Inspiring Excellence

Course Code:	**CSE111**
Course Title:	**Programming Language II**
Classwork No:	**05**
Topic:	**OOP (HAS-A relationship and access modifier)**
Number of tasks:	**3**

Task 1

Please write the **Student** and **Department** class with the necessary properties so that the provided driver code generates the output given below. Make sure the **ID and CGPA** attributes in the **'Student'** class are private and cannot be accessed directly from outside of the class.

Driver Code	Output

`s1 = Student("Akib", 22301010, 3.29)`
`s2 = Student("Reza", 22101010, 3.45)`
`s3 = Student("Ruhan", 23101934, 4.00)`
`print("1==========")`
`cse = Department("CSE")`
`cse.findStudent(22112233)`
`print("2==========")`
`cse.addStudent(s1,s2,s3)`
`print("3==========")`
`cse.details()`
`print("4==========")`
`cse.findStudent(22301010)`
`print("5==========")`
`s4 = Student("Nakib",22301010,3.22)`
`cse.addStudent(s4)`
`print("6==========")`
`s4.setId(21201220)`
`cse.addStudent(s4)`
`print("7==========")`
`cse.details()`
`print("8==========")`
`s5 = Student("Sakib",22201010,2.29)`
`cse.addStudent(s5)`
`print("9==========")`
`cse.details()`
	1==========
	Student with this ID doesn't exist, Please give a valid ID
	2==========
	Welcome to CSE department, Akib
	Welcome to CSE department, Reza
	Welcome to CSE department, Ruhan
	3==========
	Department Name: CSE
	Number of student:3
	Details of the students:
	Student name: Akib, ID: 22301010, cgpa: 3.29
	Student name: Reza, ID: 22101010, cgpa: 3.45
	Student name: Ruhan, ID: 23101934, cgpa: 4.0
	4==========
	Student info:
	Student Name: Akib
	ID: 22301010
	CGPA: 3.29
	5==========
	Student with the same ID already exists, Please try with another ID
	6==========
	Welcome to CSE department, Nakib
	7==========
	Department Name: CSE
	Number of student:4
	Details of the students:
	Student name: Akib, ID: 22301010, cgpa: 3.29
	Student name: Reza, ID: 22101010, cgpa: 3.45
	Student name: Ruhan, ID: 23101934, cgpa: 4.0
	Student name: Nakib, ID: 21201220, cgpa: 3.22
	8==========
	Welcome to CSE department, Sakib
	9==========
	Department Name: CSE
	Number of student:5
	Details of the students:
	Student name: Akib, ID: 22301010, cgpa: 3.29
	Student name: Reza, ID: 22101010, cgpa: 3.45
	Student name: Ruhan, ID: 23101934, cgpa: 4.0

	Student name: Nakib, ID: 21201220, cgpa: 3.22 Student name: Sakib, ID: 22201010, cgpa: 2.29

Task 2

Class Description:

Spaceship: This class represents a spaceship. Each spaceship has a **name** and a **capacity** (the maximum weight it can carry).

Cargo: This class represents a piece of cargo. Each cargo item has a **name** and a **weight**. Both attributes should be **private** which means they cannot be accessed directly from outside of the class.

A `Spaceship` contains (HAS) `Cargo`. That means each spaceship can carry multiple cargo items, but the total weight of the cargo cannot exceed the spaceship's capacity.

Your task is to design the **Spaceship** and **Cargo** class with necessary properties so that the given output is produced for the provided driver code.

