University of South Australia

UniSA STEM

Object-Oriented Programming

Assignment
Tons-o-Dice
Part 2- Implementation

Introduction

The assignment is intended to provide you with the opportunity to put into practice what you have learnt in the course by applying your object-oriented programming knowledge and skills. It will require you to apply object-oriented methods to design, implement, and test a text-based application software where users can play dice games.

The assignment is worth for 30% of the assessment in this course, and consists of two parts:

- Part 1 – Design (10%)
- Part 2 – Implementation (20%)

This document specifies the tasks for Part 2 of the assignment which is Python implementation of the software, while it should be read in conjunction with the Part 1 specification on design provided as a separate document on the course website.

This assignment (Part 2) is an **individual task** that will require an **individual submission**. You are required to **submit your work via course website as prescribed in the Submission Details section.**

This document is a kind of specification of the required end product that will be generated by implementing the assignment. Like many specifications, it is written in English and hence will contain some imperfectly specified parts. **Please make sure you seek clarification** if you are not clear on any aspect of this assignment.

Course Objectives

By completing this assignment, you are expected to partially fulfill the following course objects:

- CO1. Convert a problem statement into an object oriented solution
- CO2. Use inheritance and polymorphism to solve problems
- CO3. Debug and fix code defects
- CO4. Apply international coding style standards
- CO5. Plan and conduct testing of software
- CO6. Analyse and explain the behaviour of object oriented programs

Assignment Learning Outcomes

- Implement an Object-Oriented designed software in Python.
- Apply object-oriented principles including abstraction, data hiding, encapsulation, inheritance, and polymorphism to software implementation.
- Practice Python programming skills including commenting with docstrings, handling exceptions, conducting unit tests, and version control with git.

The Assignment Task Specification

The task for the Part 2 of the assignment is to implement Tons-o-Dice in Python, a text-based application software designed in Part 1 of the assignment. The assignment will require you to revise your design in UML, implement, test, and debug the software practising object-oriented programming principles and Python programming skills. In addition, the assignment will require you to use version control to keep track of your implementation as you progress through the assignment and to document your classes appropriately.

Requirement 0. Header comment with personal details.

All of the Python files you submit must have the following block of comments at the top including the filename, description, and your student details:

```
#
# File: filename.py
# Descrition: A brief description of this Python module.
# Author: Steve Jobs
# Student ID: 12345678
# This is my own work as defined by
#  the University's Academic Misconduct Policy.
#
```

Requirement 1. Correct implementation in Python according to the specification in Part 1.

Your implementation must be in Python 3.9 or above. Your submission must implement all the features to correctly produce the behaviours and output as specified in the Assignment Part 1 specification document. Make sure you have a good review of the "The Assignment Task Specification" section in the Assignment Part 1 specification document which describes the features and behaviours of the software in details with sample outputs. The submission will be marked against these specifications and sample output provided in the Assignment Part 1 document.

Requirement 2. Apply proper Object-Oriented design principles to implementation.

Your implementation must practice Object-Oriented design principles as following:

1) Abstraction

   Your implementation must include definition of **at least 8 classes** including the `TonsODice` class. All of your code must belong to one of the classes you define, except the import statements and the following code which will run the software:

   ```
   tod = TonsODice()

   tod.run()
   ```

   This `run()` method in the `TonsODice` class must be the starting point of your software and call methods of other objects in the system.

2) Data Hiding and Encapsulation

All of the data attributes defined in your classes must be **private**. In addition, following conditions must be met on particular attributes:

- Player's name must not be changeable once initialised (i.e., must be read only).
- The number of chips of each player must be only allowed for being increased or decreased by an asked amount but never to be set with a value (i.e., no setter method, but with methods to increase or decrease).
- The number of games played and won must be only allowed for being increased by one. (i.e., no setter method, but with a method to increase).

3) Inheritance and Polymorphism

There should be at least one inheritance relationship used and at least one instance of practicing polymorphism (i.e., overriding a method in the superclass). Overriding an initialiser or string conversion method is not counted (i.e., you must override a method you defined in the superclass). A subclass must not repeat the code already in its superclass.

Requirement 3. Documentation with docstrings.

Your code must be documented appropriately using docstrings. If you are not familiar with docstrings, please review week 1 tutorial or the section "Explaining yourself" in the textbook on page 42 for more information.

Each class and each method should have at least one docstring which can be brief. The responsibilities that you have specified in the UML diagram can be used as a starting point for the class docstring.

Document as you go, not all at the end. One strategy is to write a brief comment about what the "1 thing" the method is supposed to do when you declare it, then refer to that comment while implementing it: this helps maintain focus when implementing the method and helps stop yourself from making it do more than the '1' thing it is supposed to do.

Requirement 4. Use consistent coding style.

Your Python code must adhere to consistent coding style. Your class names must be in `CapitalCamelCase` style, while the methods and attributes could be in either `smallCamelCase` or `under_score` style. While you may use one or the other it should be consistent throughout your code. If you are not sure, follow the Python style guide available here: https://www.python.org/dev/peps/pep-0008/

Requirement 5. Use exception handling.

