**ІЄРАРХІЯ КЛАСІВ.**

**Варіант 1. Працівник.**

**(Сhanged)Клас Працівник абстракний.** Отже, клас працівник буде батьківським класом, від якого ми будемо наслідувати інші. Він буде мати 4 захищених(protected) поля, а саме: 1. string surname(Прізвище)

2. int experience(Досвід роботи)

3. double MoneyPerHour(Зарплатню за годину);

4. int minHours(Мінімальна к-сть годин)

Також він буде мати публічні конструктори(За замовчуванням, з параметрами, копіювання) і два віртуальних, метода які є зі значенням 0: один буде вираховувати зарплату(Salary), другий - констатний, виводити себе на екран(print ), які будуть застосовуватись для його підкласів, а також віртуальний деструктор.

Клас НЕ може створювати об’єкти типу Employer, оскільки являється абстракним.

**Клас програміст.** Клас буде моделювати сутність програміст. Цей клас буде підкласом Emlployer(public наслідування), він унаслідує від нього всі його захищені поля, а також буде мати своє особисте приватне поле - ActualyWorkedHours, яке буде зберігати в собі години, які дійсно відпрацьовані. Цей клас унаслідує публічні методи від класу Emlployer, проте перевизначить(override) їх. Метод Salary буде так само вираховувати зарплату, проте порівнювати ActualyWorkedHours < minHours, якщо істина, то зарплата буде зменшена на 25%. Метод print буде перевизначено і спочатку він виведе те, що виводить у Employer, додасть строку, яка буде відображати посаду, а також відпрацьовані години. Клас буде мати свої конструктори, які будуть викликати спочатку конструктор класу Employer та додавати ще одне поле ActualyWorkedHours

**Клас Керівник.** Клас буде моделювати сютність менеджер. Клас буде підкласом Emplorer(public наслідування), унаслідує всі захіщені поля, буде мати особисте приватне поле int workers(кс-ть. підлеглі). Також унаслідує всі методи від Employer, проте перевизначить їх. Salary буде обраховувати (40 годин \* на зарплату за годину) \* (1 + 0.01 за кожного воркера). Print виведе для початку те, що виводить Employee, додасть строку з посадою менеджер, а також кількість підлеглих. Клас буде мати свої конструктори, які спочатку будуть викликати конструктор Employer(minHours завжди 40!) і додавати поле workers.

**Клас керівник проектів.** Клас буде моделювати сутність керівник проектів. Він буде підкласом Manager(public наслідування)(отже буде наслідувати клас Employer?), унаслідує всі захищені поля класу Manager(Employer 🡺 Manager 🡺 Project manager??), буде мати власне поле-контейнер team, що зберігає кілька об’єктів класу Manager. Клас унаслідує всі віртуальні методи Manager, перевизначить їх. Salary буде спочатку рахувати ставку за відпрацьовані години і додавати 100 за кожного керівника. Метод print додасть строку, що буде виводити посаду та кількість підлеглих керівників. Клас буде мати власні конструктори, які спочатку будуть викликати конструктори Manager і додавати поле team.

--------------------------------------- Continue ---------------------------------------

"Дует" складається з двох програмістів, "тріо" - з трьох, "менторство" - з програміста наставника і стажера програміста, "ланка" - з двох керівників, "центр" - з керівника проекту і двох звичайних керівників, "група" - з керівника і двох програмістів. Кожна команда має назву, лідера (першого з учасників команди), вміє повідомляти дані лідера, вартість місячної оплати за роботу команди (сума доходів учасників, але ментор отримує +2% до звичної зарплати), окремо зарплати програмістів, керівників. Команди, де є програмісти, можуть призначати на певний проєкт, команди керівників - відправляти на тренінги. Поміркуйте над варіантами інших команд: за участі інженерних працівників, наприклад.

