College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
IN710: Object-Oriented Systems Development
Level 7, Credits 15

Assessment 01: Design Patterns

Assessment Overview

For this assessment, you will use Python/OO design patterns with Jupyter Notebook to solve 15 problems. This will consist of five written, five programming & five unit testing problems. All programming solutions should adhere to the design pattern's UML diagram as described in the lecture slides. You are allowed to deviate from this as long as you are using the correct description/template. In addition, marks will also be given for code elegance, functionality, robustness & git usage.

Assessment Table

Assessment Activity	**Weighting**	**Learning Outcomes**	**Assessment Grading Scheme**	**Completion Requirements**
Exams 1-5	30%	1, 2	CRA	Cumulative
Practicals	20%	2, 3	CRA	Cumulative
Design Patterns	25%	2, 3	CRA	Cumulative
MVT	25%	2, 3	CRA	Cumulative

Conditions of Assessment

This assessment will need to be completed by Friday, 08 May 2020 at 5pm.

Pass Criteria

This assessment is criterion-referenced with a cumulative pass mark of 50%.

Submission Details

You must submit your program files via **GitHub Classroom**. Here is the link to the repository you will be using for your submission – https://classroom.github.com/a/yvLS66q1. For ease of marking, please submit the marking sheet with your name & student id number via **Microsoft Teams** under the **Assignments** tab.

Authenticity

All parts of your submitted assessment must be completely your work and any references must be cited appropriately.

Policy on Submissions, Extensions, Resubmissions & Resits

The school's process concerning **Submissions, Extensions, Resubmissions and Resits** complies with Otago Polytechnic policies. Students can view policies on the Otago Polytechnic website located at https://www.op.ac.nz/about-us/governance-and-management/policies.

Extensions

Please familiarise yourself with the assessment due dates. If you need an extension, please contact your lecturer before the due date. If you require more than a week's extension, a medical certificate or support letter from your manager may be needed.

Resubmissions

Students may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are completed within a short time frame (usually no more than 5 working days) and usually must be completed within the timing of the course to which the assessment relates. Resubmissions will be available to students who have made a genuine attempt at the first assessment opportunity. The maximum grade awarded for resubmission will be C-.

Learning Outcomes

At the successful completion of this course, students will be able to:

1. Discuss theoretical and pragmatic issues surrounding design and implementation of enterprise software systems.
2. Analyse a problem statement for a complex software system and design an appropriate class architecture for the problem solution.
3. Design and implement components of large software systems following industry standard software engineering methodologies and producing industry-quality code.

Instructions

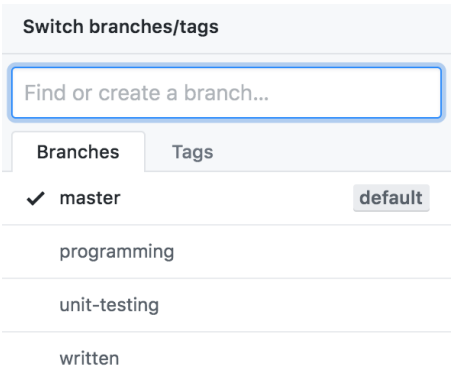
In this assessment, you are provided with **01-assessment.ipynb** containing all 15 problems. All 15 solutions must be written in the notebook. External files, for example, helper files will not be accepted for marking.

Git Usage - Learning Outcomes: 2, 3

In this task, you are required to:

1. Create three branches for the written, programming & unit testing problems. **Note:** branches must be named as specified in the screenshot below. Names deviating from this will not be accepted.
2. Add, commit & push your solution to GitHub after completion of each problem. **Note:** non-descriptive commit messages will not be accepted.

Here is the expected GitHub repository branch structure:



Written - Learning Outcomes: 2, 3

In this task, you are given five real-world problems. For each problem, you are required to:

1. Choose an OO design pattern which you think provides the best solution.
2. Describe the relationships between classes & objects pertaining to your solution.
3. Create an UML diagram detailing your solution. Display your UML diagram in the notebook using markdown.

