Answer all questions. (Total 100 marks)

Question 1

Formulate classes and use objects to store and compute tabular information.

A small grocery in your neighbourhood needs to track the items that it carries. An item is either perishable or non-perishable. Test data for three items is given in Table Q1.1 below:

Item Code	Item Name	Quantity on Hand	Cost Price($)	On Offer	Expiry
P101	Real Raisins	10	2	Yes	10 Dec 2018
NP210	Tan Baking Paper	25	2	No	NA
P105	Eggy Soup Tofu	14	1.85	Yes	26 Nov 2018

Table Q1.1

The class Item is an abstract class. It has the following methods:

- Getter and setter methods for both quantity on hand and cost price. Use the property decorator.
- A method that changes the attribute for whether the item is on offer. If the item is not on offer, make the item to be on offer. If the item is on offer, make it to be not on offer.
- A method for the selling price which is the sum of its cost price and a profit margin, dependent on the type of item.
- A method for the offer price which is the selling price minus a discount. The discount rate is dependent on the type of item.
- Abstract methods for computing the profit margin and the discount rate when the item is on offer.
- A string method that returns all its attributes except the cost price as a string. The string also includes the selling price if the item is not on offer or the offer price if the item is on offer.

Two examples are shown here:

```
P009   Tan Baking Paper   Available =  25 $3.90 (No Offer)
P009   Tan Baking Paper   Available =  25 $3.51 **Offer**
```

The class Perishable has an additional attribute: expiryDate. It has the following methods:

- A method that returns the profit margin which is 25% of its cost price.
- A method that returns the discount rate when the perishable item is on offer. The discount is dependent on the number of days before expiry according to Table Q1.2:

Days away from expiry	Discount
At least 30 days	10% of selling price
At least 15 days but less than 30 days	20% of selling price
Less than 15 days	30% of selling price

Table Q1.2

- The string method returns a string that includes the expiry date and the type of item.

Two examples are shown here:

```
P009   Real Rasins Available = 10 $2.50 (No Offer) Expiry date: 20 Nov 2018 Perishable Item
P009   Real Rasins Available = 10 $1.75 **Offer**  Expiry date: 20 Nov 2018  Perishable Item
```

The class NonPerishable has no additional attribute. It has the following methods:

- A method that returns the profit margin which is 30% of its cost price.
- A method that returns the discount rate of 10% for non-perishable item on offer.
- The string method returns a string that includes the type of item.

Two examples are shown here:

```
P009   Tan Baking Paper   Available = 25 $3.90 (No Offer)  Non-Perishable Item
P009   Tan Baking Paper   Available = 25 $3.51 **Offer**   Non-Perishable Item
```

(a) Define the class Item.

(10 marks)

(b) Define the class Perishable.

(6 marks)

(c) Define the class NonPerishable.

(4 marks)

(d) Suppose there is a class Grocer that has a collection of Item objects called items. Define the following methods to perform the following tasks:

- List items.
- Update the quantity on hand of an item when given an item code and a quantity to add to the quantity on hand. Return True if operation is successful and False otherwise.

(3 marks)

(e) Discuss **TWO (2)** instances for each of **THREE (3)** SOLID principles you have applied. Explain how you have applied them in your solution.

(12 marks)

Question 2

Apply the principles of object-oriented design in solving the following problem.

A module has a code, a name and has one or more assessment components. For each assessment component, the component type, the minimum required score for passing the component and the weightage are recorded.

Test data for two modules is given in Table Q2.1 below:

Module Code	Module Name	Component	Minimum score required	weightage
MA101	Introduction to Computing Math	Quiz Assignment Exam	50 45 40	1.5 3 10
IT128	Introduction to Programming	Practical Quiz Exam	60 40 50	1 0.5 3

Table Q2.1

You are given the class Module below:

```
class Module():
    def __init__(self, modCode, modName):
        self._modCode = modCode
        self._modName = modName
        self._minScoresNWeights = {}

    def setMinScoreNWeight(self, componentType, minScore, weight):
        self._minScoresNWeights[componentType] = [minScore, weight]

    def listMinScoresNWeights(self):
        s = 'Minimum Scores for Clearing Components and their weights:\n'
        keys = list(self._minScoresNWeights.keys())
        keys.sort()
        for k in keys:
            s += '{:10}: {:3d}  {:3}\n'.format(k, \
                                        self._minScoresNWeights[k][0],\
                                        self._minScoresNWeights[k][1])
           return s


    def __str__(self):
        return 'Code: {:5s}  Name: {:30s}\n{}'.\
            format(self._modCode, self._modName,  self.listMinScoresNWeights())
```

The class Module has a module code and a module name as attributes. In addition, it has a collection to record the component types, the minimum scores required to pass the components and their weightage. The following methods perform the following tasks:

(i) **setMinScoreNWeight**

If the component is new, it adds the component into the collection, along with the minimum score and the weightage.

