華僑中學

Hwa Chong Institution
Secondary 4 Examination 2023

CANDIDATE NAME	

CLASS				REGISTER NUMBER		

Computing

2 Hours

Candidates to answer in the template files provided in the thumb drive.

READ THESE INSTRUCTIONS FIRST

Answer **all** questions and **save** your work constantly.

You are reminded of the need for clear presentation in your answers.

The number of marks is given in brackets [ ] at the end of each question or part question.

The total of the marks for this paper is **60**.

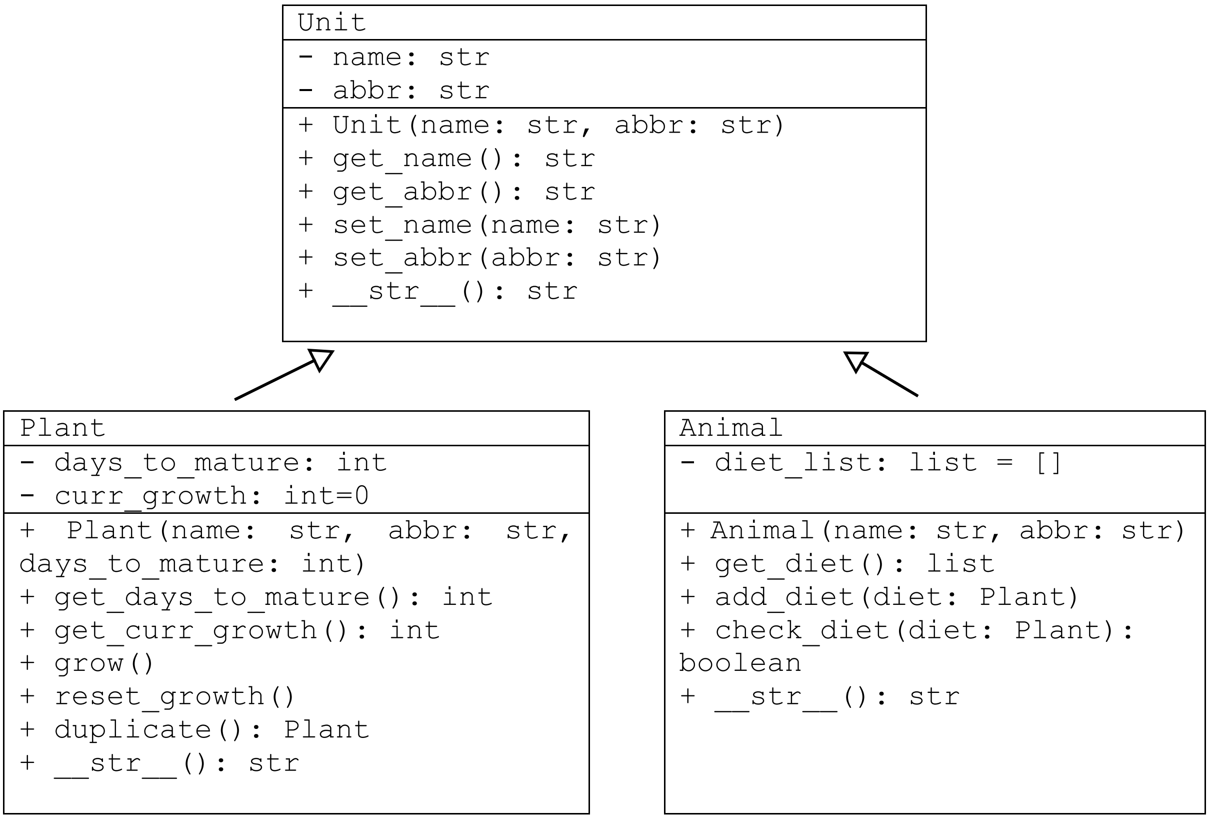
You are strongly encouraged to manage your time and spend **1 hour** for each section.

Section	Score
A	
B	
Total	

This document consists of **16** printed pages.

Section A - Python

You are tasked to implement an object-oriented simulation of plants and animals in a farm. All `Plant` and `Animal` objects are considered subclass objects of the `Unit` class. You may refer to the following UML class diagram for reference.



```
Unit
- name: str
- abbr: str
+ Unit(name: str, abbr: str)
+ get_name(): str
+ get_abbr(): str
+ set_name(name: str)
+ set_abbr(abbr: str)
+ __str__(): str
```

```
Plant
- days_to_mature: int
- curr_growth: int=0
+ Plant(name: str, abbr: str,
days_to_mature: int)
+ get_days_to_mature(): int
+ get_curr_growth(): int
+ grow()
+ reset_growth()
+ duplicate(): Plant
+ __str__(): str
```

```
Animal
- diet_list: list = []
+ Animal(name: str, abbr: str)
+ get_diet(): list
+ add_diet(diet: Plant)
+ check_diet(diet: Plant):
boolean
+ __str__(): str
```

Task 1.1

Implement `Unit` class according to the UML class diagram and following attributes / methods specifications. [5]

Attributes/Methods	**Specification**

`- name: str`
`- abbr: str`
	Each `unit` object should have private attributes `name` and `abbr` (abbreviation).

`+ Unit(name: str, abbr: str)`
`+ get_name(): str`
`+ get_abbr(): str`
`+ set_name(name: str)`
`+ set_abbr(abbr: str)`
	Constructor, getter and setter methods for `Unit` class.