The patterns you will use are:

- Builder
- Flyweight
- Proxy
- State
- Template

Real-World Problem 1

Consider the Air New Zealand frequent flyer benefits. Initially, a member will be assigned to the silver tier. As a member accumulates more flights, the member will move up to either the gold or elite tier. At these tiers, members can access the lounge, extra baggage & cabin upgrades.

The Air New Zealand frequent flyer benefits work in yearly cycles. At the end of a member's yearly cycle, their flight activity is calculated & they will be assigned to one of the three tiers. A member is required to fly 25 times in a given yearly cycle to remain assigned to the silver tier, 50 times to remain assigned to the gold tier & 75 times to remain assigned to the elite tier.

Real-World Problem 2

Consider a collection of sorting algorithms (bubble sort, selection sort, insertion sort, etc). Each sorting algorithm has a specific structure which is independent from the items being sorted. Eventually, the sorting algorithm will need to compare at least two items. Of course, this depends on the type of items being compared. For example, we can easily compare two integers - true or false, does x equal y? What happens if we want to compare people objects. It can be done by comparing person a's last name with person b's last name. However, the comparison could be made on another field, such as first name, mobile number, email address, etc. There is a clear distinction between the sorting algorithm & the operation that is dependent on the data. The same distinction exists for other algorithms such as searching.

Real-World Problem 3

Consider a new social media application similar to Facebook, Instagram & Twitter. A feature of this application/system would allow people who do not intend to post, tweet, etc to sign up only to view other user's feeds. It would be an ideal situation for every new user to begin with a clean feed & not allocate any storage space until they have added friends/users.

Another feature of this application/system would allow users to logged on & access their friend's feed. All actions (write on wall, send gifts, etc) performed by the user would originate at the browser & be sent across the network to the nearest server.

Real-World Problem 4

Consider a photo library application. We want to display a library of photos at any one time. We want to be able to scroll up & down through the library without noticeable lag. Photos should be preloaded into memory & should remain in memory while the application is running.

Also consider a photo grouping application. We want to be able to organise photos into groups. Photos can belong to many different groups. Potentially, the number of photos to display could increase exponentially. If all of the photos do fit in memory & given they belong to different groups means that any photo could display at irregular times, resulting in a lot of disk transfers.

Real-World Problem 5

Consider a clothing company that sells genuine & fake handbags. Each handbag is made up of several parts including the body, strap, accessories, etc. Generally, the more expensive the bag, the more parts it will have. Obviously, each bag will have a different list of parts. For example, handbag may have a plastic body, full grain leather strap & a sterling silver logo metal plate.

Programming - Learning Outcomes: 2, 3

In this task, you are given five programming problems. You will be required to:

1. Refactor an existing program by implementing the specified OO design pattern
2. Use the given scenario to write a program using the specified OO design pattern

Programming problems are specified in the **01-assessment.ipynb** file.

Unit Testing - Learning Outcomes: 2, 3

In this task, you are given five unit testing problems. You will be required to:

1. Create a test case class for each programming problem
2. Create unit tests covering all class methods

Unit testing problems are specified in the **01-assessment.ipynb** file.