If the component is existing, it updates the minimum score and the weightage given a component type, the new minimum score and the new weightage.

(ii) `listMinScoresNWeights`
Return the details of the components (type, minimum score and weightage) for the module, as a string.

(iii) `__str__`
Return a string with all its attributes including the collection as a string. For example,

For an example:

```
Code: MA101  Name: Introduction to Computing Math
Minimum Scores for Clearing Components and their weights:
Assignment:  45    3
Exam      :  40   10
Quiz      :  50  1.5
```

(a) Define a user exception class called ModuleScoreException.

(2 marks)

(b) Define the class ModuleScore which has a module as its attribute. In addition, it has a collection to record the scores for each component.

When a student did not perform a component, in this case, the quiz component for IT128, no score for that component is recorded as shown in Table Q2.2 below:

For Module	Component	Score
MA101	Quiz Assignment Exam	75 63 50
IT128	Practical Exam	85 45

Table Q2.2

Include the following methods to perform the following tasks:

- `addScore`
  Add a component and the score to the collection. The method raises an exception if the component is not a component for its module or if the new score is invalid. An invalid score is less than 0 or greater than 100.

  (5 marks)

- `updateScore`
  Update the score of a given a component of the module. The method raises an exception if the component is not a component for its module or if the new score is invalid.

  (2 marks)

- `removeScore`
  Remove a component and the score from the collection. The method raises an exception if the score of the component is not recorded yet for its module.

  (2 marks)

- `scoreStr`
  Return a string with the details of the components, scores and relative weightages computed using the method `getTotalWeight`.

  For example:

```
Scores
Assignment:  63 @weight 0.2
Exam      :  50 @weight 0.7
Quiz      :  75 @weight 0.1
```

  If `getTotalWeight` raises an exception, the method returns the exception error message as the score string.

  (2 marks)

- `hasPassedComponents`
  Return True if all the scores are at least the required minimum for the respective components. If any component is not cleared at the required minimum, return False

  (2 marks)

- `getTotalWeight`
  Sum the weightages of the component in its module and return the sum. The method raises an exception if the sum is 0 or negative.

  (2 marks)

- `finalScore`
  Compute and return the final score through these steps:
  - Get the total weightages of its components in its module. If the method `getTotalWeight` raises an exception, handle it by returning -1 as the final score.
  - Otherwise, calculate the relative weightages of the components by dividing each weight by the total of the weights.
  - Sum the product of the scores with the relative weights of the components.

  (2 marks)

- `grade`
  Return the grade of the score using Table Q2.3. If the score is -1, return 'Invalid Grade' instead.

Score	Grade
Failing any component or having a final score below 50	F
Score at least 50 but less than 60	D
Score at least 60 but less than 70	C
Score at least 70 but less than 80	B
Score at least 80	A

Table Q2.3

(2 marks)

- `__str__`
  The method returns all its attributes including the collection as a string.

  For example:

```
Code: MA101  Name: Introduction to Computing Math
Minimum Scores for Clearing Components and their weights:
Assignment:  45    3
Exam      :  40   10
Quiz      :  50  1.5

Scores
Assignment:  63 @weight 0.2
Exam      :  50 @weight 0.7
Quiz      :  75 @weight 0.1
Score 55 Grade D
```

  If the final score is -1, indicate the score as 'Invalid Score'.

  For example:

```
Code: MA101  Name: Introduction to Computing Math
Minimum Scores for Clearing Components and their weights:
Assignment:  45    3
Exam      :  40  -10
Quiz      :  50  1.5

Total weights of component -5.5 is invalid Score Invalid Score
Grade Invalid Grade
```

(2 marks)

(c) A student may take many modules. Test data for one student is given in Table Q2.4 below:

Number	Name	For Module	Component Type	Score
12345X	Peter Lim	MA101	Quiz Assignment Exam	75 63 50
		IT128	Practical Exam	85 45

Table Q2.4

Define the class Student which has a number and a name as attributes. In addition, it has a collection of ModuleScore objects. Include the following methods to perform the following tasks:

- `addModule`
  Add a new ModuleScore object to the collection. The module code must be unique. Return True if operation is successful and False if otherwise.
- `searchModule`
  Search for a ModuleScore object, given a module code. The method returns the ModuleScore object if located. Otherwise it returns none.
- `removeModule`
  Remove a ModuleScore object, given a module code. Return True if operation

is successful and False if otherwise.

- `listModules`
  Return the details of the collection of ModuleScore objects
- `__str__`
  Return all its attributes including the collection as a string.