Driver Code	Output

`# Creating spaceships`
`falcon = Spaceship("Falcon", 50000)`
`apollo = Spaceship("Apollo", 100000)`
`enterprise = Spaceship("Enterprise", 220000)`
`print("1.==========")`
`# Creating cargo`
`gold = Cargo("Gold", 20000)`
`platinum = Cargo("Platinum", 25000)`
`dilithium = Cargo("Dilithium", 50000)`
`trilithium = Cargo("Trilithium", 70000)`
`neutronium = Cargo("Neutronium", 80000)`
`print("2.==========")`
`# Loading cargo onto spaceships`
`falcon.load_cargo(gold)`
`falcon.load_cargo(platinum)`
`falcon.display_details()`
`print("3.==========")`
	1.==========
	2.==========
	Spaceship Name: Falcon
	Capacity: 50000
	Current Cargo Weight: 45000
	Cargo: ['Gold', 'Platinum']
	3.==========
	Spaceship Name: Apollo
	Capacity: 100000
	Current Cargo Weight: 20000
	Cargo: ['Gold']
	4.==========
	Warning: Unable to load Neutronium inside Falcon. Exceeds capacity by 75000.
	5.==========
	Spaceship Name: Enterprise
	Capacity: 220000
	Current Cargo Weight: 200000
	Cargo: ['Dilithium', 'Trilithium', 'Neutronium']

```
apollo.load_cargo(gold)  # Apollo will not reach its total capacity
apollo.display_details()
print("4.==================================")
falcon.load_cargo(neutronium)  # This should exceed Falcon's capacity
print("5.==================================")
enterprise.load_cargo(dilithium)
enterprise.load_cargo(trilithium)
enterprise.load_cargo(neutronium)  # This should not exceed Enterprise's capacity
enterprise.display_details()
```

Task 3

For this task, you will be designing classes for a music player app. There are 2 classes involved:

The **Song** class has:

- 3 private instance variables: `title`, `artist` and `duration`.
  - **`title (private) :`** A string containing the song title.
  - **`artist (private) :`** A string containing the artist title.
  - **`duration (private) :`** An integer representing the song duration time in **seconds**.
- An **`overloaded parameterized constructor`** to initialize the instance variables.
  - Parameters for `title` and `duration` are **required** but `artist` is **optional**.
  - Default value for `artist`: "`Unknown Artist`".
- **`info() :`** A method to **return** a string formatted as:
  "`{title} by {artist} ({min}:{sec})`".
  - One example: "`Never Gonna Give You Up by Rick Astley (3:33)`"

The **`Playlist`** class has:

- 3 private instance variables: `name`, `song_list` and `now_id`.
  - **`name (private) :`** A name for the playlist.
  - **`song_list (private) :`** A list containing **Song** objects.
  - **`now_id (private) :`** An integer representing the current song index in **`song_list`**.
- An **`overloaded parameterized constructor`** to initialize the instance variables.
  - Parameters for `name` is **required**, `song_list` is **optional** and `now_id` is always set by **default**.
  - Default value for `song_list` is `an empty list` and `now_id` is `-1`.
- **`play_next() :`** A method that updates `now_id` to the next index. If this method is called while the last song is being played, it will change the `now_id` variable to the first index of the `song_list`.

- **`play_prev()` :** Updates now_id to the previous index. If this method is called while the first song in the playlist is being played, it will show an error message like this, “Cannot go back from the first song.”, then play the first song again.
- **`add_songs(*songs)` :** A method that adds **all Song objects from the parameter** `songs` to its `song_list`.
- **`remove_song(song_obj)` :** A method to remove/delete `song_obj` from `song_list`.
- **`show_playlist()` :** A method that prints some playlist information.
  - An example output would be:

```
Playlist: Classical Orchestra
Total Songs: 3
1. The Flower Duet by Leo Delibes (1:26)
2. Winter by Vivaldi (3:43)
3. Ride of the Valkyries by Wagner (2:17)
```

- Appropriate **`getter`** and **`setter`** methods for accessing the variables.
- When displaying song details, **`play_next()`**, **`play_prev()`** and **`show_playlist()`** methods **must** call the **`info()`** method for each **Song** object.

Your tasks are:

- Implement the **Song** and **`Playlist`** classes ensuring proper encapsulation. All data members in both classes should be private and have corresponding getter and setter methods.
- Implement the necessary methods described above.
- You can inspect the following code snippet for better understanding: link . The link contains a demo driver code to make you understand what you are expected to do in each method. But in practice you have to write your own driver code to check if each and every method is working properly or not.