`+ __str__(): str`	Return a string in the following format:
	`{name}: ({abbr})`
	
	e.g.
	`Grass: (g)`

Task 1.2

Implement Plant and Animal sub-classes according to the UML class diagram and the following attributes / methods specifications. [10]

Plant Class	**Specification**
- days_to_mature: int - curr_growth: int=0	days_to_mature is a positive integer value, indicating the number of days needed for the plant to mature. curr_growth is the current number of days that the plant has grown. It should be initialized with a default value of 0.
+ Plant(name: str, abbr: str, days_to_mature: int)	Constructor of the Plant class, takes in the values of name, abbr and days_to_mature; and initialize curr_growth to a default value of 0.
+ get_days_to_mature(): int + get_curr_growth(): int	Getter methods for the class.

+ grow()
+ reset_growth()
	grow() will increase the curr_growth value by 1.
	
	reset_growth() will reset the curr_growth value to 0.

+ duplicate(): Plant	When a plant is matured, it will duplicate and return a new Plant object with the same attribute values, and the curr_growth of the original Plant object will be reset to 0.
+ __str__(): str	Return a string in the following format: {name}: ({abbr}) Days to Mature: {days_to_mature} Current Growth: {curr_growth} e.g. Grass: (g) Days to Mature: 2 Current Growth: 0

Animal Class	**Specification**
- diet_list: list = []	Diet_list is a list containing Plant objects that the animal can eat. It should be initialized as an empty list.
+ Animal(name: str, abbr: str)	Constructor of the Animal class, takes in the values of name and abbr; and initialize diet_list to an empty list.
+ get_diet(): list	get_diet() is the getter method which returns the diet_list.
+ add_diet(diet: Plant)	This method will add a new Plant object, diet, into the diet_list.
+ check_diet(diet: Plant): Boolean	Iterate through the diet_list, and use abbr value to compare, if the given Plant object diet can be eaten by the animal. You may assume the abbr of Plants are all unique, and no two kinds of Plants would share the same abbr value.
+ __str__(): str	Return a string in the following format: {name}:({abbr}) eats: {diet1_name}, {diet2_name} … e.g. Sheep: (S) eats: Grass, Corn

Task 1.3

Create Plant and Animal objects based on the following input and generate test cases to test your class implementation. [4]

You may assume that all Plants would use lower case letters for their abbr values; and all animals would use upper case letters for their abbr values.

Create the following Plant objects (name, abbr, days_to_mature):
"Grass", "g", 2
"Corn", "c", 3
"Wheat", "w", 4

Create the following Animal objects (name, abbr):
"Sheep", "S"
and add the following Plant objects into its diet_list:
Grass, Corn

Use print statement to print out all the Plant and Animal objects.

For each of the 3 Plant objects created earlier, check if it can be eaten by the Animal object (Sheep) based on its diet_list.

Task 1.4

Implement `Farm` class according to the UML class diagram and following attributes / methods specifications. [7]

Farm
- size: int - map: list = []

+ Farm(size: int)
+ reset_map()
+ get_size(): int
+ set_size(size: int)
+ add_unit(unit: Unit, row: int, col: int)
+ get_unit(row, col): Unit
+ display()

<table>
<tr><th>Attributes/Methods</th><th>Specification</th></tr>
<tr><td>- size: int
- map: list = []</td><td>size indicates the dimension of the squarish map.

map is a 2-dimensional list, which is initialized as a size x size 2d list filled with None values.</td></tr>
<tr><td>+ Farm(size: int)</td><td>Constructor of the Farm class, takes in the values of size; and initialize map to a size x size 2d list filled with None values.</td></tr>
<tr><td>+ reset_map()</td><td>Reset the map, re-generate a size x size 2d list filled with None values.</td></tr>
<tr><td>+ get_size(): int
+ set_size(size: int)</td><td>Getter and setter methods for size.</td></tr>
<tr><td>+ add_unit(unit: Unit, row: int, col: int)</td><td>Add a Unit object to the (row, col) position.</td></tr>
<tr><td>+ get_unit(row, col): Unit</td><td>Get the current object from the (row, col) position.</td></tr>
<tr><td>+ display()</td><td>Output the map with borders and Unit objects stored in the map to the user.

For example, the following map has:
a Plant object grass (g) at (0,0);
and an Animal object Sheep (S) at (3, 4).

+-----+
|g |
| |
| |
| S|
| |
+-----+</td></tr>
</table>

Task 1.5

In the Farm class, implement the following additional methods to make the Farm class more realistic. [4]

Attributes/Methods	**Specification**
+ plant_grow(row, col)	Increment the curr_growth attribute of the Plant object by 1. If the Plant object reaches maturity, indicated by the days_to_mature value, it should find a **random available** position within the 3x3 grid surrounding the current object. Subsequently, duplicate the current Plant object and place it at the chosen location.
+ animal_eat(row, col)	The Animal object attempts to look for food in the 3x3 grid surrounding the current object. If there is food matching to its dietary preference, the Animal object should randomly select one of these available Plant objects, move to the plant's position, and eat it. If there isn't any food matching to its dietary preference, the Animal object should randomly move to an empty position in the 3x3 grid surrounding its current position.

Note: There is no need to consider the actions required by the Plant and Animal objects after the grow and eat actions.

- End of Section A -