Your implementation must use exception handling for dealing with incorrect user inputs or any other type of exceptions. For example, your program should not halt (a.k.a. crash) if the user gives non-numerical input where a number is expected, but rather handle such exceptions using try-except statements.

Requirement 6. Version control using git.

You must use git version control to keep track of your progress during implementation. If you are not familiar with git, please review the material on Week 6.
(Note: Only the local git repository is required in this assignment. You are not required to use an online version control repository, but if you choose to, please make sure your repository is **private**.)

You should perform regular commits as you implement features and fix errors (at least 10 commits must be made). Ensure each commit comment contains a short subject line. Your commit comments should reflect the context of the changes that were made in each commit—a fellow developer can always diff the contents to see exactly what changed but that does not provide the context.

Your submission should comprise your entire version control repository specific to the assignment. For example, you must create the git repository in the directory for your assignment project. Do not commit your assignment to a repository created in a parent folder or any other location. Your repository must contain all source files required to run your program.

Requirement 7. Conduct testing.

Your submission must include Python module(s) for testing. It is recommended that you test individual methods as you go, rather than trying to test the whole application through the menu-driven interface. Writing code to test specific elements will speed up development as you will not need to constantly enter data through the menu interface.

Your submission should have code for testing the methods of each class you define.

Requirement 8. Updated UML diagram

Update the UML diagram as you implement the program. You will need to submit an updated UML diagram with your submission.

Work Plan

To complete this assignment Part 2, you will need to perform the following steps:

1. Read the assignment specification for Part 2 carefully, in conjunction with the specification for assignment Part 1.
2. Setup your project folder with a git version control. Create Python file(s) you need and make sure they are added to the repository. Make sure you include a block of comments at the beginning of each file.
3. Implement the classes you've identified from your assignment Part 1 UML diagram. The TonsODice class and its run() method will be a good starting point, then expand to other classes.
4. Always include docstrings for documentation as you go. If you do this later, you may forget something, and it is more effort to insert documentation at a later stage.
5. Remember to use consistent coding style, and also make frequent commits to git repository.
6. Make sure your code adheres to the OOP design principles, including abstraction, data hiding, encapsulation, inheritance, and polymorphism. (Check Requirement 2 for more details.)
7. Update the UML diagram to reflect your implementation.
8. Start testing the software early: (1) Write tests for methods as you develop them. (2) Play the game and try out different things.
9. Include exception handling where necessary based on test results.
10. Submit your assignment (including both the zip archive of your Python project folder and the UML diagram).

Submission Details

You MUST submit the following files:

1) A zip archive (named after your student id, e.g., jobst123.zip) including your project folder which contains:
- One or more Python files that contain the implementation and test modules.
- Version control directory ('.git' folder)

2) Updated UML Class Diagram in **uxf** file, including a note with your student details.
- The file MUST be generated from the UMLet modelling tool and be legible: a photo of a hand-drawn diagram will NOT be accepted.

The submission is **due by the date and time specified on the submission link on the course website**.

This assignment is an **individual task** that will require an **individual submission.**

There will be **no extensions or late submissions for this course without one of the following exceptions:**

1. A medical certificate is provided that has the timing and duration of the illness and an opinion on how much the student's ability to perform has been compromised by the illness. Please note if this information is not provided the medical certificate WILL NOT BE ACCEPTED. Late assessment items will not be accepted unless a medical certificate is presented to the Course Facilitator. The certificate must be produced as soon as possible and must cover the dates during which the assessment was to be attempted. In the case where you have a valid medical certificate, the due date will be extended by the number of days stated on the certificate up to five working days.
2. A Learning and Teaching Unit councillor contacts the Course Facilitator on your behalf requesting an extension. Normally you would use this if you have events outside your control adversely affecting your course work.
3. Unexpected work commitments. In this case, you will need to attach a letter from your work supervisor with your application stating the impact on your ability to complete your assessment.
4. Military obligations with proof.

Applications for **extensions must be lodged with the Course Facilitator before the due date** of the assignment, or as soon as possible after the incident took place.

Note: Equipment failure, loss of data, 'Heavy work commitments' or late starting of the course are not sufficient grounds for an extension. Make sure you make a back-up of your files frequently.

Marking Criteria

Your submission will be marked against the specifications and the marking criteria outlined below. Penalties will be applied if the submission is not submitted in the correct format.

Criteria	Mark
Header comment with student details	5
Correct behaviour and output according to specification	10
Adheres to Object-Oriented design principles (abstraction, data hiding, encapsulation, inheritance, and polymorphism)	25
Documentation with docstrings	10
Consistent coding style	10
Exception handling	10
Git version control	10
Testing	10
Updated UML diagram	10
Submission not in correct format or missing student details.	Up to -15
Total Possible Marks	**100**

Academic Misconduct

This assignment is an individual task that will require an individual submission.

Students are reminded that they should be aware of the academic misconduct guidelines available from the University of South Australia website.

Deliberate academic misconduct, such as plagiarism, is subject to penalties. Information about Academic integrity can be found in Section 9 of the Assessment Policies and Procedures Manual at:

https://i.unisa.edu.au/policies-and-procedures/codes/assessment-policies/