IN710 Object-Oriented Systems Development
Semester 1, 2020

Assessment 01: Design Patterns
Assessment Rubric

	10-9	8-7	6-5	4-0
Written, Programming & Unit Testing	Written solutions thoroughly explain the relationships between classes & objects pertaining to the correct choice of OO design patterns. UML diagrams thoroughly visualize the relationships between classes & objects pertaining to the correct written solutions. Programming solutions thoroughly implement relationships between classes & objects pertaining to the given OO design patterns. Test cases thoroughly test each unit of code pertaining to the programming solutions.	Written solutions mostly explain the relationships between classes & objects pertaining to the correct choice of OO design patterns. UML diagrams mostly visualize the relationships between classes & objects pertaining to the correct written solutions. Programming solutions mostly implement relationships between classes & objects pertaining to the given OO design patterns. Test cases mostly test each unit of code pertaining to the programming solutions.	Written solutions explain some relationships between classes & objects pertaining to the correct choice of OO design patterns. UML diagrams visualize some relationships the between classes & objects pertaining to the correct written solutions. Programming solutions implement some relationships & objects pertaining to the given OO design patterns. Test cases test some units of code pertaining to the programming solutions.	Written solutions do not or do not fully explain the relationships between classes & objects. Incorrect OO design pattern used. UML diagrams do not or do not fully visualize the relationships between classes & objects. Incorrect OO design pattern used. Programming solutions do not or do not fully implement relationships between classes & objects pertaining to the given OO design patterns. Test cases do not or do not fully test each unit of code pertaining to the programming solutions.
Git Usage	Git commit messages thoroughly describe the context of each solution. Git branches thoroughly named & describe the context of each solution.	Git commit messages mostly describe the context of each solution. Git branches mostly named & describe the context of each solution.	Git commit messages explain describe context of each solution. Git branches named & thoroughly describe the context of each solution.	Git commit messages do not or do not full describe the context of each solution. Git branches incorrectly named & do not or do not fully describe the context of each solution.

IN710 Object-Oriented Systems Development
Semester 1, 2020

Code Elegance	Programs thoroughly demonstrate code elegance on the following: • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to the KISS, SOLID & YAGNI principles. • Efficient algorithmic approach. • Correct use of setup & teardown in test case classes. • Code adhere to pycodestyle style guide. Programs thoroughly code commented & demonstrates thorough understanding of the given OO design pattern.	Programs mostly demonstrate code elegance on the following: • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to the KISS, SOLID & YAGNI principles. • Efficient algorithmic approach. • Correct use of setup & teardown in test case classes. • Code adhere to pycodestyle style guide. Programs mostly code commented & demonstrates a clear understanding of the given OO design pattern.	Programs demonstrate code elegance on some of the following: • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to the KISS, SOLID & YAGNI principles. • Efficient algorithmic approach. • Correct use of setup & teardown in test case classes. • Code adhere to pycodestyle style guide. Programs code commented & demonstrates some understanding of the given OO design pattern.	Programs do not or do not fully demonstrate code elegance on any of the following: • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to the KISS, SOLID & YAGNI principles. • Efficient algorithmic approach. • Correct use of setup & teardown in test case classes. • Code adhere to pycodestyle style guide. Programs not or not fully code commented & demonstrates no understanding of the given OO design pattern.
Functionality & Robustness	Programs thoroughly demonstrate functionality & robustness on the following: • Classes adhere to a general OO design pattern architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Correct exception handling, e.g., input values, abstract methods not implemented.	Programs mostly demonstrate functionality & robustness on the following: • Classes adhere to a general OO design pattern architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Correct exception handling, e.g., input values, abstract methods not implemented.	Programs demonstrate functionality & robustness on some of the following: • Classes adhere to a general OO design pattern architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Correct exception handling, e.g., input values, abstract methods not implemented.	Programs do not or do not fully demonstrate code elegance on any of the following: • Classes adhere to a general OO design pattern architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Correct exception handling, e.g., input values, abstract methods not implemented.

Marking Cover Sheet

Assessment 01: Design Patterns

IN710: Object-Oriented Systems Development

Level 7, Credits 15

Bachelor of Information Technology

Name: _____ Date: _____

Learner ID: _____

Assessor's Name: _____

Assessor's Signature: _____

Criteria	Out Of	Weighting	Final Result
Written, Programming & Unit Testing	**10**	**60**	
Git Usage	**10**	**10**	
Code Elegance	**10**	**15**	
Functionality & Robustness	**10**	**15**	
		Final Result	**/100**
This assessment is worth 25% of the final mark for the Object-Oriented Systems Development course.			