and complete all learning activities.
- Take an active role in the weekly tutorial and workshop.
- Java API documentation https://docs.oracle.com/en/java/javase/22/docs/api/index.html

Referencing Style Resource

NA

Task Submission

You are required to submit the following items:

- The JAVA project with all your source code files. **Zip your project into a file called username-A2.zip and upload the file.**
- The link to your GitHub repository. **Add the link in the comments.**
- The link to your short video explaining your code part by part. **Add the link in the comments.**

Resubmit policy: This assessment is not eligible for a re-submit.

Assessment Criteria

Please refer to the marking rubric for more details. Marking criteria include:

- Java code compiles
- Use of correct coding style, including the use of comments
- Accuracy of coding
- Use of suitable coding structures
- Correct submission and naming conventions of assessment items as required

Academic Integrity

At Southern Cross University, academic integrity means behaving with the values of honesty, fairness, trustworthiness, courage, responsibility and respect in relation to academic work.

The Southern Cross University Academic Integrity Framework aims to develop a holistic, systematic and consistent approach to addressing academic integrity across the entire University. For more information, see: SCU Academic Integrity Framework

NOTE: **Academic Integrity breaches include** unacceptable use of generative artificial intelligence (GenAI) tools, the use of GenAI has not been appropriately acknowledged or is beyond the acceptable limit as defined in the Assessment, poor referencing, not identifying direct quotations correctly, close paraphrasing, plagiarism, recycling, misrepresentation, collusion, cheating, contract cheating, fabricating information.

At SCU the use of GenAI tools is acceptable, *unless it is beyond the acceptable limit as defined in the Assessment Item by the Unit Assessor*.

GenAI May be Used

Generative artificial intelligence (GenAI) tools, such as ChatGPT, **may be used** for this assessment task. If you use GenAI tools, you must use these ethically and acknowledge their use. To find out how to reference GenAI in your work, consult the referencing style for your unit via the Library referencing guides. If you are not sure how to, or how much, you can use GenAI tools in your studies, contact your Unit Assessor. If you use GenAI tools without acknowledgment, it may result in an academic integrity breach against you, as described in the Student Academic and Non-Academic Misconduct Rules, Section 3.

You **may use** Generative Artificial Intelligence (GenAI) tools, such as ChatGPT or Copilot, for this assessment task **to get some ideas or insight about particular problems in code**. It is similar when you try to find a snippet of code for doing a particular task in Stack Overflow. For example, **you must make your own effort to modify, refine, or improve the code to solve the assessment problems**. Think of it as a tool – a quick way to access information – or a (free) tutor to answer your questions. However, just as if you Googled something, you still need to evaluate the information to determine its accuracy and relevance. ***If you have used a GenAI tool in this assessment, you must document how you used it and how it assisted you in completing your assessment tasks***. ***Failing to do that will be subject to an academic integrity investigation***.

You **cannot use AI to generate a complete solution code**. You need to put your own effort into building the solution code for the assessments to demonstrate the required skills. Refer to assessment information in the Assessment Tasks and Submission area for details.

Special Consideration

Please refer to the Special Consideration section of Policy.
https://policies.scu.edu.au/document/view-current.php?id=140

Late Submissions & Penalties

Please refer to the Late Submission & Penalties section of Policy.
https://policies.scu.edu.au/view.current.php?id=00255

Grades & Feedback

Assessments that have been submitted by the due date will receive an SCU grade. Grades and feedback will be posted to the 'Grades and Feedback' section on the Blackboard unit site. Please allow 7 days for marks to be posted.

… continued on next page ...

Assessment Rubric

Marking Criteria and % allocation	High Distinction (85–100%)	Distinction (75–84%)	Credit (65–74%)	Pass (50–64%)	Fail 0–49%
Apply object-oriented programming principles and develop advanced class design to solve theme park scenario (Parts 1-2) **(ULOs 2-3)** **25%** Part 1 (People, Visitor, Staff, Rides classes): 20% Part 2 (Abstract class & interface): 5%	Demonstrates **exceptional** understanding and application of object-oriented principles (abstraction, encapsulation, inheritance, polymorphism, class) to solve complex theme park problem. Designs classes that are **highly cohesive and effectively organised** using advanced class design. Implements advanced class **effectively and efficiently without errors** to manage theme park data.	Demonstrates a **solid understanding** and application of object-oriented principles **with minor errors or omissions** in solving theme park problem. Designs classes that **generally adhere** to principles of cohesion and advanced class design **with occasional errors or less effective** implementation. Implements advanced class **with some errors or oversights** in managing theme park data with creativity and innovation.	Demonstrates a **basic understanding** of object-oriented principles but **with notable gaps or errors** in applying them to theme park problem. Designs classes that show **basic cohesion** but lack sophistication in advanced class design. Implements advanced class with **significant errors or oversights** in managing theme park data.	Demonstrates a **minimal understanding** of object-oriented principles with **substantial errors or misunderstandings** in applying them to theme park problem. Designs classes that are **poorly organised or inefficient**. Implements advanced class **with fundamental errors** or **lacks basic understanding** of managing theme park data.	**Fails to demonstrate** a basic understanding of object-oriented principles, with critical errors or complete lack of application to theme park problem. Designs classes that are **fundamentally flawed** and **do not meet** basic requirements. **Fails to implement advanced class** effectively, **with critical errors** in managing theme park data or **no implementation**.
Apply and develop various collections to manage waiting list and ride usage problems using	Demonstrates **exceptional** understanding and application of advanced collections	Demonstrates a **solid understanding** and application of advanced collections **with minor errors or**	Demonstrates a **basic understanding** of advanced collections but **with notable gaps or errors** in applying	Demonstrates a **minimal understanding** of advanced collections with **substantial errors or misunderstandings** in	**Fails to demonstrate** a basic understanding of advanced collections, with critical errors or complete lack of