For example:

```
Student: 12345X     Peter Lim
Code: IT128  Name: Introduction to Programming
Minimum Scores for Clearing Components and their weights:
Exam      :  50    3
Practical :  60    1
Quiz      :  40  0.5

Scores
Exam      :  45 @weight 0.7
Practical :  85 @weight 0.2
Score 49 Grade F

Code: MA101  Name: Introduction to Computing Math
Minimum Scores for Clearing Components and their weights:
Assignment:  45    3
Exam      :  40   10
Quiz      :  50  1.5

Scores
Assignment:  63 @weight 0.2
Exam      :  50 @weight 0.7
Quiz      :  75 @weight 0.1
Score 55 Grade D
```

(8 marks)

(d) Explain the similarity and difference in the **TWO (2)** relationships:

(i) Module and ModuleScore

(ii) ModuleScore and Student

(4 marks)

Question 3

(a) Design and develop a simple graphical user interface (GUI) to allow two players to play a guessing game with event handling.

(i) Figure Q3.1 shows how the GUI initially looks like: the Play Again button is disabled, but the Submit button is enabled.

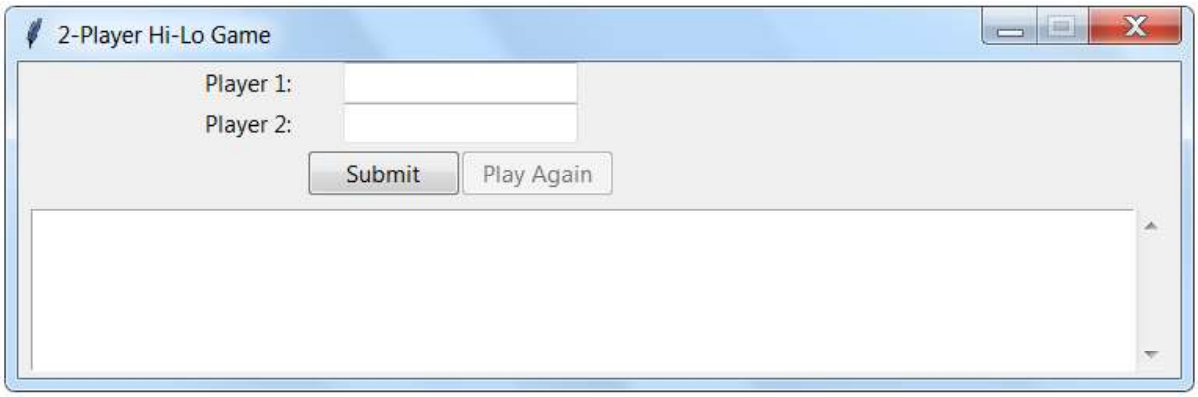

Figure Q3.1

Figure Q3.2 shows how the GUI looks like when the Submit button has been clicked twice: The player who guesses a number closer to the hidden number wins a point. If both players' guesses are equidistant to the hidden number, they both do not get any point.

Figure Q3.2

(11 marks)

(ii) Figure Q3.3 shows a player who makes a correct guess earns 2 points, in addition to the 1 point he earns if his number is closer to the hidden number.

(6 marks)

(iii) Figure Q3.3 shows when the hidden number is guessed, the Play Again button is enabled. The players can then start a new game, with both of their scores reset to zero.

(3 marks)

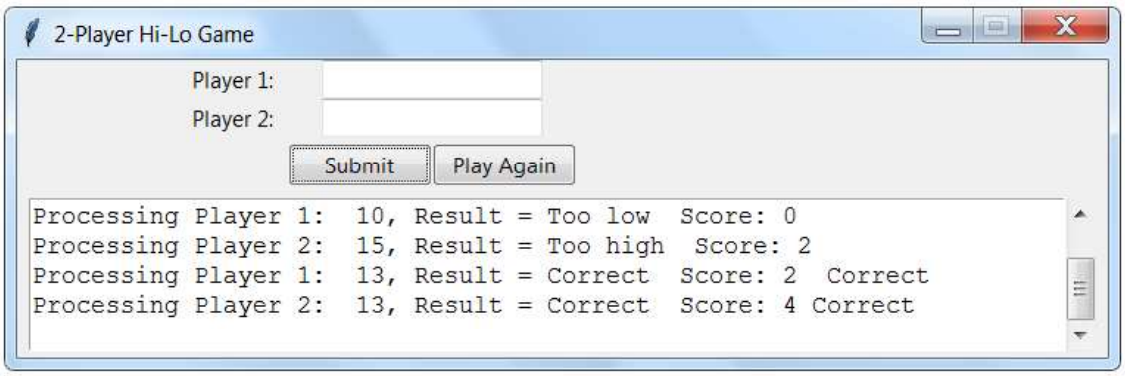

Figure Q3.3

(b) Make **THREE (3)** improvements to the GUI so that

(i) The submit button is disabled when the Play Again button is enabled.
(ii) The score of the players continues to the next game.
(iii) It incorporates your choice of improvement to the UI, with either additional game rule or buttons, or both.

(10 marks)

----- END OF ECA PAPER ----