**Клас Наставник.** Клас буде моделювати сутність програміста наставника. Клас буде дочірним класом Programmer, унаслідує всі його поля, а також додаткове поле int Pupils, яке буде зберігати кількість стажерів яких він навчає. Salary буде перевизначимо, якщо Наставник має більше 3 стажерів, то формула його зарплати ActualyWorkedHours < minHours, якщо істина, то зарплата буде зменшена на 25%. І одночасно Pupils > 3, якщо істина, то (MoneyPerHour \* minHour)\*(1+ 0.05 \* за кожного учня). Print перевизначимо так, що він буде виводити те саме що програміст, але там де він виводив посаду в дужках додамо(Teacher). Конструктор = Конструктор Programer + Pupils

**Клас Стажер.** Клас буде моделювати сутність стажера, буде дочірним класом Programer, унаслідує всі поля і буде мати своє поле int TrainedHours, що буде показувти кількість відстажованих годин. Salary перевизначимо так, щоб: ActualyWorkedHours < minHours, якщо істина, то зарплата буде зменшена на 25% і додатково TrainedHours < 100, якщо істина то зарплата зменшена на 30%.

(Тобто він повинен відстажуватись 100 годин щоб стати звичайним програмістом). Print перевизначимо так, що він буде виводити те саме що програміст, але там де він виводив посаду в дужках додамо(Pupil). Конструктор = Конструктор Programer + TrainedHours

**Team**

**Клас Команда абстрактний.** Команда - абстрактний клас, який буде основою для подальшого наслідування. Буде мати такі поля:

1. Employer\*\* Team – масив в який ми будемо записувати різних працівників.
2. String Name – назва команди.
3. Employe\* Leader – лідер команди

І такі віртуальні, методи:

1. Void ShowLeader – метод, який буде показувати інформацію про лідера(Leader->print()).
2. Double TeamSalary – метод, який має виводити спільну зарплату команди.
3. Void ToProject – відправити команду на проєкт(Вивести на екран фразу Команда «Назва» була відправлена на проєкт).
4. Void ToTraining – відправити команду на тренінг(Вивести на екран фразу Команда «Назва» була відправлена на тренінг).
5. Void Task – метод, що буде показувати завдання команди.

1, 3 ,4 методи вже реалізовані, адже їх не потрібно буде перевизначати для підкласів.

Віртуальний деструктор, який буде звільнювати пам'ять.

Конструктор буде заповнювати всі поля заданими користувачем даними.

Команда буде заповнюватись через цикл, а Leader отримає адресу на 0 об’єкт в масиві Team.

Для кожного класу TeamSalary буде обчислюватись за допомогою циклу(Спочатку зарплата лідера, потім в циклі інших учасників і це все додати)!

**Клас Дуо.** Дочірний клас Team, наслідує від нього все. Конструктор буде викликати конструктор для Team, проте Employer\*\* Team[n] обов’язково n = 2!.

Name = “Duo”

ToProject перевизначимо “Команда була відправлена на проєкт»

TeamSalary перевизначимо так, що для двох учасників команди буде застосовано метод Salary, для 1 учасника(Лідера) буде Leader->Salary() \* 1.02, потім два значення будуть \ додаватись. Метод Task перевизначимо так, щоб він виводив на екран «Команда займається тестуванням». ToTraining перевизначимо – «Команда не може відправитись на тренінг оскільки в складі не має Керівника». ShowLeader тут буде виглядати ось так Leader(Team[0]->print().)

Конструктор буде додавати два об’єкта класу Programer.

**Клас Тріо**. Клас тріо буде наслідувати клас Дуо. Від нього наслідує все, проте Employer\*\* Team[n] обов’язково n = 3!.

Name = “Trio”

TeamSalary, замість 2 учасників сумуються 3, ToTraining, ShowLeader – не потрібно перевизначати, а ToTraining перевизначимо, щоб виводив на екран “Команда займається роботою з серверами”

Конструктор буде додавати два об’єкта класу Programer.

