NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES
FAST - PESHAWAR CAMPUS

Subject: Software Construction and Development Lab (CL-2001)
Instructor: Muhammad Saood Sarwar
Lab Task : 2

Question 1 : Employee Management System

You are tasked with building an advanced **Employee Management System** that handles different types of employees in a company. The system must adhere to software construction principles, including clean code, modularity, and encapsulation. Implement the following:

Base Class: Employee

1. **Attributes**:
    - name (str): Employee's full name.
    - age (int): Employee's age.
    - salary (float): Monthly salary of the employee.
    - position (str): Job position of the employee.
    - hire_date (str): Date of hiring the employee.
2. **Constructor**: Initializes the attributes.
3. **Destructor**: Prints a message when an object is deleted: "[Employee Name] has been removed from the system."
4. **Methods**:
    - **__str__()**: Returns a readable string representing the employee details: "Employee: [name], Position: [position]".
    - **__repr__()**: Returns a more developer-oriented representation: "Employee([name], [age], [salary], [position])".
    - **display_details()**: Prints employee details (name, age, salary, position, hire date).
    - **annual_bonus()**: Calculates and returns the annual bonus as 10% of the salary.
    - **promote(new_position, increment)**: Promotes the employee to a new position and increases their salary by the increment amount.
    - **demote(new_position, decrement)**: Demotes the employee to a new position and decreases their salary by the decrement amount. Ensure the salary doesn't go below a certain threshold.
    - **retirement_age()**: Returns how many years are left until the employee reaches the retirement age of 65.
    - **increase_salary(percentage)**: Increases the employee's salary by a given percentage using a lambda function.

- **compare_salary(other_employee)**: Compares the employee's salary with another employee using a method, returning "Higher", "Lower", or "Equal" based on the salary comparison.

Derived Class: Manager (inherits from Employee)

1. **Additional Attribute**:
    - team_size (int): Number of people the manager is responsible for.
2. **Additional Methods**:
    - **__str__()**: Overrides the base method to include the team size: "Manager: [name], Team Size: [team_size]".
    - **annual_bonus()**: Override to calculate the bonus as 15% of salary + 1% of salary for each team member.
    - **increase_team_size(new_members)**: Increases the team size by a specified number of new members.
    - **reduce_team_size(lost_members)**: Reduces the team size by a specified number of members.
    - **mentor_employee(employee)**: Prints a message indicating the manager is mentoring another employee.

Function with *args and **kwargs:

Write a function update_employee_info(employee, *args, **kwargs) that updates the employee's details dynamically. It should:

- Use *args to update position and salary.
- Use **kwargs to update additional attributes like name, age, and hire date.

Helper Function:

Write a helper function total_annual_cost(*employees) that takes a variable number of Employee or Manager objects and calculates the total annual salary and bonuses of all employees.

Instructions:

1. Use meaningful variable and method names following clean code principles.
2. Write test cases to demonstrate the following:
    - Creating multiple Employee and Manager objects.
    - Testing the functionality of __str__(), __repr__(), display_details(), promote(), demote(), retirement_age(), and increase_salary() methods.
    - Using update_employee_info() to dynamically update employee information using both *args and **kwargs.
    - Calculating total annual costs of multiple employees using total_annual_cost().