National University of Computer and Emerging Sciences

# OOP (CS1004)

Date: May 10th 2024

**Course Instructor(s)**

Ms. Mahnoor, Ms. Fatima, Ms. Shaharbano

## Lab Final Exam (A)

**Total Time: 2 Hours**

**Total Marks: 100**

**Total Questions: 03**

**Semester:** SP-2024

**Campus:** Karachi

**Dept.** Computer Science

Student Name _____ Roll No _____ Section _____ Student Signature _____

***Question Q1*** ***Weightage: 15;*** **Marks: 30; CLO 2**

---

Q1: : You've been tasked with creating a robust multimedia library system capable of handling various types of media, including videos and audios.

In designing this system, you're required to implement an abstract class named `Media` that forms the backbone for managing different media items.

The `Media` class includes crucial attributes such as `max_items`, `items` (a dynamic array of media item storing categories of media), and `item_count`. Moreover, it outlines the blueprint for essential methods like `add_media`, `remove_media`, and `update_media`.

Expanding upon the `Media` class are two specialized classes: `Video` and `Audio`.

The `Video` class extends `Media` with additional attributes such as `resolution`, `duration`, and an array of `video_titles`.

Similarly, the `Audio` class inherited from `Media` with some additional attributes like `artist`, `pitch`, and an array of `audio_titles`.

Both the `Video` and `Audio` classes meticulously override the methods inherited from the abstract base class `Media`, according to the specific requirements of video and audio management.

Your challenge is to Use polymorphism concept to add, remove and update media item efficiently across different types of media.

Note: The item is a pointer array stores the media item of audios and videos accordingly.

# National University of Computer and Emerging Sciences

***Question Q2*** ***Weightage: 15;* Marks: 30; CLO 3**

---

Imagine you're tasked with designing a class named `SensitiveDatabase` to handle secret data of an agency. This class includes private attributes such as `data_storage`, an array storing secret message (you can use simple messages i-e "Meeting will be at 10:00 agenda is search operation") that is stored dynamically, `data_count` storing the size of each string stored in `data_storage` and `encryption_key`, a key crucial for encryption purposes. To interact with the database securely, three methods, `fetch_data`, `update_data`, and `remove_data`, are provided.

In addition to these, there are two global functions granted special access to the private attributes of the `SensitiveDatabase` class:

1. `access_sensitive_data`: This function accesses and prints the attributes of the `SensitiveDatabase`. If the string passed to it is empty, it utilizes the `update_data` method to update the attributes as 'No message recieved' & encrypt using (String+2).
2. `perform_complex_computation`: This function executes a complex computation, involving encryption of data using the encryption key. The encryption formula used is ` (String + key) % 26`.

Your challenge is to implement these global functions to access the private attributes and methods of the `SensitiveDatabase` class, ensuring data integrity and encapsulation. Additionally, ensure proper error handling mechanisms are in place to address potential issues such as empty input strings, out-of-bounds access, and memory allocation failures.

***Question Q3*** ***Weightage: 20;* Marks: 40; CLO 4**

---

You're tasked with crafting a class hierarchy using templates to manage data about individuals engaged in sports. The foundational class is named `Person`, equipped with attributes such as `Name`, `Age`, and `Work`. It furnishes the following functionalities:

- A method named `getName` that retrieves the individual's name.
- A method called `getAge` that retrieves the individual's age.
- A method labeled `getWork` that furnishes details regarding the individual's occupation.
- A method named `getSalary` that provides the individual's salary.

A default constructor for `Person` is provided, initializing all variables using a generic implementation.

Additionally, there exist two generic derived classes:

1. `Player`: Inherits from the superclass `Person`, with an overridden function `getWork () ` to supply game-specific details. This class includes attributes like `RUNS`, `Matches`, `ODIs`, `T20s`, `Wickets`, `Fours`, and `Sixes`, pertinent to batting and fielding averages.
2. `Umpire`: Also inheriting from the `Player` class, `Umpire` further overrides the `getWork ()` function to furnish statistics pertaining to the umpiring career. Parameters within this class encompass `ump_career_start_year`, `noT20s`, `noODIs`, `noTests`, etc.

Both `Player` and `Umpire` classes possess parameterized constructors and accessor/mutator functions for their attributes. Additionally, a display function is provided to exhibit the details of each class.

Moreover, special functions are implemented for both `Player` and `Umpire` classes to compute and return the salary. These functions utilize operator overloading for pre and post increment operations, augmenting the salary by 10% and 15%, respectively, based on any distinctive characteristics.