In [3]:
```
class Employee:
    def __init__(self, name, age, salary, position, hire_date=None, min_s
        self.name = name
        self.age = age
        self.salary = salary
        self.position = position
        self.hire_date = hire_date  # Optional parameter
        self.min_salary = min_salary  # Minimum salary threshold

    def __str__(self):
        # Used when you print the object or convert it to a string
        return f"Employee: {self.name}, Position: {self.position}"

    def __repr__(self):
        # Detailed string for developers
        return f"Employee('{self.name}', {self.age}, {self.salary}, '{sel

    def __del__(self):
        # Destructor: Called when an object is deleted
        print(f"{self.name} has been removed from the system.")

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Salary: {self.salary}")
        print(f"Position: {self.position}")
        if self.hire_date:
            print(f"Hire Date: {self.hire_date}")
        else:
            print("Hire Date: Not specified")

    def annual_bonus(self):
        return 0.10 * self.salary

    def promote(self, new_position, increment):
        self.position = new_position
        self.salary += increment
        print(f"{self.name} has been promoted to {self.position} with a s
        print(f"New Salary: {self.salary}")

    def demote(self, new_position, decrement):
        self.position = new_position
        new_salary = self.salary - decrement

        # Ensure salary doesn't fall below the minimum salary threshold
        if new_salary < self.min_salary:
            print(f"Cannot demote. Salary can't go below the threshold of
        else:
            self.salary = new_salary
            print(f"{self.name} has been demoted to {self.position} with
            print(f"New Salary: {self.salary}")

    def retirement_age(self):
        retirement_age = 65
        years_left = retirement_age - self.age
        return max(0, years_left)

    def increase_salary(self, percentage):
        # Lambda function to calculate the salary increase
        increase = lambda salary, percentage: salary + (salary * percenta
        self.salary = increase(self.salary, percentage)
        print(f"{self.name}'s salary has been increased by {percentage}%.

    def compare_salary(self, other_employee):
        if self.salary > other_employee.salary:
            return "Higher"
        elif self.salary < other_employee.salary:
            return "Lower"
        else:
            return "Equal"


class Manager(Employee):
    def __init__(self, name, age, salary, position, team_size=0, hire_dat
        # Inherit attributes from the Employee class using super()
        super().__init__(name, age, salary, position, hire_date, min_sala
        self.team_size = team_size  # New attribute specific to Manager

    def __str__(self):
        return f"Manager: {self.name}, Team Size: {self.team_size}"

    def annual_bonus(self):
        base_bonus = 0.15 * self.salary  # 15% of salary
        team_bonus = 0.01 * self.salary * self.team_size  # 1% for each t
        return base_bonus + team_bonus

    def increase_team_size(self, new_members):
        self.team_size += new_members
        print(f"Team size increased by {new_members}. New team size: {sel

    def reduce_team_size(self, lost_members):
        if lost_members > self.team_size:
            print("Cannot reduce team size below zero.")
        else:
            self.team_size -= lost_members
            print(f"Team size decreased by {lost_members}. New team size:

    def mentor_employee(self, employee):
        print(f"{self.name} is mentoring {employee.name}")


# Function to update employee information
def update_employee_info(employee, *args, **kwargs):
    # Update position and salary with *args
    if len(args) == 2:
        employee.position, employee.salary = args
    # Update other attributes with **kwargs
    for key, value in kwargs.items():
        if hasattr(employee, key):
            setattr(employee, key, value)

# Helper function to calculate total annual cost of employees
def total_annual_cost(*employees):
    total_cost = 0
    for employee in employees:
        total_cost += employee.salary * 12 + employee.annual_bonus()
    return total_cost
```

```

# Example usage
mgr = Manager("Yousaf Maaz", 35, 80000, "Engineering Manager", team_size=
emp = Employee("Rehan Khan", 28, 60000, "Software Engineer")

# Display manager details
print(mgr)

# Calculate manager's annual bonus
print(f"Annual Bonus: {mgr.annual_bonus()}")

# Increase and decrease team size
mgr.increase_team_size(3)
mgr.reduce_team_size(5)

# Mentoring another employee
mgr.mentor_employee(emp)

# Display employee and manager details
print(emp)
print(mgr)


# Example usage with Employee class
emp1 = Employee("Yousaf Maaz", 30, 50000, "Software Engineer", "2023-05-1
emp2 = Employee("Rehan Khan", 28, 60000, "Data Scientist", "2022-06-15")

emp1.display_details()
emp2.display_details()

# Comparing salaries
comparison = emp1.compare_salary(emp2)
print(f"Comparison between {emp1.name} and {emp2.name}'s salaries: {compa

# Other operations
bonus = emp1.annual_bonus()
print(f"Annual Bonus: {bonus}")

emp1.promote("Senior Software Engineer", 10000)
emp1.demote("Junior Software Engineer", 15000)
years_to_retirement = emp1.retirement_age()
print(f"Years left until retirement: {years_to_retirement}")
emp1.increase_salary(10)

# Using __str__ and __repr__:
print(emp1)
print(repr(emp1))

# Dynamically updating employee info
update_employee_info(emp1, "Team Lead", 70000, name="Ahmed Khan", age=31)
emp1.display_details()

# Calculating total annual cost of employees
total_cost = total_annual_cost(emp1, emp2, mgr)
print(f"Total Annual Cost: {total_cost}")
```

```
Manager: Yousaf Maaz, Team Size: 10
Annual Bonus: 20000.0
Team size increased by 3. New team size: 13
Team size decreased by 5. New team size: 8
Yousaf Maaz is mentoring Rehan Khan
Employee: Rehan Khan, Position: Software Engineer
Manager: Yousaf Maaz, Team Size: 8
Name: Yousaf Maaz
Age: 30
Salary: 50000
Position: Software Engineer
Hire Date: 2023-05-10
Name: Rehan Khan
Age: 28
Salary: 60000
Position: Data Scientist
Hire Date: 2022-06-15
Comparison between Yousaf Maaz and Rehan Khan's salaries: Lower
Annual Bonus: 5000.0
Yousaf Maaz has been promoted to Senior Software Engineer with a salary in
crease of 10000.
New Salary: 60000
Yousaf Maaz has been demoted to Junior Software Engineer with a salary dec
rease of 15000.
New Salary: 45000
Years left until retirement: 35
Yousaf Maaz's salary has been increased by 10%. New Salary: 49500.0
Employee: Yousaf Maaz, Position: Junior Software Engineer
Employee('Yousaf Maaz', 30, 49500.0, 'Junior Software Engineer')
Name: Ahmed Khan
Age: 31
Salary: 70000
Position: Team Lead
Hire Date: 2023-05-10
Total Annual Cost: 2551400.0
```

```
In [ ]:
```