advanced collection classes (Parts 3-5) **(ULOs 2-3)** **45%** Part 3 (Queue): 15% Part 4 (Linked List & Sorting): 15+5% Part 5 (Run the ride): 10%	to solve complex waiting list and ride usage problems. Designs classes that are **highly cohesive and effectively organised** using advanced class design. Implements advanced collections **effectively and efficiently without errors** to manage waiting list and ride usage data.	**omissions** in solving waiting list and ride usage problems. Designs classes that **generally adhere** to principles of cohesion and advanced class design **with occasional errors or less effective** implementation. Implements advanced collections **with some errors or oversights** in managing waiting list and ride usage data. creativity and innovation.	them to waiting list and ride usage problems. Designs classes that show **basic cohesion** but lack sophistication in advanced class design. Implements advanced collections with **significant errors or oversights** in managing waiting list and ride usage data data.	applying them to waiting list and ride usage problems. Designs classes that are **poorly organised or inefficient**. Implements advanced collections **with fundamental errors** or **lacks basic understanding** of managing waiting list and ride usage data.	application to waiting list and ride usage problems. Designs classes that are **fundamentally flawed** and **do not meet** basic requirements. **Fails to implement advanced collections** effectively, **with critical errors** in managing waiting list and ride usage data or **no implementation**.
Apply and develop I/O mechanism to manage theme data (Parts 6-7) **(ULOs 2,4)** **17.5%** Part 6 (Writing file): 7.5% Part 7 (Reading file): 10%	Demonstrates **exceptional** understanding of built-in I/O mechanism and develops a mechanism to manage theme park data **highly cohesive and effectively organised**.	Demonstrates a **solid understanding** of built-in I/O mechanism and develops a mechanism to manage theme park data **with occasional errors or less effective** implementation.	Demonstrates a **basic understanding** of built-in I/O mechanism and develops a mechanism to manage theme park data but **with notable gaps or errors.**	Demonstrates a **minimal understanding** of built-in I/O mechanism and develops a mechanism to manage theme park data with **substantial errors or misunderstandings or poorly organised/inefficient**.	**Fails to demonstrate** a basic understanding of built-in I/O mechanism and develops a mechanism to manage theme park data, with **critical errors** or **do not meet** basic requirements.

Accuracy, efficiency, validations and compatibility **(ULOs 2-4)** **5%**	Demonstrates **exceptional accuracy, efficiency, validations** to serve the objectives and requirements. JAVA code can be **compiled and run without any issues.**	Demonstrates **a solid accuracy, efficiency, and validations** with minor errors or omissions in serving the objectives and requirements. JAVA code can be **compiled and run without any issues.**	Demonstrates a **basic accuracy, efficiency, and validations** with **notable gaps of errors** in serving the objectives and requirements. JAVA code is **compiled** and can be run **with some minor issues.**	Demonstrates a **basic accuracy, efficiency, and validations** with **notable gaps of errors** in serving the objectives and requirements. JAVA code is **compiled** and can be run **with major issues.**	**Fails to demonstrate** a basic accuracy, efficiency, and validations to serve the objectives and requirements. JAVA code is not compiled and has significant errors/fails to be run.
Concept understanding (Comment, GitHub, video) **(ULOs 2-4)** **7.5%**	Demonstrates a **profound understanding** of object-oriented programming principles and advanced class design for the case study **through code comments, work progress in GitHub, and video**.	Demonstrates a **thorough understanding** of object-oriented programming principles and advanced class design for the case study **through code comments, work progress in GitHub, and video.**	Demonstrates a **basic understanding** of object-oriented programming principles and advanced class design for the case study **through code comments, work progress in GitHub, and video.**	Demonstrates a **minimal understanding** of object-oriented programming principles and advanced class design for the case study **through code comments, work progress in GitHub, and video.**	**Fails to demonstrate** a basic understanding of object-oriented programming principles and advanced class design for the case study **through code comments, work progress in GitHub, and video.**

Description of SCU Grades

High Distinction:

The student's performance, in addition to satisfying all of the basic learning requirements, demonstrates distinctive insight and ability in researching, analysing and applying relevant skills and concepts, and shows exceptional ability to synthesise, integrate and evaluate knowledge. The student's performance could be described as outstanding in relation to the learning requirements specified.

Distinction:

The student's performance, in addition to satisfying all of the basic learning requirements, demonstrates distinctive insight and ability in researching, analysing and applying relevant skills and concepts, and shows a well-developed ability to synthesise, integrate and evaluate knowledge. The student's performance could be described as distinguished in relation to the learning requirements specified.

Credit:

The student's performance, in addition to satisfying all of the basic learning requirements specified, demonstrates insight and ability in researching, analysing and applying relevant skills and concepts. The student's performance could be described as competent in relation to the learning requirements specified.

Pass:

The student's performance satisfies all of the basic learning requirements specified and provides a sound basis for proceeding to higher-level studies in the subject area. The student's performance could be described as satisfactory in relation to the learning requirements specified.

Fail:

The student's performance fails to satisfy the learning requirements specified.