**Клас Менторство.** Клас буде утілювати сутність менторство, буде наслідуватись від Duo, унаслідує всі поля, і конструктор буде схожий, тільки замість Programer + Programer буде Intern + Mentor. Salary, ToShowLeader, ToTraining, не має потреби перевизначати, Task можемо перевизначити, щоб виводило «Ментор навчає стажера користуватись базами даних компанії»

**Клас Ланка.** Буде наслідувати клас Team, конструктор буде викликати конструктор Team, проти Employer\*\* Team[n] обов’язково n = 2!. Конструктор буде додавати 2 об’єкта класу Manager

Name = “Link”

Метод TeamSalary буде перевизначено ідентично, як у класі Дуо. ToProject перевизначимо «Команда не може відправитись на проєкт, оскільки у офісі є важливіші справи». ToTraining перевизначимо «Команда відправилась на тренінг керівницького складу». Task “Команда займається переговорами з замовниками»

**Клас Центр.** Оскільки ми вже маємо 2 керівників в Ланці будемо наслідувати Центр від Ланки. Конструктор буде викликати конструктор Team, проти Employer\*\* Team[n] обов’язково n = 3!. Конструктор буде додавати 2 об’єкта класу Manager і 1 ProjectManager

Name = “Center”

Перевизначити треба лише Task “Команда забезпечує якісний взаємозвяок між працівниками та керівниками”

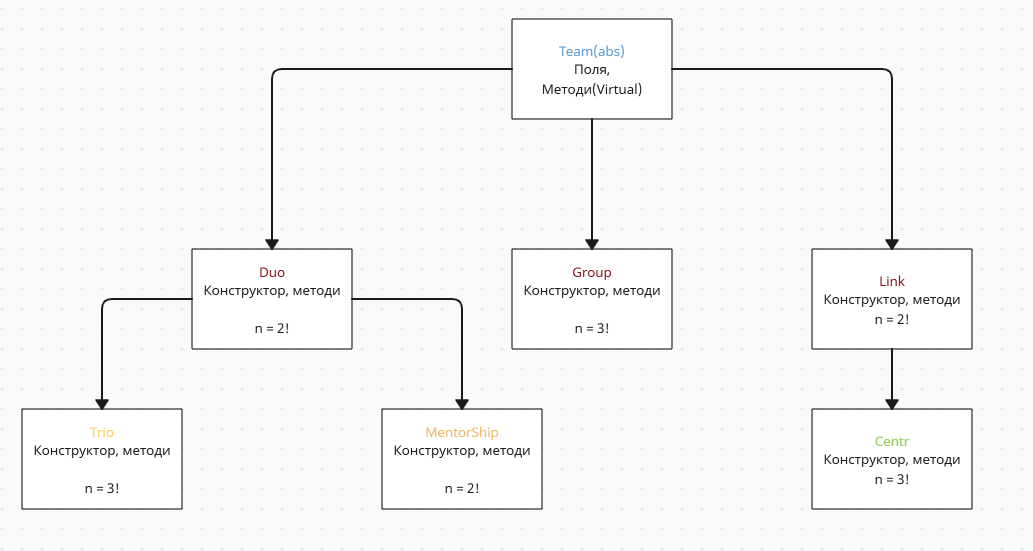
**Клас Група.** Цей клас будемо наслідувати від Team, адже тут змішані Керівник та Програмісти. Він успадкує все. Конструктор буде викликати конструктор Team, проти Employer\*\* Team[n] обов’язково n = 3!. Конструктор буде додавати 1 об’єкт класу Manager та 2 об’єкти класу Programer.

Name = “Group”.

TeamSalary перевизначимо так, як і в минулих класах. ToProject перевизначимо “Команда була відправлена на проєкт». ToTraining "Команда відправилась на загальний тренінг підвищення кваліфікації". Task “Команда займається покращенням сервісів компанії»

**Загальне. Для кожного класу ми будемо викликати конструктор Team додаючи потрібні нам данні, яких не має в самому конструкторі. Обчислення загальної зарплати у всіх однакове. Розміщення різних класів в одному масиві реалізовується завдяки масиву вказівників на конкретні об’єкти конкретних класів. Eployer є базовим абстракним класом для реалізації сутності всіх працівників, а Team для команд.**

**TEAM БЛОК СХЕМА**

****

**ДОДАТОК ДО ПОПЕРЕДЬОЇ БЛОК СХЕМИ ПРАЦІВНИКІВ**

**Зображення, що містить текст, схема, знімок екрана, ряд

Автоматично згенерований опис**