Promo Revision Set 2

Topics covered:

- OOP
- File processing
- Data validation
- `random` module
- Stack ADT

Estimated time to complete: 2.5 hours

Task 1

A school wants to use Object-Oriented Programming (OOP) to model the staff working in the school.

The `Staff` class will store the following data:

- `Name: STRING`
- `Date: STRING` with 8 digits representing the date the staff joined the school in DDMMYYYY format.
  For example, `20062015` represents 20 June 2015.

The `Staff` class will have the following methods:

- `display_name()`: displays a string which is a unique identifier constructed as follows: `<Name>_<Year>`, where `<Year>` is the year the staff member joined the school. For example, `"John Tan_2016".`
- `length_of_service()`: returns the number of years (rounded down) that the staff has worked at this school compared against the system's date.

Task 1.1

Write program code to define the `Staff` class. [6]

The `Teacher` class inherits from `Staff`. The `Teacher` class will store the following additional data:

- `Classes_taught: STRING` that contains the two classes taught by the teacher

and the following methods:

- `display_name()` displays a string which is a unique identifier constructed as follows: `<Classes_taught>_<Name>_<Year>.` For example, `"1SA4_1SB8_Peter Lim_2018"`. This method overrides `display_name()` in the parent class.

Task 1.2

Write program code to define the `Teacher` class. [5]

The text file `Teachers_info.txt` stores detail regarding every teacher in the following format:

`<Name>, <Subj_class1>, <Subj_class2>, <Date>`

The data contained in `Teachers_info.txt` should conform to the following specification:

- `Name`: The first character of every distinct word in the name is capitalized. It should only contain letters and spaces, and there are no digits or punctuation. Example: `Lim Ann Lynn`
- `Subj_class`: 4-character string beginning with either `1` or `2`, followed by two capital letters and ending with a digit between `1` and `9`. Example: `1PH4`
  `Subj_class1` and `Subj_class2` follow the same format.
- `Date`: 8-digit string representing the date the staff joined the school in DDMMYYYY format. The date should be a valid date.
  - January, March, May, July, August, October and December have 31 days;
  - April, June, September and November have 30 days;
  - February has 29 days if the year is a multiple of 4, and 28 days otherwise.
  - The year should be between 1970 and 2021 (both inclusive).

Task 1.3

Write code to:

- Check that the data contained in `Teachers_info.txt` conforms to the above specified format.
- Write all error free entries to the file `Teachers_validated.txt`
- Create a `Teacher` object for every teacher with valid information
- Create a list of all the `Teacher` objects created. [16]

Task 2

A programmer is writing a program to implement a role-playing computer game using Object-Oriented Programming (OOP).

The players have to collect `Food` items. A `Food` item has the following attributes:

- `name : STRING`
- `value : INTEGER`

and the following methods:

- `get_name()`
- `get_value()`

A `player` takes on the role of a `Person`. A `Person` has the following attributes:

- `name : STRING`
- `health : INTEGER` which is initialised at a value of `100`
- `strength : INTEGER` which is initialised at a value of `100`

and the following methods:

- `get_name()`
- `get_health()`
- `get_strength()`
- `eat(Food)` adds the value of the `Food` to the `strength`. The code should display the new `strength` of the `player`.
- `attack(opponent)` where `opponent` is another `Person`.

For the attack method:

- A random integer `r` between `1` and `10` (inclusive) is generated.
- If the `strength` of the `player` is less than `r`, then the `player` does not have enough `strength` to attack and there is no change to the `health` of `the opponent`.
- If the `strength` of the player is at least `r`, then the attack is successful and the `health` of the `opponent` is decreased by `r`.
    - If the `health` of the `opponent` is now negative, then the `opponent` has been defeated.
- The `strength` of the player is decreased by `r`.

The are two sub-classes of the `Person` class – `Healer` and `Warrior`.

The `Healer` sub-class has one additional method:

- `heal(patient)` where `patient` is another `Person`.

For the `heal` method:

- A random integer `r` between `1` and `10` (inclusive) is generated.
- If the `strength` of the `player` is less than `r`, then the `player` does not have enough `strength` to heal and there is no change to the `health` of the `patient`.
- If the `strength` of the `player` is at least `r`, then the healing is successful and the `health` of the `patient` is increased by `r`, up to a maximum of `100`.

The `attack` method for a `Warrior` is twice as effective. This means that if the player has enough `strength` to attack, the `health` of the `opponent` is decreased by `2*r`, while the `strength` of the player is decreased by `r`.

Task 2.1

Write program code to define the class `Food`. [3]

Task 2.2

Write program code to define the class `Person`.

The code should display appropriate messages about the outcome of `attack`, including the new `value` of the `health` of the `opponent`. [10]

Task 2.3

Use appropriate inheritance to write program code to define the class `Healer`.

The code should display appropriate messages about the outcome of `heal`, including the new `value` of the `health` of the `patient`. [4]

Task 2.4

Use appropriate inheritance and polymorphism to write program code to define the class `Warrior`. [2]

Task 2.5

Test your code with the following steps in order:

- Create a `Food` item with `name 'Cheese'` and `value 10`.
- Create a `Warrior` with `name 'Sam'`.
- Create a `Healer` with `name 'Alex'`.
- Create a `Person` with name `'Jan'`.
- `'Jan'` attacks `'Sam'`.
- `'Sam'` attacks `'Jan'`.
- `'Alex'` heals `'Jan'`.
- `'Sam'` eats `'Cheese'`.

[3]

[ACJC/Prelim/20 (modified)]

Task 3

A stack is an abstract data type (ADT) which has the Last-In-First-Out (LIFO) property.

Task 3.1

Write program code to implement the following stack operations:

- `make_stack()` to construct an empty stack using an empty list.
- `push(stack,item)` to push an `item` into the `stack`.
- `pop(stack)` to remove and return the element on the top of the `stack`. Return `None` if the stack is empty.
- `size(stack)` to return the size of the `stack`. A size of zero indicates that the `stack` is empty.

(Note that this question does not require you to use OOP or having a fixed-sized stack. You may use Python list operations to complete this question.) [4]

Arithmetic expressions can be represented using the *Postfix* notations. They do not require the use of parenthesis or any understanding of precedence in the mathematical operators. The expressions can be easily evaluated using a stack ADT.

Some examples of the *Postfix* and the corresponding *Infix* notations:

Postfix notation	Infix notation
`'123*+'`	`'1+2*3'`
`'12+3/'`	`'(1+2)/3'`
`'12+43-*'`	`'(1+2)*(4-3)'`

We will only use single digit integers and the operators `+`, `-`, `*` and `/` in this question.

The following algorithm can be used to evaluate the expression written in the *Postfix* notation:

- Create an empty stack to store the element.
- Scan each element in the given expression, from left to right, and do the following:
  - if the element is a number, push it into the stack.
  - if the element is an operator, pop the last two numbers out from the stack, evaluate with the operator and push the result back into the stack.
- After scanning the full expression, pop the number out of the stack to obtain the evaluated answer for the expression.

Task 3.2

Write program code `postfix(string)` to evaluate the `string` for the arithmetic expression in the *Postfix* notation. [6]

[YIJC/Promo/20]